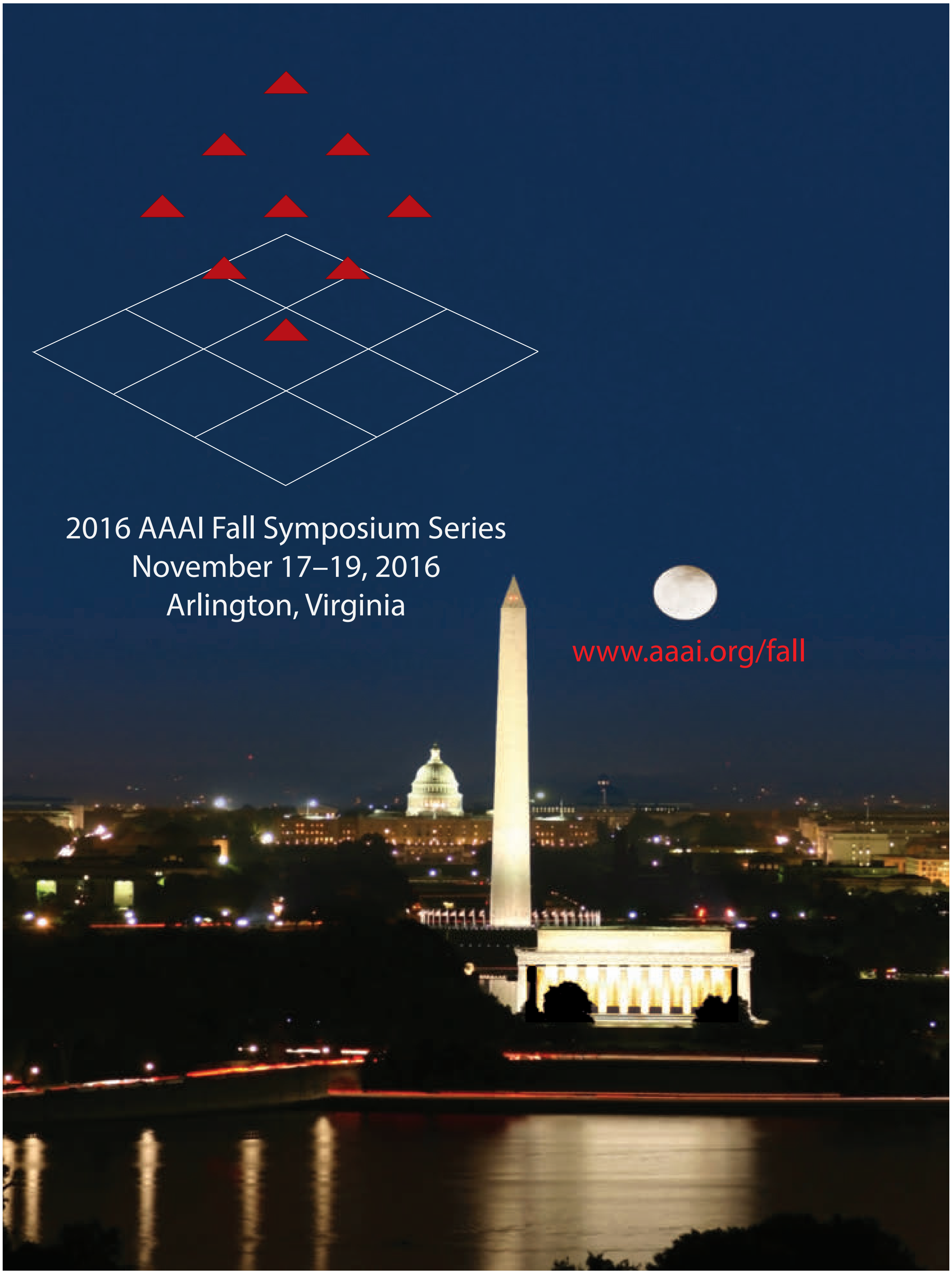
AI magazine

Volume 37 Number 3 Fall 2016



2016 AAAI Fall Symposium Series
November 17–19, 2016
Arlington, Virginia

www.aaai.org/fall

AI magazine

VOLUME 37, NUMBER 3 Fall 2016 ISSN 0738-4602

Cover: *Answer Set Programming by James Gary, New York, New York.*

The guest editors for the 2016 special issue on Answer Set Programming are Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczyński.

AI magazine

An Official Publication of the Association for the Advancement of Artificial Intelligence

aimagazine.org

ISSN 0738-4602 *(print)* ISSN 2371-9621 *(online)*

Submissions

Submissions information is available at http://aaai.org/ojs/index.php/aimagazine/information/authors. Authors whose work is accepted for publication will be required to revise their work to conform reasonably to *AI Magazine* styles. Author's guidelines are available at aaai.org/ojs/index.php/aimagazine/about/submissions#authorGuidelines. If an article is accepted for publication, a new electronic copy will also be required. Although *AI Magazine* generally grants reasonable deference to an author's work, the *Magazine* retains the right to determine the final published form of every article.

Calendar items should be posted electronically (at least two months prior to the event or deadline). Use the calendar insertion form at aimagazine.org. News items should be sent to the News Editor, *AI Magazine*, 2275 East Bayshore Road, Suite 160, Palo Alto, CA 94303. (650) 328-3123. Please do **not** send news releases via either e-mail or fax, and do **not** send news releases to any of the other editors.

Advertising

AI Magazine, 2275 East Bayshore Road, Suite 160, Palo Alto, CA 94303, (650) 328-3123; *Fax* (650) 321-4457. *Web:* aimagazine.org. Web-based job postings can be made using the form at https://www.aaai.org/Forms/jobs-submit.php.

Microfilm, Back, or Replacement Copies

Replacement copies (for current issue only) are available upon written request and a check for $10.00. Back issues are also available (cost may differ). Send replacement or back order requests to AAAI. Microform copies are available from ProQuest Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106. Telephone (800) 521-3044 or (734) 761-4700.

Copying Articles for Personal Use

Authorization to photocopy items for internal or personal use, or the internal or personal use of specific clients, or for educational classroom use, is granted by AAAI, provided that the appropriate fee is paid directly to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923. Telephone: (978) 750-8400. Fax: (978) 750-4470. Website: www.copyright.com. E-mail: info@copyright.com. This consent does **not** extend to other kinds of copying, such as for general distribution, resale, advertising, Internet or internal electronic distribution, or promotion purposes, or for creating new collective works. Please contact AAAI for such permission.

Address Change

Please notify AAAI eight weeks in advance of a change of address. Send electronically via MemberClicks or by e-mailing us to membership16@aaai.org.

Subscriptions

AI Magazine (ISSN 0738-4602) is published quarterly in March, June, September, and December by the Association for the Advancement of Artificial Intelligence (AAAI), 2275 East Bayshore Road, Suite 160, Palo Alto, CA 94303, telephone (650) 328-3123. *AI Magazine* is a direct benefit of membership in AAAI. Membership dues are $145.00 individual, $75.00 student, and $285.00 academic / corporate libraries. Subscription price of $50.00 per year is included in dues; the balance of your dues may be tax deductible as a charitable contribution; consult your tax advisor for details. Inquiries regarding membership in the Association for the Advancement of Artificial Intelligence should be sent to AAAI at the above address.

PERIODICALS POSTAGE PAID at Palo Alto CA and additional mailing offices. *Postmaster:* Change Service Requested. Send address changes to *AI Magazine*, 2275 East Bayshore Road, Suite 160, Palo Alto, CA 94303.

Editorial

Passing the Torch ...

David B. Leake

Photo courtesy Kevin Sapp.

This issue is my last as editor in chief of *AI Magazine*. It was a tremendous honor and privilege for me to lead *AI Magazine,* which I have done since 1999. It was a special pleasure to work with an outstanding team of volunteers—-the editorial board, column editors, and others—-and with the authors and reviewers, as well as with Mike Hamilton, managing editor, and the AAAI staff. As my administrative duties have expanded at Indiana University, where I am now executive associate dean of the School of Informatics and Computing, the time has come for me to pass the torch.

The editorship provided me with a birds-eye view of the field of AI that brought its stunning progress into focus. Research advances and the integration of AI into everyday life today give artificial intelligence unprecedented practical impact. The tremendous activity in diverse AI subareas underlines the importance of the magazine's mission of making advances across the field of AI accessible to the broad AI community.

Ashok Goel, of Georgia Tech, who is currently editor in chief elect, will become editor in chief with the winter issue. Goel is a professor of computer science and cognitive science in the School of Interactive Computing at Georgia Institute of Technology. He brings to the editorship both extensive leadership experience and broad scientific expertise in human-centered computing, artificial intelligence and cognitive science, with special focus on computational design, discovery, and creativity. In consultation with the *AI Magazine* Editorial Board, the AAAI Publications Committee, the AAAI Executive Council, and many leaders in the AI community, he has developed an inspiring and ambitious long-term vision for building on the magazine's strengths and launching new initiatives for coming issues.

I would like again to thank all those who have contributed to *AI Magazine.* during my tenure and to express my deep appreciation for those with whom I have worked as editor in chief. I welcome Ashok to the helm. The AAAI Executive Council has appointed me as editor emeritus and I look forward to continuing my engagement in that role.

It is an exciting time for *AI Magazine,* and for AI.

[illegible]

Artificial Intelligence and Interactive Digital Entertainment

The Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-16)

October 8–12, 2016

Embassy Suites by Hilton San Francisco Airport – Waterfront
Burlingame, California

Conference Chair
Nathan Sturtevant (University of Denver)

Colocated with the ACM SIGGRAPH Motion in Games (MIG) 2016 Conference

Workshops: October 8–9

www.aiide.org

Editorial

Answer Set Programming: An Introduction to the Special Issue

Gerhard Brewka, Thomas Eiter, Mirosław Truszczyński

■ *This editorial introduces answer set programming, a vibrant research area in computational knowledge representation and declarative programming. We give a brief overview of the articles that form this special issue on answer set programming and of the main topics they discuss.*

What is answer set programming, or ASP for short? Why is it drawing attention and continually gaining in acceptance as a computational problem-solving approach? These are the two key questions the seven articles in this special issue aim to answer. In a nutshell, ASP is a declarative problem solving paradigm — declarative, as all it requires users to do is to describe what the problem is, and not how to solve it. What distinguishes ASP from other declarative paradigms, like satisfiability (SAT) or constraint solving (CSP), is its underlying modeling language and the semantics involved. Problems are specified using logic programminglike rules, with some convenient extensions facilitating compact and readable problem descriptions. Sets of such rules, or answer set programs, come with an intuitive, well-defined and, by now, well-accepted semantics. This semantics has its roots in research in knowledge representation, in particular nonmonotonic reasoning, and avoids the pitfalls of earlier attempts such as the procedural semantics of Prolog based on negation as finite failure.

This semantics was originally called the stable-model semantics and was defined for normal logic programs only, that is, programs consisting of rules with a single atom in the head and any finite number of atoms, possibly preceded by default negation, not, in the body. Stable models were later generalized to broader classes of programs, where the semantics can no longer be defined in terms of sets of atoms, which is a natural representation of classical models. Instead, it was defined by means of some sets of literals. For this reason the term *answer set* was adopted as more adequate (although answer sets also have a straightforward interpretation as models, albeit three-valued ones).

Over the last decade or so, ASP has evolved into a vibrant and active research area that produced not only theoretical insights, but also highly effective and useful software tools and interesting and promising applications. The articles collected in this issue discuss these and other related aspects of

ASP: its theoretical underpinnings, language design, modeling methodology, principles behind processing answer set programs, development of fast processing software, applications, and some closely related formalisms to ASP of similar functionality and effectiveness.

Answer Sets and the Language of Answer Set Programming by Vladimir Lifschitz, who together with Michael Gelfond introduced stable models and answer sets back in 1988, lays the theoretical foundations of ASP by defining the basic language of ASP and the notion of answer sets.

The article by Tomi Janhunen and Ilkka Niemelä, The Answer Set Programming Paradigm, introduces the general methodology for representing and solving problems using ASP and, in several examples, illustrates the key elements of the process.

The two major steps in processing answer set programs are grounding and solving. Grounding eliminates the variables by constructing in a smart way the collection of relevant ground rules. Solving exploits search techniques similar to those used by SAT solvers to find the answer sets of the resulting ground program. These two steps are discussed by Benjamin Kaufmann, Nicola Leone, Simona Perri, and Torsten Schaub in the article Grounding and Solving in Answer Set Programming.

In the article Modeling and Language Extensions, Martin Gebser and Torsten Schaub give additional modeling examples and describe further useful language constructs. In particular, they discuss optimization methods, that is, methods to identify those answer sets that are most preferred according to some user-defined criteria.

Several excellent ASP systems are available now. In their paper Systems, Engineering Environments, and Competitions, Yuliya Lierler, Marco Maratea, and Francesco Ricca provide an overview of currently available and most commonly used answer set solvers and discuss methods behind them. They also discuss the state of the art in the emerging area of integrated answer set program development environments. Finally, they report on the series of answer set competitions that have been held regularly since 2007 and have had a major impact on the effectiveness of the ASP tools.

Next, Esra Erdem, Michael Gelfond, and Nicola Leone discuss applications. Their article Applications of Answer Set Programming demonstrates how some prototypical knowledge representation problems can be addressed in ASP, and then goes on to present real-world applications of ASP in robotics and bioinformatics, as well as several industry-grade ones.

Finally, the article First-Order Logic with Inductive Definitions for Model-Based Problem Solving, by Maurice Bruynooghe, Marc Denecker, and Mirek Truszczyński, complements this special issue by broadening the view on declarative problem solving. Not only logic programs under the answer set semantics but also other logic-based formalisms can be used to generate intended models. The authors illustrate this observation with the example of first-order logic extended with inductive definitions, a formalism closely related to ASP but, in some important respects, different from it.

We thank all authors who have contributed to this special issue on answer set programming. We appreciate their efforts to make it coherent and informative. We hope you will find the articles in the collection interesting and helpful. In addition to a good presentation of ASP topics, the articles provide many references to further reading that can serve as an excellent entry point to the rich literature on ASP. Finally, we hope that this special issue gives an idea why the field attracts so much attention and carries so much promise. And if you think ASP might be the way to tackle the problem that has just landed on your desk, just give it a try!

Acknowledgements

Gerhard Brewka was partly supported by DFG, research unit 1513. Thomas Eiter acknowledges support by the Austrian Science Fund (FWF), projects P26471 and P24090.

Gerhard Brewka is a professor of intelligent systems at Leipzig University, Germany. His research focuses on knowledge representation, in particular nonmonotonic reasoning, logic programming, preference and inconsistency handling, and computational models of argumentation. He served as president of EurAI (formerly ECCAI), the European Association of AI, and of Knowledge Representation Inc. He is a member of the IJCAI Board of Trustees and conference chair of IJCAI-16. In 2002, Brewka became a EurAI Fellow. He was an editor and associate editor of the journals *Artificial Intelligence Research* and *Artificial Intelligence*. He is now on the Editorial Board of *AI Magazine*.

Thomas Eiter is a professor of knowledge-based systems in the Faculty of Informatics at Technische Universität Wien, Austria. He worked in different fields of computer science and AI, but his main area is knowledge representation and reasoning, where he has published extensively; his current interests are declarative problem solving and computational reasoning methods. Eiter was on the project team that built the DLV system, an ASP solver that was state of the art through many years. Eiter has served on on a number of editorial boards, including *Artificial Intelligence Journal* and the *Journal of Artificial Intelligence Research*, and has served on several steering committees, including those of the Association of Logic Programming and Knowledge Representation and Reasoning Inc. (where he was president from 2014-2015). He has also chaired various conferences, including, most recently, KR 2014 and ICLP 2015. Eiter was elected a Fellow of the European Association of AI (EurAI, formerly ECCAI) in 2006 and corresponding member of the Austrian Academy of Sciences in 2007.

Mirosław Truszczyński is a professor of computer science at the University of Kentucky. His research interests include knowledge representation, nonmonotonic reasoning, logic programming, and constraint satisfaction. He has published more than 180 technical papers, coauthored a research monograph on nonmonotonic logics, and edited ten article collections and conference proceedings. His paper Stable Logic Programming, a joint work with Victor Marek, helped launch the field of answer set programming. Truszczyński served on the executive committee of the Association of Logic Programming, was chair of the steering committee of Nonmonotonic Reasoning Workshops, and was president of Knowledge Representation Inc. He served as an editor and associate editor on the boards of *Journal of Artificial Intelligence Research* and *Artificial Intelligence Journal*. He is now editor-in-chief of *Theory and Practice of Logic Programming* and an associate editor of *AI Communications*. He is an AAAI Fellow.

Answer Sets and the Language of Answer Set Programming

Vladimir Lifschitz

■ *Answer set programming is a declarative programming paradigm based on the answer set semantics of logic programs. This introductory article provides the mathematical background for the discussion of answer set programming in other contributions to this special issue.*

Answer set programming (ASP) is a declarative programming paradigm introduced by Marek and Truszczyński (1999) and Niemelä (1999). It grew out of research on knowledge representation (van Harmelen, Lifschitz, and Porter 2008), nonmonotonic reasoning (Ginsberg and Smith 1988), and Prolog programming (Sterling and Shapiro 1986). Its main ideas are described in the article by Janhunen and Niemelä (2016) and in other contributions to this special issue.

In this introductory article my goal is to discuss the concept of an answer set, or stable model, which defines the semantics of ASP languages. The answer sets of a logic program are sets of atomic formulas without variables ("ground atoms"), and they were introduced in the course of research on the semantics of negation in Prolog. For this reason, I will start with examples illustrating the relationship between answer sets and Prolog and the relationship between answer set solvers and Prolog systems. Then I will review the mathematical definition of an answer set and discuss some extensions of the basic language of ASP.

Prolog and Negation as Failure

Simple Prolog rules can be understood as rules for generating new facts, expressed as ground atoms, from facts that are given or have been generated earlier. For example, the Prolog program

```
p(1). p(2). p(3).
q(2). q(3). q(4).
r(X) :- p(X), q(X).
```

consists of six facts ("1, 2, and 3 have property p; 2, 3, and 4 have property q") and a rule: for any value of X, r(X) can be generated if p(X) and q(X) are given or have been generated earlier.[1] In response to the query ?- r(X) a typical Prolog system will return two answers, first X = 2 and then X = 3.

Let us call this program Π_1 and consider its modification Π_2, in which the "negation as failure" symbol \+ is inserted in front of the second atom in the body of the rule:

```
p(1). p(2). p(3).
q(2). q(3). q(4).
r(X) :- p(X), \+ q(X).
```

The modified rule allows us, informally speaking, to generate r(X) if p(X) has been generated, assuming that any attempt to generate q(X) using the rules of the program would fail. Given the modified program and the query ?- r(X) Prolog returns one answer, X = 1.

What is the precise meaning of conditions of this kind, "any attempt to generate … using the rules of the program would fail"? This is not an easy question, because the condition is circular: it attempts to describe when a rule *R* "fires" (can be used to generate a new fact) in terms of the set of facts that can be generated using all rules of the program, including *R* itself. Even though this formulation is vague, it often allows us to decide when a rule with negation is supposed to fire. It is clear, for instance, that there is no way to use the rules of Π_2 to generate q(1), because this atom is not among the given facts and it does not match the head of any rule of Π_2. We conclude that the last rule of Π_2 can be used to generate r(1).

But there are cases when the circularity of the above description of negation as failure makes it confusing. Consider the following program Π_3, obtained from Π_2 by replacing the facts in the second line with a rule:

```
p(1). p(2). p(3).
q(3) :- \+ r(3).
r(X) :- p(X), \+ q(X).
```

The last rule justifies generating r(1) and r(2), there can be no disagreement about this. But what about r(3)? The answer is yes if any attempt to use the rules of the program to generate q(3) fails. In other words, the answer is yes if the second rule of the program does not fire. But does it? It depends on whether the last rule can be used to generate r(3) — the question that we started with.

The first precise semantics for negation as failure was proposed by Clark (1978), who defined the process of program completion — a syntactic transformation that turns Prolog programs into first-order theories. The definition of a stable model, or answer set, proposed ten years later (Gelfond and Lifschitz 1988), is an alternative explanation of the meaning of Prolog rules with negation. It grew out of the view that an answer set of a logic program describes a possible set of beliefs of an agent associated with this program; see the paper by Erdem, Gelfond, and Leone (2016) in this special issue. Logic programs are similar, in this sense, to autoepistemic theories (Moore 1985) and default theories (Reiter 1980).[2] The definition of an answer set, reproduced in this article, adapts the semantics of default logic to the syntax of Prolog.

We will see that program Π_3, unlike Π_1 and Π_2, has two answer sets. One answer set authorizes including X=3 as an answer to the query ?- q(X) but not as an answer to the query ?- r(X); according to the other answer set, it is the other way around. In this sense, program Π_3 does not give an unambiguous specification for query answering. Programs with several answer sets are "bad" Prolog programs.

In answer set programming, on the other hand, programs with several answer sets (or without answer sets) are quite usual and play an important role, like equations with several roots (or without roots) in algebra.

Answer Set Solvers

How does the functionality of answer set solvers compare with Prolog?

Each of the programs Π_1, Π_2, and Π_3 will be accepted as a valid input by an answer set solver, except that the symbol \+ for negation as failure should be written as not. Thus Π_2 becomes, in the language of answer set programming,

```
p(1). p(2). p(3).
q(2). q(3). q(4).
r(X) :- p(X), not q(X).
```

and Π_3 will be written as

```
p(1).  p(2).  p(3).
q(3) :- not r(3).
r(X) :- p(X), not q(X).
```

Unlike Prolog systems, an answer set solver does not require a query as part of the input. The only input it expects is a program, and it outputs the program's answer sets. For instance, given program Π_1, it will find the answer set

```
p(1) p(2) p(3) q(2) q(3) q(4) r(2) r(3)
```

From the perspective of Prolog, this is the list of all ground queries that would generate the answer yes for this program. For program Π_2, the answer set

```
p(1) p(2) p(3) q(2) q(3) q(4) r(1)
```

will be calculated. Given Π_3 as input, an answer set solver will find two answer sets:

```
Answer:  1
p(1) p(2) p(3) q(3) r(1) r(2)
Answer:  2
p(1) p(2) p(3) r(3) r(1) r(2)
```

Definition of an Answer Set: Positive Programs

I will review now the definition of an answer set, beginning with the case when the rules of the program do not contain negation, as in program Π_1 discussed earlier. By definition, such a program has a unique answer set, which is formed as follows.

First, we *ground* the program by substituting specific values for variables in its rules in all possible ways. The result will be a set of rules of the form

$$A_0 \text{ :- } A_1, \ldots, A_n. \tag{1}$$

where each A_i is a ground atom. (We think of "facts," such as p(1) in Π_1, as rules of form (1) with n = 0 and with the symbol :- dropped.) For instance, grounding turns Π_1 into

```
p(1). p(2). p(3).
q(2). q(3). q(4).
r(1) :- p(1), q(1).
r(2) :- p(2), q(2).
r(3) :- p(3), q(3).
r(4) :- p(4), q(4).
```

The answer set of the program is the smallest set S of ground atoms such that for every rule (1) obtained by grounding, if the atoms $A_1, \ldots, A_n$ belong to S then the head A_0 belongs to S too.

For instance, in the case of program Π_1 this set S includes (1) the facts in the first two lines of the grounded program, (2) the atom r(2), because both atoms in the body of the rule with the head r(2) belong to S, and (3) the atom r(3), because both atoms in the body of the rule with the head r(3) belong to S.

The following program contains two symbolic constants, block and table:

```
number(1). number(2). number(3).
location(block(N)) :- number(N).
location(table).
```

Grounding turns the second rule into

```
location(block(1)) :- number(1).
location(block(2)) :- number(2).
location(block(3)) :- number(3).
```

The answer set of this program consists of the atoms

```
number(1)  number(2)  number(3)  location(block(1))
location(block(2))  location(block(3))  location(table)
```

Definition of an Answer Set: Programs with Negation

In the general case, when the rules of the given program may contain negation, grounding gives a set of rules of the form

$$A_0 \text{ :- } A_1, \ldots, A_m, \text{ not } A_{m+1}, \ldots, \text{ not } A_n. \tag{2}$$

where each A_i is a ground atom. (To simplify notation, we showed all negated atoms at the end.) For instance, the result of grounding Π_2 is

```
p(1). p(2). p(3).
q(2). q(3). q(4).
r(1) :- p(1), not q(1).
r(2) :- p(2), not q(2).
r(3) :- p(3), not q(3).
r(4) :- p(4), not q(4).
```

To decide whether a set S of ground atoms is an answer set, we form the *reduct* of the grounded program with respect to S, as follows. For every rule (2) of the grounded program such that S does not contain any of the atoms $A_{m+1}, \ldots, A_n$, we drop the negated atoms from (2) and include the "positive part" (1) of the rule in the reduct. All other rules are dropped from the grounded program altogether. Since the reduct consists of rules of form (1), we already know how to calculate its answer set. If the answer set of the reduct coincides with the set S that we started with then we say S is an answer set of the given program.

For instance, to check that the set

$$\{p(1),\ p(2),\ p(3),\ q(2),\ q(3),\ q(4),\ r(1)\} \tag{3}$$

is an answer set of Π_2, we calculate the reduct of the grounded program with respect to this set. The reduct is

```
p(1). p(2). p(3).
q(2). q(3). q(4).
r(1) :- p(1).
```

(The last three rules of the grounded program are not included in the reduct because set (3) includes q(2), q(3), and q(4).) The answer set of the reduct is indeed the set (3) that we started with. If we repeat this computation for any set S of ground atoms other than (3) then the result may be a subset of S, or a superset of S, or it may partially overlap with S, but it will never coincide with S. Consequently (3) is the only answer set of Π_2.

Intuitively, the reduct of a program with respect to S consists of the rules of the program that "fire" assuming that S is exactly the set of atoms that can be generated using the rules of the program. If the answer set of the reduct happens to be exactly S then we conclude that S was a "good guess."

The concept of an answer set can be defined in many other, equivalent ways (Lifschitz 2010).

Extensions of the Basic Language

Arithmetic. Rules may contain symbols for arithmetic operations and comparisons, for instance:

```
p(1). p(2).
q(1). q(2).
r(X+Y) :- p(X), q(Y), X<Y.
```

The answer set of this program is

```
p(1) p(2) q(1) q(2) r(3)
```

(In view of the condition X < Y in the body, the only values substituted for the variables in the process of grounding are X = 1, Y = 2.)

Disjunctive Rules (Gelfond and Lifschitz 1991). The head of a rule may be a disjunction of several atoms (often separated by bars or semicolons), rather than a single atom. For instance, the rule

```
p(1) | p(2).
```

instructs the solver to include p(1) or p(2) in each answer set. The answer sets of this one-rule program are

```
Answer: 1
p(1)
Answer: 2
p(2)
```

Choice Rules (Niemelä and Simons 2000). Enclosing the list of atoms in the head in curly braces represents the "choice" construct: choose in all possible ways which atoms from the list will be included in the answer set. For instance, the one-rule program

```
{ p(1) ; p(2) }.
```

has 4 answer sets:

```
Answer:  1

Answer: 2
p(1)
 Answer: 3
p(2)
Answer: 4
p(1)   p(2)
```

A choice rule may specify bounds on the number of atoms that are included. The lower bound is shown to the left of the expression in braces, and the upper bound to the right. For instance, the one-rule program

```
1 { p(1) ; p(2) }.
```

has 3 answer sets — answers 2–4 from the previous example. The one-rule program

```
{ p(1) ; p(2) } 1.
```

has 3 answer sets as well — answers 1–3.

Constraints. A constraint is a disjunctive rule that has 0 disjuncts in the head, so that it starts with the symbol :-. Adding a constraint to a program eliminates the answer sets that satisfy the body of the constraint. For instance, the answer sets of the program

```
{ p(1) ; p(2) }.
:- p(1), not p(2).
```

are answers 1, 3, and 4 from the preceding list. Answer 2 violates the constraint, because it includes p(1) and does not include p(2).

Classical Negation (Gelfond and Lifschitz 1991). Atoms in programs and in answer sets can be preceded by the "classical negation" sign (–) that should be distinguished from the negation as failure symbol (not). This is useful for representing incomplete information. For instance, the answer set

```
p(a) p(b) -p(c) q(a) -q(c)
```

can be interpreted as follows: a and b have property p, and c does not; a has property q, and c does not; whether b has property q we do not know. A rule of the form

$– A$:- not A.

containing classical negation in the head and negation as failure in the body expresses the "closed world assumption" for the atom A: A is false if there is no evidence that A is true. The rule

```
p(T+1) :- p(T), not -p(T+1).
```

expresses the "frame default" (Reiter 1980) in the language of answer set programming: if p was true at time T and there is no evidence that p became false at time T + 1 then p was true at time T + 1.

Input languages of many answer set solvers include other useful extensions of the basic language, such as aggregates (Faber, Leone, and Pfeifer 2004), weak constraints (Buccafuri, Leone, and Rullo 1997), consistency-restoring rules (Balduccini and Gelfond 2003), and P-log rules (Chitta, Gelfond, and Rushton 2009).

Extending the Definition of an Answer Set

The problem of extending the definition of an answer set to additional constructs, such as those reviewed in the previous section, can be approached in several ways. One useful idea is to treat expressions in the bodies and heads of rules as logical formulas written in alternative notation. For instance, we can think of the list in the body of (2) as a conjunction of literals:

$$A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n.$$

A choice expression $\{A_1; \ldots ; A_n\}$ can be treated as a conjunction of "excluded middle" formulas:

$$(A_1 \vee \neg A_1) \wedge \cdots \wedge (A_n \vee \neg A_n)$$

(Ferraris and Lifschitz 2005). Under this approach, the rules of a grounded program are expressions of the form $F \leftarrow G$, where F and G are formulas built from ground atoms using conjunction, disjunction, and negation.[3]

The definition of the reduct was extended to such rules by Lifschitz, Tang, and Turner (1999). In the process of constructing the reduct of a rule $F \leftarrow G$ with respect to a set S of ground atoms, every subformula that begins with negation is replaced by a logical constant: by true if it is satisfied by S, and by false otherwise.

Gebser et al. (2015) defined the syntax and semantics of many constructs implemented in the solver CLINGO using a generalization of this approach that allows the formulas F and G to contain implication, and that allows conjunctions and disjunctions in F and G to be infinitely long.

Acknowledgements

Thanks to Gerhard Brewka, Martin Gebser, Michael Gelfond, Tomi Janhunen, Amelia Harrison, Amanda Lacy, Yuliya Lierler, Nicola Leone, and Mirek Truszczyński for comments on a draft of this article. This research was partially supported by the National Science Foundation under Grant IIS-1422455.

Notes

1. In Prolog programs, a period indicates the end of a rule. Capitalized identifiers are used as variables. The symbol :- reads "if"; it separates the "head" of the rule (in this case, the atom r(X)) from its "body" (the pair of atoms p(X), q(X)). Answer set programming inherited from Prolog these syntactic conventions and terminology.

2. The relationship between Prolog and autoepistemic logic was described by Gelfond (1987).

3. A more radical version of this view is to think of the whole rule $F \leftarrow G$ as a propositional formula — as the implication $G \rightarrow F$ "written backwards" (Ferraris 2005). It is also possible to avoid the reference to grounding in the definition of an answer set and to treat rules with variables as first-order formulas (Ferraris, Lee, and Lifschitz 2011).

References

Balduccini, M., and Gelfond, M. 2003. Logic Programs with Consistency-Restoring Rules. Paper presented at the 2003 AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning, 24–26 March, Stanford University, Stanford CA.

Buccafuri, F.; Leone, N.; and Rullo, P. 1997. Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering* 12(5): 845–860. dx.doi.org/10.1109/69.877512

Chitta, B.; Gelfond, M.; and Rushton, N. 2009. Probabilistic Reasoning with Answer Sets. *Theory and Practice of Logic Programming* 9(1): 57–144. dx.doi.org/10.1017/S1471068408003645

Clark, K. 1978. Negation as Failure. In *Logic and Data Bases,* ed. H. Gallaire and J. Minker. New York: Plenum Press. 293–322. dx.doi.org/10.1007/978-1-4684-3384-5_11

Erdem, E.; Gelfond, M.; and Leone, N. 2016. Applications of ASP. *AI Magazine* 37(3).

Faber, W.; Leone, N.; and Pfeifer, G. 2004. Recursive Aggregates in Disjunctive Logic Programs: Semantics and Complexity. In *Logics in Artificial Intelligence, 9th European Conference* (JELIA). Lecture Notes in Computer Science 3229. Berlin: Springer.

Ferraris, P. 2005. Answer Sets for Propositional Theories. In *Proceedings of Logic Programming and Nonmonotonic Reasoning, 8th International Conference* (LPNMR 2005). Lecture Notes in Computer Science 3662, 119–131. Berlin: Springer. dx.doi.org/10.1007/11546207_10

Ferraris, P., and Lifschitz, V. 2005. Weight Constraints as Nested Expressions. *Theory and Practice of Logic Programming* 5(1–2): 45–74. dx.doi.org/10.1017/S1471068403001923

Ferraris, P.; Lee, J.; and Lifschitz, V. 2011. Stable Models and Circumscription. *Artificial Intelligence* 175(1): 236–263. dx.doi.org/10.1016/j.artint.2010.04.011

Gebser, M.; Harrison, A.; Kaminski, R.; Lifschitz, V.; and Schaub, T. 2015. Abstract Gringo. *Theory and Practice of Logic Programming* 15(4–5): 449–463. dx.doi.org/10.1017/S1471068415000150

Gelfond, M. 1987. On Stratified Autoepistemic Theories. In *Proceedings of 6th National Conference on Artificial Intelligence* (AAAI), 207–211. San Mateo, CA: Morgan Kaufmann, Publishers.

Gelfond, M., and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Proceedings of International Logic Programming Conference and Symposium,* ed. R. Kowalski and K. Bowen, 1070–1080. Cambridge, MA: The MIT Press. dx.doi.org/10.1007/BF03037169

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9(3–4): 365–385.

Ginsberg, M., and Smith, D. 1988. Reasoning About Action I: A Possible World Approach. *Artificial Intelligence* 35(3): 165–195. dx.doi.org/10.1016/0004-3702(88)90011-2

Janhunen, T., and Niemelä, I. 2016. The Answer Set Programming Paradigm. *AI Magazine* 37(3).

Lifschitz, V. 2010. Thirteen Definitions of a Stable model. In *Fields of Logic and Computation: Essays Dedicated to Yuri Gurevich on the Occasion of his 70th Birthday.* Lecture Notes in Computer Science Volume 6300, 488–503. Berlin: Springer. dx.doi.org/10.1007/978-3-642-15025-8_24

Lifschitz, V.; Tang, L. R.; and Turner, H. 1999. Nested Expressions in Logic Programs. *Annals of Mathematics and Artificial Intelligence* 25(3–4): 369–389. dx.doi.org/10.1023/A:1018978005636

Marek, V., and Truszczyński, M. 1999. Stable Models and an Alternative Logic Programming Paradigm. In *The Logic Programming Paradigm: A 25-Year Perspective,* ed. K. Apt, V. W. Marek, M. Truszczyński, D. S. Warren. Berlin: Springer Verlag. 375–398. dx.doi.org/10.1007/978-3-642-60085-2_17

Moore, R. 1985. Semantical Considerations on Nonmonotonic Logic. *Artificial Intelligence* 25(1):75–94. dx.doi.org/10.1016/0004-3702(85)90042-6

Niemelä, I. 1999. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. *Annals of Mathematics and Artificial Intelligence* 25(3–4): 241–273. dx.doi.org/10.1023/A:1018930122475

Niemelä, I., and Simons, P. 2000. Extending the Smodels System with Cardinality and Weight Constraints. In *Logic-Based Artificial Intelligence,* ed. J. Minker. Dordrecht, The Netherlands: Kluwer. 491–521. dx.doi.org/10.1007/978-1-4615-1567-8_21

Reiter, R. 1980. A Logic for Default Reasoning. *Artificial Intelligence* 13(1–2): 81–132. dx.doi.org/10.1016/0004-3702(80)90014-4

Sterling, L., and Shapiro, E. 1986. The Art of Prolog: Advanced Programming Techniques. Cambridge, MA: The MIT Press.

van Harmelen, F.; Lifschitz, V.; and Porter, B., eds. 2008. *Handbook of Knowledge Representation.* Amsterdam: Elsevier.

Vladimir Lifschitz is a professor of computer science at the University of Texas at Austin. His research interests are in computational logic and knowledge representation.

James Crawford, Conference Chair
G. Michael Youngblood, Conference Cochair

The Twenty-Ninth Annual Conference on Innovative Applications of Artificial Intelligence

(IAAI-17)

February 4–9, 2017

San Francisco, California USA

Please Join Us!

www.aaai.org/iaai17

The Answer Set Programming Paradigm

Tomi Janhunen, Ilkka Niemelä

■ *In this article, we give an overview of the answer set programming paradigm, explain its strengths, and illustrate its main features in terms of examples and an application problem.*

Answer set programming (ASP, for short) is a declarative programming paradigm for solving search problems and their optimization variants. In ASP a search problem is modeled as a set of statements (a program) in a logic programming type of a language in such a way that the answer sets (models) of the program correspond to the solutions of the problem. The paradigm was first formulated in these terms by Marek and Truszczyński (1999) and Niemelä (1999). The ASP paradigm has its roots in knowledge representation and nonmonotonic logics research as described by Marek, Niemelä, and Truszczyński (2011) in a historic account on the development of the paradigm. A recent and more technical overview of ASP has been contributed by Brewka, Eiter, and Truszczyński (2011).

The ASP paradigm is most widely used with the formalism of logic programming under the semantics given by answer sets (Gelfond and Lifschitz 1988, 1990). The term *answer sets* was proposed by Gelfond and Lifschitz (1991) for sets of literals, by which programs in an extended syntax are to be interpreted where the classical negation operator and disjunctions of literals are allowed in the heads of program rules. Lifschitz's article (2016) in this special issue gives an introduction to the notion of an answer set and the language of ASP, as well as a comparison to Prolog systems. An alternative approach to ASP has been to use directly first-order logic as the basis and extend it with inductive definitions. The details can be found in the articles by Denecker and Vennekens (2014), Denecker and Ternovska (2008), East and Truszczyński (2006), and the one by Bruynooghe et al. (2016) in this issue of *AI Magazine*.

A main reason for the increasing interest in ASP is the availability of fast software tools that make it possible to tackle problems of practical importance. Most of the current software tools employ two steps commonly referred to as grounding and solving, reflecting the definition of answer sets for programs with variables (Lifschitz 2016). The idea is to separate concerns so that the grounding phase takes care of the evaluation of more complicated data structures and variable instantiations using logic programming and deductive database techniques, and then the solving phase focuses on search for answer sets for a much simpler type of programs by employing advanced search methods. The articles by Kaufmann et al. (2016) and by Gebser and Schaub (2016) in this issue provide more information on the solving and grounding techniques.

There is a growing number of successful applications of ASP including molecular biology (Gebser et al. 2010a, 2010b), decision support system for space shuttle controllers (Balduccini, Gelfond, and Nogueira 2006), phylogenetic inference (Erdem 2011, Koponen et al. 2015), product configuration (Soininen and Niemelä 1998, Finkel and O'Sullivan 2011) and repair of web-service work flows (Friedrich et al. 2010). Erdem, Gelfond, and Leone (2016) give an account of the applications of ASP in this issue.

On the one hand, ASP is closely related to logic programming and Prolog and, on the other hand, to constraint programming (CP), propositional satisfiability (SAT), and linear or integer programming (LP/IP). Unlike Prologlike logic programming ASP is fully declarative and neither the order of rules in a program nor the order of literals in the rules matter. Moreover, Prolog systems are tailored to find proofs or answer substitutions to individual queries whereas ASP systems are finding answer sets corresponding to complete solutions to a problem instance. The basic idea in ASP is very close to the paradigm of CP, SAT, or LP/IP where problems are represented by constraints and where systems are tailored to find satisfying variable assignments corresponding to complete solutions.

However, there are significant differences. The ASP paradigm allows for a very systematic approach to problem representation through uniform encodings where the problem statement can be developed independently of data on a particular instance. This leads to a large degree of elaboration tolerance. The ASP approach enables structured representation of problems where more complicated constraints are composed of simpler ones using rules. On the other hand, rules enable one to encode conditions that are challenging (like representing disjunctive constraints or other basic relational operations on constraints) or not available at all (like recursive constraints) when comparing to CP or LP/IP paradigms. Because of these properties ASP allows for incremental development of an application and supports well rapid prototyping.

The goal of this article is to provide an up-to-date overview of the ASP paradigm and illustrate its usage with examples as well as a more comprehensive application problem. We proceed as follows. In the next section, we explain the fundamental ideas of the ASP paradigm. The use of the paradigm and its main features are then illustrated by developing ASP encodings for an application problem step by step. The application considered in this article is about designing a locking scheme for a building so that certain safety requirements are met. Having introduced the basic paradigm, we briefly address main ways to implement ASP — either using native answer set solvers or translators that enable the use of solver technology from neighboring disciplines. We end the article with a summary and discussion of future prospects. In addition, we illustrate the potential computational hardness of our application problem by explaining its connection to the NP-complete decision problem Exact-3-SAT.

Basic ASP Paradigm

The conceptual model of the ASP paradigm is depicted in figure 1. We start by explaining how to understand search problems at an abstract level and then illustrate how ASP is typically employed to solve such problems using the approach illustrated in the figure. Finally, we address a number of features and attractive properties of the paradigm.

Problem Solving

The ASP paradigm provides a general-purpose methodology for solving search and optimization problems encountered in many real-world applications. To get started, the key step is to identify and formalize the problem to be solved, that is, to work out a problem statement. Typically this consists of clarifying what the potential solutions of the problem are like and then setting the conditions that solutions should satisfy. Solving the problem means that given the data on an instance of the problem we should find one or more solutions that satisfy the given conditions (see the topmost arrow in figure 1). For illustration, we use the task of finding a seating arrangement for a dinner as the first simple example. The respective problem statement could read as formulated next.

Example 1 (Seating Arrangement Problem)

A certain group of people, say persons $p_1, \dots, p_n$, are invited for dinner. There are tables $t_1, \dots, t_k$ with the respective capacities $c_1, \dots, c_k$ available for seating such that $c_1 + \dots + c_k >= n$. The host has some prior

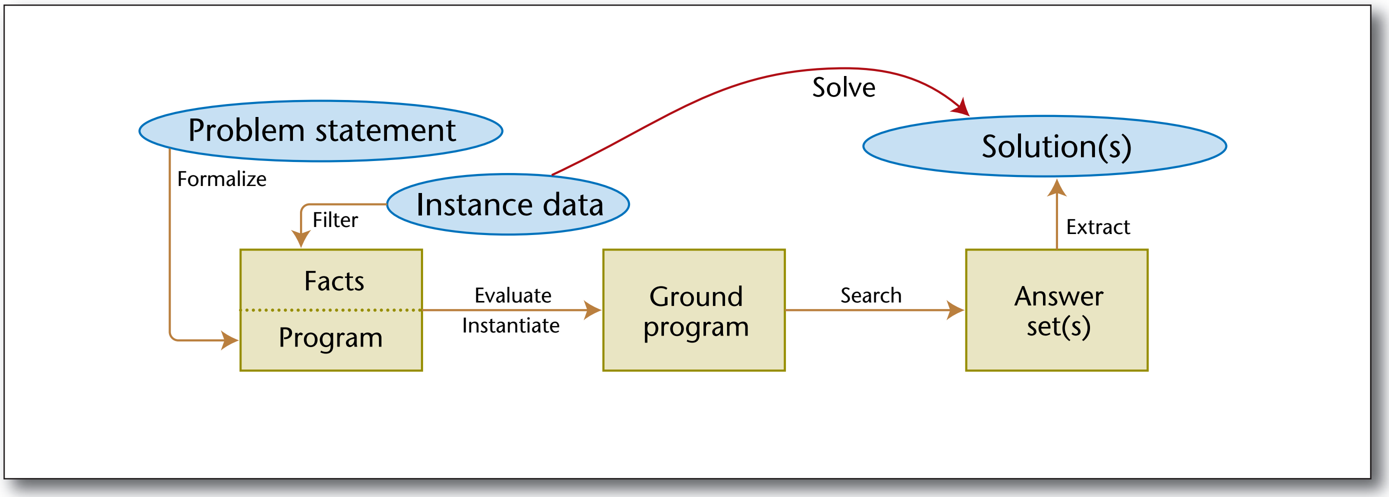


Figure 1. Conceptual Model of the ASP Paradigm.

knowledge about the relationships of the guests: there are both friends and enemies among the invitees. This information should be taken into account when designing the arrangement. A solution to this problem is a mapping $s(p_i) = t_j$ of persons p_i to tables t_j so that the mutual relationships are respected.

The problem statement in example 1 uses mathematical symbols to abstract the details of the problem such as the number and the identity of persons involved and the collection of tables available for seating. This reflects an important methodological feature, namely the separation of instance data from the actual problem statement. The point is that the problem can be stated without listing all details for a particular instance of the problem. In case of the seating arrangement problem, the instance data would consist of the names of invitees together with lists of tables and their capacities, and the pairs of persons who are known to be either friends or enemies. More concretely put, suppose that we have a group of 20 people: Alice, Bob, John, and others. There are four tables, seating 7, 6, 5, and 4 people, respectively. Moreover, we know that Alice likes Bob, Bob likes John, and so on. Given all such pieces of information, the goal is (1) to find at least one solution that fulfills the criteria set in the problem statement of example 1, or (2) to show that no solution exists. Given what we know so far, we can expect solutions where Alice, Bob, and John are seated together at one of the four tables available. However, if we state additionally that Alice and John dislike each other, for instance, the seating problem instance under consideration has no solutions.

ASP Encoding

But how do we achieve the aforementioned goal using ASP and get the problem solved? As suggested by figure 1, we should formalize the problem statement by writing down a (logic) program. Before we can really do this, we should have a basic understanding of syntax, also introduced in the article by Lifschitz (2016) in this issue. In ASP, programs consist of rules, that is, statements of the form

head :– body_1, body_2, ..., body_n.

The intuitive reading of this rule is that the head can be inferred if (and only if) the body conditions body_1, body_2, ..., body_n have been inferred by any other rules in the program. The conditions in the rule are either *atomic statements* (also called *atoms*) like seat(a,1) for Alice being seated at table 1, or *count-bounded* sets of atoms

l { atom_1; ...; atom_k } u

where at least l but at most u atoms among atom_1, ..., atom_k should be inferable. The cardinality constraint above can also be expressed in terms of a counting aggregate

#count{atom_1; ...; atom_k}

where appropriate bounds can be incorporated using relation symbols <, <=, >, >=, and =. Atoms can also be negated using the operator not for default negation. A rule with an empty body (n = 0) stands for a fact whose head holds unconditionally. As a further special case, a rule without a head stands for a constraint whose body body_1, body_2, ..., body_n must not be satisfied. In this article, we do not consider extensions of rules by classical negation nor disjunctions in rule heads (Gelfond and Lifschitz 1991).

We are now ready to describe typical steps in writing down a program in ASP, resulting in an encoding[1] given as listing 1. First, we have to decide how to represent the instance data. Sometimes this requires some form of filtering in order to identify which pieces of information are relevant in view of solving the problem. This is easy for the seating problem. The persons involved are listed in line 2 using predicate symbol person/1 and constant symbols a, b, j, ... as abbreviations for the names of persons in question.

```
 1  % Instance
 2  person(a). person(b). person(j).
 3  likes(a,b). likes(b,j). ...
 4  dislikes(a,j). dislikes(j,a). ...
 5  tbl(1,7). tbl(2,6). tbl(3,5). tbl(4,4).
 6
 7  % Rules and constraints
 8  1 { seat(P,T): tbl(T,_) } 1 :- person(P).
 9  :- #count{seat(P,T): person(P)}>C, tbl(T,C).
10  :- likes(P1,P2), seat(P1,T1), seat(P2,T2),
11     person(P1), person(P2),
12     tbl(T1,_), tbl(T2,_), T1 != T2.
13  :- dislikes(P1,P2), seat(P1,T), seat(P2,T),
14     person(P1), person(P2), tbl(T,_).
```

Listing 1. Encoding the Seating Problem in ASP.

Predicates likes/2 and dislikes/2 are used in lines 3–4 to represent (potentially incomplete)[2] information concerning friendship and dislike, respectively. Finally, the identities and capacities of tables are declared by the facts listed in line 5 using predicate tbl/2. Overall, we have obtained a set of facts as the representation of instance data.

The second step concerns the actual program formalizing the problem statement. Writing down the rules is of course a creative activity, which one learns best by doing, but in ASP one can concentrate on defining the relevant concepts (relations) in terms of rules, as well as thinking about conditions on which certain relations should hold. To understand the outcome of the formalization in listing 1, let us give the intuitive readings for the rules involved. The rule in line 8 stipulates that every person P must be seated at exactly one table T. A few constraints follow. The capacities of tables are enforced in line 9: it is unacceptable if more than C persons are seated at table T which seats at most C persons. Moreover, if person P1 likes person P2, they should not be seated at different tables T1 and T2. This constraint is expressed in lines 10–12. The other way around, if P1 does not like P2, then they should not be seated at the same table T. The respective rule is given in lines 13–14. The rules and constraints in lines 8–14 explained so far form a uniform encoding of the seating problem, as the representation is independent of any problem instance described by facts of the type in lines 2–5.

So far, we have demonstrated the modeling philosophy of ASP in terms of a simple application. The later section on locking design provides further insights into modeling and typical design decisions made. Yet further information is available in the articles of Bruynooghe, Denecker, and Truszczyński (2016) and Gebser and Schaub (2016) in this issue.

ASP Solving

It remains to explain how the encoding from listing 1 solves the problem instance in practice. First, the rules of the program have to be instantiated and evaluated with respect to the present facts. This means, for example, that the rule in line 8 yields an instance

```
1 { seat(a,1); seat(a,2); seat(a,3); seat(a,4) } 1.
```

when P is replaced by a and T ranges over the available tables 1, 2, 3, and 4. This particular instance concerns the seating of Alice. While instantiating the rules also some evaluations take place. For example, when handling the rule in line 9 for table 1 with capacity 7 the lower bound C of the constraint is substituted by the value 7. The ground program, also indicated in figure 1, is typically generated by running a dedicated tool, that is, a *grounder,* on the input. After that the search for answer sets can be performed by invoking an answer set solver. Finally, the solution(s) of the original problem instance are obtained by extracting relevant part(s) from the answer set(s) found. For the encoding under consideration, this means that whenever an occurrence of seat(P, T) is contained in an answer set, then person P is supposed to be seated at table T. Using the notions from example 1, we would have the required mapping *s* from persons P to tables T. If no answer set can be found, then a problem instance has no solutions. This is actually the case for the instance described by lines 2–5 in listing 1, since it is impossible to place Alice, Bob, and John at the same table due to their relations. However, if the facts in line 4 are removed, obtaining answer sets is still feasible — the relationships of other guests permitting.

Beyond Basic ASP

The basic paradigm illustrated in figure 1 solves the problem at hand by eventually finding one or more solutions to the problem, or by showing that no solution exists. If there are multiple solutions to the problem, then it may be desirable to select the best solution among the alternatives using some criterion such as price, capacity, and so on. This turns the problem into an optimization problem. In ASP, objective functions for such problems can be defined in terms of optimization statements like

$$\#minimize \{ w_1,1: atom_1; \ldots; w_n,n: atom_n \}.$$

The statement above assigns weights $w_1, \ldots, w_n$ to atoms $atom_1, \ldots, atom_n$, respectively, and the goal is to minimize the sum of weights for atoms contained in an answer set — when evaluated over all answer sets. As regards the seating arrangement problem, the respective optimization problem could deal with obviously inconsistent settings like the one described above. Rather than satisfying all constraints resulting from the mutual relations of persons, the goal would be to satisfy as many as possible. In the preceding example, this would mean that either Alice is seated at the same table as Bob, or Bob is seated with John, but Alice and John are placed at different tables.

Besides the optimization of solutions, there are also other reasoning modes of interest. It is sometimes interesting to see how much the solutions are alike. In cautions reasoning, the idea is to check whether a certain atom is present in all or absent from some answer set. For instance, if seat(a,1) is for some reason contained in all answer sets, then Alice will be unconditionally seated at the first table and no options remain to this end. Cautious reasoning corresponds to basic query evaluation over answer sets and it can be implemented by adding a constraint to the program. In the case of our example, the constraint would read :- seat(a,1). indicating that we would like to find any counterexample, that is, an answer set *not* containing seat(a,1). Alternatively, cautious reasoning can be implemented by solvers as a special reasoning mode while searching for answer sets. Brave reasoning is the dual of cautious reasoning and then the presence in some or absence from all answer sets is required. Again, this can be implemented by adding a constraint or as a special reasoning mode.

It is also possible to enumerate answer sets and, hence, count their number. For certain applications, the number of solutions could actually be an interesting piece of information. In product configuration (see, for example, Soininen and Niemelä [1998]), this could be the number of variants that a production line should be able to produce. There are also complex use cases of ASP. In incremental solving, the idea is to compute partial solutions to a problem (or show their nonexistence) by calling an ASP solver several times and by extending the instance data on the fly. Various kinds of planning problems (with an increasing plan length) typically fall into this category. The latest developments even suggest multishot solving (Gebser et al. 2014) where solver calls are freely mixed and the ground programs used upon solver calls may evolve in more complex ways.

Constraints over Infinite Domains

Since grounding is an inherent part of ASP work flow, the basic paradigm is based on Boolean or finite-domain variables only. However, certain applications call for variables over infinite domains such as integers and reals. For instance, there have been proposals to extend ASP rules by linear inequalities (Gebser, Ostrowski, and Schaub 2009; Liu, Janhunen, and Niemelä 2012; Mellarkod, Gelfond, and Zhang 2008) as well as difference constraints (Janhunen, Liu, and Niemelä 2011). From the modeling perspective, the goal of such extensions is to increase the expressive power of ASP suitably so that new kinds of applications become feasible. For instance, referring back to the seating problem in listing 1, we could refine the specification for each person P by introducing integer variables e(P) and l(P) denoting the points of time when P enters and leaves the table in question. Using difference constraints, we could state a specification given as listing 2. Intuitively, the rules in lines 1 and 2 insist that person P stays at the table from 5 to 90 minutes. The constraint in lines 3–5 refines the last one from listing 1. It is not allowed that any two persons P1 and P2 who dislike each other are seated at the same table at the same time. It is important to notice that when the constraint in line 1 is instantiated for Alice, the resulting constraint is :- 1(a)–e(a) < 5. Thus, the infinity of the underlying domain is not reflected to the size of the resulting ground program. Naturally, the interpretation of 1(a) and e(a) as integer variables must be dealt with by the implementation of such constraints.

```
1  :- l(P)-e(P)<5, person(P).
2  :- l(P)-e(P)>90, person(P).
3  :- l(P1)-e(P2)>0, l(P2)-e(P1)>0,
4     dislikes(P1,P2), person(P1), person(P2),
5     seat(P1,T), seat(P2,T), tbl(T,_).
```

Listing 2. Examples of Difference Constraints.

Application: Locking Design

Having introduced the ASP paradigm on a general level, we now illustrate its main features in terms of an application problem where the goal is to design a locking scheme for a building. This is to be understood comprehensively, that is, we are not just interested in locks but also anything else that can affect accessibility in a building. For simplicity, we consider a single floor. A sample floor plan of such a building is depicted in figure 2. There are 12 rooms altogether, numbered from 1 to 12 in the figure.

Given this domain, our objectives are as follows. First, we describe the domain in a uniform way by selecting adequate predicates for the representation of domain information. Second, we take one concrete design goal from this domain into consideration. To this end, we concentrate on the configuration of locks installed on (potential) doors between the rooms in such a way that certain accessibility criteria are met. A particular safety requirement is that the floor can be effectively evacuated in case of an emergency. The idea is to develop ASP encodings for a design problem like this and, at the same time, illuminate the basic line of thinking and typical primitives used when modeling in ASP.

Uniform Encoding

The goal is to choose predicate symbols and the respective relations that are needed to represent an instance of the application problem at hand. To abstract the physical coordinates of the rooms, we rather represent the adjacency relation of rooms in

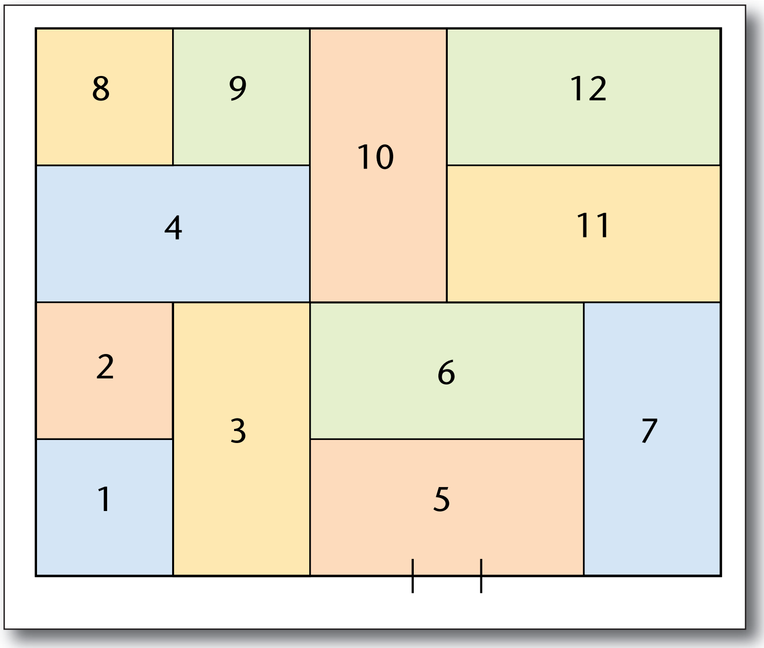


Figure 2. Floor Plan for the Rooms 1–12.

```
1  room(R1) :- adj(R1,R2).
2  room(R2) :- adj(R1,R2).
3  pot(R1,R2) :- adj(R1,R2).
4  pot(R1,R2) :- adj(R2,R1).
5  otherexit :- exit(X), X>1.
6  exit(1) :- not otherexit.
```

Listing 3. Domain Rules for Locking Design.

terms of a predicate adj/2. For simplicity, we also assume that this relation captures the potential of installing doors between any adjacent rooms. The floor plan of figure 2 can be represented by constants 1..12 for the rooms and the following facts:

adj(1,2). adj(1,3). adj(2,3). adj(2,4). … adj(11,12).

In total, there are 21 such facts and they are sufficient for the purposes of our examples to describe the interconnections of the rooms. For space efficiency, the adjacency information is represented asymmetrically, that is, adj(X,Y) is reported only if X<Y. In addition, the rooms having exits are reported using a unary predicate exit/1. For the running example in figure 2, this is captured by the fact exit(5). Now, if the given floor plan were changed in one way or another, or a completely different floor plan were taken into consideration, this should be reflected in the facts describing the problem instance. The other rules describing the application problem are based on these two predicates, hence making the encoding uniform. As typical in ASP encodings, some subsidiary domain predicates are defined in order to make the description of the actual problem easier. Some domain rules for the locking design problem are collected in listing 3 and explained next.

Relational Operations

The rules in lines 1–2 of listing 3 are used to extract room information from the adjacency information by a simple projection operation. As a result room(R) is true for only those values of R that actually appear in the adjacency information. In principle, a door between two rooms provides symmetric access from a room to another. Thus, the adjacency relation is not well-suited as such for the description of accessibility and we form the union of the accessibility relation with its reverse relation using rules in lines 3–4. The relation pot/2 stands for potential access depending on instrumentation such as locks, handles, press buttons, and so on.

Defaults

To illustrate the use of defaults in encodings, we have included the rules in lines 5–6 of listing 3. The rule in line 5 defines the condition otherexit/0 meaning that some other room than the room 1 has an exit. The rule in line 6 ensures that, by default, there is an exit at room 1. This is to hold unless another exit has been declared for the particular problem instance. There can be multiple exits. For instance, if there are two exits at rooms 1 and 5, this can be stated explicitly using facts exit(1) and exit(5). Adding these facts overrules the default in line 6 because otherexit can be inferred by the rule in line 5.

Defining the Search Space

Typical ASP encodings include a part where the solution candidates for the problem being formalized are generated. This can be achieved by expressing a number of choices that aim at capturing the varying aspects of solutions. As regards syntax, such choices can be expressed in terms of choice rules whose heads are count-bounded sets of atoms. Bounds can also be omitted if an arbitrary choice is of interest. As explained above, the access from a room to another can be asymmetric due to physical constructions. In particular, this is true for emergency situations where persons try to leave the building as soon as possible but might have no keys to unlock any door. For simplicity, we introduce a two-argument predicate evac/2 that is used to express the existence of an evacuation route from a room to another. Given adjacent rooms R1 and R2, such a design choice can be made in terms of a choice rule

```
1  reach(R,R) :- room(R).
2  reach(R1,R2) :-
3    reach(R1,R3), evac(R3,R2),
4    room(R1), pot(R3,R2).
5
6  ok(R) :- room(R), reach(R,X), exit(X).
7  :- not ok(R), room(R).
8
9  #minimize{1,R1,R2: evac(R1,R2), pot(R1,R2)}.
```

Listing 4. ASP Encoding of the Evacuation Plan.

```
{ evac(R1,R2) } :- pot(R1,R2).
```

The intuitive reading is that if pot(R1,R2) is true, then the truth value of evac(R1,R2) is subject to a choice. Hence, the selection of evacuation routes between rooms is formalized. Note that the analogous normal rule

```
evac(R1,R2) :- pot(R1,R2).
```

would falsify evac(R1,R2) by default if pot(R1,R2) were false, for example, rooms R1 and R2 were not adjacent. Since the relation pot/2 is symmetric, this gives rise to four different scenarios if pot(R1,R2) and thus also pot(R2,R1) is true.

Evacuation in one direction is possible if either evac(R1,R2) or evac(R2,R1) holds. If they are both true, this allows for bidirectional evacuation between R1 and R2. If such an option is not considered safe, it is easy to introduce an integrity constraint to exclude such a possibility in general:

```
:- evac(R1,R2), evac(R2,R1), pot(R1,R2).
```

If both evac(R1,R2) and evac(R1,R2) are false, then there is no connection between rooms R1 and R2 in case of an emergency. It remains to ensure that there exists an overall evacuation plan, that is, it is possible to reach at least one exit of the building from every room.

Recursive Definitions

The existence of an evacuation plan is governed by constraints that concern the mutual reachability of rooms, to be formalized using a predicate reach/2. The first two rules of listing 4 give a recursive definition for this predicate. Every room R is reachable from itself: the corresponding base case is given in line 1. The recursive case is formulated in lines 2–4: the reachability of R2 from R1 builds on the reachability of an intermediate room R3 from R1 and the condition that R3 can be evacuated to R2 (compare with line 3).

Constraining Solutions

The essential constraint on the evacuation plan is given in lines 6–7 of listing 4. Any given room R is considered to be OK, if some exit X is reachable from it (line 6). The auxiliary predicate ok/1 is defined in order to detect this aspect for each room. The actual constraint (line 7) excludes scenarios where some of the rooms would not be OK. Last, we want to minimize the number of evacuation connections by the objective function given in line 9. Using the encoding devised so far and an ASP solver, it is possible to check for the floor plan of figure 2 that the minimum number of connections is 11. This is clear since there are 12 rooms in total each of which (except room 5) must be connected to some other room for the purpose of evacuation. But ASP solvers can find out more for our running example. For instance, it is possible to enumerate and count all possible evacuation plans with 11 connections. In fact, there are 22 020 such plans and further constraints can be introduced to identify the most suitable ones. It is indeed the case that the current requirements allow for very long evacuation routes through the building of figure 2 such as

$$7 \rightarrow 6 \rightarrow 11 \rightarrow 12 \rightarrow 10 \rightarrow 9 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow 5.$$

Given this observation, the lengths of routes seem important. Thus, we now pay special attention to the number of evacuation *steps,* that is, moves from a room to another, and from the room perspective. The number of steps ought to be limited.

Elaboration Tolerance

It is straightforward to modify the recursive encod-

```
1  step(0..s).
2
3  reach(R,R,0) :- room(R).
4  reach(R1,R2,S+1) :-
5    reach(R1,R3,S), evac(R3,R2),
6    room(R1), pot(R3,R2), step(S), step(S+1).
7
8  ok(R) :- room(R), reach(R,X,S),
9    exit(X), step(S).
```

Listing 5. Revised ASP Encoding of the Evacuation Plan.

ing so that the number of steps is reflected. The revised encoding is presented as listing 5. The domain for steps is first declared by the rule in line 1 where the maximum number of steps s is determined from the command line of the grounder. The base case in line 3 simply states that each room R is reachable from itself in zero steps. The main modification in the recursive case (lines 4–5) concerns counting: the number of steps S is increased by one to S+1 whenever a further step is made. However, since both S and S+1 must be members of the domain of steps, the maximum value is effectively determined by the constant s in line 1. Given the floor plan of figure 2 and s=2, no evacuation plans can be found. By increasing s by one, solutions with 11 connections are found again and there are only 152 plans where the number of evacuation steps is at most three.

In summary, we have now tackled one particular aspect of locking design, that is, ensuring that an evacuation plan exists for a building. In reality further requirements are imposed on evacuation plans making the problem computationally more and more challenging. For instance, it can be shown that if we incorporate conditions which can make rooms along an evacuation route mutually exclusive, for example, for certain security reasons, it is unlikely that we are able to find a polynomial time algorithm for solving the problem (mathematically expressed the problem becomes NP-complete). This justifies well the use of powerful search methods like ASP for tackling the problem. For readers interested in computational complexity, we sketch the justifications of computational hardness in the sidebar.

Computing Answer Sets

So far, we have concentrated on the conceptual model of figure 1 with an emphasis on the modeling side. As regards the actual computation of answer sets, grounding and solving were also identified as the main steps involved. Grounders are implemented either as stand-alone tools, such as the state-of-the-art grounder GRINGO,[3] or integrated as a front end of the solver. Native answer set solvers are able to handle ground logic programs directly and, hence, truly implement the search step illustrated in the figure. Typically, this step is the most demanding one from the computational perspective. A number of answer set solvers have been developed in the history of ASP and we mention here DLV,[4] CLASP[3], and WASP[5] since they are actively maintained and developed at the moment. The article by Kaufmann et al. (2016) in this issue gives a more detailed account of grounding and solving. If ASP is extended by constraints which cannot be directly handled by the ASP solver being used, the typical solution is to isolate extensions from rules themselves and to treat them by appropriate solvers externally. This leads to an architecture where two or more solvers are cooperating and interacting in analogy to SAT modulo theories (SMT) solvers. Then each sort of constraints can be handled by native algorithms.

Translation-Based ASP

The other constraint-based disciplines discussed in the introduction offer similar solver technology at the user's disposal for handling, in particular, the search phase. However, they cannot be used straightforwardly, as ground programs are not directly understood by such solvers and certain kinds of transformations become indispensable. The idea of translation-based ASP is to translate (ground) logic programs into other formalisms so that a variety of solvers can be harnessed to the task of computing answer sets. Such an approach can be understood as a refinement of the search step in figure 1. There are existing translations from ASP, for example, to SAT (Janhunen 2004), and its extension as SMT (Niemelä 2008), and mixed integer programming (MIP) (Liu, Janhunen, and Niemelä 2012). These translations indicate the realizability of ASP in other formalisms and they have all been implemented by translators in the ASPTOOLS[6] collection. They offer another way of implementing the search phase in ASP using off-the-shelf solvers as black boxes. This approach is already competitive in certain application problems and it can be seen as an effort to combine the expressive power of the modeling language offered by ASP with the high performance of existing solvers. Translations are also useful when implementing language extensions in a single target language. For instance, the idea of (Janhunen, Liu, and Niemelä 2011) is to translate programs enriched by difference constraints into difference logic altogether. The strength is that a single solver is sufficient for the search phase, but on the other hand, the original structure of constraints may be lost.

Cross Translation

The translations mentioned above are based on very

Locking Design Can Be Computationally Challenging

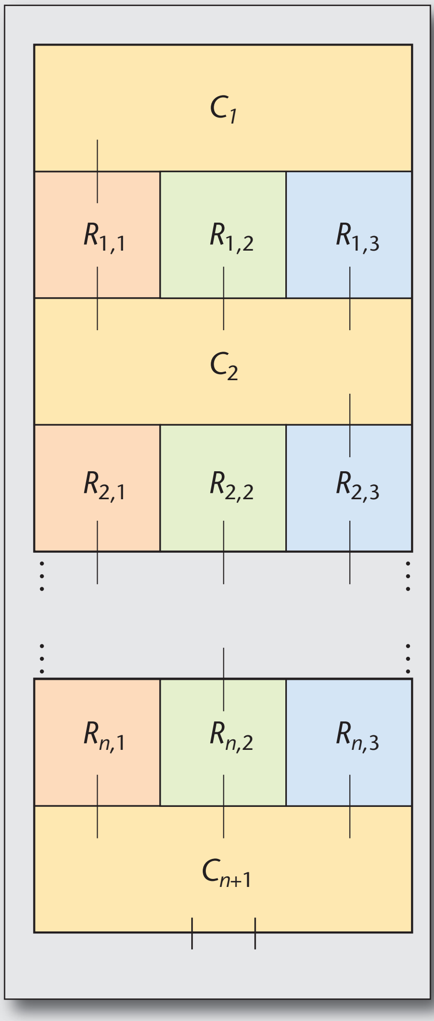


Figure 3. Floor Plan and Evacuation Routes for the NP-Completeness Proof.

It is not surprising that finding a locking scheme satisfying given conditions can become computationally challenging when more involved conditions need to be satisfied. Here we consider the problem of finding a locking scheme that allows an evacuation plan such that for each room there is exactly one evacuation direction and the evacuation routes respect a given set of room conflicts, that is, a set of pairs of rooms (R_1, R_2) such that when following the evacuation routes if you enter room R_1, then you cannot enter room R_2. We show that this locking design problem is NP-complete indicating that it is unlikely that a polynomial time algorithm for solving this problem can be found. See, for example, Papadimitriou (1994) for an introduction to computational complexity and the required concepts used next.

Technically, the NP-completeness of a problem can be shown by establishing a reduction computable in polynomial time from a known NP-complete problem to the problem and showing that it can be checked in polynomial time that a potential solution satisfies the required conditions for the problem. As such a known NP-complete problem we use the Exact-3-SAT problem where we are given a conjunction of 3-literal clauses and the problem is to find a truth assignment that satisfies exactly one literal in each of the clauses.

Reduction from Exact-3-SAT

Any given 3-SAT instance C_1&…&C_n can be transformed into a floor plan illustrated in figure 3. For each 3-literal clause $C_i = l_{i,1}|l_{i,2}|l_{i,3}$, we introduce a corridor C_i connected to rooms $R_{i,1}$, $R_{i,2}$, and $R_{i,3}$ that are connected to corridor C_{i+1}. Moreover, rooms $R_{i,1}$, $R_{i,2}$, and $R_{i,3}$ do not have doors in between. The (only) exit is located next to corridor C_{n+1}, which means that all corridors and rooms must be eventually evacuated through it. Moreover, each room $R_{i,j}$ is labeled by the respective literal $l_{i,j}$, the idea being that $l_{i,j}$ is satisfied if C_i is evacuated through the room $R_{i,j}$. Consequently, if there are two rooms labeled by complementary literals (that is, a Boolean variable x and its negation), then those rooms are in conflict. This means that evacuation routes involving any pair of conflicting rooms are not feasible. It is also easy to see that the floor plan in figure 3 and the associated set of conflicts can be computed in polynomial time.

It can be shown that a 3-SAT instance C_1&…& C_n has a satisfying truth assignment such that each clause has exactly one literal satisfied if and only if for the corresponding floor plan there is a locking scheme that allows an evacuation plan such that (1) for each room there is exactly one evacuation direction and (2) the evacuation routes respect the set of room conflicts arising from the complementary literals. The key observation is that for the corresponding floor plan evacuation is possible only if there is a route from C_1 to C_{n+1} such that for each i = 1, … ,n the route visits exactly one of the rooms $R_{i,1}$, $R_{i,2}$, and $R_{i,3}$ and all room conflicts are respected. A satisfying truth assignment such that each clause has exactly one literal satisfied gives directly such a route, and if such a route is available, it gives directly an appropriate truth assignment where literals corresponding to the visited rooms in the route are satisfied.

Moreover, it is clear that given a locking scheme with exactly one evacuation direction for each room, whether evacuation is possible and all room conflicts are respected can be checked in polynomial time.

similar technical ideas but yield representations of the ground program in completely different formats. Since the development of several translators brings about extra programming work, it would be highly desirable to integrate the variety of translators in a single tool - having options for different back-end formats. This is not as simple as that due to the wide variety of formats under consideration.

However, this issue is partly solved by a recent translation from ASP to SAT modulo acyclicity (Gebser, Janhunen, and Rintanen 2014) where graph-based constraints are interconnected with ordinary logical constraints (that is, clauses). The translation can be implemented by instrumenting a ground logic program with certain additional rules and meta information formalizing the underlying recursion mechanism in terms of the acyclicity constraint. This leads to a new implementation strategy for translation-based ASP: the choice of the target formalism can be postponed until the last step of translation where the constraints are output in a particular solver format. This idea is analogous to cross compilation in the context of compiling conventional programming languages and hence we coin the term cross translation for ASP. In the current implementation of this idea, a back-end translator transforms the instrumented program into other kinds of constraints understood by SMT, MIP, and pseudo-Boolean (PB) solvers, for instance. Interestingly, by implementing an additional acyclicity check inside a native ASP solver, the instrumented program can also be processed directly by the solver (Bomanson et al. 2015), which offers yet another approach to answer set computation.

Summary and Future Prospects

This article provides an introduction to the ASP paradigm as well as explains its main features — first generally, but also in terms of examples. We also discuss the two mainstream approaches to implementing the search for answer sets using either native solvers, or translators combined with solver technology offered by neighboring disciplines.

Towards Universal Modeling

There is a clear trend in the area of constraint-based modeling where methods and techniques are being transferred from one discipline to another. Various ideas from knowledge representation, logic programming, databases, and Boolean satisfiability served as a starting point for the ASP paradigm. But there are signs of knowledge transfer in the other direction as well. For instance, ASP solvers have been integrated into logic programming systems such as XSB (Rao et al. 1997). Advanced query evaluation mechanisms of ASP (Faber, Greco, and Leone 2007) are also relevant for deductive databases. The very idea of answer sets has been brought to the context of CP by introducing so-called bound-founded variables (Aziz, Chu, and Stuckey 2013). Quite recently, the algorithms for projected answer set enumeration have been exported for model counting in the context of SAT (Aziz et al. 2015).

We foresee that the exchange and incorporation of ideas and technologies in this way is gradually leading towards a universal approach where the user may rather freely pick the right language for expressing constraints of his or her interest. The underlying reasoning system is then supposed to (1) take care of required translations transparently and (2) forward the resulting constraints for a solver architecture that can realize the search for answers. The first attempts to define a modular framework for multilanguage modeling have already been made (Järvisalo et al. 2009; Lierler and Truszczyński 2014; Tasharrofi and Ternovska 2011). However, a lot of work remains to be done in order to realize the universal modeling scenario. Our experience from integrating various kinds of tools suggests that finding a universal format for the constraints of interest is one of the key issues for tool interoperability. There are existing formats, such as the DIMACS format in SAT, the Smodels format in ASP, and the FlatZinc format in CP, that can be used as starting points for designing the universal format.

Acknowledgments

We gratefully acknowledge support from the Finnish Centre of Excellence in Computational Inference Research (COIN) funded by the Academy of Finland (under grant #251170). We also thank Martin Gebser, Michael Gelfond, Torsten Schaub, and Mirek Truszczyński for their comments on a preliminary draft of this article.

Notes

1. The encodings presented in this article are directly executable using contemporary ASP grounders and solvers compatible with the ASP-core-2 language specification (ASP-CORE-2 Input Language Format, 2012. F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, and T. Schaub).

2. However, ASP builds on the closed world assumption (CWA): the given information is treated as complete information and the problem is solved under this assumption.

3. potassco.sourceforge.net

4. www.dlvsystem.com

5. github.com/alviano/wasp.git

6. research.ics.aalto.fi/software/asp

References

Aziz, R.; Chu, G.; Muise, C.; and Stuckey, P. 2015. #∃SAT: Projected Model Counting. In *Theory and Applications of Satisfiability Testing (SAT 2015) 18th International Conference,* Lecture Notes in Computer Science 9340, 121–137. Berlin: Springer. dx.doi.org/10.1017/S147106841300032X

Aziz, R.; Chu, G.; and Stuckey, P. 2013. Stable Model Semantics for Founded Bounds. *Theory and Practice of Logic Programming* 13(4–5): 517–532.

Balduccini, M.; Gelfond, M.; and Nogueira, M. 2006. Answer Set Based Design of Knowledge Systems. *Annals of Mathematics and Artificial Intelligence* 47(1–2): 183–219. dx.doi.org/10.1007/s10472-006-9026-1

Bomanson, J.; Gebser, M.; Janhunen, T.; Kaufmann, B.; and Schaub, T. 2015. Answer Set Programming Modulo Acyclicity. In *Logic Programming and Nonmonotonic Reasoning — 13th International Conference, LPNMR 2015,* 143–150. Lecture Notes in Computer Science 9345. Berlin: Springer. dx.doi.org/10.1007/978-3-319-23264-5_13

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer Set Programming at a Glance. *Communications of the ACM* 54(12): 92–103. dx.doi.org/10.1145/2043174.2043195

Bruynooghe, M.; Denecker, M.; and Truszczyński, M. 2016. First Order Logic with Inductive Definitions for Model-Based Problem Solving. *AI Magazine* 37(3).

Denecker, M., and Ternovska, E. 2008. A Logic of Nonmonotone Inductive Definitions. *ACM Transactions on Computational Logic* 9(2). dx.doi.org/10.1145/1342991.1342998

Denecker, M., and Vennekens, J. 2014. The Well-Founded Semantics Is the Principle of Inductive Definition, Revisited. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference (KR 2014).* Palo Alto, CA: AAAI Press.

East, D., and Truszczyński, M. 2006. Predicate-Calculus-Based Logics for Modeling and Solving Search Problems. *ACM Transactions on Computational Logic* 7(1): 38–83. dx.doi.org/10.1145/1119439.1119441

Erdem, E. 2011. Applications of Answer Set Programming in Phylogenetic Systematics. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday,* ed. M. Balduccini and T. C. Son. Lecture Notes in Computer Science Volume 6565, 415–431. dx.doi.org/10.1007/978-3-642-20832-4_26

Erdem, E.; Gelfond, M.; and Leone, N. 2016. Applications of ASP. *AI Magazine* 37(3).

Faber, W.; Greco, G.; and Leone, N. 2007. Magic Sets and Their Application to Data Integration. *Journal of Computer and Systems Sciences International* 73(4): 584–609. dx.doi.org/10.1016/j.jcss.2006.10.012

Finkel, R., and O'Sullivan, B. 2011. Reasoning about Conditional Constraint Specification Problems and Feature Models. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing* (AI EDAM) 25(2): 163–174. dx.doi.org/10.1017/S0890060410000600

Friedrich, G.; Fugini, M.; Mussi, E.; Pernici, B.; and Tagni, G. 2010. Exception Handling for Repair In Service-Based Processes. *IEEE Transactions on Software Engineering* 36(2): 198–215. dx.doi.org/10.1109/TSE.2010.8

Gebser, M., and Schaub, T. 2016. Modeling and Language Extensions. *AI Magazine* 37(3).

Gebser, M.; Guziolowski, C.; Ivanchev, M.; Schaub, T.; Siegel, A.; Thiele, S.; and Veber, P. 2010a. Repair and Prediction (Under Inconsistency) in Large Biological Networks with Answer Set Programming. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference (KR 2010).* Palo Alto, CA: AAAI Press. dx.doi.org/10.1109/ICTAI.2010.62

Gebser, M.; Janhunen, T.; and Rintanen, J. 2014. Answer Set Programming as SAT Modulo Acyclicity. In *ECAI 2014 — 21st European Conference on Artificial Intelligence,* Frontiers in Artificial Intelligence and Applications 263, 351–356. Amsterdam: IOS Press.

Gebser, M.; Kaminski, R.; Obermeier, P.; and Schaub, T. 2014. Ricochet Robots Reloaded: A Case-Study in Multi-Shot ASP Solving. In *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation: Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday,* ed. T. Eiter, H. Strass, M. Truszczyński, S. Woltran, Lecture Notes in Computer Science Volume 9060, 17–32. Berlin: Springer.

Gebser, M.; König, A.; Schaub, T.; Thiele, S.; and Veber, P. 2010b. The BioASP Library: ASP Solutions for Systems Biology. In *22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010,* 383–389. Piscataway, NJ: Institute for Electrical and Electronics Engineers.

Gebser, M.; Ostrowski, M.; and Schaub, T. 2009. Constraint Answer Set Solving. In *Logic Programming, 25th International Conference, ICLP 2009,* Lecture Notes in Computer Science 5649, 235–249. Berlin: Springer. dx.doi.org/10.1007/978-3-642-02846-5_22

Gelfond, M., and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium,* 1070–1080. Cambridge, MA: The MIT Press.

Gelfond, M., and Lifschitz, V. 1990. Logic Programs with Classical Negation. In *Logic Programming, Proceedings of the Seventh International Conference,* 579–597. Cambridge, MA: The MIT Press. dx.doi.org/10.1007/BF03037169

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9(3/4): 365–385.

Janhunen, T. 2004. Representing Normal Programs with Clauses. In P*roceedings of the 16th European Conference on Artificial Intelligence,* ECAI'2004, 358–362. Amsterdam: IOS Press.

Janhunen, T.; Liu, G.; and Niemelä, I. 2011. Tight Integration of Non-Ground Answer Set Programming and Satisfiability Modulo Theories. Paper presented at the First Workshop on Grounding and Transformation for Theories with Variables, GTTV 2011, Vancouver, BC, Canada, 16 May.

Järvisalo, M.; Oikarinen, E.; Janhunen, T.; and Niemelä, T. 2009. A Module-Based Framework for Multi-Language Constraint Modeling. In *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009,* Lecture Notes in Computer Science 5753, 155–168. dx.doi.org/10.1007/978-3-642-04238-6_15

Kaufmann, B.; Leone, N.; Perri, S.; and Schaub, T. 2016. Grounding and Solving in Answer Set Programming. *AI Magazine* 37(3).

Koponen, L.; Oikarinen, E.; Janhunen, T.; and Säilä, L. 2015. Optimizing Phylogenetic Supertrees Using Answer Set Programming. *Theory and Practice of Logic Programming* 15(4-5): 604–619. dx.doi.org/10.1017/S1471068415000265

Lierler, Y., and Truszczyński, M. 2014. Abstract Modular Inference Systems and Solvers. In *Practical Aspects of Declarative Languages — 16th International Symposium, PADL 2014,* Lecture Notes in Computer Science 8324, 49–64. Berlin: Springer. dx.doi.org/10.1007/978-3-319-04132-2_4

Lifschitz, V. 2016. Answer Sets and the Language of Answer Set Programming. *AI Magazine* 37(3).

Liu, G.; Janhunen, T.; and Niemelä, I. 2012. Answer Set Programming via Mixed Integer Programming. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference (KR 2012)*, 32–42. Palo Alto, CA: AAAI Press.

Marek, V., and Truszczyński, M. 1999. Stable Models and an Alternative Logic Programming Paradigm. In *The Logic Programming Paradigm: A 25-Year Perspective,* ed. K. Apt, V. Marek, M. Truszczyński, and D. Warren, 375–398. Berlin: Springer.

Marek, V.; Niemelä, I.; and Truszczyński, M. 2011. Origins of Answer-Set Programming — Some Background and Two Personal Accounts. In *Nonmonotonic Reasoning: Essays Celebrating Its 30th Anniversary,* ed. G. Brewka, V. Marek, M. Truszczynński. London: College Publications. 233–258. dx.doi.org/10.1007/978-3-642-60085-2_17

Mellarkod, V.; Gelfond, M.; and Zhang, Y. 2008. Integrating Answer Set Programming and Constraint Logic Programming. *Annals of Mathematics and Artificial Intelligence* 53(1–4): 251–287. dx.doi.org/10.1007/s10472-009-9116-y

Niemelä, I. 1999. Logic Programming with Stable Model Semantics as a Constraint Programming Paradigm. *Annals of Mathematics and Artificial Intelligence* 25(3–4): 241–273.

Niemelä, I. 2008. Stable Models and Difference Logic. *Annals of Mathematics and Artificial Intelligence* 53(1–4): 313–329. dx.doi.org/10.1007/s10472-009-9118-9

Papadimitriou, C. 1994. *Computational Complexity.* Boston, MA: Addison-Wesley.

Rao, P.; Sagonas, K.; Swift, T.; Warren, D.; and Freire, J. 1997. XSB: A System for Efficiently Computing WFS. In *Logic Programming and Nonmonotonic Reasoning, 4th International Conference, LPNMR'97,* Lecture Notes in Computer Science 1265, 431–441. Berlin: Springer.

Soininen, T., and Niemelä, I. 1998. Developing a Declarative Rule Language for Applications in Product Configuration. In *Practical Aspects of Declarative Languages, First International Workshop, PADL '99,* Lecture Notes in Computer Science 1551, 305–319. Berlin: Springer. dx.doi.org/10. 1007/3-540-49201-1_21

Tasharrofi, S., and Ternovska, E. 2011. A Semantic Account for Modularity in Multi-Language Modelling of Search Problems. In *Frontiers of Combining Systems, 8th International Symposium, FroCoS 2011,* Lecture Notes in Computer Science 6989, 259–274. Berlin: Springer. dx.doi.org/10.1007/978-3-642-24364-6_18

Tomi Janhunen (Tomi.Janhunen@aalto.fi) is a senior university lecturer at Aalto University in the Department of Computer Science. He holds the title of docent from Aalto University. Janhunen received his doctoral degree in theoretical computer science from Helsinki University of Technology in Finland in 1998. His primary research interests are in knowledge representation and automated reasoning, especially in answer set programming, extensions of Boolean satisfiability, and translations between logical formalisms. He has been a member in the program committees of 50 conferences and workshops in his research area.

Ilkka Niemelä (Ilkka.Niemela@aalto.fi) is a professor of computer science at Aalto University and serves currently as the provost of Aalto University. He received a doctoral degree in computer science in 1993 from Helsinki University of Technology. His research interests include automated reasoning, constraint programming, knowledge representation, computational complexity, computer-aided verification, automated testing, and product configuration. He is one of the principal investigators of the Finnish Center of Excellence in Computational Inference Research. He has served as the editor-in-chief of *Theory and Practice of Logic Programming* and is a Fellow of the European Association for Artificial Intelligence.

Grounding and Solving in Answer Set Programming

Benjamin Kaufmann, Nicola Leone, Simona Perri, Torsten Schaub

■ *Answer set programming is a declarative problem-solving paradigm that rests upon a work flow involving modeling, grounding, and solving. While the former is described by Gebser and Schaub (2016), we focus here on key issues in grounding, or how to systematically replace object variables by ground terms in an effective way, and solving, or how to compute the answer sets, of a propositional logic program obtained by grounding.*

Answer set programming (ASP) combines a high-level modeling language with effective grounding and solving technology. Moreover, ASP is highly versatile by offering various elaborate language constructs and a whole spectrum of reasoning modes. The work flow of ASP is illustrated in figure 1.

At first, a problem is expressed as a logic program. A grounder systematically replaces all variables in the program by (variable-free) terms, and the solver takes the resulting propositional program and computes its answer sets (or aggregations of them).

ASP's success is largely due to the availability of a rich modeling language (Gebser and Schaub 2016) along with effective systems. Early ASP solvers SModels (Simons, Niemelä, and Soininen 2002) and DLV (Leone et al. 2006) were followed by SAT[1]-based ones, such as ASSAT (Lin and Zhao 2004) and Cmodels (Giunchiglia, Lierler, and Maratea 2006), before genuine conflict-driven ASP solvers such as clasp (Gebser, Kaufmann, and Schaub 2012a) and WASP (Alviano et al. 2015) emerged. In addition, there is a continued interest in mapping ASP onto solving technology in neighboring fields, like SAT or even MIP[2] (Janhunen, Niemelä, and Sevalnev 2009; Liu, Janhunen, and Niemelä 2012), and in the automatic selection of the appropriate solver by heuristics (Maratea, Pulina, and Ricca 2014).

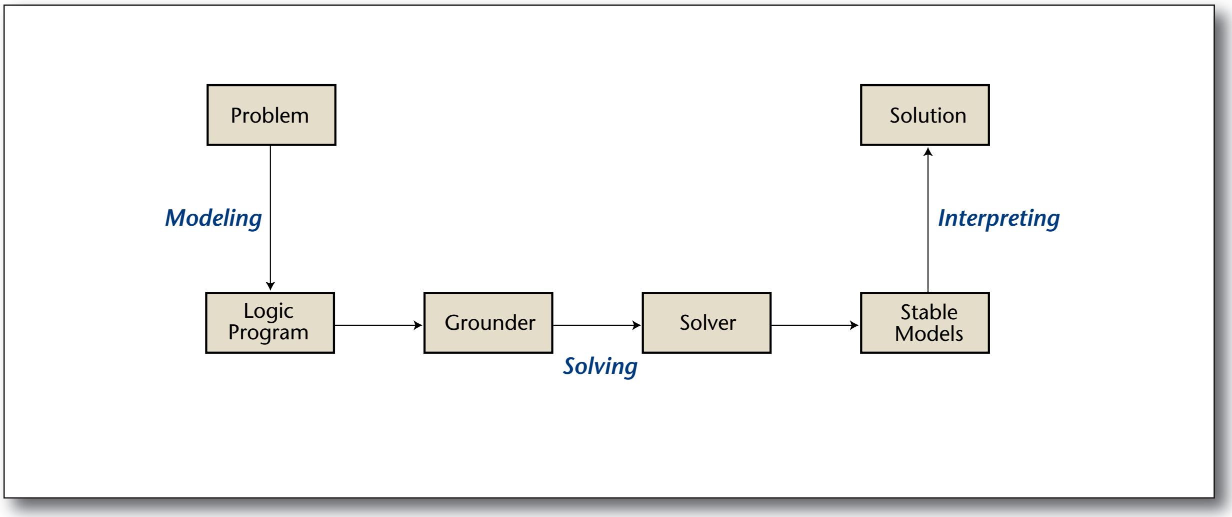


Figure 1. The Work Flow of Answer Set Programming.

By contrast, modern grounders like (the one in) DLV (Faber, Leone, and Perri 2012) or GrinGo (Gebser et al. 2011) are based on seminaive database evaluation techniques (Ullman 1988) for avoiding duplicate work during grounding. Grounding is seen as an iterative bottom-up process guided by the successive expansion of a program's term base, that is, the set of variable-free terms constructible from the signature of the program at hand. Other grounding approaches are pursued in GIDL (Wittocx, Mariën, and Denecker 2010), Lparse (Syrjänen 2001), and earlier versions of GrinGo (Gebser, Schaub, and Thiele 2007). The latter two bind nonglobal variables by domain predicates to enforce ω- or λ-restricted (Syrjänen 2001; Gebser, Schaub, and Thiele 2007) programs that guarantee a finite grounding, respectively.

In what follows, we describe the basic ideas and major issues of modern ASP grounders and solvers, also in view of supporting ASP's language constructs and reasoning modes.

Grounding

Modern ASP systems perform their computation by first generating a ground program that does not contain any variable but has the same answer sets as the original program. This phase, usually referred to as *grounding* or *instantiation,* solves a complex problem. In the case in which input nonground programs can be assumed to be fixed (data complexity), this task is polynomial. However, as soon as variable programs are given in input, grounding becomes EXPTIME-hard, and the produced ground program is potentially of exponential size with respect to the input program. To give an idea of that, consider the following program containing only one rule, and two facts:

```
obj(0). obj(1).
tuple(X1, ..., Xn)  :- obj(X1), ..., obj(Xn).
```

The ground instantiation of the rule contains 2^n ground rules, corresponding to the number of n-tuples, over a set of two elements. For more details about complexity of ASP the reader may refer to Dantsin et al. (2001).

Grounding, hence, may be computationally very expensive having a big impact on the performance of the whole system, as its output is the input for an ASP solver, that, in the worst case, takes exponential time in the size of the input. Thus, a näıve grounding which replaces the variables with all the constants appearing in the program (thus producing the full instantiation) is undesirable from a computational point of view. Indeed, most of the ground atoms appearing in the full instantiation are not derivable from the program rules, and all generated ground rules containing these atoms in the positive bodies are useless for answer set computation. For instance, consider the following program:

```
c(1, 2).
a(X) | b(Y) :- c(X, Y).
```

The full instantiation of the only rule appearing in the program contains four ground instances:

```
a(1) | b(1) :- c(1, 1).
a(2) | b(1) :- c(2, 1).
a(2) | b(2) :- c(2, 2).
a(1) | b(2) :- c(1, 2).
```

However, the first three ground rules are useless. They will never be applicable because their bodies contain atoms c(1, 1) and c(2, 1), and c(2, 2) that are not derivable from the program (they do not appear in the head of any rule).

ASP grounders, like GrinGo or the DLV instantiator, employ smart procedures that are geared toward efficiently producing a ground program that is considerably smaller than the full instantiation but preserves the semantics. In the following, we first give an informal description of the grounding computation. Then we introduce the problem of dealing with function symbols, which may lead to infinite groundings. Finally we overview some optimization strategies.

The Instantiation Procedure

In this subsection, we provide a description of the basic instantiation procedure, which is adopted by the most popular grounders, GrinGo and the DLV instantiator. For clarity, the description is informal, and presents a simplified version of the actual instantiation strategy. For instance, we do not take into account extensions of the basic language like choice rules or aggregates (Lifschitz 2016; Alviano and Leone 2015, 2016). Full details can be found in the work by Faber, Leone, and Perri (2012) and Gebser et al. (2011).

The core of the grounding phase is the process of rule instantiation. Given a rule *r* and a set of ground atoms *S*, which represents the extensions of the predicates, it generates the ground instances of *r*. Such a task can be performed by iterating on the body literals looking for possible substitutions for their variables. Grounders impose a safety condition, which requires that each rule variable appear also in a positive body literal. Thus, for the instantiator, it is enough to have a substitution for the variables occurring in the positive literals.

To clarify this process, consider the following (nonground) rule:

```
a(X) | b(Y) :- p(X, Z), q(Z, Y).
```

Now, assume that the set of extensions S = {p(1, 2), q(2, 1), q(2, 3)} is given. Then, the instantiation starts by looking for a ground atom in *S* matching with p(X, Z). Therefore p(X, Z) is matched with p(1, 2) and the substitution for X and Z is propagated to the other body literals, thus leading to the partially ground rule body p(1, 2), q(2, Y). Then, q(2, Y) is instantiated with the matching ground atom q(2, 1) and a ground rule a(1)| b(1):- p(1, 2), q(2, 1) is generated. Now, in order to find other possible rule instances, a backtracking step is performed, the binding for variable Y is restored and a new match for q(2, Y) is searched, finding q(2, 3). The new match is applied, leading to another rule instance a(1) | b(3) :– p(1, 2), q(2, 3). Then, the process goes on, by backtracking again to q(2, Y), and then to p(X, Z), because there are no more matches for q(2, Y). Given that also no further matches are possible for p(X, Z), the instantiation of the rule terminates, producing only two ground rules:

```
a(1) | b(1) :- p(1, 2), q(2, 1).
a(1) | b(3) :- p(1, 2), q(2, 3).
```

Roughly, the body literals are instantiated from left to right, starting from the first one. The instantiation of the generic body literal *L* consists in searching in *S* for a ground atom *A* matching with *L;* if such a matching is found, then the variables in *L* are bound with the constants in *A*, the substitution is propagated to the other body literals, and the next literal in the body is considered. If such a matching atom is not found, a backtracking step to a previous literal *L*′ is performed, some variable bindings are restored, and the process goes on by looking for another matching for *L*′. When all body literals have been instantiated, an instance for the rule *r* is found and the process continues by backtracking again to some previous literal, in order to find other substitutions. A crucial aspect of this process is how the set of ground atoms *S* containing the extensions of the predicates is computed. When a program is given as input to a grounder, it usually contains also a set of ground atoms, called *Facts*. It constitutes the starting point of the computation. In other words, initially *S* = *Facts*. During instantiation, the set *S* is expanded with the ground atoms occurring in the head of the newly generated ground rules. For instance, in the previous example, the ground atoms a(1) and b(1) are added to *S* and they will possibly be used for the instantiation of other rules. Thus, the extensions of the predicates are built dynamically. In order to guarantee the generation of all useful ground instances a particular evaluation order should be followed. If a rule r_1 defines (that is, has in the head) a predicate *p,* and another rule r_2 contains *p* in the positive body, then r_1 has to be evaluated before r_2 since r_1 produces ground atoms needed for instantiating r_1. Complying with such evaluation orders ensures that the produced ground program has the same answer sets of the full instantiation, but is possibly smaller (Faber, Leone, and Perri 2012).

To produce proper evaluation orders, grounders make use of structural information provided by a directed graph, called *Dependency Graph,* that describes how predicates depend on each other. This graph induces a partition of the input program into subprograms, associated with the strongly connected components, and a topological ordering over them. The subprograms are instantiated one at a time starting from the ones associated with the lowest components in the topological ordering.

Recursive rules within a subprogram, that is, rules where some body predicate depends, directly or transitively, on a predicate in the head, are instantiated according to a seminaive database technique (Ullman 1988). Their evaluation produces ground atoms needed for their own evaluation, thus, several iterations are performed, until a fixpoint is reached. At each iteration, for the predicates involved in the recursion, only the ground atoms newly derived during the previous iteration are taken into account.

To illustrate this, consider the following problem, called *Reachability:* Given a finite directed graph, compute all pairs of nodes (*a, b*) such that *b* is reachable from *a* through a nonempty sequence of arcs. This problem can be encoded by the following ASP program:

```
reach(X, Y) :- arc(X, Y).
reach(X, Y) :- arc(X, U), reach(U, Y).
```

The set of arcs is represented by the binary relation arc. A fact arc(a, b) means that the graph contains an arc from a to b; the set of nodes is not explicitly represented.

The program computes a binary relation reach containing all facts reach(a, b) such that *b* is reachable from *a* through the arcs of the input graph *G*. In particular, the first (nonrecursive) rule states that *b* is directly reachable from *a,* if there is an arc from *a* to *b;* while the second (recursive) rule states that *b* is transitively reachable from *a,* if there is a path in the graph from *a* to *b*.

The instantiation of this program is performed by first evaluating the nonrecursive rule on the set *S* containing the arcs. Assuming that *S* = { arc(1, 2), arc(2, 3), arc(3, 4) } three ground instances are produced:

```
reach(1, 2) :- arc(1, 2).
reach(2, 3) :- arc(2, 3).
reach(3, 4) :- arc(3, 4).
```

The ground atoms reach(1, 2), reach(2, 3), and reach(3, 4) are added to set *S* and the evaluation of the recursive rule starts. The first iteration is performed, producing rules

```
reach(1, 3) :- arc(1, 2), reach(2, 3).
reach(2, 4) :- arc(2, 3), reach(3, 4).
```

Then, reach(1, 3), reach(2, 4) are added to *S* and another iteration starts.
To avoid duplicate rules, for the recursive predicate reach, only the two newly generated ground atoms are used, producing:

```
reach(1, 4) :- arc(1, 2), reach(2, 4).
```

Now, reach(1, 4) is added to *S*. Another iteration is performed. Nothing new can be produced. The fixpoint is reached and the evaluation terminates.

Optimizations

Substantial effort has been spent on sophisticated algorithms and optimization techniques aimed at improving the performance of the instantiation process. In the following we briefly recall the most relevant ones.

The dynamic magic sets technique (Alviano et al. 2012) is a rewriting-based optimization strategy used by the DLV system. It extends the Magic Sets technique originally defined for standard Datalog for optimizing query answering over logic programs. Given a query, the Magic Sets technique rewrites the input program to identify a subset of the program instantiation which is sufficient for answering the query. The restriction of the instantiation is obtained by means of additional "magic" predicates, whose extensions represent relevant atoms with regard to the query. Dynamic Magic Sets, specifically conceived for disjunctive programs, inherit the benefits provided by standard magic sets and additionally allow to exploit the information provided by the magic predicates also during the answer set search. Magic sets turned out to be very useful in many application domains, even on some co-NP complete problems like consistent query answering (Manna, Ricca, and Terracina 2015).

Other techniques have been developed for optimizing the rule instantiation task (Faber, Leone, and Perri 2012). In particular, since rule instantiation is essentially performed by evaluating the relational join of the positive body literals, an optimal ordering of literals in the body is a key issue for the efficiency of the procedure, just like for join computation. Thus, an efficient body reordering criterion specifically conceived for the rule instantiation task has been proposed. Moreover, a backjumping algorithm has been developed (Perri et al. 2007), which reduces the size of the ground programs, avoiding the generation of useless rules, but fully preserving the semantics.

In the last few years, in order to make use of modern multicore and multiprocessor computers, a parallel instantiator has been developed. It is based on a number of strategies allowing for the concurrent evaluation of parts of the program, and is endowed with advanced mechanisms for dealing with load balancing and granularity control (Perri, Ricca, and Sirianni 2013).

Dealing with Function Symbols

Function symbols are widely recognized as an important feature for ASP. They increase the expressive power and in some cases improve the modeling capabilities of ASP, allowing the support of complex terms like lists, and set terms. Functions can also be employed to represent, through skolemization, existential quantifiers, which are receiving an increasing attention in the logic programming and database communities (Gottlob, Manna, and Pieris 2015). However, the presence of function symbols within ASP programs has a strong impact on the grounding process, which might even not terminate. Consider, for instance, the program:

```
p(0).
p(f(X)) :- p(X).
```

The instantiation is infinite; indeed the grounding of the recursive rule, at the first iteration adds to the set of extensions *S* the ground atom p(f(0)), which is used in the next iteration, producing p(f(f(0)) and so on. Despite this, grounders like the one in DLV and GrinGo are a way to deal with recursive function symbols and guarantee termination whenever the program belongs to the class of the so called finitely ground programs (Calimeri et al. 2008). Intuitively, for each program *P* in this class, there exists a finite ground program *P′* having exactly the same answer sets as *P*. Program *P′* is computable for finitely ground programs, thus answer sets of P are computable as well. Notably, each computable function can be expressed by a finitely ground program; membership in this class is not decidable, but it has been proven to be semidecidable (Calimeri et al. 2008).

For applications in which termination needs to be guaranteed a priori, the ASP grounders can make use of a preprocessor implementing a decidable check, which allows the user to statically recognize whether the input program belongs to a smaller subclass of the finitely ground programs (Syrjänen 2001; Gebser, Schaub, and Thiele 2007; Lierler and Lifschitz 2009; Calimeri et al. 2008). For instance, the grounder of DLV is endowed with a checker (which can also be disabled) for recognizing argument-restricted programs (Lierler and Lifschitz 2009). Earlier versions of GrinGo, in order to guarantee finiteness, accepted input programs with a domain restriction, namely λ-restricted programs (Gebser, Schaub, and Thiele 2007). From series 3, GrinGo removed domain restrictions and the responsibility to check whether the input program has a finite grounding is left to the user.

Solving

Modern ASP solvers rely upon advanced conflict-driven search procedures, pioneered in the area of satisfiablity testing (SAT; [Biere et al. 2009]).[3] Conflicts are analyzed and recorded, decisions are taken in view of conflict scores, and back-jumps are directed to the origin of a conflict.

While the general outline of search in ASP is arguably the same as in SAT, the extent of ASP requires a much more elaborate approach. First, the stable model's semantics enforces that atoms are not merely true but provably true (Lifschitz 2016). Second, the rich modeling language of ASP comes with complex language constructs. In particular, disjunction in rule heads and nonmonotone aggregates lead to an elevated level of computational complexity, which imposes additional search efforts. Finally, ASP deals with various reasoning modes. Apart from satisfiability testing, this includes enumeration, projection, intersection, union, and (multiobjective) optimization of answer sets, and moreover combinations of them, for instance, the intersection of all optimal models.[4] The first two issues bring about additional inferences, the latter require flexible solver architectures.

The restriction of modern SAT solvers to propositional formulas in conjunction normal form allows for reducing inferences to unit propagation along with the usual choice operations. In contrast, traditional ASP solving deals with an abundance of different inferences for propagation, which makes a direct adaption of conflict-driven search procedures virtually impossible. The key idea is thus to map inferences in ASP onto unit propagation on nogoods[5] (Gebser, Kaufmann, and Schaub 2012a), which traces back to a characterization of answer sets in propositional logic (Lin and Zhao 2004). Let us illustrate this by program P (thereby restricting ourselves to normal rules):[6]

$$P = \left\{ \begin{array}{ll} \texttt{a:-notb,} & \texttt{b:-nota,} \\ \texttt{x:-a,notc,} & \texttt{x:-y,} \\ \texttt{y:-x,b} & \end{array} \right\}$$

Interpreting this program in propositional logic results in the set $RF(P)$ of implications:

$$RF(P) = \left\{ \begin{array}{ll} a \leftarrow \neg b, & b \leftarrow \neg a, \\ x \leftarrow a \wedge \neg c \vee y, & \\ y \leftarrow x \wedge b & \end{array} \right\}$$

Note that we replaced default negation not by classical negation ¬ and combined both rules with head x while leaving the direction of the implications untouched (for readability). Now, the set $RF(P)$ has twelve classical models, many of which contain atoms not supported by any rule. (This is important because the stable models semantics insists on provably true atoms.) For instance, c is not supported by any rule, as is b whenever a is true as well.

Models containing unsupported atoms are eliminated by turning the implications in $RF(P)$ into equivalences (Clark 1978). Doing so for each atom yields the set $CF(P)$ of equivalences:

$$CF(P) = \left\{ \begin{array}{ll} a \leftrightarrow \neg b, & b \leftrightarrow \neg a, \\ x \leftrightarrow a \wedge \neg c \vee y, & \\ y \leftrightarrow x \wedge b, & c \leftrightarrow \bot \end{array} \right\}$$

This strengthening results in three models of $CF(P)$, one entailing atom b only, another making b, x, y true, and finally one in which a, x hold, respectively. The first two models differ in making both x and y true or not. A closer look at the original program P reveals that x and y support each other in a circular way. Whether or not such a circular derivation is harmful depends upon the existence of a valid external support (Lin and Zhao 2004), provided by an applicable rule whose head is in the loop but none of its positive antecedents belongs to it. In our case, this can be accomplished by the formula in $LF(P)$:

$$LF(P) = \left\{ \; (x \vee y) \rightarrow a \wedge \neg c \; \right\}$$

The formula expresses that an atom in the loop (consisting of x and y) can only be true if an external support of x or y is true. Here the only external support is provided by rule $x \leftarrow a,$ not c in P, as reflected by the consequent in $LF(P)$. That is, x or y can only be true if the latter rule applies. Since no other loops occur in P, the set $CF(P) \cup LF(P)$ provides a characterization of P's answer sets (Lin and Zhao 2004), one making atom b true and another a, x.

Note that in general the size of $CF(P)$ is linear in that of a program P, whereas the size of $LF(P)$ may be exponential in P (Lifschitz and Razborov 2006). Fortunately, satisfaction of $LF(P)$ can be tested in linear time for logic programs facing no elevated complexity (previously discussed), otherwise this test is co-

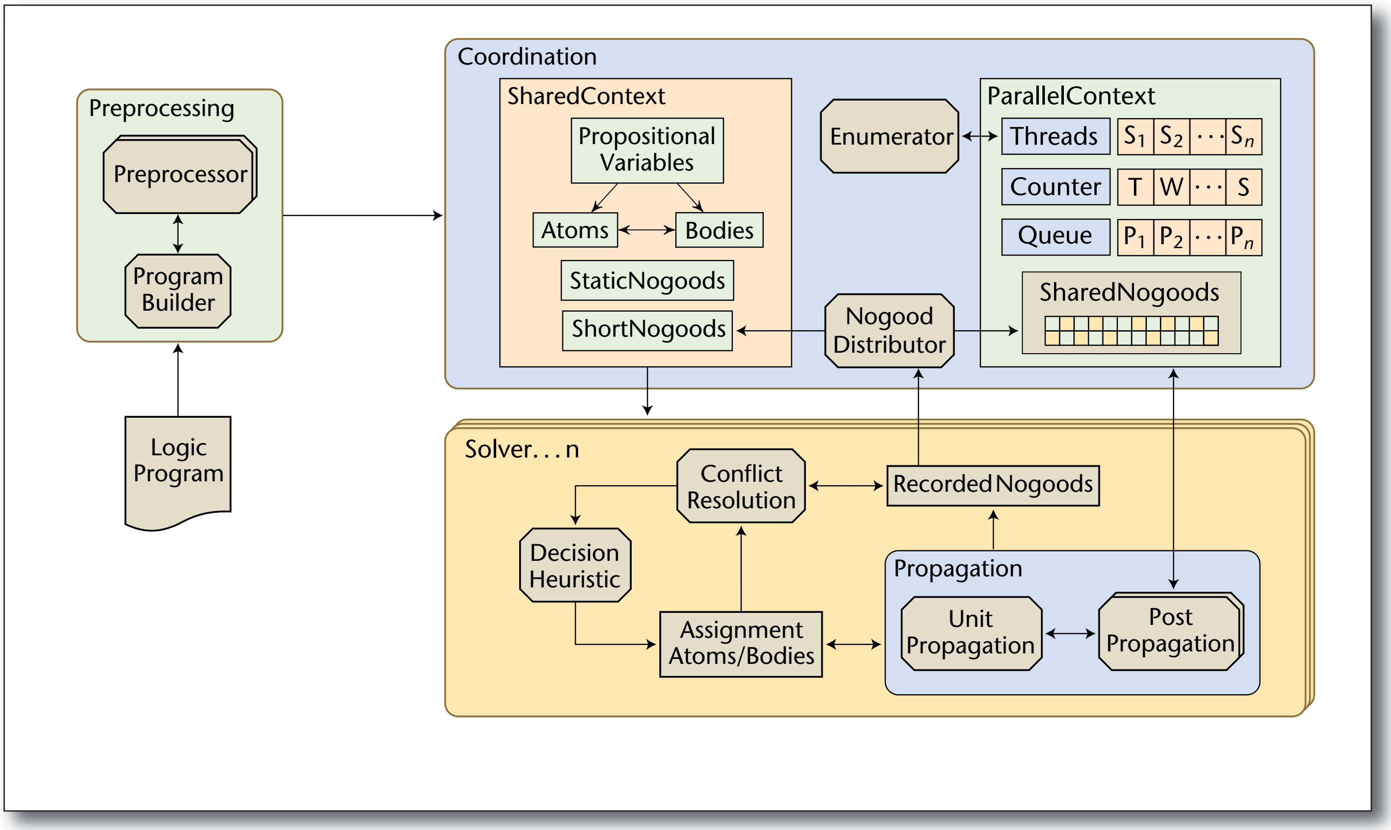


Figure 2. The Multithreaded Architecture of the ASP Solver Clasp.

NP-complete (Leone, Rullo, and Scarcello 1997).

The translation of programs into nogoods employed by modern ASP solvers follows the above characterization but takes the space issue into account. Given a program *P,* the nogoods expressing *CF*(*P*) are explicitly represented in an ASP solver, while the ones in *LF*(*P*) are only made explicit upon violation. This violation is detected by so-called unfounded set algorithms (Leone, Rullo, and Scarcello 1997; Gebser et al. 2012). Although we do not detail this here, we mention that aggregates are treated in a similar way by dedicated mechanisms unless they can be translated into nogoods in a feasible way (Gebser et al. 2009). Finally, let us make this more concrete by looking at the system architecture of clasp, depicted in figure 2.

The preprocessing component takes a (disjunctive) logic program and translates it into an internal representation. This is done in several steps. First, the given program, *P,* is simplified by semantic preserving translations as well as equivalence detection (Gebser et al. 2008). The simplified program P' is then translated into nogoods expressing $CF(P')$, which are subject to clausal simplifications adapted from corresponding SAT techniques. The resulting static nogoods are kept in the shared context component, as are parts of the dependency graph of *P* in order to reconstruct members of *LF*(*P*) on demand. Often more than three quarters of the nogoods obtained from $CF(P')$ are binary or ternary. Hence, such short nogoods are stored in dedicated data structures (and shared during parallel solving). Each solver instance implements a conflict-driven search procedure, as sketched at the outset of this section. Of particular interest is propagation, distinguishing between unit and post propagation. The former computes a fixed point of unit propagation. More elaborate propagation mechanisms can be added through post propagators. For instance, for programs with loops, this list contains a post propagator implementing the unfounded set checking procedure. Similarly, clasp's extension with constraint processing, clingcon (Ostrowski and Schaub 2012), as well as dlvhex (Eiter et al. 2006) use its post propagation mechanism to realize additional theory-specific propagations. The parallel execution of clasp allows for search space splitting as well as running competitive strategies. In both cases, learned conflict nogoods (as well as bounds in case of optimization) are exchanged between solver instances, each of which can be configured individually (see Gebser, Kaufmann, and Schaub [2012b] for details on multithreading). Finally, the enumerator is in charge of handling the various reasoning modes; once a solver finds a model, the enumerator tells it how to continue. This can be done by finding a next model in case of enumeration, or a better model in case of optimization.

Conclusion

Answer set programming combines a high-level modeling language with effective grounding and solving technology. This materializes in off-the-shelf ASP systems, whose grounding and solving engines can be used as black-box systems with standardized interfaces. Also, ASP is highly versatile by offering various complex language constructs and reasoning modes. As a side effect, many ASP solvers can also be used for MAX-SAT[7], SAT, and PB[8] solving. As a consequence, ASP faces a growing range of applications, as detailed by Erdem, Gelfond, and Leone (2016).

Acknowledgments

The first and last author were partially funded by DFG grants SCHA 550/8 and SCHA 550/9. The second and third author were partially supported by MIUR under PON project SI-LAB BA2KNOW Business Analitycs to Know, and by Regione Calabria, programme POR Calabria FESR 2007-2013, projects ITravel PLUS and KnowRex: Un sistema per il riconoscimento e lestrazione di conoscenza.

Notes

1. Satisfiablity testing.
2. Mixed integer programming.
3. This technology is usually referred to as conflict-driven clause learning.
4. Actually, this is a frequent reasoning mode used in underspecified application domains such as bioinformatics (Erdem, Gelfond, and Leone 2016).
5. Nogoods express inadmissible assignments (Dechter 2003).
6. *RF*(*P*), *CF*(*P*), and *LF*(*P*) stand for the rule, completion, and loop formulas of *P*.
7. Maximum satisfiability problem.
8. Pseudo-Boolean.

References

Alviano, M., and Leone, N. 2015. Complexity and Compilation of GZ-Aggregates in Answer Set Programming. *Theory and Practice of Logic Programming* 15(4-5): 574–587. dx.doi.org/10.1017/S147106841500023X

Alviano, M., and Leone, N. 2016. On the Properties of GZ-Aggregates in Answer Set Programming. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence* (IJCAI-16). Palo Alto, CA: AAAI Press.

Alviano, M.; Dodaro, C.; Leone, N.; and Ricca, F. 2015. Advances in WASP. In *Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15),* ed. F. Calimeri, G. Ianni, and M. Truszczyński, 40–54. Berlin: Springer. dx.doi.org/10.1007/978-3-319-23264-5_5

Alviano, M.; Faber, W.; Greco, G.; and Leone, N. 2012. Magic Sets for Disjunctive Datalog Programs. *Artificial Intelligence* 187: 156–192. dx.doi.org/10.1016/j.artint.2012.04.008

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2009. *Handbook of Satisfiability.* Amsterdam: IOS Press.

Calimeri, F.; Cozza, S.; Ianni, G.; and Leone, N. 2008. Computable Functions in ASP: Theory and Implementation. In *Proceedings of the Twenty-Fourth International Conference on Logic Programming (ICLP'08),* ed. M. Garcia de la Banda and E. Pontelli, 407–424. Berlin: Springer. dx.doi.org/10.1007/978-3-540-89982-2_37

Clark, K. 1978. Negation as Failure. In *Logic and Data Bases,* ed. H. Gallaire and J. Minker, 293–322. New York: Plenum Press. dx.doi.org/10.1007/978-1-4684-3384-5_11

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys* 33(3): 374–425. dx.doi.org/10.1145/502807.502810

Dechter, R. 2003. *Constraint Processing.* San Francisco: Morgan Kaufmann Publishers.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2006. DLVHEX: A Prover for Semantic-Web Reasoning under the Answer-Set Semantics. In *Proceedings of the International Conference on Web Intelligence (WI'06),* 1073–1074. Los Alamitos, CA: IEEE Computer Society. dx.doi.org/10.1109/wi.2006.64

Erdem, E.; Gelfond, M.; and Leone, N. 2016. Applications of ASP. *AI Magazine.* 37(3).

Faber, W.; Leone, N.; and Perri, S. 2012. The Intelligent Grounder of DLV. In *Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz,* ed E. Erdem, J. Lee, Y. Lierler, and D. Pearce. Berlin: Springer. 247–264. dx.doi.org/10.1007/978-3-642-30743-0_17

Gebser, M., and Schaub, T. 2016. Modeling and Language Extensions. *AI Magazine* 37(3).

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2009. On the Implementation of Weight Constraint Rules in Conflict-Driven ASP Solvers. In *Proceedings of the Twenty-Fifth International Conference on Logic Programming (ICLP'09),* ed. P. Hill and D. Warren, 250–264. Berlin: Springer. dx.doi.org/10.1007/978-3-642-02846-5_23

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice.* Synthesis Lectures on Artificial Intelligence and Machine Learning. San Rafael, CA: Morgan and Claypool Publishers.

Gebser, M.; Kaminski, R.; König, A.; and Schaub, T. 2011. Advances in Gringo Series 3. In *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11),* ed. J. Delgrande, and W. Faber, 345–351. Berlin: Springer. dx.doi.org/10.1007/978-3-642-20895-9_39

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012a. Conflict-Driven Answer Set Solving: From Theory to Practice. *Artificial Intelligence* 187–188: 52–89. dx.doi.org/10.1016/j.artint.2012.04.001

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012b. Multi-Threaded ASP Solving with Clasp. *Theory and Practice of Logic Programming* 12(4–5): 525–545. dx.doi.org/10.1017/S1471068412000166

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2008. Advanced Preprocessing for Answer Set Solving. In *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08),* ed. M. Ghallab, C. Spyropoulos, N. Fakotakis, and N. Avouris, 15–19. Amsterdam: IOS Press.

Gebser, M.; Schaub, T.; and Thiele, S. 2007. Gringo: A New Grounder for Answer Set Programming. In *Logic Programming and Nonmonotonic Reasoning, 9th International Conference,* Lecture Notes in Computer Science 4483, ed. C. Baral, G. Brewka, and J. Schlipf, 266–271. Berlin: Springer. dx.doi.org/10.1007/978-3-540-72200-7_24

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2006. Answer Set Programming Based on Propositional Satisfiability. *Journal of Automated Reasoning* 36(4): 345–377. dx.doi.org/10.1007/s10817-006-9033-2

Gottlob, G.; Manna, M.; and Pieris, A. 2015. Polynomial Rewritings for Linear Existential Rules. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence* (IJCAI-15), 2992–2998. Palo Alto, CA: AAAI Press.

Janhunen, T.; Niemelä, I.; and Sevalnev, M. 2009. Computing Stable Models via Reductions to Difference Logic. In *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007,* Lecture Notes in Computer Science 4483, ed. E. Erdem, F. Lin, and T. Schaub, 142–154. Berlin: Springer. dx.doi.org/10.1007/978-3-642-04238-6_14

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic* 7(3): 499–562. dx.doi.org/10.1145/1149114.1149117

Leone, N.; Rullo, P.; and Scarcello, F. 1997. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation. *Information and Computation* 135(2): 69–112. dx.doi.org/10.1006/inco.1997.2630

Lierler, Y., and Lifschitz, V. 2009. One More Decidable Class of Finitely Ground Programs. In *Logic Programming, 25th International Conference,* Lecture Notes in Computer Science 5649, 489–493. Berlin: Springer. dx.doi.org/10.1007/978-3-642-02846-5_40

Lifschitz, V. 2016. Answer Sets and the Language of Answer Set Programming. *AI Magazine* 37(3).

Lifschitz, V., and Razborov, A. 2006. Why Are There So Many Loop Formulas? *ACM Transactions on Computational Logic* 7(2): 261–268. dx.doi.org/10.1145/1131313.1131316

Lin, F., and Zhao, Y. 2004. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. *Artificial Intelligence* 157(1–2): 115–137. dx.doi.org/10.1016/j.artint.2004.04.004

Liu, G.; Janhunen, T.; and Niemelä, I. 2012. Answer Set Programming via Mixed Integer Programming. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2012),* ed. G. Brewka, T. Eiter, and S. McIlraith, 32–42. Palo Alto, CA: AAAI Press.

Manna, M.; Ricca, F.; and Terracina, G. 2015. Taming Primary Key Violations to Query Large Inconsistent Data via ASP. *Theory and Practice of Logic Programming* 15(4–5): 696–710. dx.doi.org/10.1017/S1471068415000320

Maratea, M.; Pulina, L.; and Ricca, F. 2014. A Multi-Engine Approach to Answer-Set Programming. *Theory and Practice of Logic Programming* 14(6): 841–868. dx.doi.org/10.1017/S1471068413000094

Ostrowski, M., and Schaub, T. 2012. ASP Modulo CSP: The Clingcon System. *Theory and Practice of Logic Programming* 12(4–5): 485–503. dx.doi.org/10.1017/S1471068412000142

Perri, S.; Ricca, F.; and Sirianni, M. 2013. Parallel Instantiation of ASP Programs: Techniques and Experiments. *Theory and Practice of Logic Programming* 13(2): 253–278. dx.doi.org/10.1017/S1471068411000652

Perri, S.; Scarcello, F.; Catalano, G.; and Leone, N. 2007. Enhancing DLV Instantiator by Backjumping Techniques. *Annals of Mathematics and Artificial Intelligence* 51(2–4): 195–228. dx.doi.org/10.1007/s10472-008-9090-9

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence* 138(1–2): 181–234. dx.doi.org/10.1016/S0004-3702(02)00187-X

Syrjänen, T. 2001. Omega-Restricted Logic Programs. In *Logic Programming and Nonmonotonic Reasoning, 6th International Conference,* Lecture Notes in Computer Science 2173, 267–279, ed. T. Eiter, F. Faber, and M. Truszczyński. Berlin: Springer. dx.doi.org/10.1007/3-540-45402-0_20

Ullman, J. 1988. *Principles of Database and Knowledge-Base Systems.* Rockville, MD: Computer Science Press.

Wittocx, J.; Mariën, M.; and Denecker, M. 2010. Grounding FO and FO(ID) with Bounds. *Journal of Artificial Intelligence Research* 38: 223–269.

Benjamin Kaufmann is a postdoctoral researcher at the University of Potsdam, Germany. He mainly works on efficient search-based reasoning and optimization techniques for Boolean satisfiability and related approaches. In particular, he is the lead developer of clasp, a state-of-the-art solving system in answer set programming.

Nicola Leone is a professor of computer science at University of Calabria, where he heads the Department of Mathematics and Computer Science and leads the AI Lab. He was professor of database systems at Vienna University of Technology until 2000. He is internationally renowned for his research on knowledge representation, answer set programming (ASP), and database theory, and for the development of DLV, a state-of-the-art ASP system that is popular worldwide. He has published more than 250 papers in prestigious conferences and journals, and has more than 8000 citations, with h-index 46. He is a fellow of ECCAI (now EurAI) and recipient of a Test of Time award from the Association for Computing Machinery.

Simona Perri is an associate professor at University of Calabria, Italy. She received her MSc in mathematics (2000) and a Ph.D. in computer science and mathematics (2005) from the University of Calabria, Italy. She is a member of the team that designed and maintains DLV, one of the major answer set programming systems, and is cofounder of DLVSystem Ltd, a spin-off company of the University of Calabria.

Torsten Schaub is university professor at the University of Potsdam, Germany, and holds an international chair at Inria Rennes, France. He is a fellow of ECCAI and the current president of the Association of Logic Programming. His current research focus, on answer set programming (ASP) and its applications, materializes at potassco.sourceforge.net, the home of the open source project Potassco bundling software for ASP developed at Potsdam.

Modeling and Language Extensions

Martin Gebser, Torsten Schaub

■ *Answer set programming (ASP) has emerged as an approach to declarative problem solving based on the stable model semantics for logic programs. The basic idea is to represent a computational problem by a logic program, formulating constraints in terms of rules, such that its answer sets correspond to problem solutions. To this end, ASP combines an expressive language for high-level modeling with powerful low-level reasoning capacities, provided by off-the-shelf tools. Compact problem representations take advantage of genuine modeling features of ASP, including (first-order) variables, negation by default, and recursion. In this article, we demonstrate the ASP methodology on two example scenarios, illustrating basic as well as advanced modeling and solving concepts. We also discuss mechanisms to represent and implement extended kinds of preferences and optimization. An overview of further available extensions concludes the article.*

Answer set programming (ASP) (Brewka, Eiter, and Truszczyński 2011) is a paradigm of declarative problem solving with roots in knowledge representation, logic programming, constraint satisfaction and optimization. Formally, ASP is based on the stable model semantics for logic programs (Gelfond and Lifschitz 1991), detailed by Lifschitz (2016) in this issue. As illustrated by Janhunen and Niemelä (2016), also in this issue, logic programs can be used to compactly represent search and optimization problems within the first two levels of the polynomial time hierarchy (Faber, Pfeifer, and Leone 2011; Ferraris 2011).[1] On the one hand, the attractiveness of ASP is due to an expressive modeling language, where concepts like (first-order) variables, negation by default, and recursion enable uniform problem representations in terms of facts specifying an instance along with a general problem encoding (Schlipf 1995). On the other hand, powerful ASP systems, described by Kaufmann et al. (2016) in this issue, are available off the shelf and automate the grounding of an encoding relative to a problem instance as well as the search for answer sets corresponding to problem solutions.

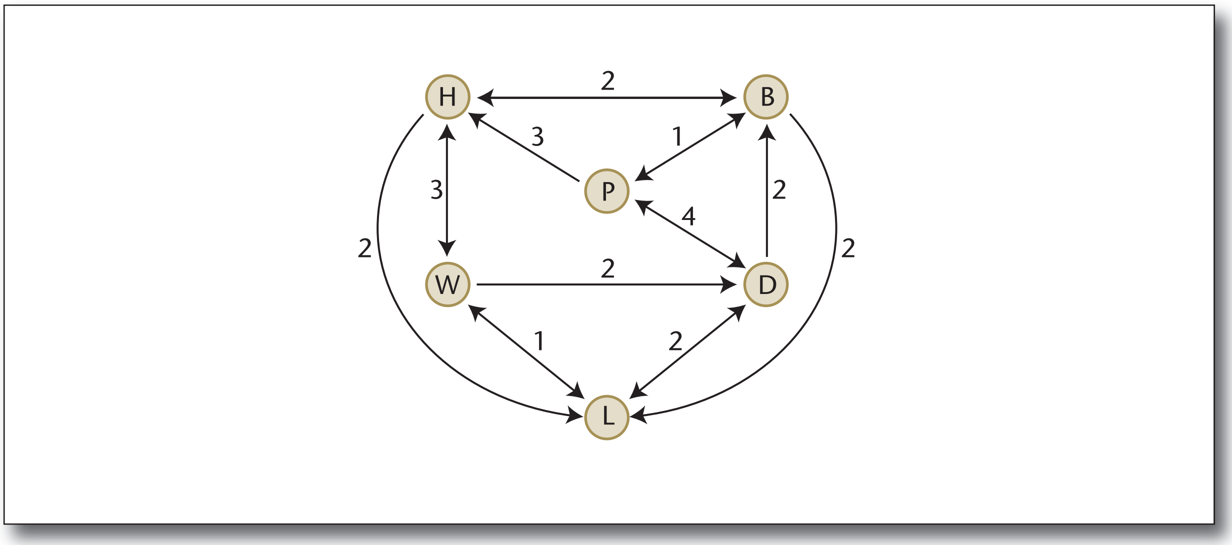


Figure 1. Places Connected by Links Associated with Costs.

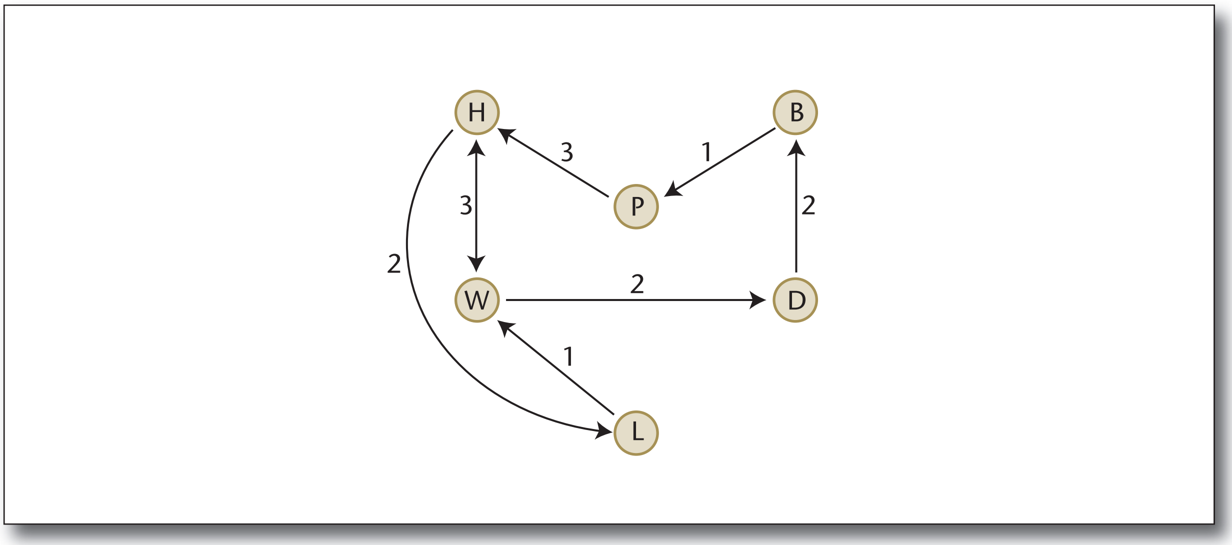


Figure 2. The Shortest Round Trip for the Places in Figure 1.

In this article, we detail the ASP modeling methodology on two example scenarios. To begin with, we elaborate on the use of traditional one-shot solving, where a problem is tackled by means of singular grounding and search processes. We particularly focus on a conceptual generate-and-test pattern (Eiter, Ianni, and Krennwallner 2009; Leone et al. 2006; Lifschitz 2002) as a best practice method to conceive legible yet efficient problem encodings. Further information regarding, among others, tool support for logic program development, elaboration of tolerant ways to represent extensive application domains, and alternative modeling languages is provided by Lierler, Maratea, and Ricca (2016), Erdem, Gelfond, and Leone (2016), as well as Bruynooghe, Denecker, and Truszczyński (2016) in this issue.

In our second example scenario, we take advantage of multishot solving, a powerful extension of traditional ASP methods in which grounding and search are interleaved to process a series of evolving subtasks in an iterative manner. Rather than processing each subtask from scratch, multishot solving gradually expands the representation of a problem, where grounding instantiates novel problem parts and search can reuse conflict information. Such incremental reasoning fits the needs in dynamic domains like, for example, logistics, policies, or robotics. In particular, we address a planning problem, where the (minimal) number of actions required to achieve a goal is usually not known a priori, while theoretical limits are prohibitively high as regards grounding.

The presentation of the two main approaches to modeling and solving is complemented by a survey of mechanisms to represent and implement extended kinds of preferences and optimization. An overview of further extensions conceived for demanding application problems concludes the article.

Modeling the Traveling Salesperson Problem

For illustrating the principal modeling concepts in ASP, let us consider the well-known traveling salesperson problem (TSP). A TSP instance consists of a number of places, each of which must be included within a round trip visiting every place exactly once, as well as links between places, specifying potential successors along with associated costs.

For example, figure 1 displays an instance with six places — the German cities Berlin, Dresden, Hamburg, Leipzig, Potsdam, and Wolfsburg, each denoted by its first letter. The cities are linked by train connections, available in either one or both directions, and their respective durations in hours constitute the costs. For example, Berlin and Potsdam are mutually linked, and it takes one hour to travel between the two neighboring cities, while four hours are needed from Potsdam to Dresden or vice versa. Moreover, a train connects Potsdam to Hamburg within three hours, but it does not operate the other way round. Further train connections interlink the other cities, and the question is how to arrange a shortest round trip visiting all cities.

When we construct a round trip manually, we may first fix some place to start the trip from, say Potsdam, and then proceed by opportunistically picking links to yet unvisited cities. for example, we can find a round trip leading from Potsdam to Berlin, Hamburg, Leipzig, Wolfsburg, Dresden, and then back to Potsdam. The connections taken in this trip add up to a total duration of 1 + 2 + 2 + 1 + 2 + 4 = 12 hours. However, the true shortest round trip shown in figure 2 takes only 11 hours. To find such shortest round trips, and also in case the train connections or cities to visit change, we aim at a general method for arbitrary places and links between them.[2] In what follows, we thus apply the ASP methodology to model shortest round trips for any TSP instance provided as input.

Problem Instance

The common practice in ASP is to represent a problem at hand uniformly, distinguishing between a particular instance and a general encoding (Marek and Truszczyński 1999, Niemelä 1999, Schlipf 1995). That is, we first need to fix a logical format for specifying places and links with associated costs. For example,

```
place(b).  % Berlin
place(d).  % Dresden
place(h).  % Hamburg
place(l).  % Leipzig
place(p).  % Potsdam
place(w).  % Wolfsburg
link(b,h,2).    link(b,l,2). link(b,p,1).
link(d,b,2).    link(d,l,2). link(d,p,4).
link(h,b,2).    link(h,l,2). link(h,w,3).
link(l,d,2).    link(l,w,1).
link(p,b,1).    link(p,d,4). link(p,h,3).
link(w,d,2).    link(w,h,3). link(w,l,1).
```

Listing 1. Instance Specifying the Places in Figure 1 as Facts.

the cities and connections displayed in figure 1 are described in terms of the facts given in listing 1. These facts are based on two predicates, place/1 and link/3, where 1 and 3 denote the arities of respective relations. The letters used as arguments of facts over place/1 stand for corresponding cities, for example, *p* refers to Potsdam, and writing such constants in lowercase follows logic programming conventions. Moreover, facts over link/3 specify the available connections, for example, link(*p*, *b*, 1), link(*p*, *d*, 4), and link(*p*, *h*, 3) provide those from Potsdam to Berlin, Dresden, and Hamburg along with their associated durations, as displayed in figure 1. The durations are given by integers, on which ASP systems support arithmetic operations,[3] while the names used for cities and predicates have no particular meaning beyond identifying places or relations, respectively. Also note that the facts constitute a set, so that the order of writing them is immaterial, which distinguishes ASP from logic programming languages having a procedural flavor, such as Prolog.

Problem Encoding

The main modeling task consists of specifying the intended outcomes, that is to say, shortest round trips, in terms of the conditions they must fulfill. To this end, let us first formulate such requirements in natural language:

(a) Every place is linked to exactly one successor in a trip.
(b) Starting from an arbitrary place, a trip visits all places and then returns to its starting point.
(c) The sum of costs associated with the links in a trip ought to be minimal.

```
1   % DOMAIN
2   start(X) :- X = #min{Y : place(Y)}.
3   % GENERATE
4   {travel(X,Y) : link(X,Y,C)} = 1 :- place(X).
5   % DEFINE
6   visit(X)     :- start(X).
7   visit(Y)     :- visit(X), travel(X,Y).
8   % TEST
9   :- place(Y),   not visit(Y).
10  :- start(Y),   #count{X : travel(X,Y)} < 1.
11  :- place(Y),   #count{X : travel(X,Y)} > 1.
12  % OPTIMIZE
13  :~travel(X,Y), link(X,Y,C). [C,X]
14  % DISPLAY
15  #show travel/2.
```

Listing 2. Encoding of Round Trips with Regard To Facts as in Listing 1.

Apart from a system-specific #show directive for output projection, the encoding in listing 2 is written in the syntax of the ASP-Core-2 standard language (www.mat.unical.it/aspcomp2013/ASPStandardization).

While these conditions are sufficient to characterize shortest round trips, the requirement in (b) further implies that every place has some predecessor. Given that (a) limits the number of links in a round trip to the number of places, the following condition must hold as well:

(d) Every place is linked to exactly one predecessor in a trip.

In summary, trips meeting the requirements in (a) and (b) are subject to the optimality criterion in (c), and (d) expresses an implied property. The conditions at hand provide a mental model for the ASP encoding furnished in the following.

The encoding shown in listing 2 is based on a conceptual generate-and-test pattern (Eiter, Ianni, and Krennwallner 2009; Leone et al. 2006; Lifschitz 2002). Accordingly, it is structured into several parts, distinguished by their concerns as well as typical constructs among those presented by Lifschitz (2016) in this issue. The purposes of the parts indicated by comments in lines beginning with % are as follows.

A DOMAIN part specifies auxiliary concepts that can be derived from facts and are shared by all answer sets.

A GENERATE part includes nondeterministic constructs, usually choice or disjunctive rules, to provide solution candidates.

A DEFINE part characterizes relevant properties of solution candidates, where the inherent features of fixpoint constructions and negation by default suppress false positives and enable a compact representation.

A TEST part usually consists of integrity constraints that deny invalid candidates whose properties do not match the requirements on solutions.

An OPTIMIZE part makes use of optimization statements or weak constraints to associate solutions with costs subject to minimization.

A DISPLAY part declares output predicates to which the printing of answer sets ought to be restricted in order to make reading off solutions more convenient.

In what follows, we elaborate on respective encoding parts.

DOMAIN

The first part, denoted by DOMAIN, includes the rule in line 2 to determine the lexicographically smallest identifier among places in an instance as (arbitrary) starting point for the construction of a round trip. To this end, the identifiers given by facts over place/1 are taken as values for the variable Y, and the smallest value, selected through a #min aggregate, is used to instantiate the variable X recurring in the head start(X). Note that, as usual in logic programming, variable names begin with uppercase letters, and recurrences within the same scope, that is, a rule, are substituted with common values. Relative to the facts in listing 1, the rule in line 2 is thus instantiated to

```
start(b) :- b = #min{
b : place(b); d : place(d); h : place(h);
p : place(p); 1 : place(1); w : place(w)}.
```

Since the predicate place/1 is entirely determined by facts, the above rule can be simplified to a derived fact start(*b*). In general, a DOMAIN part contains deterministic rules specifying relevant auxiliary concepts, so that they do not need to be provided per instance in a redundant fashion. Rather, including such rules in an encoding increases elaboration tolerance and exploits the capabilities of common grounders, which evaluate deterministic parts.

GENERATE

The second part, indicated by GENERATE, gathers nondeterministic constructs such that alternative selections among the derivable atoms provide distinct solution candidates. In line 4, we use a choice rule (Simons, Niemelä, and Soininen 2002) to express that, for every place in an instance, exactly one link from the place must be picked for a round trip. The rule constitutes a schema that applies to each place identifier taken as value for the variable X. For example, considering Potsdam and the three connections from there, it yields

```
{travel(p, b) : link(p, b, 1);
 travel(p, d) : link(p, d, 4);
 travel(p, h) : link(p, h, 3)} = 1 :- place(p).
```

Further simplifying this rule in view of facts over place/1 and link/3 leads to

```
{travel(p, b); travel(p, d); travel(p, h)} = 1.
```

That is, any answer set must include exactly one of the options travel(*p, b*), travel(*p, d*), and travel(*p, h*), reflecting that either Berlin, Dresden, or Hamburg has to succeed Potsdam in a round trip. As the same schema applies to other cities as well, atoms over the predicate travel/2 in an answer set represent a trip meeting the requirement in (a). However, the rule in line 4 leaves open which successor per place shall be picked, and hence it is called choice rule.

DEFINE

While the predicate travel/2 provides sufficient information to reconstruct a trip from an answer set, the requirement in (b) that all places must be visited is yet unaddressed. In order to test this condition, the DEFINE part includes rules analyzing which places are visited from the starting point fixed in the DOMAIN part before. To begin with, the rule in line 6 derives the starting point as visited, for example, visit(*b*) follows from start(*b*) relative to the facts in listing 1. The rule in line 7 further collects places reachable through the connections indicated by travel /2. For example, the following derivation chain is activated by atoms over travel/2 that represent the connections shown in figure 2:

```
visit(b) :- start(b).
visit(p) :- visit(b), travel(b, p).
visit(h) :- visit(p), travel(p, h).
visit(l) :- visit(h), travel(h, l).
visit(w) :- visit(l), travel(l, w).
visit(d) :- visit(w), travel(w, d).
```

Given that the involved connections form a round trip, all atoms over visit/1 follow through a sequence of rules rooted in start(b). However, if Hamburg were linked to Berlin instead of Leipzig, no such sequence would yield visit(*l*), visit(*w*), and visit(*d*). Atoms lacking a noncircular derivation are unfounded and exempt from answer sets (Van Gelder, Ross, and Schlipf 1991). In turn, answer sets encompass fix-point constructions for expressing concepts like, for example, induction and recursion. A DEFINE part makes use of this to derive predicates indicating relevant properties of a solution candidate at hand. As in DOMAIN parts, the contained rules are deterministic, yet their evaluation relies on nondeterministically generated solution candidates. In our case, visit/1 provides all places reached by taking connections in the trip from a fixed starting point.

TEST

The predicates characterizing solution candidates as well as their relevant properties are inspected in the TEST part in order to eliminate invalid candidates. This is accomplished by means of integrity constraints, that is, rules of denial with an implicitly false head, written by leaving the left side of :- blank. Regarding the conditions for round trips, the GENERATE part already takes care of (a), while the requirement in (b) remains to be checked. To this end, the integrity constraint in line 9 expresses that all places must be visited from the starting point given by start/1. for example, if Leipzig were not reached, a contradiction would be indicated through :- place(*l*), **not** visit(*l*). Note that **not** visit(*l*) makes use of negation by default, which applies whenever visit(*l*) is unfounded. Importantly, negation by default does not offer any derivation (by contraposition). As a consequence, the above integrity constraint is not interchangeable with a rule like visit(*l*) :- place(*l*).

If given such a rule, we could simply conclude visit(*l*), regardless of reachability. Unlike that, integrity constraints do not modify solution candidates or predicates providing their properties, but merely deny unintended outcomes. The distinction between constructs for deriving and evaluating atoms is an important modeling concept, here used to check that all places are indeed reached from a fixed starting point.

For the requirement in (b), we still have to make sure that a trip at hand returns to its starting point. Since every place is linked to one successor only and all but one final connection are needed to visit places different from the starting point given by start/1, it is sufficient to check that a (final) connection returning to the starting point exists. This condition is imposed by the integrity constraint in line 10, and relative to the facts in listing 1 it is instantiated to

```
:- start(b), #count{d : travel(d, b);
   h : travel(h, b); p : travel(p, b)} < 1.
```

The #count aggregate provides the number of atoms among travel(*d, b*), travel(*h, b*), and travel(*p, b*), representing connections returning to Berlin, included in an answer set. If neither connection is taken, this number is zero, in which case the success of the < 1 comparison indicates a contradiction. In turn, some connection must lead back to Berlin, but it can only be taken once all places are visited.

The checks through the integrity constraints in lines 9 and 10 establish that answer sets represent round trips meeting the requirement in (b). Since (a) is handled in the GENERATE part, the rules up to line 10 are already sufficient to characterize round trips. However, the implied property in (d) also states that a place cannot be linked to several predecessors. While this condition may seem apparent to humans, it relies on a counting argument taking the number of connections in a trip and the necessity that every place must be linked to some predecessor into account. Given that ASP solvers do not apply such reasoning, it can be beneficial to formulate nontrivial implied properties as redundant constraints. This is the motivation to include the integrity constraint in line 11, making explicit that a place cannot be linked to several predecessors. For example, regarding connections leading to Berlin, the schema yields

```
:- place(b), #count{d : travel(d, b);
```

```
$ clingo tsp-ins.lp tsp-enc.lp

Answer: 1
    travel(b,l) travel(l,w) travel(w,d)
    travel(d,p) travel(p,h) travel(h,b)
Optimization: 14

Answer: 2
    travel(b,p) travel(p,h) travel(h,w)
    travel(w,l) travel(l,d) travel(d,b)
Optimization: 12

Answer: 3
    travel(b,p) travel(p,h) travel(h,l)
    travel(l,w) travel(w,d) travel(d,b)
Optimization: 11

OPTIMUM FOUND
```

Listing 3. clingo Run on Facts and Encoding in Listing 1-2.

h : travel(h, b); p : travel(p, b)} > 1.

In view of the > 1 comparison relative to the #count aggregate, a contradiction is indicated as soon as connections from two cities among Dresden, Hamburg, and Potsdam to Berlin are picked for a round trip. Respective restrictions to a single predecessor apply to cities other than Berlin as well.

OPTIMIZE
After specifying solution candidates and requirements, the OPTIMIZE part addresses the optimality criterion in (c). To this end, the weak constraint in line 13 associates every place with the cost of the link to its successor in a round trip.[4] Focusing on the three connections from Berlin, we obtain

:~ travel(b, h), link(b, h, 2). [2, b]
:~ travel(b, l), link(b, l, 2). [2, b]
:~ travel(b, p), link(b, p, 1). [1, b]

Weak constraints resemble integrity constraints, but rather than eliminating solution candidates to which the expressed conditions apply, the lists enclosed in square brackets are gathered in a set. The sum of integers included as their first elements constitutes the total cost associated with an answer set and is subject to minimization. Regarding connections from Berlin, the fraction of the total cost is either 1 for Potsdam or 2 in case of Hamburg and Leipzig. Given that Hamburg and Leipzig cannot both succeed Berlin in a round trip, there is no urge to keep their respective lists distinct, for example, by adding the identifier h or l as an element. By reusing the same list instead, we actually reduce the number of factors taken into account in the total cost calculation, which can in turn benefit the performance of ASP solvers.[5]

DISPLAY
The final part, denoted by DISPLAY, includes the #show directive in line 15, declaring travel/2 as output predicate. This does not affect the meaning of the encoding, but instructs systems like clingo (Gebser et al. 2014) to restrict the printing of answer sets to atoms over travel/2. Indeed, facts as well as places given by start/1 and visit/1 are predetermined by an instance, and only the connections provided by travel/2 characterize a particular round trip.

Solution Computation

Assuming that the facts in listing 1 and the encoding in listing 2 are stored in text files called tsp-ins.lp and tsp-enc.lp, the output of a clingo run is given in listing 3. We see that clingo finds three round trips of decreasing cost, listed in lines beginning with Optimization: below the atoms over travel/2 in a corresponding answer set. While the first round trip is arbitrary and merely depends on heuristic aspects of the search in clingo, the second must be of smaller cost, and likewise the third. The latter cannot be improved any further, indicated by OPTIMUM FOUND in the last line, as it represents the shortest round trip shown in figure 2. For the instance at hand, this is the only round trip of cost 11, and some arbitrary witness among all optimal answer sets is determined in general.[6] Nondeterminisms, such as the (optimal) answer set found, are thus left up to the search in an ASP solver, while an encoding merely specifies requirements on intended outcomes. This distinguishes ASP from traditional logic programming languages like Prolog, in which programs have a procedural semantics based on the order of writing rules.

Summary

While the well-known TSP is conceptually simple, it gives room for exploring diverse modeling concepts and designs. Let us recap the main principles of the above ASP method.

A uniform problem representation separates facts describing an instance from a general problem encoding. The latter consists of schemata, expressed in terms of variables, that specify solutions for any problem instance. Such high-level modeling is crucial for elaboration tolerance, meaning that changes in a problem specification can be addressed by modest modifications of the representation. For example, when round trips shall be approximated for instances

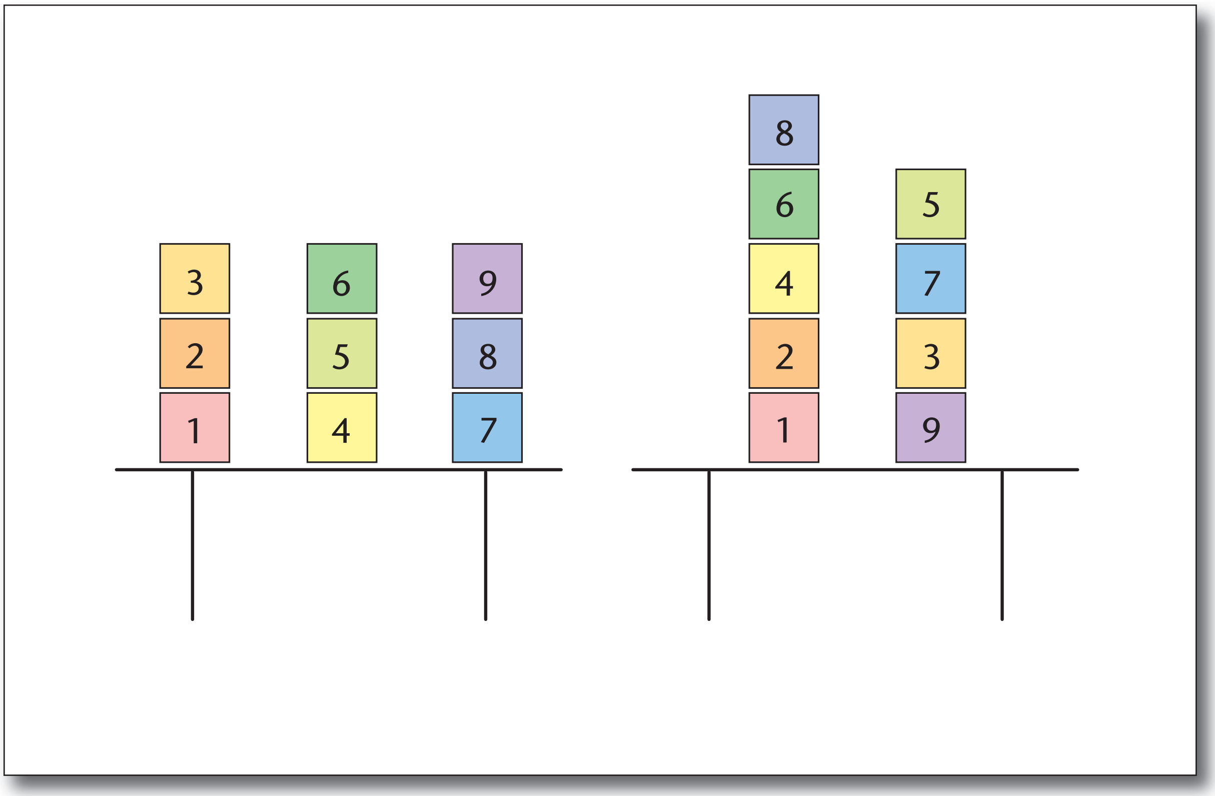


Figure 3. Initial and Goal Situation for Blocks World Planning.

that have no solution otherwise, the integrity constraint requiring all places to be visited can easily be turned into a weak constraint for admitting exceptions.

An ASP encoding is usually structured into parts addressing different concerns in a generate and test conception. The key parts, nicknamed GENERATE, DEFINE, TEST, and OPTIMIZE, provide solution candidates, analyze their relevant properties, eliminate invalid candidates, and evaluate solution quality.

The typical constructs used within the GENERATE, DEFINE, TEST, and OPTIMIZE parts are nondeterministic (choice) rules, deterministic rules, integrity constraints, or weak constraints, respectively. Deterministic rules make use of the expressivity of answer sets encompassing fix points, induction, and recursion. While TEST parts should typically stay compact regarding sufficient conditions, redundant constraints expressing nontrivial implied properties can benefit the search in an ASP solver. Weak constraints in an OPTIMIZE part can be made more effective by reducing the number of factors taken into account to evaluate solution quality.

An ASP encoding merely specifies requirements, but not how answer sets representing (optimal) solutions shall be computed. While admissible outcomes are fixed by the semantics, the way to find them is left up to ASP solvers.

Modeling the Blocks World Planning Problem

Beyond traditional one-shot solving, where a problem instance is fed to an isolated search process, multishot solving addresses series of evolving subtasks in an iterative manner. This is of interest in dynamic domains, such as logistics, policies, or robotics, dealing with recurrent tasks in a changing environment. To illustrate respective scenarios, we consider blocks world planning (Slaney and Thiébaux 2001), where blocks must be restacked on a table to bring them from their initial positions into a goal configuration.

Figure 3 displays an example scenario with nine blocks. In the initial situation, shown on the left, the blocks are arranged in three stacks, and the two stacks on the right constitute the goal situation. To change the configuration, a free block at the top of some stack can be moved on top of another stack or to the table. That is, a block offers room for at most one other block on top of it, while any number of blocks can be put on the table. A naive strategy to establish the goal situation thus consists of successively moving all blocks to the table, and then build up required stacks from the bottom. For the displayed scenario, this results in six moves to the table plus seven moves to construct the stacks on the right. However, the interest is to perform as few moves as

```
init(3,2). init(6,5). init(9,8).
init(2,1). init(5,4). init(8,7).
init(1,0). init(4,0). init(7,0). table(0).
goal(8,6).
goal(6,4). goal(5,7).
goal(4,2). goal(7,3).
goal(2,1). goal(3,9).
goal(1,0). goal(9,0).
```

Listing 4. Instance Specifying Situations in Figure 3 as Facts.

```
1   #program base.
2   % DOMAIN
3   do(X,Z)  :- init(X,Y),    not table(Y), table(Z).
4   do(X,Y)  :- goal(X,Y),    not table(Y).
5   on(X,Y,0) :- init(X,Y).

7   #program check(t).
8   % TEST
9   :- query(t), goal(X,Y),  not on(X,Y,t).
11  #program step(t).
12  % GENERATE
13  {move(X,Y,t) : do(X,Y)} = 1.
14  % DEFINE
15  move(X,t) :- move(X,Y,t).
16  on(X,Y,t)   :- move(X,Y,t).
17  on(X,Y,t)   :- on(X,Y,t–1), not move(X,t).
18  lock(Y,t)   :- on(X,Y,t–1), not table(Y).
19  firm(X,t)   :- on(X,Y,t), table(Y).
20  firm(X,t)   :- on(X,Y,t), firm(Y,t).
21  % TEST
22  :- lock(X,t),  move(X,t).
23  :- lock(Y,t),  move(X,Y,t).
24  :- init(Y,Z),  #count{X : on(X,Y,t)} > 1.
25  :- init(X,Z),  #count{Y : on(X,Y,t)} > 1.
26  :- init(X,Z),  not firm(X,t).

28  % DISPLAY
29  #show move/3.
```

Listing 5. Blocks World Encoding with Regard to Facts as in Listing 4.

needed, and in the following we show how shortest plans can be found using multishot solving.

Problem Instance

Similar to one-shot solving, applied to the TSP before, a problem instance is described in terms of facts. Those representing the situations displayed in figure 3 are given in listing 4, where the predicates init/2 and goal/2 specify the respective positions of blocks. In addition, table(0) declares 0 as identifier for the table, which is at the bottom of stacks in both the initial and the goal configuration.

Problem Encoding

To exploit the multishot solving capacities provided by the clingo system (Gebser et al. 2014), the encoding given in listing 5 is composed of three subprograms. Their names and parameters are introduced by #program directives, and a subprogram includes the rules up to the next such directive (if any). In the context of planning, the subprograms are dedicated to the following concerns:

(1) A base subprogram is processed once for providing auxiliary concepts along with setting up an initial configuration. (2) A check(*t*) subprogram is parametrized by a constant *t*, serving as a placeholder for successive integers starting from 0. For each time point taken as a value to replace *t* with, integrity constraints impose goal conditions. They include a dedicated atom query(*t*), provided by clingo for the current last time point, while obsolete conditions are deactivated to reflect an increased plan length. (3) A step(*t*) subprogram is likewise parametrized, yet *t* is replaced with successive integers starting from 1. This subprogram specifies transitions in terms of rules for picking actions, deriving atoms that represent a successor configuration, and asserting the validity of a transition. In contrast to check(*t*), such rules are not withdrawn but joined with others obtained at later time points.

The subprograms are further structured into conceptual DOMAIN, GENERATE, DEFINE, and TEST parts. Moreover, the DISPLAY part declares move/3 as output predicate (for all subprograms) through the #show directive in line 29, while the solving process of clingo focuses on shortest plans without requiring any OPTIMIZE part.

base

The first subprogram, called base, contributes a DOMAIN part consisting of the rules from lines 3 to 5. The idea of the predicate do/2 is to provide moves that could be relevant to a shortest plan. In particular, the rule in line 3 expresses that moving a block to the table can be useful for accessing the stack underneath, but only if such a stack exists and the block is not already on the table in the initial situation. Given the stacks on the left in figure 3, we thus derive that the blocks numbered 2, 3, 5, 6, 8, and 9 may be moved to the table. In addition, the rule in line 4 indicates moves to goal positions different from the table. Regarding the goal configuration on the right in figure 3, we obtain corresponding moves for all blocks but those numbered 1 and 9. As a result, derived facts over do/2 yield at most two relevant moves per block, while other moves may be legal but cannot belong to shortest plans.[7] The remaining rule in line 5 maps initial positions to derived facts over on/3, where the integer 0 denotes a time point associated with the initial configuration.

check(*t*)
The subprogram check(*t*) is parametrized by a constant *t* that is handled by clingo and replaced with successive integers starting from 0. It contributes a TEST part, including the integrity constraint in line 9, to deny plans such that some goal position is not yet established at the last time point referred to by *t*. This is accomplished by means of a dedicated atom query(*t*), provided by clingo for the current last time point and deactivated when proceeding to the next integer to replace *t* with. For example, the initial position of block 3 on the left in figure 3 does not match its goal position on the right, and a contradiction is indicated through

```
:- query(0), goal(3, 9), not on(3, 9, 0).
```

However, query(0) holds only as long as 0 is the last time point, while query(1) is used for 1 instead, and so on.

step(*t*)
The third subprogram, denoted by step(*t*), specifies transitions to time points referred to by its parameter *t*, serving as a placeholder for successive integers starting from 1. To begin with, the choice rule in line 13 constitutes the GENERATE part for picking one among the moves taken as relevant in the base subprogram. Note that the current time point is used as third argument in atoms over move/3, while do/2 remains fixed, regardless of the time point.

The deterministic rules in the DEFINE part from line 15 to 20 derive further atoms characterizing a transition at hand. A block changing its position is extracted through projection to move/2. Atoms over on/3, representing a successor configuration, are derived from a move as well as inertia applying to all blocks but the one that is moved. Again harnessing projection, the predicate lock/2 indicates blocks that were not on top of a stack and can thus not participate in legal moves. Finally, the predicate firm/2 provides blocks rooted on the table in a successor configuration, where noncircular derivations similar to those for places reachable in the TSP have the table as their starting point.

The TEST part, including the integrity constraints from line 22 to 26, then eliminates inexecutable plans. Moves involving inaccessible blocks are ruled out in line 22 and 23, which is actually sufficient to check that a plan can be executed.

Notably, the first of these integrity constraints reuses the projection to move/2, as only the moved block is of interest here. In addition, line 24 to 26 impose redundant state constraints, making explicit that, in any configuration, no block is under or on several objects and all blocks are rooted on the table.[8] For example, this expresses that block 3 cannot be at its goal position in between the blocks numbered 7 and 9 as long as the third stack displayed on the left in figure 3 is intact, no matter the performed moves.

```
$ clingo blocks-ins.lp blocks-enc.lp

Solving...
Solving...
Solving...
Solving...
Solving...
Solving...
Solving...
Solving...
Solving...
Solving...

 Answer: 1
   move(9,0,1) move(6,0,2) move(3,9,3)
   move(8,0,4) move(7,3,5) move(5,7,6)
   move(4,2,7) move(6,4,8) move(8,6,9)
```

Listing 6. clingo Run on Facts and Encoding in Listings 4 and 5.

Solution Computation

The output of clingo run on the facts in listing 4 and the encoding in listing 5, stored in text files blocks-ins.lp and blocks-enc.lp, is given in listing 6. The 10 lines saying *Solving...* indicate that 10 time points, namely successive integers from 0 to 9, have been used for the parameter of the check(*t*) subprogram. Apart from 0, they are also applied to the step(*t*) subprogram describing transitions, while base is processed just once at the beginning. Failed attempts to find an answer set for time points from 0 to 8 mean that there is no plan consisting of a respective number of moves. In turn, the plan found for time point 9 is shortest. The contained atoms over move/3 mainly convey that moving the blocks numbered 6, 8, and 9 to the table allows for building up the goal stacks. Alternative shortest plans, which can be obtained by enumerating answer sets, include a move of block 5, rather than block 8, to the table.

Summary

The blocks world is a dynamic domain, in which actions change the state of the environment over time. Shortest plans to progress from an initial to a goal situation can be found using multishot solving according to some basic principles:

An instance is provided by facts specifying the objects of interest along with initial and goal conditions.

A general problem encoding furnishes three subprograms, called base, check(*t*), and step(*t*). The latter are parametrized by a constant, here denoted *t*, serving as a placeholder for successive integers starting from 0 or 1, respectively.

The base subprogram is processed once at the beginning. It typically contributes a DOMAIN part setting up auxiliary concepts as well as atoms representing an initial configuration.

Occurrences of parameter *t* in the check(*t*) subprogram are successively replaced with integers from 0. The common purpose is to impose goal conditions by means of integrity constraints in a TEST part. By using a dedicated atom query(*t*) in integrity constraints, obsolete conditions are deactivated when proceeding to the next integer.

The step(*t*) subprogram is processed analogously to check(*t*), yet starting from integer 1 instead of 0. This predestinates step(*t*) to specify the transition to a successor configuration associated with *t*. The constructs typical for the GENERATE, DEFINE, and TEST parts are used to provide candidates, derive atoms characterizing them, and eliminate invalid transitions. Invariant properties can be expressed by incorporating redundant state constraints.

While facts as well as the base subprogram are processed only at the beginning, multishot solving by clingo iteratively adds rules obtained by replacing the parameters of the check(*t*) and step(*t*). subprograms with successive integers. This corresponds to gradually increasing the plan length until an answer set representing a shortest plan is found. The required length is often not known a priori, and multishot solving allows for discovering it.

Preferences and Optimization

The identification of preferred, or optimal, solutions is often indispensable in real-world applications, as illustrated on the TSP and blocks world scenarios above. In many cases, this also involves the combination of various qualitative and quantitative preferences. In fact, optimization statements representing objective functions based on summation or counting are integral concepts of ASP systems since their beginnings (manifested by #minimize and #maximize statements [Simons, Niemelä, and Soininen 2002] or weak constraints [Leone et al. 2006]). The built-in repertoire of current ASP systems also covers set-inclusion-based optimization (Gebser et al. 2015).

Other approaches to optimizing relative to specific and often more complex types of preference are furnished by dedicated external systems. Such approaches can be categorized into two classes (compare Delgrande et al. [2004]). On the one hand, we find prescriptive approaches to preference that take an order on rules and then enforce this order during the construction of optimal answer sets (Brewka and Eiter 1999). Such prescriptive approaches do not lead to an increase in computational complexity, which makes them amenable to implementation by compilation (Delgrande, Schaub, and Tompits 2003) or metainterpretation (Eiter et al. 2003). On the other hand, we have descriptive approaches that impose preferences among the answer sets of a program (Brewka, Niemelä, and Truszczyński 2003; Sakama and Inoue 2000; Son and Pontelli 2006). Unlike the former, these approaches typically lead to an elevated level of complexity, which makes their efficient implementation more challenging. The asprin system (Brewka et al. 2015) offers a general and flexible framework for computing optimal answer sets relative to preferences among them.[9] In particular, its library comprises all afore-cited descriptive approaches and further allows for freely combining preferences of qualitative and quantitative nature.

Further Extensions

The previous sections presented some popular modeling features and extensions, for example, relative to propositional satisfiability (SAT), going along with the ASP methodology. These include uniform problem representations using (first-order) variables within encodings, aggregates expressing collective conditions on sets, optimization, and multishot solving capacities. While such concepts already provide rich facilities for modeling and solving complex computational problems, we conclude with a (nonexhaustive) overview of further extensions.

Similar to disjunctive rules, nonmonotone recursive aggregates (Faber, Pfeifer, and Leone 2011; Ferraris 2011) allow for expressing problems at the second level of the polynomial time hierarchy. Finite-domain constraints specifying quantitative conditions can be addressed through dedicated backends (Aziz, Chu, and Stuckey 2013; Balduccini 2011; Mellarkod, Gelfond, and Zhang 2008; Ostrowski and Schaub 2012) or compilation (Banbara et al. 2015; Drescher and Walsh 2010). Moreover, translation approaches allow for handling real numbers (Bartholomew and Lee 2013; Liu, Janhunen, and Niemelä 2012). Extended functionalities like multishot solving are realized by combining ASP systems with scripting languages (Gebser et al. 2014). Further details regarding the integration of ASP systems with imperative languages or external information sources are provided by Lierler, Maratea, and Ricca (2016) and Erdem, Gelfond, and Leone (2016) in this issue. As also discussed in the latter article, high-level problem representations, for example, specified in terms of action languages, can in turn be mapped to ASP through corresponding front ends.

Acknowledgments

The authors were partially funded by DFG grants SCHA 550/8 and SCHA 550/9. We are grateful to Esra Erdem, Francesco Ricca, Gerhard Brewka, Mirek Truszczyński, and Wolfgang Faber for their comments on a preliminary draft of this article.

Notes

1. See, for example, Papadimitriou (1994) for an introduction to computational complexity.

2. Computing a shortest round trip is FPNP-complete (Papadimitriou 1994); that is, it can be accomplished by means of a polynomial number of queries to an NP-oracle.

3. Extensions to real numbers are presented by Bartholomew and Lee (2013) and Liu, Janhunen, and Niemelä (2012).

4. The weak constraint corresponds to the optimization statement #minimize {C, X : travel(X, Y), link(X, Y, C)}.

5. An even more elaborate penalization scheme based on relative cost differences is presented by Gebser et al. (2012, section 8.3).

6. Optimal answer sets can be enumerated using dedicated reasoning modes of clingo (Gebser et al. 2015).

7. More elaborate conditions to further restrict potential moves are provided by Slaney and Thiébaux (2001), and respective ASP encodings are presented by Gebser et al. (2012, section 8.2). While such domain knowledge as well as the encoding in listing 5 are specific to Blocks World Planning, domain-independent approaches to model actions and change are discussed by Erdem, Gelfond, and Leone (2016) in this issue.

8. Similar constraints are also included in encodings presented by Erdem and Lifschitz (2003), Gebser et al. (2012), and Lifschitz (2002) and further pave the way to partially ordered plans with parallel actions.

9. The only requirement is that evaluating a preference must be encodable in ASP (and thus have a complexity not beyond the second level of the polynomial time hierarchy)
.

References

Aziz, R.; Chu, G.; and Stuckey, P. 2013. Stable Model Semantics for Founded Bounds. *Theory and Practice of Logic Programming* 13(4–5): 517–532. dx.doi.org/10.1017/S1471068 41300032X

Balduccini, M. 2011. Industrial-Size Scheduling with ASP+CP. In *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-11),* volume 6645 of Lecture Notes in Artificial Intelligence, ed. J. Delgrande and W. Faber, 284–296. Berlin: Springer. dx.doi.org/10.1007/978-3-642-20895-9_33

Banbara, M.; Gebser, M.; Inoue, K.; Ostrowski, M.; Peano, A.; Schaub, T.; Soh, T.; Tamura, N.; and Weise, M. 2015. Aspartame: Solving Constraint Satisfaction Problems with Answer Set Programming. In *Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-15)*, volume 9345 of Lecture Notes in Artificial Intelligence, ed. F. Calimeri, G. Ianni, and M. Truszczyński, 112–126. Berlin: Springer. dx.doi.org/10.1007/978-3-319-23264-5_10

Bartholomew, M., and Lee, J. 2013. Functional Stable Model Semantics and Answer Set Programming modulo Theories. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI-13)*, ed. F. Rossi, 718-724. Palo Alto, CA: AAAI Press.

Brewka, G., and Eiter, T. 1999. Preferred Answer Sets for Extended Logic Programs. *Artificial Intelligence* 109(1–2): 297–356. dx.doi.org/10.1016/S0004-3702(99)00015-6

Brewka, G.; Delgrande, J.; Romero, J.; and Schaub, T. 2015. Asprin: Customizing Answer Set Preferences without a Headache. In *Proceedings of the Twenty-Ninth National Conference on Artificial Intelligence (AAAI-15)*, 1467–1474. Palo Alto, CA: AAAI Press.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer Set Programming at a Glance. *Communications of the ACM* 54(12): 92–103. dx.doi.org/10.1145/2043174.2043195

Brewka, G.; Niemelä, I.; and Truszczyński, M. 2003. Answer Set Optimization. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, ed. G. Gottlob and T. Walsh, 867–872. San Francisco: Morgan Kaufmann Publishers.

Bruynooghe, M.; Denecker, M.; and Truszczyński, M. 2016. ASP with First-Order Logic and Definitions. *AI Magazine* 37(3).

Delgrande, J.; Schaub, T.; and Tompits, H. 2003. A Framework for Compiling Preferences in Logic Programs. *Theory and Practice of Logic Programming* 3(2): 129–187. dx.doi.org/10.1017/S1471068402001539

Delgrande, J.; Schaub, T.; Tompits, H.; and Wang, K. 2004. A Classification and Survey of Preference Handling Approaches in Nonmonotonic Reasoning. *Computational Intelligence* 20(2): 308–334. dx.doi.org/10.1111/j.0824-7935.2004.00240.x

Drescher, C., and Walsh, T. 2010. A Translational Approach to Constraint Answer Set Solving. *Theory and Practice of Logic Programming* 10(4–6): 465–480. dx.doi.org/10.1017/S1471068410000220

Eiter, T.; Faber, W.; Leone, N.; and Pfeifer, G. 2003. Computing Preferred Answer Sets by Meta-Interpretation in Answer Set Programming. *Theory and Practice of Logic Programming* 3(4–5): 463–498.

Eiter, T.; Ianni, G.; and Krennwallner, T. 2009. Answer Set Programming: A Primer. In *Fifth International Reasoning Web Summer School (RW-09)*, volume 5689 of Lecture Notes in Computer Science, ed. S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M. Rousset, and R. Schmidt, 40–110. Berlin: Springer-Verlag.

Erdem, E., and Lifschitz, V. 2003. Tight Logic Programs. *Theory and Practice of Logic Programming* 3(4–5): 499–518. dx.doi.org/10.1017/S1471068403001765

Erdem, E.; Gelfond, M.; and Leone, N. 2016. Applications of ASP. *AI Magazine* 37(3).

Faber, W.; Pfeifer, G.; and Leone, N. 2011. Semantics and Complexity of Recursive Aggregates in Answer Set Programming. *Artificial Intelligence* 175(1): 278–298. dx.doi.org/10.1016/j.artint.2010.04.002

Ferraris, P. 2011. Logic Programs with Propositional Connectives and Aggregates. *ACM Transactions on Computational Logic* 12(4): 25. dx.doi.org/10.1145/1970398.1970401

Gebser, M.; Kaminski, R.; Kaufmann, B.; Romero, J.; and

Schaub, T. 2015. Progress in Clasp Series 3. In *Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-15)*, volume 9345 of Lecture Notes in Artificial Intelligence, ed. F. Calimeri, G. Ianni, and M. Truszczyński, 368–383. Berlin: Springer. dx.doi.org/10.1007/978-3-319-23264-5_31

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. San Rafael, CA: Morgan and Claypool Publishers.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2014. Clingo = ASP + Control: Preliminary Report. In *Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP-14)*, Theory and Practice of Logic Programming, Online Supplement .arXiv Volume: 1405.3694v1. Ithaca, NY: Cornell University Library.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9(3/4): 365–385. dx.doi.org/10.1007/BF03037169

Janhunen, T., and Niemelä, I. 2016. The Answer Set Programming Paradigm. *AI Magazine* 37(3).

Kaufmann, B.; Leone, N.; Perri, S.; and Schaub, T. 2016. Grounding and Solving in Answer Set Programming. *AI Magazine* 37(3).

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic* 7(3): 499–562. dx.doi.org/10.1145/1149114.1149117

Lierler, Y.; Maratea, M.; and Ricca, F. 2016. Systems, Engineering Environments, and Competitions. *AI Magazine* 37(3).

Lifschitz, V. 2002. Answer Set Programming and Plan Generation. *Artificial Intelligence* 138(1–2): 39–54. dx.doi.org/10.1016/S0004-3702(02)00186-8

Lifschitz, V. 2016. Answer Sets and the Language of Answer Set Programming. *AI Magazine* 37(3).

Liu, G.; Janhunen, T.; and Niemelä, I. 2012. Answer Set Programming via Mixed Integer Programming. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR-12)*, ed. G. Brewka, T. Eiter, and S. McIlraith, 32–42. Palo Alto, CA: AAAI Press.

Marek, V., and Truszczyński, M. 1999. Stable Models and an Alternative Logic Programming Paradigm. In *The Logic Programming Paradigm: A 25-Year Perspective*, ed. K. Apt, V. Marek, M. Truszczyński, and D. Warren, 375–398. Berlin: Springer-Verlag. dx.doi.org/10.1007/978-3-642-60085-2_17

Mellarkod, V.; Gelfond, M.; and Zhang, Y. 2008. Integrating Answer Set Programming and Constraint Logic Programming. *Annals of Mathematics and Artificial Intelligence* 53(1–4): dx.doi.org/10.1007/s10472-009-9116-y

Niemelä, I. 1999. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. *Annals of Mathematics and Artificial Intelligence* 25(3–4): 241–273. dx.doi.org/10.1023/A:1018930122475

Ostrowski, M., and Schaub, T. 2012. ASP modulo CSP: The Clingcon System. *Theory and Practice of Logic Programming* 12(4–5): 485–503. dx.doi.org/10.1017/S1471068412000142

Papadimitriou, C. 1994. *Computational Complexity*. Boston, MA: Addison-Wesley.

Sakama, C., and Inoue, K. 2000. Prioritized Logic Programming and its Application to Commonsense Reasoning. *Artificial Intelligence* 123(1–2): 185–222. dx.doi.org/10.1016/S0004-3702(00)00054-0

Schlipf, J. 1995. The Expressive Powers of the Logic Programming Semantics. *Journal of Computer and System Sciences* 51: 64–86. dx.doi.org/10.1006/jcss.1995.1053

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence* 138(1–2): 181–234. dx.doi.org/10.1016/S0004-3702(02) 00187-X

Slaney, J., and Thiébaux, S. 2001. Blocks World Revisited. *Artificial Intelligence* 125(1–2): 119–153. dx.doi.org/10.1016/S0004-3702(00) 00079-5

Son, T., and Pontelli, E. 2006. Planning with Preferences Using Logic Programming. Theory and Practice of Logic Programming 6(5): 559–608. dx.doi.org/10.1017/S147106 8406002717

Van Gelder, A.; Ross, K.; and Schlipf, J. 1991. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM* 38(3): 620–650. dx.doi.org/10.1145/116825.116838

Martin Gebser is a postdoctoral researcher at the University of Potsdam, Germany. His research interests include theoretical and practical aspects of declarative problem solving methods. In particular, Gebser contributes to the development of state-of-the-art solving systems in answer set programming and related areas. He gathered practical expertise in knowledge representation and reasoning technology by working on applications in systems biology, system synthesis, configuration, and related areas. Gebser cochaired the ICLP doctoral consortium in 2013 and 2014 as well as the answer set programming competition in 2009, 2014, and 2015.

Torsten Schaub is a university professor at the University of Potsdam, Germany, and holds an international chair at Inria Rennes, France. He is a fellow of ECCAI and the current president of the Association of Logic Programming. His current research focus lies on answer set programming and its applications, which materializes at potassco.sourceforge.net, the home of the open source project Potassco, which bundles software for ASP developed at Potsdam.

Systems, Engineering Environments, and Competitions

Yuliya Lierler, Marco Maratea, Francesco Ricca

■ *The goal of this article is threefold. First, we trace the history of the development of answer set solvers, by accounting for more than a dozen of them. Second, we discuss development tools and environments that facilitate the use of answer set programming technology in practical applications. Last, we present the evolution of the answer set programming competitions, prime venues for tracking advances in answer set solving technology.*

Answer set programming (ASP) is a prominent knowledge representation paradigm that found numerous successful industrial and scientific applications including product configuration, decision support systems for space shuttle flight controllers, large-scale biological network repairs, and team building and scheduling (see Erdem, Gelfond, and Leone [2016] for more details). The success story of ASP is largely due to its modeling language and the availability of efficient and effective answer set programming tools that encompass grounders, solvers, and engineering environments. Syntactically, simple answer set programs (or ASP programs) look like Prolog logic programs. Yet solutions to such programs are represented in ASP by sets of atoms called answer sets, and not by substitutions, as in Prolog. An answer set system typically consists of two tools, a grounder and a solver, and is used to compute answer sets. Since 2007, the series of ASP competitions has promoted the collection of challenging benchmarks as well as supplied researchers with a uniform platform for tracking the progress in the development of ASP solving technologies. More recent introduction of programming environments eased the development of ASP programs and the implementation of software systems based on ASP. In this article, we present a brief survey of (1) existing answer set grounders and solvers, (2) engineering tools and environments that support production of ASP-based applications, and (3) ASP competitions. Our goal is to provide an interested reader with an outlook on existing ASP technologies together with sufficient literature pointers rather than in-depth explanation of research and engineering ideas behind these technologies.

ASP Grounders and Solvers

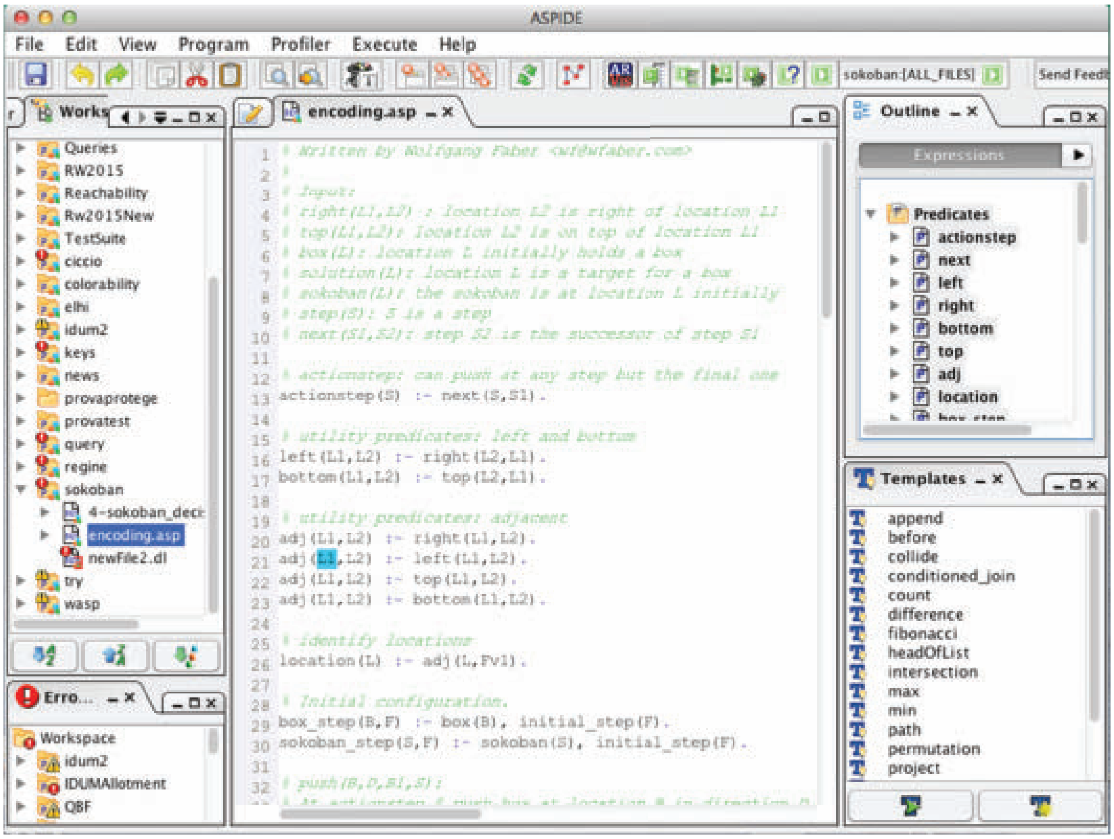
In ASP, solutions to logic programs are represented by sets of atoms called answer sets (stable models) (Gelfond and Lifschitz 1988). Answer set solvers, such as Smodels (Simons, Niemelä, and Soininen 2002), Smodels$_{cc}$ (Ward and Schlipf 2004), and DLV (Leone et al. 2006), to name some of the first implementations, compute answer sets of a given propositional logic program. Conceptually, most answer set solvers have a lot in common with satisfiability solvers (or SAT solvers), systems that compute satisfying assignments for propositional formulas in clausal normal form. Tools called grounders complement answer set solvers. A grounder is a software system that takes a logic program with variables as its input and produces a propositional program as its output so that the resulting propositional program has the same answer sets as the input program. Propositional programs are crucial in devising efficient solving procedures, yet it is the logic programming language with variables that facilitates modeling and effective problem solving in ASP.

There are three main grounders available for ASP practitioners: Lparse (Syrjänen 2001), DLV-grounder (Leone et al. 2006), and GrinGo (Gebser, Schaub, and Thiele 2007). Grounders Lparse and GrinGo are stand-alone tools that are commonly used as front ends for distinct answer set solvers. System DLV encapsulates both a grounder and a solver. However, calling the system with an option *–instantiate* produces propositional (ground) program for the given input and exits the computation without accessing the solving procedure of the system.

"Native" answer set solvers such as Smodels, Smodels$_{cc}$, and DLV are based on specialized search procedures in the spirit of the classic backtrack-search Davis-Putnam-Logemann-Loveland (DPLL) algorithm. The DPLL algorithm and its modifications are at the core of the majority of modern SAT solvers. This algorithm consists of performing three basic operations: decision, unit propagate, and backtrack. The unit propagate operation is based on a simple inference rule in propositional logic that given a formula F in clausal normal form allows utilization of knowledge about unit clauses occurring in F or being inferred so far by the DPLL procedure in order to conclude new inferences. Native answer set solvers replace the unit propagate operation of DPLL by specialized operations based on inference rules suitable in the context of logic programs. For example, Smodels implements five propagators called *Unit Propagate, All Rules Canceled, Backchain True, Backchain False,* and *Unfounded* (for details on these propagators see, for instance, Lierler and Truszczyński [2011]). In DLV the basic chronological backtrack-search was improved introducing backjumping and look-back heuristics (Maratea et al. 2008). Solver Smodels$_{cc}$ extends the algorithm of Smodels by conflict-driven backjumping and clause learning. Clause learning is an advanced solving technique that originated in SAT and proved to be powerful. The distinguishing feature about the answer set solver DLV is its ability to handle disjunctive answer set programs. In rules of such programs a disjunction of atoms in place of a single atom is allowed in the heads. The problem of deciding whether a disjunctive program has an answer set is Σ_2^P-complete. The other systems capable of dealing with such programs are GNT, Cmodels, claspD, and WASP. Brochenin, Lierler, and Maratea (2014) survey the key features of disjunctive answer set solvers.

Answer sets of a "tight" logic program are in a one-to-one correspondence with models of completion, a propositional logic formula proposed by Clark (1977). This observation immediately leads to an idea that answer sets of a tight logic program can be found by running a SAT solver on clausified program's completion. Tightness is a simple syntactic condition that many interesting ASP applications satisfy. An inception of a SAT-based answer set solver Cmodels (Giunchiglia, Lierler, and Maratea 2006) is due to this fact. It starts its computation by forming completion of an input program. Then Cmodels calls a SAT solver for enumerating models of program's completion. Lin and Zhao (2004) proposed a concept of loop formula so that given a program, extending its completion by its loop formulas results in a propositional formula, whose models are in a one-to-one correspondence with answer sets. In general case, the number of loop formulas can be exponentially larger than the size of a program. Nevertheless, solvers ASSAT (Lin and Zhao 2004) and Cmodels found means to utilize the concept of a loop formula in order to compute answer sets of a program. This computation typically requires multiple interactions with a SAT solver. Loop formulas are related to so called unfounded sets, which is the basis behind Unfounded propagator often employed in answer set solvers. Both ASSAT and Cmodels take advantage of conflict-driven backjumping and clause learning available in SAT technology that they rely on.

Answer set solver clasp (Gebser, Kaufmann, and Schaub 2012a) borrows the ideas from both native and "loop formula"–based solvers. Just as Cmodels or ASSAT, it starts its computation by forming the clausified completion of an input program. Next it implements a search procedure that relies on a unit propagator stemming from SAT on the program's completion and an Unfounded propagator stemming from native answer set solvers. System clasp imple-



```
% Written by Wolfgang Faber <wf@wfaber.com>
%
% Input:
% right(L1,L2) : location L2 is right of location L1
% top(L1,L2): location L2 is on top of location L1
% box(L): location L initially holds a box
% solution(L): location L is a target for a box
% sokoban(L): the sokoban is at location L initially
% step(S): S is a step
% next(S1,S2): step S2 is the successor of step S1

% actionstep: can push at any step but the final one
actionstep(S) :- next(S,S1).

% utility predicates: left and bottom
left(L1,L2) :- right(L2,L1).
bottom(L1,L2) :- top(L2,L1).

% utility predicates: adjacent
adj(L1,L2) :- right(L1,L2).
adj(L1,L2) :- left(L1,L2).
adj(L1,L2) :- top(L1,L2).
adj(L1,L2) :- bottom(L1,L2).

% identify locations
location(L) :- adj(L,Fv1).

% Initial configuration.
box_step(B,F) :- box(B), initial_step(F).
sokoban_step(S,F) :- sokoban(S), initial_step(F).

% push(B,D,B1,S):
```

Figure 1. The User Interface of ASPIDE.

ments conflict-driven backjumping and clause learning. The PC(ID)/answer set solvers MINISAT(ID) (Wittocx, Marïen, and Denecker 2008) and WASP (Alviano et al. 2015; Alviano and Dodaro 2016) share a lot in common with the design of clasp. Lierler and Truszczyński (2011) present a study that draws parallels between several answer set solvers.

System LP2SAT (Janhunen 2006) represents a family of "translation-based" solvers. This family relies on a translation of propositional logic programs into logic formulas so that models of the resulting formula are in one-to-one correspondence with the answer sets of the input program. This translation may add auxiliary atoms in the process and may include the normalization of aggregates as well as the encoding of level mappings for nontight problem instances. The latter can be expressed in different terms including acyclicity checking. Pseudo-Boolean and SAT formulations resulted in a variety of systems, such as LP2ACYCASP+CLASP, LP2ACYCPB+CLASP, LP2ACYCSAT+CLASP, and LP2ACYCSAT+GLUCOSE. Systems LP2DIFFZ3 and LP2DIFF+YICES utilize satisfiability modulo theory solvers (Nieuwenhuis, Oliveras, and Tinelli 2006) through a translation from logic programs to difference logic. Among other alternatives, solver LP2MIP relies on a translation into a mixed integer programming problem, and runs CPLEX as back-end, while LP2NORMAL+CLASP normalizes aggregates (of small to medium size) and uses clasp as a back-end ASP solver.

ASP systems have been also extended to exploit multicore and multiprocessor machines by introducing parallel evaluation methods. In particular, parallel techniques for the instantiation of programs were proposed as extensions of the Lparse (Pontelli, Balduccini, and Bermudez 2003) and DLV (Perri, Ricca, and Sirianni 2013) grounders. Recent approaches for extending the algorithm of clasp include that of Gebser, Kaufmann, and Schaub (2012b).

Automated algorithm selection techniques have been employed in ASP for obtaining solvers performing well across a wide heterogeneous set of inputs. The idea is to leverage a number of efficient implementations (or heuristically different variants of these) and apply machine-learning techniques for learning from a training set how to choose the "best" solver for an input program. System CLASPFOLIO (Gebser et al. 2011) combines variants of clasp, and is a representative of portfolio solving in ASP. System ME-ASP (Maratea, Pulina, and Ricca 2014), instead, implements a multiengine portfolio ASP solver, by combining several solvers. The adoption of the ASP-CORE-2 standard input language[2] allowed the application of algorithm selection techniques also to the grounding step.

Constraint answer set programming is a recent direction of research that attempts to combine advances in answer set programming with these in constraint processing. This new area has already demonstrated promising results, including the development of the solvers ACSOLVER, CLINGCON, EZC-SP, IDP, and MINGO. Lierler (2014) surveys the key features of constraint answer set programming languages and systems. This direction of research is inspired by the advances in the related field of satisfiability modulo theories.

Engineering Environments

The availability of efficient solvers makes ASP a valuable tool for many computationally intensive real-world applications. Effective large-scale software engineering requires infrastructure that includes advanced editors, debuggers, etc. These tools are usually collected in integrated development environments (IDE) that ease the accomplishment of various programming tasks by both novice and skilled software developers. The development of application programming interfaces (APIs) is also essential for allowing ASP-based solutions within large software frameworks common in the modern, highly technological world. The following subsections present an overview of both the IDEs for writing ASP programs and the available APIs for building full-fledged software systems based on ASP.

Development Environments for ASP

Several tools have been proposed in the last few years that aim at solving specific tasks arising during the development of ASP programs, including specialized editors, debuggers, testing tools, and visual programming tools. The IDEs that collect several tools in the same framework are also now available. SeaLion (Busoniu et al. 2013) is the first environment offering debugging for programs with variables. It also features unique tools for model-based engineering (using ER diagrams to model domains of answer set programs), testing through annotations, and bidirectional visualization of interpretations. The ASPIDE IDE (Febbraro, Reale, and Ricca 2011) is a comprehensive framework that integrates several tools for advanced program composition and execution. To provide an overview of insides of ASP IDEs we briefly outline key features of ASPIDE.

A snapshot of the user interface of ASPIDE is reported in figure 1. Logic programs are organized in projects collected in a workspace (displayed in the left panel in figure 1). The main editor for ASP programs (central frame in figure 1) offers code line numbering, find/replace, undo/redo, copy/paste, coloring of keywords, dynamic highlighting of predicate names, variables, strings, and comments. The editor is able to complete (on request) predicate names (learned while reading from the files belonging to the same project), as well as variable names (suggested by taking into account the rule one is currently writing). Programs can be modified in an assisted way, for instance, by considering bindings of variables, or by applying custom rewritings (that can be user defined). Syntax errors and some syntactic conditions (for instance, safety) are checked while writing and promptly outlined. ASPIDE suggests quick fixes that can be applied (on request) by automatically changing the affected part of code. Common programming patterns (such as guessing with disjunctive rules and specific constraints) are available as code templates that are expanded as rules (again on request). An outline view (left frame in figure 1) graphically represents program elements for quick access to the corresponding definition. Users accustomed to graphic programming environments can draw logic programs by exploiting a QBE-like tool for building logic rules (Febbraro, Reale, and Ricca 2010). The user can switch from the text editor to the visual one (and vice versa) thanks to a reverse-engineering mechanism from text to graphical format. The execution of ASP programs is fully customizable through a number of shortcuts, including toolbar buttons and drop-down menus, for a quick execution of files. The results are presented to the user in a view combining tabular representation of predicates and a treelike representation of answer sets. ASPIDE supports test-driven software development in the style of JUnit (see more details in Febbraro et al. [2011]).

Program development is enhanced in ASPIDE by additional tools such as the dependency graph visualizer, designed to inspect predicate dependencies and browsing the program; the debugger to find bugs (Dodaro et al. 2015); the DLV profiler; the ARVis comparator of answer sets; the answer set visualizer IDP-Draw; and the data source plug-in that simplifies the connection to external DBMSs through JDBC. Notably, ASPIDE is an extensible environment that can be extended by users providing new plug-ins that support (1) new input formats, (2) new program rewritings, and even (3) customization of the visualization/output format of solver results. System ASPIDE is written in Java and is available for all the major operating systems, including Linux, Mac OS, and Windows.[1]

Building Full-Fledged Applications with ASP

IDEs for ASP provide clear advantages for logic programmers, but are not enough to enable assisted development of full-fledged industry-level applications (Grasso et al. 2011, Ricca et al. 2012). ASP is not a full general-purpose language. Thus, ASP programs are eventually embedded in software components developed in imperative/object-oriented programming languages.

The development of APIs, which offer methods for interacting with an ASP system from an embedding program, is a necessary step in accommodating the use of ASP-based solutions within large software systems. Among the first proposals was the DLV Wrapper, a library that allows embedding ASP programs and controlling the execution of the DLV system from a Java program, and the ONTODLV API, a richer API that allows embedding ontologies and reasoning modules developed using the ONTODLP language (Ricca et al. 2009). More recently, the Potassco group from the University of Potsdam supported the embedding of ASP in Python and Lua programs using GrinGo and clasp. These interfaces provide a finer-grained access to grounder and solver functionality, and also allow incremental solving.

In APIs, however, the burden of the integration between ASP and Java is still in the hands of the programmer, who must take care of the (often repetitive and) time-consuming development of scaffolding code that executes the ASP system and gets data back and forth from logic-based to imperative representations.

These observations inspired the development of a hybrid language, called JASP (Febbraro et al. 2012), that transparently supports a bilateral interaction between ASP and Java. JASP introduces minimal syntax extensions both to Java and ASP. Its specifications are both easy to learn by programmers and easy to integrate with other existing Java technologies. The programmer can simply embed ASP code in a Java program without caring about the interaction with the underlying ASP system. An "ASP program" can access Java variables, and the answer sets that result from the execution of the ASP code are automatically stored in Java objects, possibly populating Java collections, in a transparent way. A distinctive feature of JASP is the clean separation between the two integrated programming paradigms interacting through a standard object-relational mapping (ORM) interface. JASP supports both (1) a default mapping strategy, which fits the most common programmers' requirements, and (2) custom ORM strategies, which can be specified according to the Java Persistence API (JPA) to perfectly suit enterprise application development standards. The framework also encompasses an implementation of JASP as a plug-in for the Eclipse platform, called JDLV.

Another hybrid language combining Java and ASP was proposed by Oetsch, Pührer, and Tompits (2011), which employs a radically different strategy for the interaction with Java. For instance, Java methods including constructors can be called by exploiting special atoms in ASP rules.

The ASP Competition Series

ASP competitions are the events of the ASP community, where ASP solvers are evaluated for efficiency. Since 2007, they take place biennially and are affiliated with the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR); the exception was the fifth event that took place in 2014, affiliated with the 30th International Conference on Logic Programming (ICLP). In this section we present the history and evolution, in terms of format, solvers participation, and winners, of the ASP competition series, by first summarizing the editions up to the fourth edition, and then focusing on the 2014 and 2015 events.

ASP Competitions up to 2013

In September 2002, participants of the Dagstuhl Seminar on Nonmonotonic Reasoning, Answer Set Programming, and Constraints (Brewka et al. 2002) agreed that standardization was one of the key issues for the development of ASP. This led to the the initiative to establish an infrastructure for benchmarking ASP solvers, as already in use in related research fields of SAT and constraint programming. The first informal competition took place during that workshop in Dagstuhl, featuring five systems, namely DLV, Smodels, ASSAT, Cmodels, and ASPPS. The second informal edition took place in 2005, during another Dagstuhl meeting. Since then, the ASP competitions have been established as reference events for the community.

The First ASP Competition (Gebser et al. 2007) was organized in Potsdam with two main goals. The first goal was to collect benchmarks. It was achieved through a call for benchmarks to members of the community. The second goal was to set up a fair competition environment. In the competition, only decision problems were considered. There were three categories of benchmarks involved: (1) MGS (modeling, grounding, solving), where benchmarks were specified by a problem statement, a set of instances, and the names of the predicates and their arguments to be used by programmers to encode solutions; (2) SCORE (solver, core language), where benchmarks consisted of ground normal and disjunctive programs in the format common to DLV and Lparse; and (3) SLparse (solver, Lparse language), where benchmarks consisted of ground programs in Lparse format with aggregates.

Ten ASP solvers participated, with several new solvers compared to the first informal events, namely ASPER, clasp, NOMORE, GNT, LP2SAT, and PBMODELS, thus establishing the advent of CDCL solvers and solvers based on eager translation-based approaches to ASP solving. Solvers were ranked in terms of number of solved instances: DLV won the MGS and SCORE categories, while clasp was the best solver on the SLparse category.

The Second ASP Competition (Denecker et al.

2009) was organized by Katholieke Universiteit Leuve. Differently from the precursor event, it was a model and solve team competition: A number of well-specified benchmarks (collected, again, through a call for benchmarks, and divided into categories based on complexity) had to be modeled by the participant teams and solved with a system of their choice. Moreover, optimization problems were introduced in the runnings. Sixteen solvers entered the competition: among others, IDP and approaches based on compilation into SMT participated for the first time. The Potassco team (Potsdam University) won the overall competition, and performed best on both decision and optimization problems.

In the 2011 and 2013 editions, the format consisted of two different tracks: a model and solve and a system track. The former was the continuation of the Second ASP Competition tradition, while the later was in spirit of the First ASP Competition, which aimed at fostering language standardization and at allowing participants to compete on given encodings under fixed conditions. Both tracks featured a selected suite of domains, chosen again by means of an open call for the benchmarks stage, and organized in classes based on complexity.

The Third ASP Competition (Calimeri, Ianni, and Ricca 2014) was organized by the University of Calabria. Eleven systems participated in the system track, among them the first portfolio answer set solver, CLASPFOLIO, and a number of translation-based solvers. Six teams entered the model and solve track, including the Fast Downward team from the planning community. Winners were determined with a scoring computed by the number of solved instances and the CPU time, plus the quality of the solution in case of optimization problems: claspD won the system track, while the Potassco team won the model and solve track. The portfolio solver CLASPFOLIO was the best system on the NP class, which included NP-complete problems and any problem in NP not known to be polynomially solvable.

The Fourth ASP Competition (Alviano et al. 2013) was jointly organized by TU Vienna and the University of Calabria. The design of the event was similar to the previous edition, with some important changes. The competition introduced the standard input language ASP-CORE-2.0[2] for the system track (an evolution of the ASP-CORE language proposed in 2011); exceptions were made and problem encodings in legacy formats were still admitted. Also a system track for parallel systems was introduced. Sixteen solvers entered the system track: most of these solvers participated in the earlier editions, with the notable exception of the WASP solver. Seven teams entered the model and solve track. About the results: claspD and its parallel version claspD-MT (Potsdam University) won the system track, while the Potassco team was the winner of the model and solve track.

The Fifth and Sixth ASP Competitions

The fifth and the sixth editions of the ASP competition series introduced significant modifications to the trend. We first outline the main changes, and then we speak of the two events separately.

The model and solve track was no longer an integral part of the events. Rather, it was organized as an (informal) on-site event. The reasons for this are that, first, organizing, and even more participating in, such a track requires a substantial amount of work. In addition, the participation from neighboring research communities was rather limited, probably due to the presence of competitions in the related research communities and the nonnegligible effort of participating. The first on-site event, called ASP Modeling Competition 2014, saw five participating teams. Each team was formed by three researchers and was allocated a fixed amount of time for solving a few problems.

The Fifth Answer Set Programming Competition (Calimeri et al. 2016) broke the usual timeline of the competition series in order to join the Olympic Games at the Vienna Summer of Logic, in affiliation with the 30th International Conference on Logic Programming (ICLP). It was jointly organized by the Aalto University, the University of Calabria, and the University of Genoa. This event did not feature a call for benchmarks and mostly relied on 2013 benchmarks. It was mainly conceived as a rerun of the system track of the previous event: participants to the 2013 event were invited to submit new versions of their solvers, but also new solvers were welcome. Several significant design changes and improvements in the competition settings were introduced, that is, (1) benchmark classes (called tracks in this edition) were defined based on the presence of language constructs (for example, aggregates, choice rules, presence of queries) in problem encoding rather than on a complexity basis, in order to both push the adoption of the new standard and allow participation also to solvers that may have not included all constructs, (2) novel encodings for almost all problems were proposed, to overcome some observed limitations of 2013 encodings, and (3) a simplified scoring schema for decision problems, based on solved instances only, and a scoring schema for optimization problems solely based on the solvers's ranking on solution quality, were employed. Sixteen solvers entered the competition. Answer set solver clasp was the winner on the single-processor category, while its multithreaded version clasp-MT won the multithreaded category. Interestingly, the solver LP2NORMAL+CLASP, which normalizes aggregates and then resorts to clasp, was the best solver in an intermediate track, allowing for the full ASP-CORE-2 language, except optimization statements and no head cycle-free disjunction.

The Sixth ASP Competition (Gebser, Maratea, and Ricca 2015) was jointly organized by the same institutions as the previous event. Its design maintained some choices of the last event, for example, tracks based on language features, the scoring schemes, and the adherence to the ASP-CORE-2 standard language. It also presented some novelties, for instance (1) a call for benchmarks stage focused on obtaining new benchmarks arising from applications of practical impact, and/or being ASP focused, that is, whose encodings are nontight, and (2) a benchmarks selection stage was introduced to classify instances according to their expected hardness. Moreover, a "marathon" track was added, where the best performing systems are given more time for solving hard instances. Thirteen solvers entered the competition. The winner of the regular track was the multiengine solver ME-ASP, while the winner of the marathon track was WASP.

Conclusion

Answer set programming is a thriving research field that features dozens of solvers and applications. Engineering environments for ASP facilitate the adoption of the technology by a broad spectrum of users. Quest for the ideal settings of the ASP competitions attests to the ever-changing, fast-paced life of the field, which strives to advance answer set programming.

Acknowledgments

This work was partially supported by MIUR under PON project SI-LAB BA2KNOW Business Analitycs to Know, and by Regione Calabria, programme POR Calabria FESR 2007-2013, projects ITravel PLUS and KnowRex: Un sistema per il riconoscimento e lestrazione di conoscenza.

Notes

1. Available from www.mat.unical.it/ricca/aspide.

2. The ASP-Core-2 Input Language Format is available from www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.01c.pdf.

References

Alviano, M., and Dodaro, C. 2016. Completion of Disjunctive Logic Programs. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*. Palo Alto, CA: AAAI Press.

Alviano, M.; Calimeri, F.; Charwat, G.; Dao-Tran, M.; Dodaro, C.; Ianni, G.; Krennwallner, T.; Kronegger, M.; Oetsch, J.; Pfandler, A.; Pührer, J.; Redl, C.; Ricca, F.; Schneider, P.; Schwengerer, M.; Spendier, L. K.; Wallner, J. P.; and Xiao, G. 2013. The Fourth Answer Set Programming Competition: Preliminary Report. In *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013,* Lecture Notes in Computer Science volume 8148, 42–53. Berlin: Springer.

Alviano, M.; Dodaro, C.; Faber, W.; Leone, N.; and Ricca, F. 2015. Advances in WASP. In *Logic Programming and Nonmonotonic Reasoning — 13th International Conference, LPNMR 2015,* Lecture Notes in Computer Science volume 9345, 40–54. Berlin: Springer. dx.doi.org/10.1007/978-3-319-23264-5_5

Brewka, G.; Niemelä, I.; Schaub, T.; and Truszczyński, M. 2002. Workshop on Nonmonotonic Reasoning, Answer Set Programming and Constraints. Papers presented at the Dagstuhl Seminar Nr. 02381, Nonmonotonic Reasoning, Answer Set Programming and Constraints, System Competition. September 15–20, Waldern, Germany (www.dagstuhl .de/02381).

Brochenin, R.; Lierler, Y.; and Maratea, M. 2014. Abstract Disjunctive Answer Set Solvers. In *ECAI 2014: 21st European Conference on Artificial Intelligence, Including Prestigious Applications of Intelligent Systems (PAIS 2014),* volume 263 of Frontiers in Artificial Intelligence and Applications, 165–170. Amsterdam: IOS Press.

Busoniu, P.; Oetsch, J.; Pührer, J.; Skocovsky, P.; and Tompits, H. 2013. SeaLion: An Eclipse-Based IDE for Answer-set Programming with Advanced Debugging Support. *Theory and Practice of Logic Programming* 13(4-5): 657–673. dx.doi.org/10.1017/S1471068413000410

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, I.; Kaminski, R.; Krennwallner, T.; Leone, N.; Ricca, F.; and Schaub, T. 2013. The ASP-Core-2 Input Language Format. Technical Report, Università Della Calabria, Consenza, Italy. www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.01c.pdf

Calimeri, F.; Gebser, M.; Maratea, M.; and Ricca, F. 2016. Design and Results of the Fifth Answer Set Programming Competition. *Artificial Intelligence* 231: 151–181. dx.doi.org/10.1016/j.artint.2015.09.008

Calimeri, F.; Ianni, G.; and Ricca, F. 2014. The Third Open Answer Set Programming Competition. *Theory and Practice of Logic Programming* 14(1): 117–135. dx.doi.org/10.1017/S1471068412000105

Clark, K. L. 1977. Negation as Failure. In *Logic and Data Bases,* 293–322. New York: Plenum Press.

Denecker, M.; Vennekens, J.; Bond, S.; Gebser, M.; and Truszczyński, M. 2009. The Second Answer Set Programming Competition. In *Logic Programming and Nonmonotonic Reasoning, 10th International Conference,* LPNMR 2009, Lecture Notes in Computer Science volume 5753, 637–654. Berlin: Berlin: Springer. dx.doi.org/10.1007/978-3-642-04238-6_75

Dodaro, C.; Gasteiger, P.; Musitsch, B.; Ricca, F.; and Shchekotykhin, K. 2015. Interactive Debugging of Nonground ASP Programs. In *Logic Programming and Nonmonotonic Reasoning, 13th International Conference,* LPNMR 2015, Lecture Notes in Computer Science volume 9345, 279–293. Berlin: Springer.

Erdem, E.; Gelfond, M.; Leone, N. 2016. Applications of Answer Set Programming. *AI Magazine* 37(3).

Febbraro, O.; Leone, N.; Grasso, G.; and Ricca, F. 2012. JASP: A Framework for Integrating Answer Set Programming with Java. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference* (KR-12). Palo Alto, CA: AAAI Press.

Febbraro, O.; Leone, N.; Reale, K.; and Ricca, F. 2011. Unit Testing in ASPIDE. In *Applications of Declarative Programming and Knowledge Management: 19th International Conference, INAP 2011, and 25th Workshop on Logic Programming, WLP 2011,* Lecture Notes in Computer Science volume 7773, 345–364. Berlin: Springer.

Febbraro, O.; Reale, K.; and Ricca, F. 2010. A Visual Interface for Drawing ASP Programs. In *Proceedings of the 25th Italian Conference on Computational Logic,* volume 598 of CEUR Workshop Proceedings. Aachen, Germany: RWTH Aachen University.

Febbraro, O.; Reale, K.; and Ricca, F. 2011. ASPIDE: Integrated Development Environment for Answer Set Programming. In *Logic Programming and Nonmonotonic Reasoning, 11th International Conference,* Lecture Notes in Computer Science volume 6645, 317–330. Berlin: Springer.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Schaub, T.; Schneider, M. T.; and Ziller, S. 2011. A Portfolio Solver for Answer Set Programming: Preliminary Report. In *Logic Programming and Nonmonotonic Reasoning — 11th International Conference,* LPNMR 2011, Lecture Notes in Computer Science volume 6645, 352–357. Berlin: Springer.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012a. Conflict-Driven Answer Set Solving: From Theory to Practice. *Artificial Intelligence* 187: 52–89. dx.doi.org/10.1016/j.artint.2012.04.001

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012b. Multithreaded ASP Solving with Clasp. *Theory and Practice of Logic Programming* 12(4–5): 525–545. dx.doi.org/10.1017/S1471068412000166

Gebser, M.; Liu, L.; Namasivayam, G.; Neumann, A.; Schaub, T.; and Truszczyński, M. 2007. The First Answer Set Programming System Competition. In *Logic Programming and Nonmonotonic Reasoning, 9th International Conference,* LPNMR 2007, Lecture Notes in Computer Science volume 4483, 3–17. Berlin: Springer. dx.doi.org/10.1007/978-3-540-72200-7_3

Gebser, M.; Maratea, M.; and Ricca, F. 2015. The Design of the Sixth Answer Set Programming Competition: Report. In *Logic Programming and Nonmonotonic Reasoning, 13th International Conference,* LPNMR 2015, Lecture Notes in Computer Science volume 9345. Berlin: Springer.

Gebser, M.; Schaub, T.; and Thiele, S. 2007. GrinGo: A New Grounder for Answer Set Programming. In *Logic Programming and Nonmonotonic Reasoning, 9th International Conference,* LPNMR 2007, Lecture Notes in Computer Science volume 4483, 266–271. Berlin: Springer. dx.doi.org/10.1007/978-3-540-72200-7_24

Gelfond, M., and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium,* 1070–1080. Cambridge, MA: The MIT Press.

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2006. Answer Set Programming Based on Propositional Satisfiability. *Journal of Automated Reasoning* 36(4): 345–377. dx.doi.org/10.1007/s10817-006-9033-2

Grasso, G.; Leone, N.; Manna, M.; and Ricca, F. 2011. ASP at Work: Spin-off and Applications of the DLV System. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday,* volume 6565 of Lecture Notes in Computer Science, 432–451. Berlin: Springer.

Janhunen, T. 2006. Some (In)translatability Results for Normal Logic Programs and Propositional Theories. *Journal of Applied Non-Classical Logics* 16(1-2): 35–86.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic* 7(3): 499–562. dx.doi.org/10.1145/1149114.1149117

Lierler, Y. 2014. Relating Constraint Answer Set Programming Languages and Algorithms. *Artificial Intelligence* 207(February): 1–22. dx.doi.org/10.1016/j.artint.2013.10.004

Lierler, Y., and Truszczyński, M. 2011. Transition Systems for Model Generators: A Unifying Approach. *Theory and Practice of Logic Programming* 11(4–5): 629–646. dx.doi.org/10.1017/S1471068411000214

Lin, F., and Zhao, Y. 2004. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. *Artificial Intelligence* 157(1–2): 115–137. dx.doi.org/10.1016/j.artint.2004.04.004

Maratea, M.; Ricca, F.; Faber, W.; and Leone, N. 2008. Look-Back Techniques and Heuristics in DLV: Implementation, Evaluation, and Comparison to QBF Solvers. *Journal of Algorithms* 63(1–3): 70–89. dx.doi.org/10.1016/j.jalgor.2008.02.006

Maratea, M.; Pulina, L.; and Ricca, F. 2014. A MultiEngine Approach to Answer-Set Programming. *Theory and Practice of Logic Programming* 14(6): 841–868. dx.doi.org/10.1017/S1471068413000094

Nieuwenhuis, R.; Oliveras, A.; and Tinelli, C. 2006. Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T). *Journal of the ACM* 53(6): 937–977. dx.doi.org/10.1145/1217856.1217859

Oetsch, J.; Pührer, J.; and Tompits, H. 2011. Extending Object-Oriented Languages by Declarative Specifications of Complex Objects Using Answer-Set Programming. In the Computing Research Repository CoRR abs/1112.0922. New York: Association for Computing Machinery.

Pontelli, E.; Balduccini, M.; and Bermudez, F. 2003. Nonmonotonic Reasoning on Beowulf Platforms. In *Practical Aspects of Declarative Languages, 5th International Symposium,* PADL 2003, Lecture Notes in Computer Science volume 2562, 37–57. Berlin: Springer.

Perri, S.; Ricca, F.; and Sirianni, M. 2013. Parallel Instantiation of ASP Programs: Techniques and Experiments. *Thoery and Practice of Logic Programming* 13(2): 253–278. dx.doi.org/10.1017/S1471068411000652

Ricca, F.; Gallucci, L.; Schindlauer, R.; Dell'Armi, T.; Grasso, G.; and Leone, N. 2009. OntoDLV: An ASP-Based System for Enterprise Ontologies. *Journal of Logic and Computation* 19(4): 643–670. dx.doi.org/10.1093/logcom/exn042

Ricca, F.; Grasso, G.; Alviano, M.; Manna, M.; Lio, V.; Iiritano, S.; and Leone, N. 2012. Team-Building with Answer Set Programming in the Gioia-Tauro Seaport. *Theory and Practive of Logic Programming* 12(3): 361–381. dx.doi.org/10.1017/S147106841100007X

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence* 138(1-2): 181–234. dx.doi.org/10.1016/S0004-3702(02)00187-X

Syrjänen, T. 2001. Omega-Restricted Logic Programs. In *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009,* Lecture Notes in Computer Science volume 2173, 267–279. Berlin: Springer. dx.doi.org/10.1007/3-540-45402-0_20

Ward, J., and Schlipf, J. S. 2004. Answer Set Programming with Clause Learning. In *Logic Programming and Nonmonotonic Reasoning, 7th International Conference,* LPNMR 2004, Lecture Notes in Computer Science volume 2923, 302–313. Berlin: Springer.

Wittocx, J.; Marïen, M.; and Denecker, M. 2008. The IDP System: A Model Expansion System for an Extension of Classical Logic. Paper presented at the Second Workshop on Logic and Search, Leuven, Belgium, 6–7 November.

Yuliya Lierler is an assistant professor of computer science at the University of Nebraska at Omaha. Her research interests are in the area of knowledge representation, automated reasoning, declarative problem solving, and natural language understanding. Her work spans theoretic foundations as well as practical implementations of methods for automated reasoning based on model generation, and applications of these methods. She completed her Ph.D. in computer science at the University of Texas at Austin in 2010.

Marco Maratea (www.star.dist.unige.it/~marco) is an associate professor in computer engineering at the University of Genoa, Italy. He obtained is Ph.D. at the Faculty of Engineering at University of Genoa in 2005. From 2010 to 2014 he was an assistant professor at the University of Genova. In fall 2015 he was a university lecturer at the Institute for Information Systems of the Faculty of Informatics at the Vienna University of Technology, in the DBAI group. His research interests include artificial intelligence, logic programming, and knowledge representation and reasoning, in particular answer set programming.

Francesco Ricca is an associate professor in the Department of Mathematics of the University of Calabria, Italy. He received his Laurea Degree in computer science engineering in 2002 and a Ph.D. in computer science and mathematics in 2006 from the University of Calabria, Italy. He is interested in declarative logic-based languages, consistent query answering and data integration, ontologies and, in particular, on the issues concerning their practical applications: system design and implementation, and development tools. Ricca is coauthor of more than 70 refereed articles published in international journals, collections, and conference proceedings.

Applications of Answer Set Programming

Esra Erdem, Michael Gelfond, Nicola Leone

Answer set programming (ASP) has been applied fruitfully to a wide range of areas in AI and in other fields, both in academia and in industry, thanks to the expressive representation languages of ASP and the continuous improvement of ASP solvers. We present some of these ASP applications, in particular, in knowledge representation and reasoning, robotics, bioinformatics, and computational biology as well as some industrial applications. We discuss the challenges addressed by ASP in these applications and emphasize the strengths of ASP as a useful AI paradigm.

Answer set programming (ASP) is a knowledge representation and reasoning (KR) paradigm. It has rich high-level representation languages that allow recursive definitions, aggregates, weight constraints, optimization statements, default negation, and external atoms. With such expressive languages, ASP can be used to declaratively represent knowledge (for example, mathematical models of problems, behaviour of dynamic systems, beliefs and actions of agents) and solve combinatorial search problems (for example, planning, diagnosis, phylogeny reconstruction) and knowledge-intensive problems (for example, query answering, explanation generation). The idea is to represent a problem as a "program" whose models (called "answer sets" [Gelfond and Lifschitz 1988, 1991]) correspond to the solutions of the problem. The answer sets for the given program can then be computed by special software systems called answer set solvers, such as DLV, Smodels, or clasp.

Due to the continuous improvement of ASP solvers and the expressive representation languages of ASP, ASP has been applied fruitfully to a wide range of areas in AI and in other fields. Areas of AI that include applications of ASP are planning, probabilistic reasoning, data integration and query answering, multiagent systems, natural language processing and understanding, learning, theory update/revision, preferences, diagnosis, description logics, semantic web, multicontext systems, and argumentation. Other areas that include applications of ASP are, for instance, computational biology, systems biology, bioinformatics, automatic music composition, assisted living, software engineering, bounded model checking, and robotics.

ASP has also been used in industry, for instance, for decision support systems (Nogueira et al. 2001) (used by United Space Alliance), automated product configuration (Tiihonen, Soininen, and Sulonen 2003) (used by Variantum Oy), intelligent call routing (Leone and Ricca 2015) (used by Italia Telecom) and configuration and reconfiguration of railway safety systems (Aschinger et al. 2011) (used by Siemens).

In this article, we will describe only some of these ASP applications, in particular, in knowledge representation and reasoning, robotics, bioinformatics, as well as some industrial applications. For a wide variety of ASP applications and relevant references, we refer the reader to the ASP applications table.[1]

ASP and Knowledge Representation

One of the main goals of AI is to better understand how to build software components of agents capable of reasoning and acting in a changing environment. Most AI researchers agree that to exhibit such behavior the agent should have a mathematical model of its environment and its own capabilities and goals. A logic-based approach to AI (McCarthy 1990) suggests that this model should contain a knowledge base (KB) — a collection of statements in some declarative language with precisely defined syntax and semantics. As a rule, such a KB should include commonsense knowledge — information an ordinary person is expected to know — as well as some specialized knowledge pertinent to a particular set of activities the agent is built to perform. Early proponents of logic-based AI believed that such a KB could be built in classical first-order logic (FOL) which, at the time, was commonly used for formalization of mathematical reasoning. It was quickly discovered, however, that this logic may not be a fully adequate tool for representing nonmathematical (especially commonsense) knowledge. The main problem was difficulty with using FOL for defeasible (or nonmonotonic) reasoning. In precise terms, a consequence relation $A \models F$ between statements of a declarative language is called monotonic if, for every A, B, and F, if $A \models F$ then $(A \land B) \models F$. This property guarantees that, once proven, a statement stays proven. If this condition is not satisfied, that is, if addition of new information can force a reasoner to withdraw its previous conclusion, the consequence relation is called nonmonotonic. Though not encountered in mathematics, nonmonotonicity seems to be a prevailing feature of commonsense reasoning. It is especially relevant to reasoning with so called defaults — statements of the form "Normally (typically, as a rule) elements of class C have property P." We all learn rather early in life that parents normally love their children, citizens are normally required to pay taxes, and so forth. We also learn, however, that these rules are not absolute and allow various types of exceptions. It is natural to assume that these and other defaults should be included in a reasoner's KB. Learning correct ways to reason with defaults and their exceptions is necessary for building an agent capable of using such a KB. One of the best available solutions to this problem uses the knowledge representation language CR-Prolog (Balduccini and Gelfond 2003) — a simple extension of the original ASP language of logic programs with two types of negation and epistemic disjunction.

A program Π of CR-Prolog consists of a first-order signature, a collection Π^r of standard ASP rules of the form

$$l_0 \mid \ldots \mid l_k \leftarrow l_{k+1}, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$$

and a collection Π^{cr} of consistency-restoring rules (or simply cr-rules)[2] of the form:

$$l_0 \xleftarrow{+} l_{k+1}, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n.$$

Here each l_i is a literal, that is, an atom $p(\bar{t})$ or its negation $\neg p(\bar{t})$. The last statement says that $\bar{t}$ does not have property p. In contrast, default negation *not* has an epistemic character — *not l* is often read as "it is not believed that *l* is true." Similarly, the connective | (also denoted by *or*) is often called epistemic disjunction with $l_1 \mid l_2$ being read as "l_1 is believed to be true or l_2 is believed to be true." Intuitively, a regular ASP rule *Head* ← *Body* says that if the body of the rule is believed then the reasoner *must* believe its head. A cr-rule says that if the body of the rule is believed, then the reasoner *may* possibly believe its head; however, this possibility may be used only if Π^r is inconsistent.

Informally, program Π can be viewed as a specification for answer sets — sets of beliefs that could be held by a rational reasoner associated with Π. Answer sets are represented by collections of ground literals. In forming such sets the reasoner must satisfy the rules of Π together with a so-called rationality principle, which says that the reasoner associated with the program shall believe nothing that he is not forced to believe by the program's rules. In the absence of cr-rules this idea is captured by the standard answer set semantics.

The definition of an answer set for an arbitrary CR-Prolog program is as follows. For a collection R of cr-rules, by $\alpha(R)$ we denote the collection of regular rules obtained by replacing labeled arrows in cr-rules of R by ←. A minimal (with respect to the preference relation of the program) collection R of cr-rules of Π such that $\Pi^r \cup \alpha(R)$ is consistent (that is has an answer set) is called an abductive support of Π. A set A is called an answer set of Π if it is an answer set of the regular ASP program $\Pi^r \cup \alpha(R)$ for some abductive support R of Π.

In the following, we assume that the rules that are in sans-serif font are in the ASP Core language ASP-Core-2,[3] whereas the rules in math font (for example, cr-rules) are in languages (for example, CR-Prolog) that extend the ASP Core language in different ways. The schematic variables (respectively, object constants) in rules are denoted by strings whose first letters are in uppercase (resp. in lowercase).

Example 1 [Representing Defaults]
A default "parents normally love their children" can be represented by the following rule:

```
(1) loves(P,C) :-
        parent(P,C), not -loves(P,C).
```

Consider a program P_1 consisting of this rule and a fact:

```
(2) parent(mary,john).
```

Since the answer set semantics of CR-Prolog incorporates the rationality principle and no rule of the program forces the reasoner to believe that Mary does not love John, the first rule allows it to conclude that she does. Additional information,

```
(3) -loves(mary,john)
```

will not lead to a contradiction. The new statement will render the first rule inapplicable and allow the reasoner to withdraw its earlier conclusion. Statement (3) is an example of so-called direct exceptions to defaults, that is, exceptions that directly contradict the default conclusion. The situation is not always that neat. Let us now consider a program P_2 consisting of statements (1) and (2) above, together with a new rule:

```
(4) cares(P,C) :- loves(P,C) .
```

It is easy to see that P_2 entails *cares(mary, john).* If, however, we were to learn that Mary does not care for John and expanded P_2 by

```
(5) -cares(mary,john)
```

the new program, P_3, would become inconsistent, that is, will not have answer sets. In our everyday reasoning we do not seem to have difficulties in dealing with such indirect exceptions to defaults. We would avoid the contradiction by simply concluding that Mary and John constitute an (indirect) exception to default (1). So the fact (5) is the only conclusion that can be derived from the program. To model this type of reasoning, we should make it possible for our program to recognize that the relationship between parents and their children may be not that of love, but still be able to use our default whenever possible. This is done using a cr-rule.

Indirect exceptions to default (1) can be represented as follows:

$$\neg loves(P,C) \xleftarrow{+} parent(P,C).$$

The new program P_4 consisting of regular rules (1), (2), and (4) and cr-rule (6) entails that Mary cares about John. A consistent answer set of the program can be obtained from its regular rules only and cr-rule (6) is not used. If, however, we expand P_4 by statement (5), regular rules of the program are not sufficient to avoid the contradiction. Consistency restoring rule (6) will be activated and the reasoner will conclude that Mary does not love John.

The previous example is rather general and allows for representation of different types of exceptions to defaults. More information on this can be found in Gelfond and Kahl (2014).

Reasoning about Effects of Actions

Gaining better understanding of basic principles and mathematical models of default reasoning helped researchers to move forward in solving a number of other longstanding problems of AI and KR. In this section we briefly describe an ASP-based solution of one such problem — finding logical means for representing and reasoning about direct and indirect effects of actions.

To act in a changing (dynamic) domain, a rational agent should have a mathematical model of this domain allowing it to predict such effects. Here we limit ourselves to discrete dynamic domains represented by transition diagrams whose nodes correspond to possible physical states of the domain and whose arcs are labeled by actions.

A transition (σ_1, a, σ_2) indicates that the execution of action a in state σ_1 may cause the domain to move to state σ_2. Due to the size of the diagram, the problem of finding a concise specification for it is not trivial and has been a subject of research for a comparatively long time. Its solution requires a good understanding of the nature of causal effects of actions in the presence of complex interrelations between fluents — propositions whose truth value may depend on the state of the domain.

An additional level of complexity is added by the need to specify what is not changed by actions in a concise, clear, and elaboration tolerant way. A seminal paper (Hayes and McCarthy 1969) in which the problem of finding such a specification (called the frame problem) was discussed also suggested a direction in which its possible solution could be found. The proposal was to reduce the solution of the frame problem to the problem of finding a concise, accurate, and elaboration tolerant representation of the inertia axiom — a default that says that things normally stay as they are. The search for such a representation substantially influenced AI research during the next 30 years. An interesting account of the history of this research together with some possible solutions can be found in (Shanahan 1997). We have already discussed the ways of using ASP for representing defaults and their exceptions so it shall not come as a surprise that ASP provides a good solution to the frame problem. It also turned out that rules of ASP languages can nicely capture causal relations between fluents, which led to the development of a powerful methodology for representing and reasoning about actions and their effects.

We illustrate this methodology by way of example — representation of a simple hydraulic system of a space shuttle. The example is taken from an actual decision support software system (Nogueira et al. 2001) developed to help shuttle controllers to deal with critical situations caused by multiple failures.

Example 2 [Effects of Actions]
Consider a hydraulic system viewed as a graph whose nodes are labeled by tanks containing propellant, jets, junctions of pipes, and so on. Arcs of the graph are labeled by valves that can be opened or closed by a collection of switches. The system is used to deliver propellant from tanks to a proper combination of jets.

To axiomatize the knowledge pertinent to this example, we describe the graph by a collection of statements of the form *connected*(N_1, V, N_2) — valve V labels the arc from N_1 to N_2, and *controls*(S, V) — switch S controls valve V. Fluents *pos*(S, *open*) and *pos*(S, *closed*) define positions of switch S. Fluents *pos*(V, *open*) and *pos*(V, *closed*), defining the position of a valve, and fluent *pressurized*(N), which holds when node N is reached by propellant from some tank, will be defined in terms of positions of switches of the system.

The following axiom

```
(1)  -h(pos(X,P1),I) :-
          h(pos(X,P2),I),  P1  !=  P2
```

guarantees that positions of switches and valves are mutually exclusive, that is, cannot both be true at the same time step. Here relation $h(F, I)$, where h stands for *holds*, is true if a fluent F holds (is true) at timestep I of the system's trajectory.

Now we concentrate on the representation of action *flip*(S), which flips the switch S from position *open* to position *closed* and vice versa. Note that this action has comparatively complex effects including the propagation of the delivery of propellent from tanks to other nodes of the system. The effects will be divided into direct and indirect.

The direct effect of flipping a switch S from closed to open will be given by the following axiom

```
(2)  h(pos(S,open),I+1) :-
          occurs(flip(S),I), h(pos(S,closed),I)
```

where *occurs*(A, I) is true if action A occurs (happened, is executed) at I. The rule states that if action *flip*(S) occurred at a time-step I, in which the fluent *pos*(S, *closed*) was true, then at the next step, $I + 1$, the fluent *pos*(S, *open*) would become true. A similar axiom is needed for flipping a switch from the open to closed position.

To represent indirect effects we simply need to state the relations between fluents of the domain. The next rule describes a relationship between fluents representing positions of switches and valves.

```
(3)  h(pos(V,P),I) :-
          controls(S,V), h(pos(S,P),I).
```

The rule states that if a switch is placed in a particular position, then so is the valve controlled by this switch.

The next rule describes the relationship between the values of fluent *pressurized*(N) for neighboring nodes.

```
(4)  h(pressurized(N2),I) :-
          connected(N1,V,N2),
          h(pressurized(N1),I),
          h(pos(V,open),I).
```

The rule says that if nodes N_1 and N_2 are connected by open valve V and node N_1 is pressurized then so is node N_2. We also assume that tanks are always pressurized and encode this as follows:

```
(5) h(pressurized(N),I) :- tank(N), step(I).
```

To complete the definition of this fluent we need to state that no other nodes except those defined by rules 4 and 5 are pressurized. This is done by the rule

```
(6)  -h(pressurized(N),I) :- node(N), step(I),
         not h(pressurized(N),I).
```

Suppose now that the system contains nodes n_1, n_2, and n_3 where n_1 is a tank; n_1 and n_2 are connected by valve v_1; n_2 and n_3 are connected by valve v_2; v_1 and v_2 are controlled by switches s_1 and s_2, respectively. Assume also that initially, the switches are closed. One can see that at the initial step 0, node n_1 is pressurized (axiom 5), and nodes n_2 and n_3 are not (axiom 6). To compute the effects of flipping switch s_1 let us expand the program by statement

```
occurs(flip(s1),0).
```

The direct effect of this action, determined by axiom 2, is $h(pos(s_1, open), 1)$. There are also indirect effects that follow from axioms 3, 4, and 1: $h(pos(v_1, open), 1)$, $h(pressurized(n_2), 1)$, and $\neg h(pos(v_1, closed), 1)$. To complete our formalization we need to add our solution to the frame problem, which will allow us to conclude that flipping switch s_1 does not change the status of switch s_2 and valve v_2. As discussed earlier, this can be done by simply axiomatizing the default stating that normally the value of fluent *pos*(S, *Val*) remains unchanged:

```
(7a)    h(pos(S,Val),I+1) :-
            switch(S), h(pos(S,Val),I),
            not -h(pos(S,Val),I+1).
(7b)    -h(pos(S,Val),I+1) :-
            switch(S),   -h(pos(S,Val),I),
            not h(pos(S,Val),I+1).
```

This is, of course, a typical ASP representation of a default that provides the solution to the frame problem. It guarantees that at step 1 switch s_2 is still closed. Since positions of v_2 and the value of *pressurized*(n_3) are fully determined by positions of the switches, nothing else is necessary — v_2 will remain closed and n_3 depressurized.

The ability of ASP languages to represent defaults and to express indirect effects of actions by a unidirectional implication made it a good tool for representing knowledge about dynamic domains. Nowadays, however, such knowledge is more frequently represented in so-called action languages (Gelfond and Lifschitz 1998), which are more specialized, higher-level languages designed for specifying state-action-state transition diagrams. Consider, for example, one such language, called AL (Gelfond and Kahl 2014). Axiom 2 from example 2 may be written in AL as

flip(*S*) **causes** *pos*(*S, open*) **if** *pos*(*S, closed*)

which is a special case of a dynamic causal law of AL — a statement of the form

A **causes** *F* **if** *P*.

The law says that execution of action *A* in a state that satisfies property *P* causes fluent *F* to become true in a resulting state. Axiom 4 from example 2 may be written as

$$\begin{aligned} pressurized(N_2) \quad \textbf{if} \quad & pressurized(N_1), \\ & connected(N_1, V, N_2), \\ & pos(V, open) \end{aligned}$$

which is a special case of an action language *AL* statement

F **if** *P*.

The statement guarantees that every state of the system satisfying property *P* also satisfies *F*. The inertia axioms 7a and 7b can be replaced by the simpler statement

fluent(*inertial, pos*(*S, Val*))

which indicates that the fluent *pos*(*S, Val*) is subject to the inertia axiom. Overall, use of action languages leads to substantially simpler representations and allows the system designer to avoid some ASP-related details. ASP, however, continues to play an important role in reasoning about actions.

First, answer set semantics of logic programs is often used to define semantics of action languages. Natural translations from action languages to logic programs allow us to use the notion of answer set to precisely define the effects of executing an action *A* in a state σ. Rules 1–6 of example 2 can be viewed as part of such translation from the description of our domain in an action language. The translation will also contain a more general version of axioms 7a and 7b:

```
h(F,I+1) :-
    fluent(inertial,F), h(F,I), not -h(F,I+1)
-h(F,I+1) :-
    fluent(inertial,F), -h(F,I), not h(F,I+1)
```

which provide a rather general solution of the frame problem.

Second, translation from action languages to logic programs enables us to reduce classical reasoning tasks such as prediction, planning, and finding explanations of unexpected events to computing answer sets of logic programs. An interested reader may look into Gelfond and Kahl (2014) for further details.

There are other interesting applications of ASP to classical KR problems. These include its early use for providing a declarative semantics to the negation as failure construct of the Prolog programming language (Gelfond and Lifschitz 1988) as well as comparatively recent work on combining subtle forms of logical and probabilistic reasoning (Baral, Gelfond, and Rushton 2009).

Applications of ASP to Robotics

ASP has been applied in various robotic applications, such as assembly planning, mobile manipulation, geometric rearrangement, multirobot path finding, multirobot coordination, multirobot planning, plan execution and monitoring, and human-robot interaction, to provide methods for high-level reasoning (like planning, hypothetical reasoning, diagnostic reasoning) and for declarative problem solving (like team coordination, gridization of continuous space).

For instance, Erdem, Aker, and Patoglu (2012) use ASP for planning of actions of multiple robots to collaboratively tidy up a house within a given time (figure 1). They illustrate applications of their ASP-based planning, execution and monitoring approach with dynamic simulations with PR2 robots.[4]

In another study Erdem et al. (2013) use ASP to find an optimal global plan for multiple teams of heterogeneous robots in a cognitive factory to manufacture a given number of orders within a given time (figure 2). They also use ASP for diagnosing plan failures during monitoring of plan execution (Erdem, Patoglu, and Saribatur 2015). They illustrate applications of their ASP-based planning with dynamic simulations and with an augmented reality physical implementation that utilizes Kuka youBots and Lego NXT robots controlled over Robot Operating System (ROS). They show applications of their execution-monitoring algorithm, in particular, the use of diagnostic reasoning for replanning and repairs, with dynamic simulations using Kuka youBots and a Nao humanoid robot.[5]

Havur et al. (2014) use ASP for geometric rearrangement of multiple movable objects on a cluttered surface, where objects can change locations more than once by pick and/or push actions (figure 3). They use ASP for gridization of the continuous plane for a discrete placement of the initial configurations and the tentative final configurations of objects on the cluttered surface, and for planning of robots' actions. The authors illustrate applications of their method with the CoCoA service robot, which features a holonomic mobile base and two 7 degrees of freedom (DoF) arms with grippers.[6]

Zhang, Sridharan, and Wyatt (2015) use ASP to describe objects and relations between them, and utilize this knowledge to improve localization of tar-

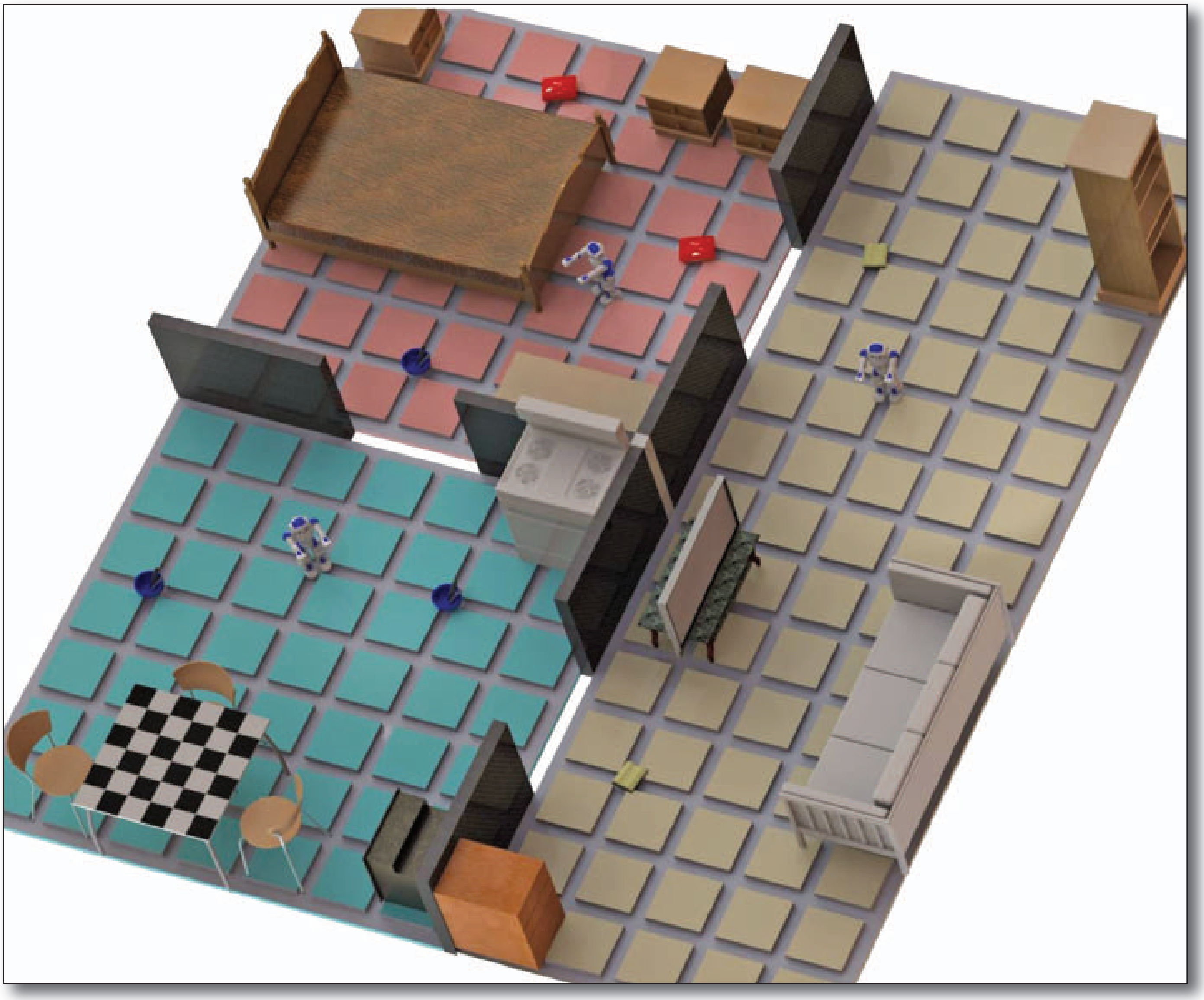


Figure 1. Multiple Robots Tidying Up a House.

get objects in an indoor domain using (primarily) visual data. Such a use of ASP has been illustrated by a physical implementation with a wheeled robot navigating in an office building.[7]

In these robotic applications, there are some important challenges from the point of view of robotic planning and diagnosis. The following discusses how ASP can handle them.

Hybrid Reasoning

One of the key challenges addressed in these robotic applications is hybrid reasoning, which can be understood as integrating high-level reasoning tasks, such as planning, hypothetical reasoning, and diagnosis, with low-level external computations. These external computations include, for instance, feasibility checks of robotic actions using probabilistic motion-planning methods, as well as automatic extraction of relevant commonsense knowledge from the existing knowledge bases available on the web. Such a variety of hybrid reasoning is possible in ASP, thanks to "external atoms" (Eiter et al. 2006). These atoms provide a general interface between high-level reasoning and low-level external computations, in the spirit of semantic attachments in theorem proving. More precisely, an external atom is an expression of the form

$$\&g\ [Y_1, \ldots, Y_n](X_1, \ldots, X_m)$$

where $Y_1, \ldots, Y_n$ and $X_1, \ldots, X_m$ are two lists of terms (called input and output lists, respectively), and g is an external predicate name. Intuitively, an external atom provides a way for deciding the values of an

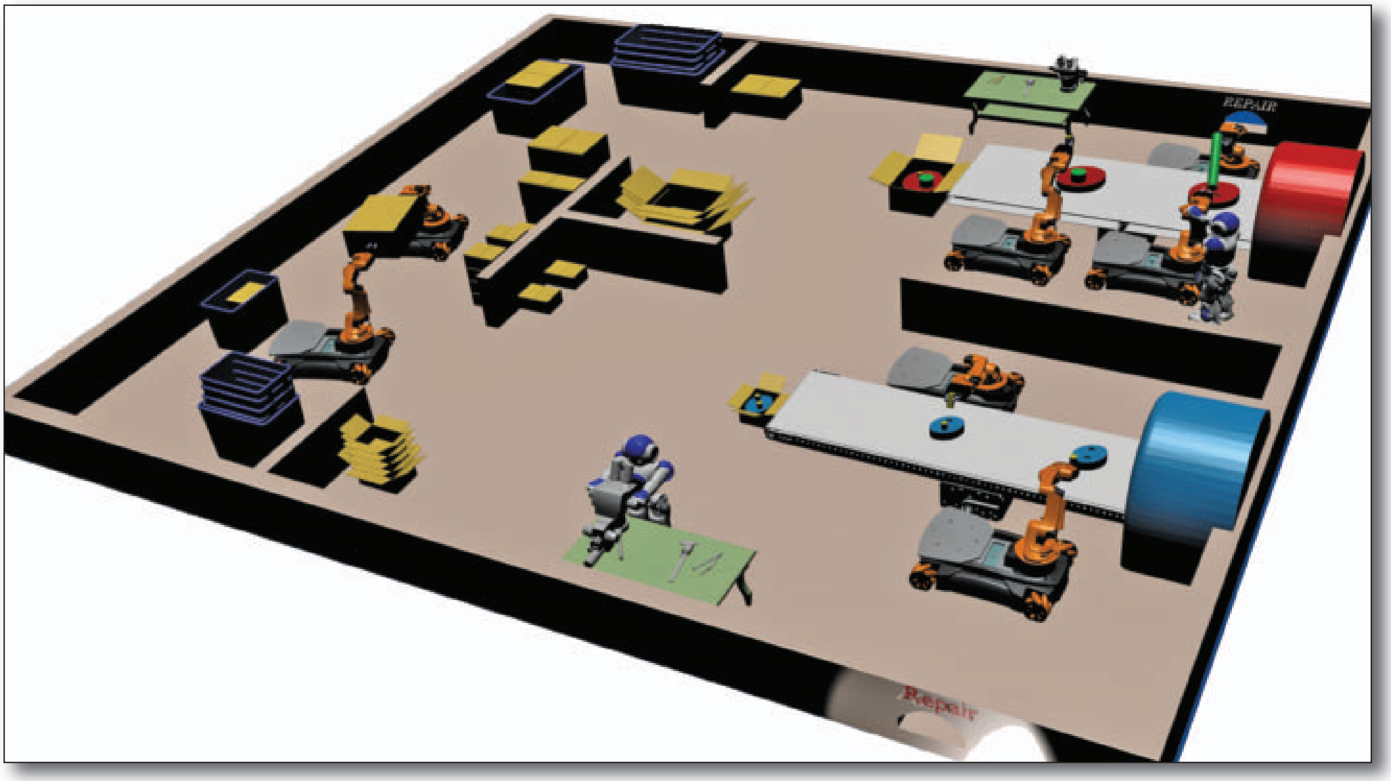


Figure 2. Multiple Robots Collaboratively Working in a Cognitive Factory.

output tuple with respect to the values of an input tuple. External atoms allow us to embed results of external computations into ASP programs. Therefore, external atoms are usually implemented in a programming language of the user's choice.

Integrating High-Level Reasoning with Low-Level Feasibility Checks

Consider, for instance, multiple robots rearranging objects on a cluttered table (Havur et al. 2014). The objects can only move when picked up and placed, or pushed by robots, and the order of pick-and-place and push operations for rearranging objects may matter to obtain a feasible kinematic solution. Therefore, motion planning (for example, finding a continuous trajectory from one configuration of the robot to another configuration) and other low-level feasibility checks alone are not sufficient to solve them. However, manipulation of objects also requires feasibility checks, such as whether the robot will be able to move an object from one location to another location without colliding with the other objects, or whether the robot will be able to reach the object on the table and grasp it without any collisions. Therefore, task planning only (for example, finding a sequence of robotic actions from an initial state to a goal state) is not sufficient to solve the problem either. These examples illustrate the necessity for a hybrid approach to planning that integrates task planning with feasibility checks.

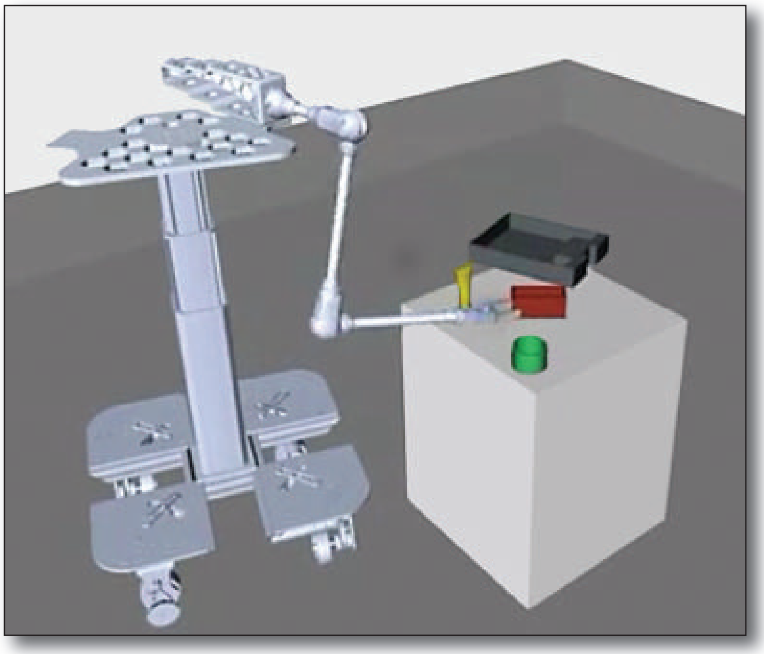


Figure 3. A Mobile Service Robot Rearranging Objects on a Cluttered Tabletop.

One of the preconditions of the action *pickPlace*(*R, O, C*) of a robot *R* picking and placing an object *O* onto location *C* is that the object *O* is graspable by the end effector of the robot. This precondition can be formalized in ASP as follows:

← *occurs*(*pickPlace*(*R, O, C*), *I*),
not &reachableGraspable[*O, R*]().

Here *&reachableGraspable*[*O, R*]() is an external atom; it returns true if and only if the end effector of the manipulator *R* can successfully reach and grasp the given object *O* according to kinematics and force-closure calculations of OPENRAVE. Note that these calculations are done in a continuous configuration space using real numbers, and thus are not possible in ASP.

One of the preconditions of the action *push*(*R, O, C*) of a robot *R* pushing an object *O* to location *C* is that the volume swept by the object *O* from its current configuration toward another configuration in *C* does not collide with other objects. This precondition can be described as follows:

← *occurs*(*push*(*R, O, C*), *I*),
not & pushPossible[*location, O, I*]().

Here *&pushPossible* is an external predicate as well: it takes as input all locations of objects at time step *I,* and checks whether the swept volume of the object *O* collides with other objects using Open Dynamics Engine (ODE).

Embedding Commonsense Knowledge in High-Level Reasoning

Consider, for instance, the housekeeping domain with multiple robots (Erdem, Aker, and Patoglu 2012). The commonsense knowledge about expected locations *Loc* of objects *Ep* (for example, dish in kitchen, bed in bedroom) can be extracted from the existing commonsense knowledge base ConceptNet (Liu and Singh 2004) by means of queries through its Python API, and can be defined by external atoms of the form *&in_place*[*Ep, Loc*](). Then, one can represent the expected locations of objects *Ep* in the house by a new fluent of the form *at_desired_location*(*Ep*) as follows:

h(*at_desired_location*(*Ep*), *I*) ←
h(*at*(*Ep, Loc*), *I*), *&in_place*[*Ep, Loc*]().

This rule formalizes that the object *Ep* is at its desired location if it is at some "appropriate" position *Loc* in the right room.

Another line of research that represents commonsense knowledge for service robotics applications is by Chen et al. (2010). In these applications, commonsense knowledge such as "a long-shape object *B* whose center-of-mass is on the table, is initially in balance if there is a can *A* on one end E_1 of it and another can *B* on the other end E_2 of it" is formulated in ASP:

```
h(balance(B,A,C),0) :-
    h(on(A,E_1),0), h(on(C,E_2),0),
    endof(E_1,B), endof(E_2,B).
```

Optimizations in Planning and Diagnosis

There are various sorts of desired optimizations in robotic applications. For instance, in planning, an optimal plan can be understood as a plan with minimum makespan or a plan with minimum total cost of actions. In diagnostic reasoning, an optimal diagnosis can be understood as a hypothesis with a smallest number of broken parts of robots. Such optimizations are possible in ASP, thanks to "optimization statements."

For instance, consider the cognitive factories domain with multiple teams of heterogeneous robots (Erdem et al. 2013; Erdem, Patoglu, and Saribatur 2015). The following expression

```
#minimize {C,R,I : h(cost(R,C),I),
                   robot(R), step(I)}
```

is used to minimize the sum of all costs *C* of robotic actions performed in a local plan, where costs of actions performed by robot *R* at every time step are defined by fluents of the form *cost*(*R, C*).

The following statement minimizes the total number of the broken parts of robots while finding a diagnosis for a discrepancy:

```
#minimize {1,P,R: broken(R,P), comp(R,P)}
```

where atoms of the form *comp*(*R, P*) describe robots and their parts, and atoms of the form *broken*(*R, P*) describe that part *P* of the robot *R* is broken.

Complex Constraints in Replanning

During plan execution, discrepancies between the observed state and the expected one may be detected that are relevant for the rest of the plan. These discrepancies may be due to an unexpected obstacle in the environment, change of locations of objects, broken robots, or failures of some actions. Once the cause of a discrepancy is detected, a new plan from the current state can be computed. While replanning, some guidance from earlier experiences and causes of discrepancies might be helpful to compute a better plan that does not fail due to the same reasons. ASP allows us to include such a guidance, by including constraints into the representation of the planning problem description. These constraints might express not only the new knowledge about the environment, robots, and/or goals, but also what sort of actions should not be executed under what conditions and when. For instance, in the housekeeping domain (Erdem, Aker, and Patoglu 2012), if some robot's plan fails because the robot cannot pick up an object that turns out to be quite heavy, the robot might want to delay asking for help as much as possible so that the other robots are not distracted. Such a complex constraint (for example, heavy objects can be picked with help only during the last three steps of the plan) can be represented in the planning problem description using ASP.

Commonsense Knowledge and Exceptions

Consider, for instance, the housekeeping domain with multiple robots (Erdem, Aker, and Patoglu 2012). Normally, the movable objects in an untidy house are not at their desired locations. Such commonsense knowledge can be described by means of defaults, as in the following rules:

```
-h(at_desired_location(Ep),I) :-
   endpoint(Ep), step(I),
   not  h(at_desired_location(Ep),I).
```

In a similar way, the tidiness of a house is defined by means of defaults:

```
-h(tidy,I) :- -h(at_desired_location(Ep),I)
h(tidy,I) :- not -h(tidy,I), step(I).
```

The second rule above expresses that the house is normally tidy. The first rule above describes the exceptions: when an object is not at its expected location, the house is untidy.

Let us now consider diagnostic reasoning in cognitive factories with multiple teams of heterogeneous robots (Erdem, Patoglu, and Saribatur 2015). Normally, the robots and their parts run smoothly. However, there may be exceptions: some parts *P* of robots *R* that are not broken currently (at time-step *I*) may get broken at the next state (at any time-step *I*). This commonsense knowledge can be represented by means of defaults as well:

```
-h(broken(R,P),I) :- comp(R,P), step(I),
   not h(broken(R,P),I)
h(broken(R,P),I+1) :- comp(R,P), step(I),
    -h(broken(R,P),I),
    not  -h(broken(R,P),I+1).
```

Other examples of the use of ASP to represent expected locations of objects, by means of defaults, and to find diagnoses, by means of cr-rules, can be found in Zhang et al. (2014).

Applications of ASP to Computational Biology and Bioinformatics

ASP has been applied in various computational biology and bioinformatics applications, providing a declarative problem-solving framework for combinatorial search problems (for example, haplotype inference, consistency checking in biological networks, phylogeny reconstruction) and providing a knowledge representation and reasoning framework for knowledge-intensive reasoning tasks (for example, integrating, query answering and explanation generation over biomedical ontologies).

Tran and Baral (2009) introduce a method to model a biological signaling network as an action description in ASP to allow prediction, planning, and explanation generation about the network. They illustrate an application of their method to generate hypotheses about the various possible influences of a tumor suppressor gene on the p53 pathway. Gebser et al. (2011) introduce a method to model biochemical reactions and genetic regulations as influence graphs in ASP, to detect and explain inconsistencies between experimental profiles and influence graphs. With this method, they compare the yeast regulatory network with the genetic profile data of SNF2 knockouts, and find the data to be inconsistent with the network.

Brooks et al. (2007) use ASP to solve the problem of reconstructing phylogenies (that is, evolutionary trees) for specified taxa, with a character-based cladistics approach. They apply their method to infer phylogenies for *Alcataenia* species as well as Indo-European languages (figure 4) and Chinese dialects; these phylogenies are found plausible by the experts. Based on these phylogenies, phylogenetic networks are reconstructed as well (Erdem, Lifschitz, and Ringe 2006).

In the NMSU-PhyloWS project (Le et al. 2012), ASP is used to query the repository CDAOStore of phylogenies. These queries are used, for instance, to find the trees that satisfy a given property (for example, whose size is smaller than a specified constant, with a specified ratio of internal nodes to external nodes), to find the similarity of two trees with respect to a distance measure (for example, the Robinson-Foulds distance), to compute clades with specific properties (for example, the minimum spanning clade for taxa in a specified tree).

Erdem et al. (2011) and Erdem and Oztok (2015) use ASP to answer complex queries over biomedical ontologies and databases that consider the relevant parts of these knowledge resources, and to generate the shortest explanations to justify these answers. They apply their methods to find answers and explanations to some complex queries related to drug discovery (for example, "What are the genes that are targeted by the drug Epinephrine and that interact with the gene DLG4?", "What are the genes related to the gene ADRB1 through a gene-gene relation chain of length at most three?" and "What are the most similar three genes that are targeted by the drug Epinephrine?") over the biomedical knowledge resources PharmGKB, DrugBank, BioGRID, CTD, SIDER, and Disease Ontology.

Dovier, Formisano, and Pontelli (2009) use ASP to study a variation of the protein structure prediction problem: the two-dimensional HP-protein structure prediction problem. The goal is to find a folding in the two-dimensional square lattice space that maximizes the number of hydropic-hydropic contacts between given amino acids.

Erdem and Ture (2008) use ASP to solve the problem of haplotype inference by pure parsimony (HIPP) and its variations. Identifying maternal and paternal inheritance is essential for finding the set of genes responsible for a particular disease. However, due to technological limitations, we have access to genotype data (genetic makeup of an individual), and determining haplotypes (genetic makeup of the par-

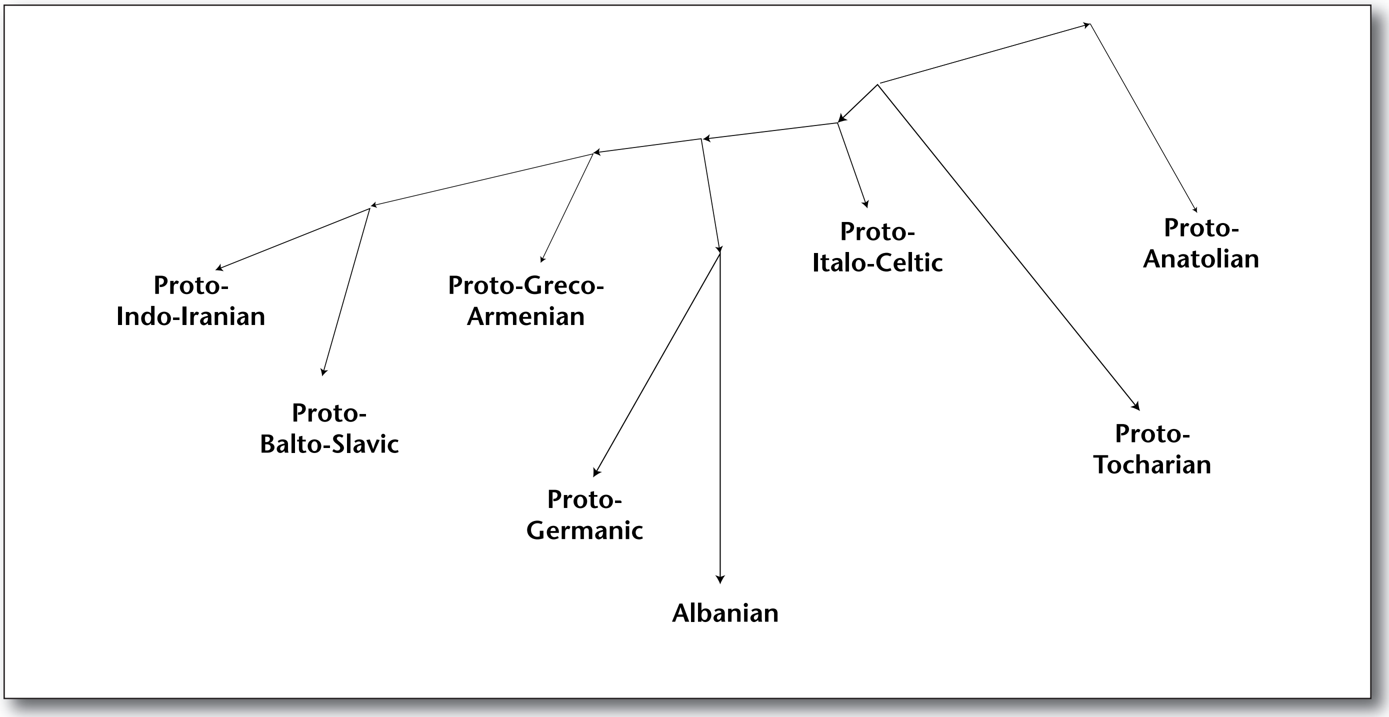


Figure 4. The Most Plausible Phylogeny Reconstructed for Indo-European Languages.

ents) experimentally is a costly and time consuming procedure. With these biological motivations, HIPP asks for the minimal number of haplotypes that form a given set of genotypes.

In these bioinformatics applications, one can identify some important challenges addressed by ASP; these challenges illustrate also the strengths of ASP.

Declarative Problem Solving

The declarative representation formalism of ASP allows us to easily include domain-specific information and constraints in the program, and thus to prevent the construction of implausible solutions. For instance, including some temporal and geographical constraints about Indo-European languages provided by historical linguists (for example, "Albanian cannot be a sister of IndoIranian or BaltoSlavic") helps in computing plausible phylogenies more efficiently.

Well-studied properties of programs in ASP allow us to easily prove the correctness of the formulation of the problem in ASP, as shown in Erdem, Lifschitz, and Ringe (2006).

With a declarative representation of the problem in ASP, one can perform various reasoning tasks, such as ontological query answering and explanation generation (Le et al. 2012; Erdem et al. 2011; Erdem and Oztok 2015), planning and diagnosis (Tran and Baral 2009), consistency checking and explanation generation (Gebser et al. 2011), and repair and prediction (Gebser et al. 2010).

Integration of Heterogeneous Knowledge

To answer complex queries over a variety of biomedical ontologies (Erdem et al. 2011), ASP allows us to extract relevant parts of them (thanks to external atoms) and integrate them by rules. For instance, the drug names can be extracted from a drug ontology, by first extracting the relevant triples from the ontology:

$$tripleD(X, Y, Z) \leftarrow \\ \quad \& \, rdf\,[\text{"URIforDrugOntology"}](X, Y, Z)$$

and then extracting drug names from the triples:

```
drugName(A) :-
    tripleD(_,"drugproperties:name",A).
```

Then, to answer queries like "What are the drugs that treat the disease depression and that do not target the gene ACYP1?" the extracted relevant knowledge can be integrated by rules as follows:

```
whatDrugs(DRG) :- cond1(DRG), cond2(DRG)
cond1(DRG) :- drugDisease(DRG,"Depression")
cond2(DRG) :- drugName(DRG),
    not  drug_gene(DRG,"ACYP1").
```

Expressivity of Representation

ASP features rich, expressive formalisms (for example, the support of recursive definitions and negation as failure), and efficient solvers that support special syntactic constructs (for example, aggregates and optimization statements).

For instance, in Gebser et al. (2011), candidates for minimal inconsistent components in an influence graph, where two distinct vertices are not reachable from each other by a cycle, can be eliminated by the following constraint:

```
:- active(U), active(V),
  not cycle(U,V), U<V.
```

where the definition of a *cycle* requires recursion:

```
reach(U,V) :- edge(U,V)
reach(U,V) :- edge(U,W),
    reach(W,V), vertex(V)
cycle(U,V) :- reach(U,V), reach(V,U), U!=V.
```

To answer queries like "What are the genes related to the gene ADRB1 through a gene-gene relation chain of length at most three?" the auxiliary concept of reachability of a gene from another gene by means of a chain of gene-gene interactions is required (Erdem et al. 2011); this concept can be defined in ASP recursively as follows:

```
geneReachable(X,1) :-
    geneGene(X,Y), startGene(Y)
geneReachable(X,N+1) :-
    geneGene(X,Z), geneReachable(Z,N),
    max_chain_length(L), 0 < N, N < L.
```

Aggregates allow concise and easy-to understand formulations of problems. According to Le et al. (2012), we can identify phylogenies by the parsimony tree length, which is defined by the total number of characters of its taxa, by the following rules:

```
parsimonyLength(T,L) :- tree(T),
    L = #count {Char: belongsChar(_,Cell,Char),
                      belongsTU(_,Cell,TU),
                      representsTU(T,_,TU)}.
```

Negation as failure is useful to represent defaults (as seen in the examples of robotics applications) and the concept of unknown. For instance, we can define that some drugs' toxicity is unknown as follows (Erdem et al. 2011):

```
unknownToxicityDrug(X) :- drugSynonym(R,X),
    not drugToxic(R), not -drugToxic(R).
```

Industrial ASP Applications

As pointed out in the introduction, the availability of efficient ASP solvers has recently enabled the implementation of many advanced ASP applications, not only in academia but also in industry. In this section, we briefly overview a number of real-world industrial applications of ASP. In particular, we will focus on ASP applications to e-tourism, workforce management, intelligent call routing, and e-medicine that have been implemented by using the DLV system, and applications to products and services configuration and decision support systems that have been implemented by using clasp and Smodels systems.

e-Tourism

ASP has been profitably applied in a couple of applications arising in the tourism industry. In the following, we overview an ASP-based application that has been integrated into an e-tourism portal and implements an intelligent advisor that selects the most promising offers for customers of a travel agency (Ricca et al. 2010). The goal is to devise a tool that helps the employees of a travel agency in finding the best possible travel solution in a short time. It can be seen as a mediator system that finds the best match between the offers of the tour operators and the requests of the tourists. The system improves the business of the travel agency by reducing the time needed to single out and sell the touristic offers, and increases the level of customer satisfaction by suggesting the offers that best match the user profile. By analyzing the touristic domain in cooperation with the staff of a travel agency, a knowledge base has been specified that models the key entities that describe the process of organizing and selling a complete holiday package. In this framework, ASP has been first used as the intelligent engine of a semantic-based information extractor (Manna, Scarcello, and Leone 2011), which analyzes the text files describing the touristic offers, extracts the relevant information (for example, place, date, prize), and classifies them in an ontology. But the main usage of ASP in this application has been for developing several search modules that simplify the task of selecting the holiday packages that best fit the customer needs. As an example, we report (a simplified version of) a logic program that creates a selection of holiday packages in figure 5.

```
%detect possible and suggested places
possiblePlace(Place) :- askFor(TripKind,_),
      placeOffer(Place, TripKind).
suggestPlace(Place) :- possiblePlace(Place),
      askFor(_,Period),
      suggestedPeriod(Place, Period),
      not badPeriod(Place, Period).
%select packages to suggest to the user
suggestOffer(O) :- touristicOffer(O, Place),
      suggestPlace(Place).
```

Figure 5. A Program That Creates a Selection of Holiday Packages.

Input predicate *askFor*(*TripKind,Period*) specifies the kind of trip requested by the customer and the period she or he wants to travel. Predicate *touristicOffer*(*Offer, Place*) specifies, for each touristic offer available at the travel agency, what place it refers to. Predicates *placeOffer*(*Place, TripKind*) and *badPeriod*(*Place, Period*) are derived by other modules of the knowledge base and define, respectively, the places that are appropriate for a kind of trip, and the periods that should be avoided for a place (for example, because of bad weather). The first two rules select, respectively, possible places (that is, the ones that offer the kind of holiday requested by the customer),

```
assign(E,Sh,Sk) | nAssign(E,Sh,Sk) :-
   hasSkill(E,Sk), employee(E,_),
   shift(Sh,Day,Dur), not absent(Day,E),
   not excluded(Sh,E),
   neededSkill(Sh,Sk),
   workedHours(E,Wh), Wh+Dur<36.
:- shift(Sh,_,_),
   neededEmployee(Sh,Sk,EmpNum),
   #count{E: assign(E,Sh,Sk)}!=EmpNum.
:- assign(E,Sh,Sk1), assign(E,Sh,Sk2),
   Sk1!=Sk2.
:- wstats(E1,Sk,LastTime1),
   wstats(E2,Sk,LastTime2),
   LastTime1>LastTime2, assign(E1,Sh,Sk),
   not assign(E2,Sh,Sk).
:- workedHours(E1,Wh1), workedHours(E2,Wh2),
   threshold(Tr), Wh1+Tr<Wh2,
   assign(E1,Sh,Sk), not assign(E2,Sh,Sk).
```

Figure 6. A Program for Computing Teams.

and places to be suggested (because they offer the required kind of holiday in the specified period). The last rule selects, within the available holiday packages, the ones that offer a holiday that matches the original input (possible offer). This is one of the several reasoning modules that have been devised for implementing the intelligent search and integrated in the e-tourism portal (Ricca et al. 2010).

Workforce Management

An interesting ASP application has been developed in the framework of the efficient management of employees of the Gioia Tauro seaport — the largest transhipment port of the Mediterranean sea. The problem that this application has dealt with is a form of work force management problem. It amounts to computing a suitable allocation of the available personnel of the seaport such that cargo ships mooring in the port are properly handled. To accomplish this task, several constraints have to be satisfied. An appropriate number of employees, providing several different skills, is required, depending on the size and load of the cargo ships. Moreover, the way an employee is selected and the specific role she will play in the team (each employee is able to cover several roles according to her skills) are subject to many conditions (for example, fair distribution of the working load, turnover of heavy or dangerous roles, employees' contract rules, and so on). To cope with this crucial problem ASP has been exploited for developing a team builder. First of all, the input — the employees and their skills — was modeled by the predicate *hasSkill*(*employee, skillName*). The specification of a shift for which a team needs to be allocated was modeled by predicate *shift*(*id, date, duration*), the skills necessary for a certain shift by *neededSkill*(*shift, skill*), weekly statistics that specify, for each employee, the last allocation date per skill by predicate *wstat*(*employee, skill, lastTime*), employees excluded due to a management decision by *excluded*(*shift, employee*), absent employees by predicate *absent*(*day, employee*), and total amount of working hours in the week per employee by predicate *workedHours*(*employee,weekHours*). A simplified version of the program computing teams is shown in figure 6.[4]

The first rule is disjunctive. It generates the search space by guessing the assignment of a number of available employees to the shift in the appropriate roles. Absent or excluded employees, together with employees exceeding the maximum number of weekly working hours, are automatically discarded. Then, inadmissible solutions are discarded by means of four integrity constraints: the first constraint discards assignments with a wrong number of employees for some skill; the second one avoids that an employee covers two roles in the same shift; the third implements the turnover of roles; and the fourth constraint guarantees a fair distribution of the workload. Note that only the kernel part of the employed logic program is reported here (in a simplified form), and many other constraints were developed, tuned, and tested.

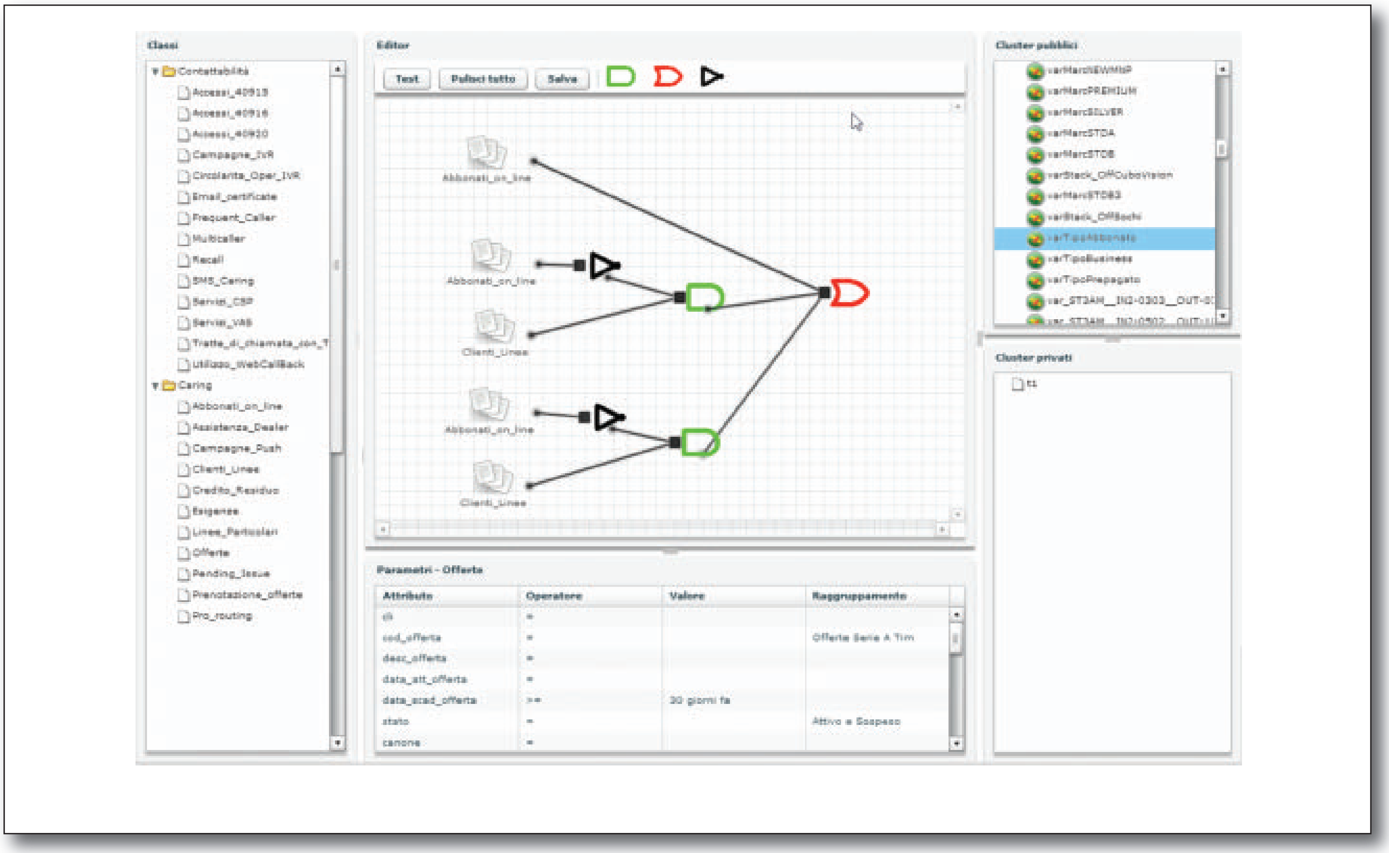
The user interface allows for modifying manually computed teams, and the system is able to verify whether the manually modified team still satisfies the constraints. In case of errors, causes are outlined and suggestions for fixing a problem are proposed. for example, if no team that satisfies all constraints can be generated, then the system suggests that the user relax some constraints. In this application, the pure declarative nature of the language allowed for refining and tuning both problem specifications and ASP programs while interacting with the stakeholders of the seaport. The system, developed by a spin-off company of the University of Calabria called Exeura s.r.l, has been used by the company ICO BLG, an automobile logistics firm in the seaport of Gioia Tauro. Further details can be found in Ricca et al. (2012).

Intelligent Call Routing

Contact centers are used by many organizations to provide remote assistance to a variety of services. Their front ends are flooded by a huge number of telephone calls every day. In this scenario the ability to route customers automatically to the most appropriate service brings a two-fold advantage: improved quality of service and reduction of costs.

The company Exeura developed a platform for customer profiling for phone-call routing based on ASP that is called zLog.[5]

The key idea is to classify customer profiles and try to anticipate their actual needs for creating a personalized experience of customer care service. Call-cen-



Parametri - Offerte

Attributo	Operatore	Valore	Raggruppamento
cli	=		
cod_offerta	=		Offerte Serie A Tim
desc_offerta	=		
data_att_offerta	=		
data_scad_offerta	>=	30 giorni fa	
stato	=		Attivo e Sospeso
canone	=		

Figure 7. A Visual Definition of Customer Categories in zLog.

ter operators can define customer categories, but it is very likely that these employees may not have the competence for defining categories with a traditional programming language. Thus, the definition of customer categories is carried out through a user-friendly visual interface (see figure 7) that allows one to specify and modify categories. Once a new category has been defined, zLog automatically generates an ASP program that provides its logical encoding and that can be executed by DLV over the database to populate the category. A category's definition criteria include customer behavioral aspects, such as recent history of contacts (for example, telephone calls to the contact center, messages sent to customer assistance) or basic customer demographics (for example, age, residence). The latter is useful, for instance, in case of natural disasters, or type of contract. When a customer calls the call center, she or he is automatically assigned to a category (based on his or her profile) and then routed to an appropriate human operator or automatic responder.

Telecom Italia employs the zLog platform in a production system that handles its call centers. Every day, more than 1 million telephone calls asking for diagnostic services reach the call centers of Telecom Italia. The needs are to optimize the operator assignment process, in order to reduce the average call response time, and to improve customer support quality. The zLog platform can detect the customer category in less than 100 milliseconds (starting from her or his telephone number) and manage more than 400 calls per second. As a result, zLog enables huge time savings for more than 1 million daily calls.

e-Medicine

Medical knowledge bases, resulting from the integration of several different databases, often present errors and anomalies that severely limit their usefulness. ASP has been successfully employed in this context. In particular, a multisource data-cleaning system, based on ASP and called DLVCleaner, has been realized, which detects and automatically corrects both syntactic and semantic anomalies in medical knowledge bases (Leone and Ricca 2015), based on ontological domain descriptions and measures for string similarities (Greco and Terracina 2013). DLVCleaner automatically generates ASP programs that are able to identify and possibly correct errors within the medical data. DLVCleaner has been employed to clean up the data stored in the tumor registries of the Calabria region, integrating information from several local health-care centers, including public hospitals, private health-care centers, family doctors, and others. The main input table consisted of

1,000,000 tuples collecting records from 155 municipalities, whereas the dictionary stored about 15,000 tuples. DLVCleaner recognized that almost 50 percent of input tuples were wrong. Moreover, 72 percent of the wrong tuples were automatically corrected by DLVCleaner, which for an additional 26 percent of the tuples suggested multiple corrections to be evaluated by the user. Only 2 percent of input tuples have been detected as wrong and not repairable.

By using ASP in this application, a simplified and flexible specification of the logic of the data-cleaning task is obtained.

Configuration and Reconfiguration of Products and Services

One of the first industrial applications of ASP (using the ASP solver Smodels) was for product configuration (Tiihonen, Soininen, and Sulonen 2003), used by Variantum Oy. The most recent configuration and reconfiguration applications have been carried out by the group of Gerhard Friedrich at Alpen-Adria Universität Klagenfurt, Austria, and deployed by Siemens.

In particular, ASP has been applied (with the ASP solver clasp) for the configuration of railway safety systems in order to compute the connection structure between sensors, indicators, and communication units. The task of this part of a railway safety system is to detect objects that entered but did not leave a section, thus blocking a track. It turned out that this configuration problem is NP-hard and is challenging for the state-of-the-art problem-solving frameworks, that is, SAT, CSP, MIP, and ASP (Aschinger et al. 2011). However, by applying ASP it was possible to solve configuration problems that could not be solved by specialized configuration tools.

Besides configuration, the reconfiguration of products and services plays an important role in practice. In many areas of configurable systems where the customer requirements change, the configured system is also subject to adaptations. ASP is applied to model the possible changes of existing systems and to compute reconfiguration solutions that optimize the adaptation actions. for example, maximize the number of reused modules and minimize the costs of additional equipment (Friedrich et al. 2011).

In addition to configuration tasks, ASP was applied to diagnose and repair systems. Friedrich et al. (2010) describe a system that computes repair plans for faulty workflow instances employing ASP. Given the workflow structure, a set of possible repair actions, and a workflow instance where an exception was triggered, a contingency plan is generated such that after the execution of this plan a correct completion of the workflow instances is achieved.

Decision Support Systems

ASP has been used by United Space Alliance to check correctness of plans and find plans for the operation of the reaction control system (RCS) of the space shuttle (Nogueira et al. 2001) (as briefly discussed in example 2). The RCS is the shuttle's system mainly for maneuvering the aircraft while it is in space. The RCS is computer controlled during takeoff and landing. While in orbit, however, astronauts have the primary control. During normal shuttle operations, there are prescripted plans that tell the astronauts what should be done to achieve certain goals. The situation changes when there are failures in the system. The number of possible sets of failures is too large to preplan for all of them. Meanwhile, RCS consists of fuel and oxidizer tanks, valves, and other plumbing to provide propellant to the maneuvering jets of the shuttle, and it consists of electronic circuitry to control the valves in the fuel lines and to prepare the jets to receive firing commands. The actions of flipping switches have many ramifications on the states of valves, and thus this application domain presents the further challenges of the ramification problem. Thanks to the expressivity of ASP in representing dynamic systems and handling the ramification problem (as explained in example 2), an intelligent system has been implemented using ASP with the ASP solver Smodels to verify and generate such preplans.

Some Challenges Addressed by ASP in Industrial Applications

To deal with industrial applications, ASP has to address various software engineering challenges. Thanks to its powerful and expressive framework, using ASP-based software development provides many advantages, such as flexibility, readability, extensibility, and ease of maintenance. Indeed, the possibility of modifying complex reasoning tasks by simply editing a text file with the ASP rules, and testing it on-site together with the customer, has been often a great advantage of ASP-based development. This aspect of ASP-based software development was a success reason especially for the workforce-management application, where the high complexity of the requirements was a main obstacle, and the availability of an executable specification language, like ASP, allowed clarifying and formalizing the requirements much more quickly together with the customer.

Realizing complex features of an application in such a way also brings about advantages of lower (implementation) costs, compared to traditional imperative languages.

Another challenge in industrial applications is computational efficiency. Fortunately, ASP solvers implement optimization techniques to handle such challenges. For instance, in the Intelligent call-routing application, an immediate response has to be given to queries over huge data sets. Thanks to the availability of the Magic Set optimization technique (Alviano et al. 2012), DLV can localize the computation to the small fragment of the database that is relevant for the specific query at hand; using this optimization technique leads to a tremendous speedup of the computation.

Conclusion

We have discussed some applications of ASP in knowledge representation and reasoning, robotics, and bioinformatics and computational biology as well as some industrial applications. In these applications, ASP addresses various challenges. For instance, representation of defaults to handle exceptions and the commonsense law of inertia to be able to reason about the effects of actions are some of the important challenges in knowledge representation and reasoning. Hybrid reasoning, reasoning about commonsense knowledge and exceptions, optimizations over plans or diagnoses are some of the important challenges addressed by ASP in robotic applications. Provability of formulation of computational problems, expressing sophisticated concepts that require recursion and/or aggregates, and integration of heterogeneous knowledge are some of the important challenges addressed by ASP in bioinformatics and computational biology. Similar challenges are also addressed in industrial applications, such as data cleaning, extraction of relevant knowledge from large databases, and software-engineering challenges. Thanks to the expressive declarative languages of ASP that support default negation, aggregates, recursion, external atoms, consistency restoring rules and optimization statements, the presence of theoretical results that help for analysis of ASP formulations, and the sophisticated methods (like Magic Sets) implemented in the ASP solvers to improve computational efficiency, these challenges can be addressed by ASP.

Acknowledgements

Thanks to Gerhard Brewka, Francesco Calimeri, Wolfgang Faber, Martin Gebser, Tomi Janhunen, Volkan Patoglu, Simona Perri, Enrico Pontelli, Francesco Ricca, Torsten Schaub, Tran Son, and Mirek Truszczyński for their comments on an earlier draft of this article. The work of Esra Erdem is partially supported by TUBITAK Grants 111E116 and 114E491 (Chist-Era COACHES). The work of Nicola Leone is partially supported by the Italian Ministry of University and Research under PON project Ba2Know (Business Analytics to Know) Service Innovation - LAB, No. PON03PE 00001 1.

Notes

1. Available at www.dropbox.com/s/pe261e4qi6bcyyh/aspAppTable.pdf?dl=0.

2. The definition of a program also includes the fourth component — a preference relation on sets of cr-rules. In what follows, we assume that a set with the smaller number of rules is preferred to that with the larger one.

3. www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.03c .pdf

4. The full version makes use of sophisticated constructs, like weak constraints and more complex aggregates (Alviano and Leone 2015).

5. www.exeura.eu/en/solution/customer-profiling.

References

Alviano, M., and Leone, N. 2015. Complexity and Compilation of GZ-Aggregates in Answer Set Programming. *Theory and Practice of Logic Programming* 15(4–5): 574–587. dx.doi.org/10.1017/S147106841500023X

Alviano, M.; Faber, W.; Greco, G.; and Leone, N. 2012. Magic Sets for Disjunctive Datalog Programs. *Artificial Intelligence* 187: 156–192. dx.doi.org/10.1016/j.artint.2012.04.008

Aschinger, M.; Drescher, C.; Friedrich, G.; Gottlob, G.; Jeavons, P.; Ryabokon, A.; and Thorstensen, E. 2011. Optimization Methods for the Partner Units Problem. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 8th International Conference, CPAIOR 2011,* Lecture Notes in Computer Science 6697, 4–19. Berlin: Springer. dx.doi.org/10.1007/978-3-642-21311-3_4

Balduccini, M., and Gelfond, M. 2003. Logic Programs with Consistency-Restoring Rules. In Logical Formalization of Commonsense Reasoning: Papers from the 2003 AAAI Spring Symposium, Technical Report SS-03-05, 9–18. Menlo Park, CA: AAAI Press.

Baral, C.; Gelfond, M.; and Rushton, J. N. 2009. Probabilistic Reasoning with Answer Sets. Theory and *Practice of Logic Programming* 9(1): 57–144. dx.doi.org/10.1017/S1471068408003645

Brooks, D. R.; Erdem, E.; Erdogan, S. T.; Minett, J. W.; and Ringe, D. 2007. Inferring Phylogenetic Trees Using Answer Set Programming. *Journal of Automated Reasoning* 39(4): 471–511. dx.doi.org/10.1007/s10817-007-9082-1

Chen, X.; Ji, J.; Jiang, J.; Jin, G.; Wang, F.; and Xie, J. 2010. Developing High-Level Cognitive Functions for Service Robots. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010),* 989–996. Richland, SC: International Foundation on Autonomous Agents and Multiagent Systems.

Dovier, A.; Formisano, A.; and Pontelli, E. 2009. An Empirical Study of Constraint Logic Programming and Answer Set Programming Solutions of Combinatorial Problems. *Journal of Experimental and Theoretical Artificial Intelligence* 21(2): 79–121. dx.doi.org/10.1080/09528130701538174

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2006. Effective Integration of Declarative Rules with External Evaluations for Semantic-Web Reasoning. In *Proceedings of the Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006,* Lecture Notes in Computer Science 4011. Berlin: Springer. dx.doi.org/10.1007/s11370-012-0119-x

Erdem, E.; Aker, E.; and Patoglu, V. 2012. Answer Set Programming for Collaborative Housekeeping Robotics: Representation, Reasoning, and Execution. *Intelligent Service Robotics* 5(4): 275–291. dx.doi.org/10.1017/S1471068413000598

Erdem, E., and Oztok, U. 2015. Generating Explanations for Biomedical Queries. *Theory and Practice of Logic Programming* 15(1):35–78. dx.doi.org/10.1017/S1471068413000598

Erdem, E., and Türe, F. 2008. Efficient Haplotype Inference with Answer Set Programming. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence,* 436–441. Menlo Park, CA: AAAI Press.

Erdem, E.; Erdem, Y.; Erdogan, H.; and Öztok, U. 2011. Finding Answers and Generating Explanations for Complex Biomedical Queries. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence.* Palo Alto, CA: AAAI Press.

Erdem, E.; Lifschitz, V.; and Ringe, D. 2006. Temporal Phylogenetic Networks and Logic Programming. *Theory and Practice of Logic Programming* 6(5): 539–558. dx.doi.org/10.1017/S1471068406002729

Erdem, E.; Patoglu, V.; and Saribatur, Z. G. 2015. Integrating Hybrid Diagnostic Reasoning in Plan Execution Monitoring for Cognitive Factories with Multiple Robots. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2015,* 2007–2013. Piscataway, NJ: Institute for Electrical and Electronics Engineers. dx.doi.org/10.1109/ICRA.2015.7139461

Erdem, E.; Patoglu, V.; Saribatur, Z. G.; Schüller, P.; and Uras, T. 2013. Finding Optimal Plans for Multiple Teams of Robots Through a Mediator: A Logic-Based Approach. *Theory and Practice of Logic Programming* 13(4-5): 831–846. dx.doi.org/10.1017/S1471068413000525

Friedrich, G.; Fugini, M.; Mussi, E.; Pernici, B.; and Tagni, G. 2010. Exception Handling for Repair in Service-Based Processes. *IEEE Transactions on Software Engineering* 36(2): 198–215. dx.doi.org/10.1109/TSE.2010.8

Friedrich, G.; Ryabokon, A.; Falkner, A. A.; Haselböck, A.; Schenner, G.; and Schreiner, H. 2011. (Re)Configuration Based on Model Generation. Paper presented at the Second Workshop on Logics for Component Configuration, Perugia, Italy, 12 September. dx.doi.org/10.4204/eptcs.65.3

Gebser, M.; Guziolowski, C.; Ivanchev, M.; Schaub, T.; Siegel, A.; Thiele, S.; and Veber, P. 2010. Repair and Prediction (Under Inconsistency) in Large Biological Networks with Answer Set Programming. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference.* Menlo Park, CA: AAAI Press.

Gebser, M.; Schaub, T.; Thiele, S.; and Veber, P. 2011. Detecting Inconsistencies in Large Biological Networks with Answer Set Programming. *Theory and Practice of Logic Programming* 11(2): 1–38. dx.doi.org/10.1017/s1471068410000554

Gelfond, M., and Kahl, Y. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. New York: Cambridge University Press. dx.doi.org/10.1017/CBO978 1139342124

Gelfond, M., and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium,* 1070–1080. Cambridge, MA: The MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9(3/4): 365–385. dx.doi.org/10.1007/BF0303 7169

Gelfond, M., and Lifschitz, V. 1998. Action Languages. *Electronic Transactions on Artificial Intelligence, ETAI* 2(3–4): 193–210.

Greco, G., and Terracina, G. 2013. Frequency-Based Similarity for Parameterized Sequences: Formal Framework, Algorithms, and Applications. *Information Sciences* 237(July): 176–195. dx.doi.org/10.1016/j.ins.2013.03.016

Havur, G.; Ozbilgin, G.; Erdem, E.; and Patoglu, V. 2014. Geometric Rearrangement of Multiple Movable Objects on Cluttered Surfaces: A Hybrid Reasoning Approach. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation, ICRA 2014,* 445–452. Piscataway, NJ: Institute for Electrical and Electronics Engineers. dx.doi.org/ 10.1109/icra.2014.6906894

Hayes, P. J., and McCarthy, J. 1969. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In *Machine Intelligence 4* ed. B. Meltzer and D. Michie. Edinburgh: Edinburgh University Press. 463–502.

Le, T.; Nguyen, H.; Pontelli, E.; and Son, T. C. 2012. ASP at work: An ASP Implementation of Phylows. In Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, 359–369. Saarbrücken, German: Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik.

Leone, N., and Ricca, F. 2015. Answer Set Programming: A Tour from the Basics to Advanced Development Tools and Industrial Applications. Tutorial Lecture presented at Reasoning Web — Web Logic Rules: 11th International Summer School 2015, Berlin, Germany, July 31 - August 4.

Liu, H., and Singh, P. 2004. ConceptNet: A Practical Commonsense Reasoning Toolkit. *BT Technology Journal* 22(4): 211–226. dx.doi.org/10.1023/B:BTTJ.0000047600.4421.6d

Manna, M.; Scarcello, F.; and Leone, N. 2011. On the Complexity of Regular-Grammars with Integer Attributes. *Journal of Computer and Systems Sciences International* 77(2): 393–421.

McCarthy, J. 1990. *Formalization of Common Sense, Papers by John McCarthy,* edited by V. Lifschitz. Norwood, NJ: Ablex.

Nogueira, M.; Balduccini, M.; Gelfond, M.; Watson, R.; and Barry, M. 2001. An A-Prolog Decision Support System for the Space Shuttle. In *Practical Aspects of Declarative Languages, Third International Symposium, PADL 2001*. Lecture Notes in Computer Science 1990, 169–183. Berlin: Springer.

Ricca, F.; Dimasi, A.; Grasso, G.; Ielpa, S. M.; Iiritano, S.; Manna, M.; and Leone, N. 2010. A Logic-Based System for E-Tourism. *Fundamenta Informatica* 105(1–2): 35–55. dx.doi.org/10.1017/S147106841100007X

Ricca, F.; Grasso, G.; Alviano, M.; Manna, M.; Lio, V.; Iiritano, S.; and Leone, N. 2012. Team-Building with Answer Set Programming in the Gioia-Tauro Seaport. *Theory and Practice of Logic Programming* 12(3): 361–381.

Shanahan, M. 1997. *Solving the Frame Problem: A Mathematical Investigation of the Commonsense Law of Inertia*. Cambridge, MA: The MIT Press.

Tiihonen, J.; Soininen, T.; and Sulonen, R. 2003. A Practical Tool for Mass-Customising Configurable Products. In *Proceedings of the 14th International Conference on Engineering Design,* 1290–1299. Edinburgh, UK: The Design Society.

Tran, N., and Baral, C. 2009. Hypothesizing about Signaling Networks. *Journal of Applied Logic* 7(3): 253–274. dx.doi.org/10.1016/j.jal.2008.10.001

Zhang, S.; Sridharan, M.; and Wyatt, J. L. 2015. Mixed Logical Inference and Probabilistic Planning for Robots in Unreliable Worlds. *IEEE Transactions on Robotics* 31(3): 699–713. dx.doi.org/10.1109/TRO.2015.2422531

Zhang, S.; Sridharan, M.; Gelfond, M.; and Wyatt, J. L. 2014. Towards an Architecture For Knowledge Representation and Reasoning in Robotics. In *Social Robotics: 6th International Conference, ICSR 2014*. Lecture Notes in Computer Science 8755, 400–410. Berlin: Springer. dx.doi.org/10.1007/978-3-319-11973-1_41

Esra Erdem is an associate professor in computer science and engineering at Sabanci University. She received her Ph.D. in computer sciences at the University of Texas at Austin (2002) and carried out postdoctoral research at the University of Toronto and Vienna University of Technology from 2002 to 2006. Her research is about the mathematical foundations of knowledge representation and reasoning and their applications to cognitive robotics and computational biology.

Michael Gelfond is a professor of computer science at Texas Tech University. He received his Ph.D. from the Institute of Mathematics of the Russian Academy of Sciences, St. Petersburg in 1974.Gelfond contributed to the development of the stable model/answer set semantics of logic programming and the answer set programming paradigm, which is founded on the stable model semantics. He is a fellow of the AAAI and recipient of three Test of Time awards from the International Association of Logic Programming.

Nicola Leone is a professor of computer science at the University of Calabria, where he heads the Department of Mathematics and Computer Science and leads the AI Lab. Until 2000 was a professor of database systems at Vienna University of Technology. He is internationally renowned for his research on knowledge representation, answer set programming, and database theory, and for the development of DLV, a state-of-the-art ASP system. He published more than 250 papers in prestigious conferences and journals, and has more than 8000 citations, with h-index 46. He is a fellow of ECCAI (now EurAI) and recipient of an ACM Test of Time award.

First-Order Logic with Inductive Definitions for Model-Based Problem Solving

Maurice Bruynooghe, Marc Denecker, Mirosław Truszczyński

■ *In answer set programming (ASP), programs can be viewed as specifications of finite Herbrand structures. Other logics can be (and, in fact, were) used toward the same end and can be taken as the basis of declarative programming systems of similar functionality as ASP. We discuss here one such logic, the logic FO(ID), and its implementation IDP3. The choice is motivated by notable similarities between ASP and FO(ID), even if both approaches trace back to different origins.*

Answer set programming (ASP), described in other articles in this issue, is based on an extension of the language of logic programming under the answer set semantics. In ASP, an instance of a computational problem is encoded by an answer set program in such a way that the Herbrand models of the program determine all solutions for the problem instance. Thus, at an abstract level, answer set programs are specifications of finite Herbrand structures (those that are models of the programs), and the key reasoning task supported by ASP systems is to compute them. That task is often referred to as model generation.

Other logics that can express constraints on Herbrand structures (or even non-Herbrand structures) could also be used as the basis for this form of declarative problem solving. Implementing the model-generation task for theories in such logics yields declarative programming tools with the same basic functionality as that of ASP systems. Two examples of such an alternative approach are the model generators NP-SPEC (Cadoli et al. 2000) and aspps (East and Truszczyński 2006) for extended Datalog. In both cases, the specification language is an extension of Datalog with clausal constraint rules. This formalism captures the class NP in the sense that any decision problem from the class NP can be specified. The system NP-SPEC offers a compiler for recursion- and negation-free NP-SPEC problems to SAT whereas aspps is a native system supporting recursive Datalog, making it suitable to model problems involving transitive closure.

A further step in this direction is the logic FO(ID). The origin of the logic FO(ID) can be traced to the fundamental problem of logic programming that also triggered the development of stable semantics and later ASP, namely, the problem of negation as failure (Lifschitz 2016). The essence of the negation problem was that negation as failure in Prolog had so many useful and intuitive applications, and yet, the original view of logic programs as sets of material implications (Horn clauses extended with negation in the body) could not account for the derivation of a single negative literal. As noted by Lifschitz (his article appears in this issue), one solution was inspired by research in nonmonotonic reasoning and consisted of adapting the semantics of default logic (Reiter 1980) to the syntax of logic programs. This led to stable semantics and later to ASP.

An alternative solution was to interpret a logic program as a definition of its predicates. In this view, a logic program consists of, for every predicate, an exhaustive list of rules that define the cases in which it is true. Unlike sets of implications, definitions entail negative information. Moreover, a case-based representation of definitions and, in particular, inductive definitions is a common way to specify definitions in mathematical texts. The view of logic programs as a definition was already implicit in Clark's first-order completion semantics (Clark 1978). However, as was well known in mathematical logic and databases, in general, inductive definitions cannot be expressed in first-order logic (FO) (Aho and Ullman 1979). Hence, Clark's semantics did not not correctly interpret recursive logic programs as inductive definitions. This weakness spurred the development of so-called canonical semantics such as the perfect model (Apt, Blair, and Walker 1988) and the well-founded model semantics (Van Gelder, Ross, and Schlipf 1991). The latter (in a suitably extended version) turned out to correctly formalize definitions, including the most common forms of inductive definitions (Denecker 1998, Denecker and Vennekens 2014). Since definitional knowledge is an important form of expert knowledge and since in general it cannot be expressed in first-order logic, it is natural to seek extensions of classical logic to incorporate it. This has been recognized by the database community, which developed such extensions in support of more expressive database query languages (Abiteboul, Hull, and Vianu 1995). It has also motivated a similar effort in knowledge representation. In particular, these considerations led to the logic FO(ID), an extension of first-order logic with inductive definitions expressed as sets of logic programlike rules under well-founded semantics that represent the base cases and (possibly) inductive cases of the definition (Denecker 2000; Denecker and Ternovska 2008).

The logic FO(ID) has a standard Tarskian model semantics. A structure satisfies a theory if it satisfies its first-order logic sentences and is a well-founded model of its definitions. Thus, the logic FO(ID) can be understood as a conjunction of its formulas and definitions and, in particular, it is a conservative extension of first-order logic. These features make the logic FO(ID) well suited as the basis of declarative knowledge representation systems.

Two systems were developed for significant fragments of the FO(ID) language: enfragmo (Aavani et al. 2012) and IDP3 (De Cat et al. 2014). The key inference task supported by these systems is (finite) model expansion: given an FO(ID) theory and a partial structure as input, the goal is to output one or more structures that are models of the theory and expand the input structure. The enfragmo system provides support for arithmetic and aggregates in its theories, and also some limited support for inductive definitions. The IDP3 system has a similar functionality as enfragmo but provides a more advanced treatment and support for inductive definitions; in addition to model expansion, it also supports several other forms of inference.

In this article, we present the logic FO(ID) and the IDP3 system. Our presentation is not formal but relies on a series of examples. We start by illustrating the importance of structures in knowledge representation and how they lead to a methodology for knowledge modeling and for declarative problem solving through the model-expansion task. We then describe the IDP3 system. This discussion is intertwined with references to the logic FO(ID), the theoretical foundation for the IDP3 system. Next we briefly discuss the relationship between FO(ID) (IDP3) and ASP and conclude with comments on the role these formalisms may play in the future of computational logic.

Structures in Knowledge Representation

First-order logic has proved to be a powerful formalism for representing knowledge largely because of two key interrelated factors. Structures used as interpretations of first-order formulas are well suited to model practical situations and application domains; and our intuitive understanding of how first-order formulas constrain the space of possible structures matches exactly the formal definition of the satisfiability relation. Our goal in this section is to present structures in their role as a fundamental abstraction for knowledge representation. When modeling a problem domain, we start by selecting symbols to denote its functions and relations. Collectively, these symbols form the vocabulary of the domain. Each symbol in the vocabulary comes with a nonnegative integer called the *arity* that denotes the number of arguments of the corresponding function or relation. If a is a relation or function symbol, we write a/k to indicate the arity of a. Function symbols of arity 0 are called *constants*.

To illustrate, let us consider a hypothetical software company that holds weekly lunch meetings for its software development teams. These lunch meetings take place on certain weekdays. Some teams are in a (scheduling) conflict (for instance, they may share a team member). The meeting days for teams in conflict must be different.

Relation symbols

team/1, *day*/1, *conflict*/2

team(x): x is a team

day(x): x is a weekday

conflict(x, y): x has a conflict with y

Function symbols (here only one)

mtng_day/1

mtng_day(x) = y: the time of lunch meeting of x is y

Figure 1. A Vocabulary of Symbols for the Lunch Meeting Domain.

Domain of S

$D^S = \{T_1, T_2, T_3, T_4, T_5, M, Tu, W, Th, F\}$

Relations

$team^S = \{T1, T2, T3, T4, T5\}$

$day^S = \{M, Tu, W, Th, F\}$

$conflict^S =$

$\{(T_1, T_2), (T_1, T_5), (T_2, T_3), (T_2, T_5), (T_3, T_4)\}$

Functions

$mtng\_day^S =$

$\{T_1 \rightarrow M, T_2 \rightarrow W, T_3 \rightarrow M, T_4 \rightarrow Tu, T_5 \rightarrow Tu,$
$M \rightarrow M, Tu \rightarrow M, W \rightarrow M, Th \rightarrow M, F \rightarrow M\}$

Figure 2. A Structure S for the Vocabulary from Figure 1.

This short text describes a problem domain. The underlined terms indicate relations and functions in that domain. Figure 1 shows symbols that could be selected to denote them, as well as their intended meaning. Together, these symbols form the vocabulary of the lunch meeting domain.

A *structure* (also called an *interpretation*) S over a vocabulary consists of a nonempty universe D and, for each symbol σ in the vocabulary, the value σ^S of σ in S (also called the interpretation of σ in S). More specifically, for every relation symbol r/k, r^S is a relation on D with k arguments (that is, $r^S \subseteq D^k$), and for every function symbol f/k, f^S is a function on D with k arguments (that is, $f^S : D^k \rightarrow D$). Figure 2 shows an example of a structure over the vocabulary from figure 1.

A structure over a vocabulary is an abstract representation of a concrete instance, or state of affairs, of a problem domain modeled in terms of this vocabulary. The universe of the structure is an abstraction of the set of objects in that instance, while the relations and functions of the structure — the values of the symbols in the vocabulary — abstract the relations and functions in the instance.

For instance, the structure S in figure 2 represents a state of affairs with five teams referred to as $T_1, \ldots, T_5$ and five (working) days referred to as $M, \ldots, F$. The structure also specifies conflicts between the teams (for example, teams T_1 and T_2 are in conflict), and an assignment of meeting days to teams (for example, team T_3 meets on Monday). Since functions in structures are total, the function $mtng\_day^S$ is also defined on days. This is redundant, as those assignments do not represent any pertinent information. The structure S is an abstraction of a possible state of affairs of our problem domain: one in which the properties of the domain mentioned in the specification of this problem domain are satisfied. Indeed, the mapping $mtng\_day^S$ assigns weekdays to teams (other assignments it makes are immaterial or, as we said, redundant), and two teams in conflict are not scheduled to have lunch on the same day.

To formally model this domain, these and other properties present in the informal description (some only implicitly) need to be expressed as sentences of the logic over the chosen vocabulary. Three properties relevant to our scenario are shown in figure 3. We see there formal sentences in the language of first-order logic over the vocabulary from figure 1, as well as their informal reading. The first sentence says that the relation *conflict* applies to teams only. It specifies the types of the arguments of the relation *conflict*. Incidentally, this information is not explicitly present in the narrative. Nevertheless, it is implicit there and can be included in any formal representation of the problem. The second sentence is of a similar nature. It describes the type of objects the function *mtng_day* maps to. The last sentence represents the *essential* constraint of the problem that teams in conflict do not hold their lunch meetings on the same day.

First-order propositions (first-order logic sentences) are true or false in structures (we also say *satisfied* or *unsatisfied*, respectively). For example, the third proposition of figure 3, expressing that conflicting teams do not meet on the same day, is true in the structure from figure 2. However, it is false in the structure that is the same except that the interpretation of conflict is extended with the pair (T_4, T_5). Indeed, we now have two teams in conflict that are scheduled to meet on the same day.

More formally, given a structure S interpreting all symbols in a first-order sentence F, we can evaluate F in S, that is, assign to it a logical value *true* or *false*. When F evaluates to true in S, we say that S is a *model* of F or that F is *satisfied* by S. The property of a sentence being true in a structure yields a satisfaction relation between structures and first-order logic sentences. It provides first-order logic sentences with a

Vocabulary: As in figure 1

Sentence:
$\forall X \forall Y\,(conflict(X, Y) \rightarrow team(X) \land team(Y))$

Reads: for all *X* and *Y,* if *X* and *Y* are in conflict then both *X* and *Y* are teams

Sentence: $\forall X \forall Y\,(mtng\_day(X) = Y \rightarrow day(Y))$

Reads: for all *X* and *Y,* if *Y* is the meeting day for *X* then *Y* is a day

Sentence: $\forall X \forall Y\,(conflict(X, Y) \rightarrow mtng\_day(X) \neq mtng\_day(Y))$

Reads: for all *X* and *Y*, if *X* and *Y* are in conflict then the meeting day for *X* is different from the meeting day for *Y*

Figure 3. Relevant Properties as First-Order Sentences.

Vocabulary: As in Figure 1

Structure: *S* defined in Figure 2

Sentence: $\forall X \forall Y\,(conflict(X, Y) \rightarrow team(X) \land team(Y))$

Evaluates to true . Indeed, for every $(a, b) \in conflict^S$, both *team*(*a*) and *team*(*b*) hold in *S* (figure 2)

Sentence: $\forall X \forall Y\,(conflict(X, Y) \rightarrow mtng\_day(X) \neq mtng\_day(Y))$

Evaluates to *true.* Indeed, for every $(a, b) \in conflict^S$,teams *a* and *b* meet on a different day (figure 2)

Figure 4. The First-Order Semantics Applied to Some Sentences from Figure 3.

semantics (the first-order semantics) that captures precisely their informal reading. Figure 4 illustrates these concepts for sentences from the language based on our example vocabulary. The second sentence would evaluate to false in case the relation $conflict^S$ contained the extra pair (T_4, T_5).

The satisfaction relation is of crucial importance. In some cases, we fully know the relevant state of affairs of the problem domain or, more precisely, we know the structure that serves as its abstract representation. However, in other cases the precise state of affairs is not known or is only partially known. In that case, our knowledge frequently consists of separate informal propositions. They implicitly specify possible states of affairs as those in which these propositions hold true. As we argued, structures are formal representations of states of affairs. Those structures that represent possible states of affairs are called *intended.* Given this terminology, knowledge representation can then be understood as the art and practice of formulating knowledge as a formal theory so that models of that theory are precisely the intended structures. For instance, the original specifications of the problem domain are correctly expressed by axioms in figure 3, which also shows their informal semantics. The structure from figure 2 is an intended one, and it indeed satisfies (is a model of) all the sentences in figure 3. That latter claim can be verified formally and is also easy to see intuitively: conflicts are between teams, the mapping $mtng\_day^S$ assigns days to teams (and also to days, but that is immaterial), and two teams in conflict are not scheduled on the same day.

To say that all models of the theory are intended structures here is slightly imprecise. For example, the theory has infinite models that hardly count as intended structures. The problem is that some implicit information such as what are weekdays and teams, is not expressed in the theory. This information is expressed in the values assigned to the symbols *team* and *day* by structures like the one in figure 2.

To recapitulate, in this setting the satisfaction relation allows us to use sentences over the fixed vocabulary to constrain structures over that vocabulary to those that satisfy the sentences. These sentences can be seen as *specifications* of classes of intended structures over that vocabulary, that is, the structures that represent those states of affairs that are possible (might be encountered in practice).

The setting we presented supports several important reasoning problems. Say the manager in our running example is reviewing a schedule proposed by one of her assistants or, more formally, the corresponding structure. The manager wants to know whether certain propositions hold for the schedule or, formally, whether the formal sentences expressing the propositions are satisfied in that structure. We call that reasoning task *model checking* or *querying*. For instance, we might want to know whether team T_2 has its meeting scheduled on the same day as team T_4 in the structure (schedule) in figure 2 (that query would evaluate to false). Model checking is a special instance of this task; it verifies that a structure satisfies the specifications, that is, that it indeed is an intended structure. In the case of our example and the structure in figure 2, it consists of verifying that all statements of the theory in figure 3 are satisfied by the structure.

Even more interesting and important is the situation when the schedule is yet to be constructed. How can the manager find one? She knows the axioms in figure 3. This information specifies the class of intended structures, each of them representing a valid instance of a lunch meeting domain. It is also reasonable to assume that she knows which teams she needs to schedule, what scheduling conflicts she

$$\begin{aligned}&mtng\_day = \\ &\quad \{T_1 \rightarrow M, T_2 \rightarrow W, T_3 \rightarrow M, T_4 \rightarrow W, T_5 \rightarrow Tu, \\ &\quad M \rightarrow M, Tu \rightarrow M, W \rightarrow M, Th \rightarrow M, F \rightarrow M\}\end{aligned}$$

Figure 5. Another Possible Schedule Function.

has to take into account, and which days are work days. This information explicitly fixes the domain of an intended structure as well as its relations team, day, and conflict (for instance, to the values they have in figure 2). Any function *mtng_day* that completes this explicitly given fragment of a structure to an intended one yields a good schedule for the setting of interest to the manager. The converse is also true. Good schedules give rise to intended structures (when combined with the explicitly given components).

The task to find the missing function, which we just described, is an example of the model-expansion problem. In model expansion we assume that the vocabulary is partitioned into input and output symbols. Given a theory (that is, a set of sentences) over the entire vocabulary and a structure over the vocabulary consisting of the input symbols, called an input structure, the goal is to extend the input structure with relations and functions for output symbols so that the resulting structure (now over the entire vocabulary) satisfies the theory.

This applies to our example scenario. Here, *team, day,* and *conflict* are input symbols and *mtng_day* is an output symbol. The input structure consists of the domain $\{T_1, \ldots, T_5, M, \ldots, F\}$ and of the relations team, day, and conflict as in figure 2. The theory specifying intended structures is given in figure 3. Under these assumptions, the model-expansion problem asks for a specific function *mtng_day* that would expand the input structure to the one satisfying the three sentences (that is, to an intended structure). That function would offer a legal schedule of lunch meetings for the five teams involved. The function shown in figure 2 is one of the possible solutions. The function in figure 5 is another one.[1]

A more involved reasoning task is assigning meeting days to teams interactively. The task involves propagation inference that calculates the valid days for every team. Each time the user assigns a meeting day to a team, the propagation inference updates the valid choices for the remaining teams. Still another task is revision, assisting a user in assigning a different meeting day to a particular team while preserving as much as possible the meeting days assigned to the other teams. All these reasoning tasks use the same theory as a specification of valid structures.

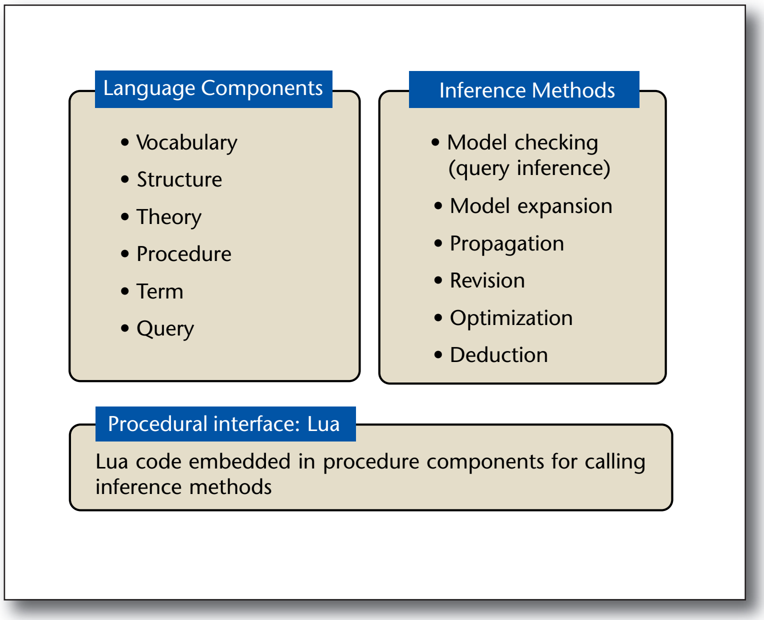


Figure 6. High-Level Representation of IDP3 as a Knowledge Base System.

To recapitulate, structures are important to us for four key reasons. First, they provide natural abstractions of states of affairs of the problem domain, in which sentences (properties) can be evaluated for satisfaction. Second, they are useful to define computational tasks in the context of logic. Third, they can be used to present input data, as in the query inference (where the value of every symbol is known), or in model expansion (where values of some symbols are not known). Fourth, they can be used as representations of answers to model-expansion problems.

The IDP3 System

We will now present a software system, IDP3, that implements the ideas presented above. In particular, IDP3 allows us to define structures, input structures and partial structures, as well as sentences to state their properties.[2] An overview of the IDP3 system is presented in figure 6. We use a series of simple examples to illustrate and discuss all key features of IDP3.

The IDP3 system separates information from the reasoning task to be performed. In this way, it facilitates the use of the same knowledge to solve diverse reasoning problems. To represent information, IDP3 uses an enriched variant of first-order logic. The information is split over three components. The first component is the vocabulary. The IDP3 syntax for describing vocabularies is illustrated in figure 7, where we again use our software team's domain as an example. The vocabulary goes beyond the basic first-order logic as it introduces not only the alphabet but also the types and the signatures of the relation and function

```
vocabulary V{
    type day
    type team
    conflict(team,team)
    mtng_day(team):day
}
```

Figure 7. An IDP3 Vocabulary Introducing Two Types, Day and Team, and a Predicate and a Function Together with Their Signatures.

```
structure S:V{
    day={M;Tu;W;Th;F}
    team={T1;T2;T3;T4;T5}
    conflict={(T1,T2);(T1,T5);(T2,T3);(T2,T5);(T3,T4)}
}
```

Figure 8. An IDP3 Structure for the Vocabulary V from Figure 7.

FO	IDP3	FO	IDP3	FO	IDP3
∧	&	≡	<=>	=	=
∨	|	¬	~	≠	~=
→	=>	∀	!	≤	=<
←	<=	∃	?	≥	>=

Figure 9. Translation Table.

Note that implication translates into a double arrow, while the translation of ≤ and ≥ is not an arrow.

```
theory T:V{
    ! a[team] b[team]: conflict(a,b) =>
                    mtng_day(a) ~= mtng_day(b).
}
```

Figure 10. An IDP3 Theory over the Vocabulary of Figure 7 (Using Correspondences from Figure 9).

symbols. In IDP3, vocabularies are assigned identifiers; the vocabulary in figure 7 is identified by *V*.

The second component is a *structure*. The IDP3 syntax for a structure for our example domain is shown in figure 8. The structure has an identifier (here *S*). It refers to a vocabulary (here *V*) and introduces the domains for the types declared in the vocabulary (here by enumerating the elements of types *day* and *team*). In addition it also enumerates the known information about the declared relations and functions. The interpretation of *conflict* is fully known, and the relation is specified by the list of its tuples. However, as nothing is known about the interpretation of *mtng_day*, nothing about this function is mentioned in the structure.

The last component used to express information is a theory. An example theory appropriate for our domain is shown in figure 10. The notation is essentially first-order logic but in a keyboard-friendly syntax. The IDP3 counterparts to standard mathematical logic notation are given in figure 9.

The theory (as the other two components) has an identifier (here *T*). The theory refers to a vocabulary (in the example, through the identifier *V*), and expresses the constraints that a structure must meet to serve as a valid abstraction of the problem domain (here, the function *mtng_day* is constrained so that teams in conflict are not scheduled for their lunch meetings on the same day). The types of the variables are optional; if omitted, type inference will derive them. In the case at hand, the type of the variables *a* and *b* can be derived from the signature of *conflict* and *mtng_day*.

For solving problems, the IDP3 system offers a procedural interface in Lua[3] and executes the procedure *main*(). In *main*(), the IDP3 user can overwrite default values of solver options, can invoke Lua functions provided by the IDP3 designers as well as standard Lua functions, and can also invoke functions written by the user in the procedural component. The information components (vocabularies, structures, and theories) are first-class citizens in the Lua code and can be passed as parameters to various functions. The procedural component in figure 11 illustrates some common use patterns of the Lua interface. In this code, *modelexpand, printmodels,* and *allmodels* are Lua functions provided with the IDP3 system, while # is the Lua operator that returns the length of a Lua sequence and *print* is the standard Lua printing function.

The *modelexpand* function invokes model-expansion inference on a theory *T* and a structure *S* (both referring to the same vocabulary), and returns a (possibly empty) Lua sequence of models of *T* that extend *S;* by default, *modelexpand* is bound to return a single model. To obtain more models, another bound can be set with an assignment to the *stdoptions.nbmodels* option. The *printmodels* function prints the number of models in a sequence of models as well as each of its models. To obtain the sequence of all models, one can use the *allmodels* function. Indexing can be used to select a particular element in a sequence; if the sequence is empty (models do not exist), the special Lua value *nil* is returned. Models are represented and printed as structure components so that they can serve as IDP3 input.

```
procedure main(){
    stdoptions.nbmodels=5
    printmodels(modelexpand(T,S))
    models=allmodels(T,S)
    print(#models)
    print(model[5])
    print(model[980])
}
```

Figure 11. Solving The Model-Expansion Problem In IDP3.

This procedure applies model-expansion inference on our theory *T* and initial structure *S* in two different ways. The first line sets the bound on the number of models to five. The second line invokes model expansion and prints the sequence of five models. The third line also invokes model expansion but returns the sequence of all models and assigns it to the Lua variable *models*. The fourth line prints the number of models in the sequence, the next line prints the fifth model, and the final line prints *nil* as there are only 960 models.

Definitions, Aggregates and Optimization

So far, we have seen two extensions of first-order logic that are available in IDP3: *types* and *partial* functions (a typed function is partial as it is only defined for the values determined by the types in the signature). Other important extensions are *aggregates* and *definitions*. To illustrate them, we elaborate on our example; at the same time we also introduce another reasoning task, optimization inference. The IDP3 code for the extended example is shown in figure 12.

In the extended scenario, we are concerned with the workload of the company cafeteria where the meetings take place. We introduce the concept of *quiet_day*, which we define as a day in which at most one team holds its meeting. As stated before, definition expressions in FO(ID) are modeled after the way definitions are expressed in text. They define one or more predicate or function symbol in terms of a set of parameter symbols; for example, the concept *quiet_day* is defined in terms of the function *mtng_day,* which we call a parameter of the definition. To distinguish a definition expression from first-order logic sentences, it is written as a set of rules placed between "**define** {" and "}."[4]

Each rule expresses one (base or inductive) case. The head and body of the rule are separated by "<–", called the *definitional* implication to distinguish it from the material implication => (both given in the IDP syntax). The head is an atomic formula of one of the defined predicates and the body can be any formula in first-order logic. In contrast with logic programming, variables are explicitly quantified. To give such formal rule sets the intuitive reading of definitions in mathematics, the semantics chosen for them is an extension of the well-founded semantics (Van Gelder, Ross, and Schlipf 1991; Denecker and Ternovska 2008), because the well-founded semantics correctly formalizes the most common forms of definitions found in text (Denecker and Vennekens 2014). The IDP3 definition given in figure 12 formally expresses the intended meaning for the concept *quiet_day:* a day *d* is a quiet day if it is not the case that two different teams (*t*1, *t*2) have their meeting day on *d*.

Similarly, we include in the vocabulary a function *nmbr_mtngs* that we want to define as the function that maps a day to the number of teams meeting on this day. The new symbol ranges over the new type *number*, which we introduce in the vocabulary as a subtype of the natural numbers (a built-in type *nat*) and specify in the structure as the set of numbers from 0 to 10. As the relation *quiet_day*, also this function is defined in terms of the parameter *mtng_day*. In general, functions are defined by sets of rules of the form $f(t_1, \ldots, t_n) = t$ <– *body*. In the case at hand, the body degenerates to "true" and is omitted. Importantly, the function value here is given by the *cardinality* aggregate #{*tm : mtng day* (*tm*) = *d*}. This aggregate represents the cardinality of the set {(*tm*) | *mtng_day*(*tm*) = *d*}, that is, the number of teams meeting on day *d*. Besides cardinality, IDP3 also supports *minimum*, *maximum*, *sum*, and *product* aggregates. They have a slightly different syntax. An overview of the supported aggregates is given in figure 13.

Assume that the manager of the cafeteria wishes to minimize the maximal workload for the cafeteria. To solve this problem, another form of inference is needed called *optimization* inference. This is done by the procedure call *minimize*(*T, S, m*) in the *main*() procedure of figure 12. The procedure call contains yet another sort of component of IDP3: the *term* component. Its role is to give a name to a term. Referred to by its name, the term can then be used inside Lua procedures. Here, the term of interest is the maximum *n* in the set of pairs (*d, n*) defined as {(*d, n*) | *n* = *nmbr_mtngs*(*d*)}. This is a simple *maximum* aggregate in which there are no extra conditions on *d*. According to the translation table of figure 13, its IDP3 syntax is *max*{*d* [*day*] : *true* : *nmbr_mtngs*(*d*)}. The middle part, *true*, is the trivially true extra condition on the selected values for *d*. The optimization inference performs a search for a model that minimizes the value of the term referred to by *m*. The call not only returns a model, but also whether optimality could be shown

```
vocabulary V{
  …
  type number isa nat
  quiet_day(day)
  nmbr_mtngs(day):number
}
structure S:V{
  …
  number={0..10}
}
theory T:V{
  …
  define {
   ! d: quiet_day(d)<- ~(? t1t2: t1=+t2&
               mtng_day(t1)=mtng_day(t2)=d).
  }
  define{
   !d: nmbr_mtngs(d)=#{tm: mtng_day(tm)=d}
  }
}
term m:V{
  max{d[day] : true : nmbr_mtngs(d)}
}
procedure main() {
  stdoptions.cpsupport=true
  models, optimal, cost = minimize(T,S,m)
  print(models[1])
  print(optimal)
  print(cost)
}
```

Figure 12. The Running Example Extended to Illustrate Aggregates, Definitions, and Optimization.

The missing lines of code should be taken from Figures 7, 8, and 10.

FO	IDP3
$\#\{\bar{x} : F\}$	#{x1 … xn : F }
$sum\{(\bar{x},t) : F\}$	sum{x1… xn: F:t }
$prod\{(\bar{x},t) : F\}$	prod{x1 … xn : F : t }
$max\{(\bar{x},t) : F\}$	max{x1… xn: F:t }
$min\{(\bar{x},t) : F\}$	min{x1… xn: F:t }

Figure 13. Translation Table for Aggregates.

and the value of the term. So, minimization is on the maximal number of meetings on the same day. In other words, the call *minimize*(*T, S, m*) returns a schedule that minimizes the maximum number of lunch meetings scheduled for a single day (informally, it offers a "balanced" schedule). The grounder of the IDP3 system is unable to derive a bound on the value of the optimization term *m*. To avoid an infinite grounding, the option *cpsupport* must be on.

Partial Information and Constructed Types

As an alternative elaboration of our example, assume it is decided that team *T*1 meets on Monday ("is certainly true") and team *T*2 does not meet on Tuesday ("is certainly false"). This partial knowledge can be expressed in the structure as shown in figure 14 (we note the use of markers <ct> and <cf>).

Alternatively, we may want to express this information in the theory. However, in the theory we can only express information about domain elements if we have symbols in the vocabulary to refer to them. Hence, we need to extend the vocabulary with constants *mon, tue,* …, *t*1, *t*2, … to denote days and teams; furthermore, the structure needs to be extended to specify the interpretation for the new constants by means of the statements *mon* = *M*, …, *t*1 = *T*1, … . Only then we can express constraints such as *mtng_day* (*t*1) = *mon* or *mtng_day* (*t*2)~=*tue* in the theory. This verbose way is a consequence of the fact that functions and constants are not limited to their Herbrand interpretation as in ASP and Prolog. A shortcut is to make use of constructed types to enforce Herbrand interpretations over certain types. Figure 15 shows how constructed types impose the same constraints on the function *mtng_day*. As the domain of these types is fixed in the vocabulary, they are not part of any structure.

More about Definitions

The definitions we discussed above are simple and can be expressed in first-order logic as equivalences. For example, the equivalence !*d* : *quiet day*(*d*) ⇔~(?*t*1 *t*2 : *t*1~=*t*2 & *mtng_day* (*t*1) = *mtng_day* (*t*1) = *d*) correctly expresses the definition of *quiet_day*. While this works for all noninductive definitions, it is well known that inductive definitions in general cannot be expressed through first-order logic equivalences.

Definitions are the most substantial extension that IDP3 offers with respect to first-order logic. Not only do they offer the designer a facility to define concepts, they also increase the expressiveness. The archetypal example of a relation that cannot be expressed in first-order logic is the transitive closure of the edges in a graph. The inductive definition of this relation, say *T,* for a graph (*N, E*) with nodes *N* and edges *E* is often stated as follows:

If (*a, b*) ∈ *E,* then (*a, b*) ∈ *T,*

If for some $c \in N$, it holds that $(a, c) \in T$ and $(c, b) \in T$, then also $(a, b) \in T$. In IDP3, we can model it as in figure 16.

To further illustrate the power of definitions, we present in figure 17 a representation of a simple graph problem that requires selecting edges among nodes so that in the resulting graph all vertices are reachable from a (given) node *root* and none of the (given) forbidden edges are selected. The main difficulty is that the set of vertices reachable from the root is not expressible in first-order logic. To overcome this problem, we introduce the auxiliary unary predicate symbol *reachable* and express it through the inductive definition provided in figure 17. Additional axioms express that the defined relation *reachable* is the set of all nodes and no edges are forbidden. An interesting aspect is that here, the defined relation *reachable* is known initially while the parameter *edge* in terms of which it is defined is unknown. Hence, IDP3 searches for an interpretation of the parameter *edge* such that the defined relation *reachable* has the given value. This sort of input/output pattern is different from that of the Prolog and Datalog systems, and it shows the declarative nature of definitions. It is a powerful aspect of IDP3 as well as ASP systems.

In the example in figure 17, one can check that the edges (*A, D*) and (*D, C*) must appear in every solution for the relation *edge*. Also, at least one of (*D, B*) or (*C, B*) must be present. Other allowed edges are not constrained. Thus, one possible value for *edge* is {(*A, D*), (*D, B*), (*D, C*)} and another one is {(*A, D*), (*C, B*), (*D, C*), (*B, D*)}.[5]

FO(ID), (IDP3), and ASP

On the conceptual level, FO(ID) and ASP are quite different. Whereas ASP has its foundation in nonmonotonic and commonsense reasoning, FO(ID) is based on a definition construct inspired by the structure of definitions used in mathematics. Negation in ASP is viewed as a nonclassical epistemic or default operator. In FO(ID), it is the definitional rule operator ← that is nonclassical, while negation in the bodies of definition rules is classical. And yet, despite these different foundations, there are strong structural relationships between ASP and FO(ID). On the language level, FO(ID)'s rule-based definition construct resembles ASP rules, and first-order logic axioms resemble ASP constraints. We illustrate these similarities with the problem of finding a Hamiltonian cycle in a directed graph. An answer set program encoding the problem is shown in figure 18.

That program has a typical structure resulting from following the generate-define-test (GDT) methodology (Lifschitz 2002) (discussed in this issue by Faber, Gebser, and Schaub [2016]). This methodology leads to three sorts of modules. The first of them generates the space of candidate solutions (in our example, the space of all subsets of the set of edges of the input

```
structure S:V{
  ...
  mtng day<ct> = {T1 –>M }
  mtng day<cf> = {T2 –>Tu }
}
```

Figure 14. Partial Knowledge in a Structure.

```
vocabulary  V{
  type day constructed from {M,Tu,W,Th,F}
  type team constructed from {T1,T2,T3,T4,T5}
  ...
}
theory  T:V{
  ...
  mtng_day(T1) = M.
  mtng_day(T2) ~= Tu.
}
```

Figure 15. Constructed Types.

```
vocabulary  V{
  type node
  edge(node,node)
  trans(node,node)
}
structure  S:V{
  edge={...}
}
theory T:V
  define {
    ! x y: trans(x,y) <– edge(x,y).
    ! x y: trans(x,y) <– ? z: trans(x,z) & trans(z,y).
  }
}
```

Figure 16. Transitive Closure of the Edges in a Graph.

```
vocabulary V {
  type node
  forbidden(node,node)
  edge(node,node)
  reachable(node)
  root:node
}
structure S:V {
  node = A..D // a shorthand for {A; B; C; D}
  forbidden = {(A,A); (A,B); (A,C); (B,A);
          (B,B); (B,C); (C,C); (C,D); (D,D)}
  root = A
}
theory T:V {
  // inductive definition of reachable
 define {
  reachable(root).
  !x:reachable(x) <- ?y:reachable(y)&edge(y,x)
 }
  // The graph is fully connected
  ! x: reachable(x).
  // No forbidden edges
  ! x y: edge(x,y) => ~ forbidden(x,y).
}
```

Figure 17. A Graph Problem.

```
generate   {In(x,y)}<–Edge(x,y)
--------------------------------------------
define     T (x, y)<–In(x, y).
           T (x,y)<–T(x,z), T(z,y)
--------------------------------------------
test       <– In((x,y), In (x,z), y ≠ z.
           <– In(x, z), In(y, z), x ≠ y
           <–Node(x),Node(y), not T(x,y).
```

Figure 18. A Generate-Define-Test ASP Program Encoding the Existence of a Hamiltonian Cycle Problem.

graph; they are possible instantiations of a relation *In*). The generate module commonly relies on the construct of choice rules (as in our example) or, alternatively, uses disjunctive rules. The second one defines some additional concepts that are useful in identifying solutions (here, the transitive closure of the relation *In*). Finally, the third one specifies constraints of the problem. These constraints narrow down the space of candidate solutions to those that represent the valid ones (here, the constraints ensure that exactly one edge comes into each node, exactly one edge leaves each node, and finally, that all nodes are connected to each other both ways; that last condition requires an auxiliary concept of the transitive closure). The horizontal lines in figure 18 make this structure explicit.

The corresponding IDP3 solution to the problem has a similar format. We present its theory component in figure 19.

The similarity is striking. The first sentence plays the role of the generate module in the program in figure 18. The definition of the transitive closure mirrors the define module. Finally, the last three sentences are the three constraints of the test module cast in the IDP3 syntax. As an aside, we note that a direct translation from natural language to the IDP3 syntax of these constraints would more likely be as in figure 20.

Almost all GDT programs can be translated into IDP3 following the idea outlined above. The encoding of the generate module does not require any special syntax. In fact, in many cases the generate part of a GDT program disappears entirely from the corresponding IDP3 theory. The converse is also true. A large class of IDP3 theories allows for automated rewritings into the language of ASP. The key in such translations is to properly construct the choice rules to "open" some of the predicates.

Similarities between ASP and FO(ID) can be found not only in the structure of programs (theories). On the system level, the core of IDP3 is a model generator that is developed using similar technologies as current ASP solvers, and offers similar functionalities.

Concluding Remarks

In this article, we focused on the model-generation task because of its natural applications in solving search and optimization problems. This is also the focus of ASP and ASP implementations. We noted that model generation can be implemented for other logics. We mentioned some of them and then described in detail the logic FO(ID) and the associated reasoning system IDP3.

However, it is important to point out that the knowledge present in both FO(ID) theories as well as in answer set programs can support many other reasoning tasks besides model generation. That observation has played a central role in the development of the system IDP3 and is reflected in its functionality (see figure 6). Similarly, it underlined some developments in ASP (see the article by Kaufmann et al. [2016] in this issue). In particular, most implementations of ASP support skeptical and brave reasoning, and add-ons facilitating abduction and planning were developed for some systems, as well (Eiter et al. 2003, 1999).

The field of computational logic has an urgent need for integrative frameworks that recognize that many reasoning tasks are needed in knowledge-intensive applications and that these tasks can all be driven by a single well-designed underlying knowledge base. Formalisms and systems discussed in this special issue are on the intersection of several related lines of research, building on the advances in classical logic, automated reasoning, logic programming, databases, satisfiability, satisfiability modulo theories, constraint programming, fix-point logics, and description logics. As such, they are well suited to play this integrative role. Their modeling capabilities, which in important respects, such as the ability to capture inductive definitions, go beyond SAT/CSP formalisms, as well as the computational effectiveness of their reasoning software demonstrate that. We posit that developing FO(ID), ASP, and related formalisms with this goal in mind is essential both for the theory of logic-based computation and for practical applications.

```
theory T:V {
   ! x y: In(x,y) => Edge(x,y).

   define {
     ! x y: T(x,y) <- In(x,y).
     ! x y z T(x,y) <- T(x,z) & T(z,y).
     }

   ! x y z: ~ (In(x,y) & In(x,z) & y ~ = z).
   ! x y z: ~ (In(x,z) & In(y,z) & x ~ = y).
   ! x y: ~ (Node(x) & Node(y) & ~ T(x,y)).
}
```

Figure 19. An IDP3 Theory Encoding the Existence of a Hamiltonian Cycle Problem.

```
! x y z: In(x,y) & In(x,z) => y = z.
! x y z: In(x,y) & In(y,z) => y = z.
! x y: T(x,y).
```

Figure 20. A Direct IDP3 Representation of the Test Constraints.

Notes

1. A more general version of the model-expansion problem takes a partially instantiated structure (a fully specified domain but possibly only partially instantiated relations and functions for all vocabulary symbols) and asks if it can be completed to a structure that satisfies the theory.

2. The IDP3 system has been developed by the Knowledge Representation and Reasoning group at the University of Leuven, dtai.cs.kuleuven.be/topics/kbs. The most recent versions of the source code and documentation, as well as other resources such as an online IDE, are available at the IDP page, dtai.cs.kuleuven.be/software/idp.

3. Lua is a scripting language (Ierusalimschy, de Figueiredo, and Celes 1996) available at www.lua. org.

4. The keyword **define** is optional.

5. This theory can be accessed and experimented with on the IDP-IDE webpage at dtai.cs.kuleuven.be/krr/idpide/?present=forbidden.

Acknowledgments

This research was supported by the project GOA 13/010 Research Fund KULeuven and projects G.0489.10, G.0357.12, and G.0922.13 of the Research Foundation, Flanders.

References

Aavani, A.; Wu, X. N.; Tasharrofi, S.; Ternovska, E.; and Mitchell, D. G. 2012. Enfragmo: A System for Modelling and Solving Search Problems with Logic. In *Proceedings of the 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning,* LPAR 2012, volume 7180 of LNCS, ed. N. Bjørner and A. Voronkov, 15–22. Springer. dx.doi.org/10.1007/978-3-642-28717-6_4

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases.* Boston, MA: Addison-Wesley.

Aho, A. V., and Ullman, J. D. 1979. The Universality of Data Retrieval Languages. In *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages*, January 1979, 110–120. New York: Association for Computing Machinery. dx.doi.org/10.1145/567752.567763

Apt, K.; Blair, H.; and Walker, A. 1988. Towards a Theory of Declarative Knowledge. In *Foundations of Deductive Databases and Logic Programming,* ed. J. Minker, 89–142. San Mateo, CA: Morgan Kaufmann. dx.doi.org/10.1016/b978-0-934613-40-8.50006-3

Cadoli, M.; Ianni, G.; Palopoli, L.; Schaerf, A.; and Vasile, D. 2000. NP-SPEC: An Executable Specification Language for Solving All Problems in NP. *Computer Languages* 26(2–4): 165–195. dx.doi.org/10.1016/S0096-0551(01)00010-8

Clark, K. 1978. Negation as Failure. In *Logic and Data Bases,* ed. H. Gallaire and J. Minker. New York-London: Plenum Press. 293–322. dx.doi.org/10.1007/978-1-4684-3384-5_11

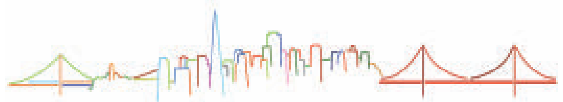
De Cat, B.; Bogaerts, B.; Bruynooghe, M.; and Denecker, M. 2014. Predicate Logic as a Modelling Language: The IDP System. Unpublished paper (CoRR abs/1401.6312). Ithaca, New York: Cornell University Library.

Denecker, M. 1998. The Well-Founded Semantics Is the Principle of Inductive Definition. In *Logics in Artificial Intelligence: European Workshop, JELIA 1998 Proceedings,* volume 1489 of Lecture Notes in Computer Science, ed. J. Dix, L. del Cerro,and U. Furbach, U., 1–16. Berlin: Springer. dx.doi.org/10.1007/3-540-49545-2_1

Denecker, M. 2000. Extending Classical Logic with Inductive Definitions. In *Computational Logic, CL 2000,* volume 1861 of Lecture Notes in Computer Science, ed. J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L. Pereira, Y. Sagiv, and P. Stuckey, 703–717. Berlin: Springer.

Denecker, M., and Ternovska, E. 2008. A Logic of Nonmonotone Inductive Definitions. *ACM Transactions on Computer Logic* 9(2): 14:1–14:52.

Denecker, M., and Vennekens, J. 2014. The Well-Founded Semantics Is the Principle of Inductive Definition, Revisit-



It's Time to Plan Your Trip to San Francisco for AAAI-17!

The Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17) and the Twenty-Ninth Conference on Innovative Applications of Artificial Intelligence (IAAI-17) will be held February 4-9 at the Hilton San Francisco Union Square in San Francisco, California, USA. San Francisco is a world-class city full of delights for every visitor. From the iconic Golden Gate bridge to its renowned art galleries, science museums, and picturesque neighborhoods, the City takes pride in its unrivaled attractions and treasures. The Hilton Hotel is located near Union Square and very close to shops, restaurants, and entertainment venues, as well as public transportation. The conference location is a great starting point to explore the City's tremendous ethnic and cultural diversity and its wide variety of offerings. San Francisco is also perfectly positioned to explore the entire Bay Area, whether for recreation or business. The Napa Valley, Pacific Coast, Silicon Valley, and East Bay are all a short distance away. As an unrivaled center for AI and tech professionals, San Francisco promises to be a vibrant home for AAAI-17.

For local information, please visit the San Francisco travel site at www.sftravel.com.

ed. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning,* KR 2014, ed. C. Baral, G. Giacomo, and T. Eiter. Palo Alto, CA: AAAI Press.

East, D., and Truszczyński, M. 2006. Predicate-Calculus-Based Logics for Modeling and Solving Search Problems. *ACM Transactions on Computational Logic* 7(1): 38–83. dx.doi.org/10.1145/1119439.1119441

Eiter, T.; Faber, W.; Leone, N.; and Pfeifer, G. 1999. The Diagnosis Frontend of the dlv System. *AI Communications* 12(1–2):99–111.

Eiter, T.; Faber, W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2003. A Logic Programming Approach to Knowledge-State Planning, II: The dlvk System. *Artificial Intelligence* 144(1–2): 157–211. dx.doi.org/10.1016/S0004-3702(02)00367-3

Faber, W.; Gebser, M.; and Schaub, T. 2016. Modeling and Language Extensions. *AI Magazine* 37(3).

Ierusalimschy, R.; de Figueiredo, L. H.; and Celes, W. 1996. Lua – An Extensible Extension Language. *Software: Practice and Experience* 26(6): 635–652. dx.doi.org/10.1002/(sici)1097-024x(199606)26:6<635::aid-spe26>3.0.co;2-p

Kaufmann, B.; Leone, N.; Perri, S.; and Schaub, T. 2016. Grounding and Solving in Answer Set Programming. *AI Magazine* 37(3).

Lifschitz, V. 2016. Answer Sets and the Language of Answer Set Programming. *AI Magazine* 37(3).

Lifschitz, V. 2002. Answer Set Programming and Plan Generation. *Artificial Intelligence* 138: 39–54. dx.doi.org/10.1016/S0004-3702(02)00186-8

Reiter, R. 1980. A Logic for Default Reasoning. *Artificial Intelligence* 13(1-2): 81–132. dx.doi.org/10.1016/0004-3702(80)90014-4

Van Gelder, A.; Ross, K. A.; and Schlipf, J. S. 1991. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM* 38(3): 620–650. dx.doi.org/10.1145/116825.116838

Maurice Bruynooghe has been, since October 2015, a professor emeritus at the Department of Computer Science of Katholieke Universiteit Leuven. After his Ph.D. work on logic programming, also at Katholieke Universiteit Leuven, was completed in 1979, he started the Declarative Languages and Artificial Intelligence research group. His research has been in logic programming, program analysis, inductive logic programming, machine learning, and knowledge representation and reasoning.

Marc Denecker is professor of computer science at Katholieke Universiteit Leuven. He is interested in all aspects of declarative information and its use for automated problem solving. His research is concerned with the formal study of knowledge and information, with semantic foundations of declarative specification languages in monotone and nonmonotonic reasoning, with the development of practical knowledge representation languages, of methodologies and inference methods including propagation, constraint programming, satisfiability checking, abduction: in general, any form of inference of use to solve problems and perform tasks using declarative knowledge. He is head of the research group Knowledge Representation and Reasoning that developed the IDP language and system that is presented in this article.

Mirosław Truszczyński is a professor of computer science at the University of Kentucky. His research interests include knowledge representation, nonmonotonic reasoning, logic programming, and constraint satisfaction. He has published more than 180 technical papers, coauthored a research monograph on nonmonotonic logics, and edited 10 article collections and conference proceedings. His paper Stable Logic Programming, a joint work with Victor Marek, helped launch the field of answer set programming. Truszczyński served on the Executive Committee of the Association of Logic Programming, was chair of the Steering Committee of Nonmonotonic Reasoning Workshops, and was president of Knowledge Representation Inc. He served as an editor and associate editor on boards of the *Journal of Artificial Intelligence Research* and *Artificial Intelligence Journal*. He is now editor-in-chief of *Theory and Practice of Logic Programming*, and an associate editor of *AI Communications*. In 2013, Truszczyński was elected Fellow of the Association for the Advancement of Artificial Intelligence (AAAI).

Symbiotic Cognitive Computing

Robert Farrell, Jonathan Lenchner, Jeffrey Kephart, Alan Webb, Michael Muller, Thomas Erickson, David Melville, Rachel Bellamy, Daniel Gruen, Jonathan Connell, Danny Soroker, Andy Aaron, Shari Trewin, Maryam Ashoori, Jason Ellis, Brian Gaucher, Dario Gil

■ *IBM Research is engaged in a research program in symbiotic cognitive computing to investigate how to embed cognitive computing in physical spaces. This article proposes five key principles of symbiotic cognitive computing: context, connection, representation, modularity, and adaptation, along with the requirements that flow from these principles. We describe how these principles are applied in a particular symbiotic cognitive computing environment and in an illustrative application for strategic decision making. Our results suggest that these principles and the associated software architecture provide a solid foundation for building applications where people and intelligent agents work together in a shared physical and computational environment. We conclude with a list of challenges that lie ahead.*

In 2011, IBM's Watson competed on the game show *Jeopardy!* winning against the two best players of all time, Brad Rutter and Ken Jennings (Ferrucci et al. 2010). Since this demonstration, IBM has expanded its research program in artificial intelligence (AI), including the areas of natural language processing and machine learning (Kelly and Hamm 2013). Ultimately, IBM sees the opportunity to develop cognitive computing — a unified and universal platform for computational intelligence (Modha et al. 2011). But how might cognitive computing work in real environments — and in concert with people?

In 2013, our group within IBM Research started to explore how to embed cognitive computing in physical environments. We built a Cognitive Environments Laboratory (CEL) (see figure 1) as a living lab to explore how people and cognitive computing come together.

Our effort focuses not only on the physical and computational substrate, but also on the users' experience. We envision a fluid and natural interaction that extends through time across multiple environments (office, meeting room, living room, car, mobile). In this view, cognitive computing systems are always on and available to engage with people in the environment. The system appears to follow individual users, or groups of users, as they change environments, seamlessly connecting the users to available input and output devices and extending their reach beyond their own cognitive and sensory abilities.

We call this *symbiotic cognitive computing:* computation that takes place when people and intelligent agents come together in a physical space to interact with one another. The intelligent agents use a computational substrate of "cogs" for visual object recognition, natural language parsing, probabilistic decision support, and other functions. The term *cog* is from

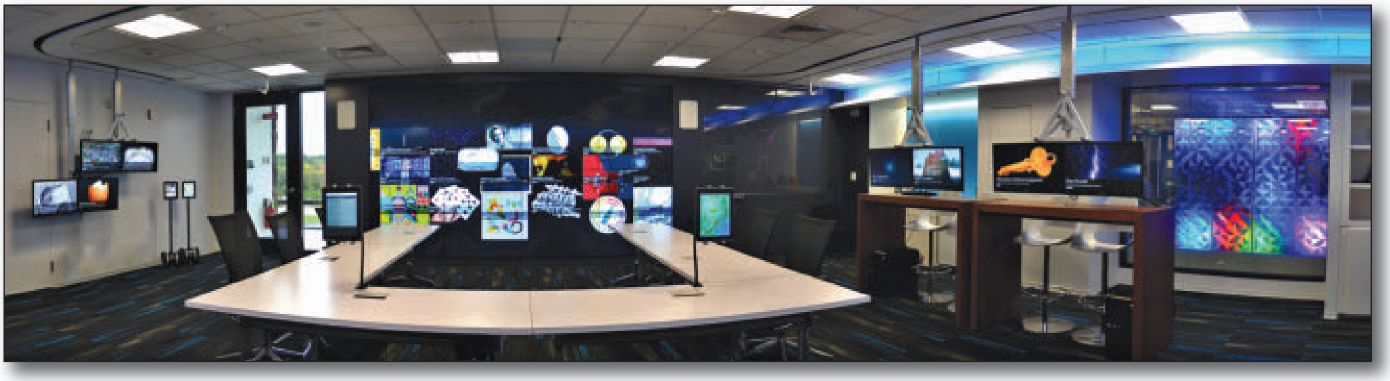


Figure 1. The Cognitive Environments Lab.

CEL is equipped with movement sensors, microphones, cameras, speakers, and displays. Speech and gesture are used to run cloud-based services, manipulate data, run analytics, and generate spoken and visual outputs. Wands, and other devices enable users to move visual elements in three dimensions across displays and interact directly with data.

the book *The Society of Mind* where Marvin Minsky likened agents to "cogs of great machines" (Minsky 1988). These cogs are available to intelligent agents through programmatic interfaces and to human participants through user interfaces.

Our long-term goal is to produce a physical and computational environment that measurably improves the performance of groups on key tasks requiring large amounts of data and significant mental effort, such as information discovery, situational assessment, product design, and strategic decision making. To date, we have focused specifically on building a cognitive environment, a physical space embedded with cognitive computing systems, to support business meetings for strategic decision making. Other applications include corporate meetings exploring potential mergers and acquisitions, executive meetings on whether to purchase oil fields, and utility company meetings to address electrical grid outages. These meetings often bring together a group of participants with varied roles, skills, expertise, and points of view. They involve making decisions with a large number of high-impact choices that need to be evaluated on multiple dimensions taking into account large amounts of structured and unstructured data.

While meetings are an essential part of business, studies show that they are generally costly and unproductive, and participants find them too frequent, lengthy, and boring (Romano and Nunamaker 2001). Despite this, intelligent systems have the potential to vastly improve our ability to have productive meetings (Shrobe et al. 2001). For example, an intelligent system can remember every conversation, record all information on displays, and answer questions for meeting participants. People making high-stakes, high-pressure decisions have high expectations. They typically do not have the time or desire to use computing systems that add to their workload or distract from the task at hand. Thus, we are aiming for a "frictionless" environment that is always available, knowledgeable, and engaged. The system must eliminate any extraneous steps between thought and computation, and minimize disruptions while bringing important information to the fore.

The remainder of this article is organized as follows. In the next section, we review the literature that motivated our vision of symbiotic cognitive computing. We then propose five fundamental principles of symbiotic cognitive computing. We list some of the key requirements for cognitive environments that implement these principles. We then provide a description of the Cognitive Environments Lab, our cognitive environments test bed, and introduce a prototype meeting-support application we built for corporate mergers and acquisitions (M&A) that runs in this environment. We wrap up by returning to our basic tenets, stating our conclusions, and listing problems for future study.

Background

In his paper "Man-Machine Symbiosis," J. C. R. Licklider (1960) originated the concept of symbiotic computing. He wrote,

> Present-day computers are designed primarily to solve preformulated problems or to process data according to predetermined procedures. … However, many problems … are very difficult to think through in advance. They would be easier to solve, and they could be solved faster, through an intuitively guided trial-and-error procedure in which the computer cooperated, turning up flaws in the reasoning or revealing unexpected turns in the solution.

Licklider stressed that this kind of computer-supported cooperation was important for real-time decision making. He thought it important to "bring computing machines effectively into processes of thinking that must go on in real time, time that moves too fast to permit using computers in conventional ways." Licklider likely did not foresee the explosive growth in data and computing power in the last several decades, but he was remarkably prescient in his vision of man-machine symbiosis.

Distributed cognition (Hutchins 1995) recognizes that people form a tightly coupled system with their environment. Cognition does not occur solely or even mostly within an individual human mind, but rather is distributed across people, the artifacts they use, and the environments in which they operate. External representations often capture the current understanding of the group, and collaboration is mediated by the representations that are created, manipulated, and shared. In activity theory (Nardi 1996), shared representations are used for establishing collective goals and for communication and coordinated action around those goals.

Work on cognitive architectures (Langley, Laird, and Rogers 2009; Anderson 1983; Laird, Newell, and Rosenbloom 1987) focuses on discovering the underlying mechanisms of human cognition. For example, the adaptive control of thought (ACT family of cognitive architectures includes semantic networks for modeling long-term memory and production rules for modeling reasoning, and learning mechanisms for improving both (Anderson, Farrell, and Sauers 1984).

Work on multiagent systems (Genesereth and Ketchpel 1994) has focused on building intelligent systems that use coordination, and potentially competition, among relatively simple, independently constructed software agents to perform tasks that normally require human intelligence. Minsky (1988) explained that "each mental agent in itself can do some simple thing that needs no mind or thought at all. Yet when we join these agents in societies — in certain very special ways — this leads to true intelligence."

Calm technology (Weiser and Brown 1996) suggests that when peripheral awareness is engaged, people can more readily focus their attention. People are typically aware of a lot of peripheral information, and something will move to the center of their attention, for example when they perceive that things are not going as expected. They will then process the item in focus, and when they are done it will fade back to the periphery.

We have used these ideas as the basis for our vision of symbiotic cognitive computing.

Principles of Symbiotic Cognitive Computing

Our work leads us to propose five key principles of symbiotic cognitive computing: context, connection, representation, modularity, and adaption. These principles suggest requirements for an effective symbiosis between intelligent agents and human participants in a physical environment.

The *context principle* states that the symbiosis should be grounded in the current physical and cognitive circumstances. The environment should maintain presence, track and reflect activity, and build and manage context. To maintain presence, the environment should provide the means for intelligent agents to communicate their availability and function, and should attempt to identify people who are available to engage with intelligent agents and with one another. To track and reflect activity, the environment should follow the activity of people and between people and among people, and the physical and computational objects in the environment or environments. It should, when appropriate, communicate the activity back to people in the environment. At other times, it should await human initiatives before communicating or acting. To build and manage context, the environment should create and maintain active visual and linguistic contexts within and across environments to serve as common ground for the people and machines in the symbiosis, and should provide references to shared physical and digital artifacts and to conversational foci.

The *connection principle* states that the symbiosis should engage humans and machines with one another. The environment should reduce barriers, distractions, and interruptions and not put physical (for example, walls) or digital barriers (for example, pixels) between people. The environment should, when appropriate, detect and respond to opportunities to interact with people across visual and auditory modalities. The environment should provide multiple independent means for users to discover and interact with agents. It should enable people and agents to communicate within and across environments using visual and auditory modalities. The environment should also help users establish joint goals both with one another and with agents. Finally, the environment should include agents that are cooperative with users in all interactions, helping users understand the users' own goals and options, and conveying relevant information in a timely fashion (Grice 1975).

The *representation principle* states that the symbiosis should produce representations that become the basis for communication, joint goals, and coordinated action between and among humans and machines. The environment should maintain internal representations based on the tracked users, joint goals, and activities that are stored for later retrieval. The environment should externalize selected representations, and any potential misunderstandings, to coordinate with users and facilitate transfer of representations across cognitive environments. Finally, the environment should utilize the internal and external representations to enable seamless context switching between different physical spaces and between different activities within the same physical space by retrieving the appropriate stored representations for the current context.

The *modularity principle* states that the symbiosis should be driven by largely independent modular composable computational elements that operate on the representations and can be accessed equally by humans and machines. The environment should provide a means for modular software components to describe themselves for use by other agents or by people in the environment. The environment should also provide a means of composing modular software components, which perform limited tasks with a subset of the representation, with other components, to collectively produce behavior for agents. The environment should provide means for modular software components to communicate with one another independently of the people in the environment.

Finally, the *adaptation principle* states that the symbiosis should improve with time. The environment should provide adequate feedback to users and accept feedback from users. Finally, the environment should incrementally improve the symbiosis from interactions with users and in effect, learn.

We arrived at these principles by reflecting upon the state of human-computer interaction with intelligent agents and on our own experiences attempting to create effective symbiotic interactions in the CEL. The context principle originates from our observation that most conversational systems operate with little or no linguistic or visual context. Breakdowns often occur during human-machine dialogue due to lack of shared context. The connection principle arises out of our observation that today's devices are often situated between people and become an impediment to engagement. The representation principle was motivated by our observation that people often resolve ambiguities, disagreements, and diverging goals by drawing or creating other visual artifacts. The use of external representations reduces the domain of discourse and focuses parties on a shared understanding. The modularity principle arose from the practical considerations associated with building the system. We needed ways of adding competing or complementary cogs without reimplementing existing cogs. The adaptation principle was motivated by the need to apply machine learning algorithms to a larger range of human-computer interaction tasks. Natural language parsing, multimodal reference resolution, and other tasks should improve through user input and feedback.

One question we asked ourselves when designing the principles was whether they apply equally to human-human and human-computer interaction. Context, connection, representation, and adaptation all apply equally well to these situations. The modularity principle may appear to be an exception, but the ability to surface cogs to both human and computer participants in the environment enables both better collaboration and improved human-computer interaction.

Cognitive environments that implement these requirements enable people and intelligent agents to be mutually aware of each others' presence and activity, develop connections through interaction, create shared representations, and improve over time. By providing both intelligent agents and human participants with access to the same representations and the same computational building blocks, a natural symbiosis can be supported.

It is impossible to argue that we have found a definitive set of principles; future researchers may find better ones or perhaps more self-evident ones from which the ones we have articulated can be derived. It may even be possible to create a better symbiotic cognitive system than any we have created and not obey one or more of our principles. We look forward to hearing about any such developments.

We are starting to realize these principles and requirements by building prototype cognitive environments at IBM Research laboratories worldwide.

The Cognitive Environments Laboratory

The Cognitive Environments Laboratory is located at the IBM T. J. Watson Research Center in Yorktown Heights, New York. The lab is meant to be a test bed for exploring what various envisioned cognitive environments might be like. It is more heavily instrumented than the vast majority of our envisioned cognitive environments, but the idea is that over time we will see what instrumentation works and what does not. The lab is focused on engaging users with one another by providing just the right technology to support this engagement.

Perhaps the most prominent feature of the CEL is its large number of displays. In the front of the room there is a four by four array of high definition monitors (1920 x 1080 pixel resolution), which act like a single large display surface. On either side of the room are two pairs of high definition monitors on tracks. These monitor pairs can be moved from the back to the front of the room along tracks inlaid in the ceiling, enabling fast and immediate reconfiguration of the room to match many meeting types and activities. In the back of the room there is an 84-inch touch-enabled 3840 x 2160 pixel display. The monitors are laid out around the periphery of the room. Within the room, visual content can either be moved programmatically or with the aid of special ultrasound-enabled pointing devices called "wands" or with combinations of gesture and voice, from monitor to monitor or within individual monitors.

In addition to the displays, the room is outfitted with a large number of microphones and speakers. There are several lapel microphones, gooseneck microphones, and a smattering of microphones attached to the ceiling. We have also experimented with array microphones that support "beam forming" to isolate the speech of multiple simultaneous speakers without the need for individual microphones.

An intelligent agent we named Celia (cognitive environments laboratory intelligent agent) senses the conversation of the room occupants and becomes a supporting participant in meetings. With the aid of a speech-to-text transcription system, the room can document what is being said. Moreover, the text and audio content of meetings is continuously archived. Participants can ask, for example, to recall the transcript or audio of all meetings that discussed "graph databases" or the segment of the current meeting where such databases were discussed. Transcribed utterances are parsed using various natural language processing technologies, and may be recognized as commands, statements, or questions. For example, one can define a listener that waits for key words or phrases that trigger commands to the system to do something. The listener can also test whether certain preconditions are satisfied, such as whether certain objects are being displayed. Commands can invoke agents that retrieve information from the web or databases, run structured or unstructured data analytics, route questions to the Watson question-answering system, and produce interactive visualizations. Moreover, with the aid of a text-to-speech system, the room can synthesize appropriate responses to commands. The system can be configured to use the voice of IBM Watson or a different voice.

In addition to the audio and video output systems, the room contains eight pan-tilt-zoom cameras, four of which are Internet Protocol (IP) cameras, plus three depth-sensing devices, one of which is gimbal mounted with software-controllable pan and tilt capability. The depth-sensing systems are used to detect the presence of people in the room and track their location and hand gestures. The current set of multichannel output technologies (that is, including screens and speakers) and multichannel input technologies (that is, keyboard, speech-to-text, motion) provide an array of mixed-initiative possibilities.

People in the CEL can simultaneously gesture and speak to Celia to manipulate and select objects and operate on those objects with data analytics and services. The room can then generate and display information and generate speech to indicate objects of interest, explain concepts, or provide affordances for further interaction. The experience is one of interacting with Celia as a gateway to a large number of independently addressable components, cogs, many of which work instantly to augment the cognitive abilities of the group of people in the room.

Dependence on a single modality in a complex environment generally leads to ineffective and inconvenient interactions (Oviatt 2000). Environments can enable higher level and robust interactions by exploiting the redundancy in multimodal inputs (speech, gesture, vision). The integration of speech and gesture modalities has been shown to provide both flexibility and convenience to users (Krum 2002). Several prototypes have been implemented and described in the literature. Bolt (1980) used voice and gesture inputs to issue commands to display simple shapes on a large screen. Sherma (2003) and Carbini (2006) extended this idea to a multiuser interaction space. We have built upon the ideas in this work in creating the mergers and acquisitions application.

Mergers and Acquisitions Application

In 2014 and 2015 we built a prototype system, situated in the Cognitive Environments Laboratory, for exploring how a corporate strategy team makes decisions regarding potential mergers and acquisitions. As depicted in figure 2, one or more people can use speech and gestures to interact with displayed objects and with Celia. The system has proven useful for exploring some of the interaction patterns between people and intelligent agents in a high-stakes decision-making scenario, and for suggesting architectural requirements and research challenges that may apply generally to symbiotic cognitive computing systems.

In the prototype, specialists working on mergers and acquisitions try to find reasonable acquisition targets and understand the trade-offs between them. They compare companies side by side and receive guidance about which companies are most aligned with their preferences, as inferred through repeated interactions. The end result is a small set of companies to investigate with a full-fledged "due diligence" analysis that takes place following the meeting.

When the human collaborators have interacted with the prototype to bring it to the point depicted in figure 2, they have explored the space of mergers and acquisitions candidates, querying the system for companies with relevant business descriptions and numeric attributes that fall within desired ranges, such as the number of employees and the quarterly revenue. As information revealed by Celia is interleaved with discussions among the collaborators, often triggered by that information, the collaborators develop an idea of which company attributes matter most to them. They can then invoke a decision table to finish exploring the trade-offs.

Figure 3 provides a high-level view of the cognitive environment and its multiagent software architecture. Agents communicate with one another through a publish-and-subscribe messaging system (the message broker) and through HTTP web services using the Representational State Transfer (REST) software design pattern. The system functions can be divided into command interpretation, command execution, agent management, decision making, text and data analysis, text-to-speech, visualization, and manage-

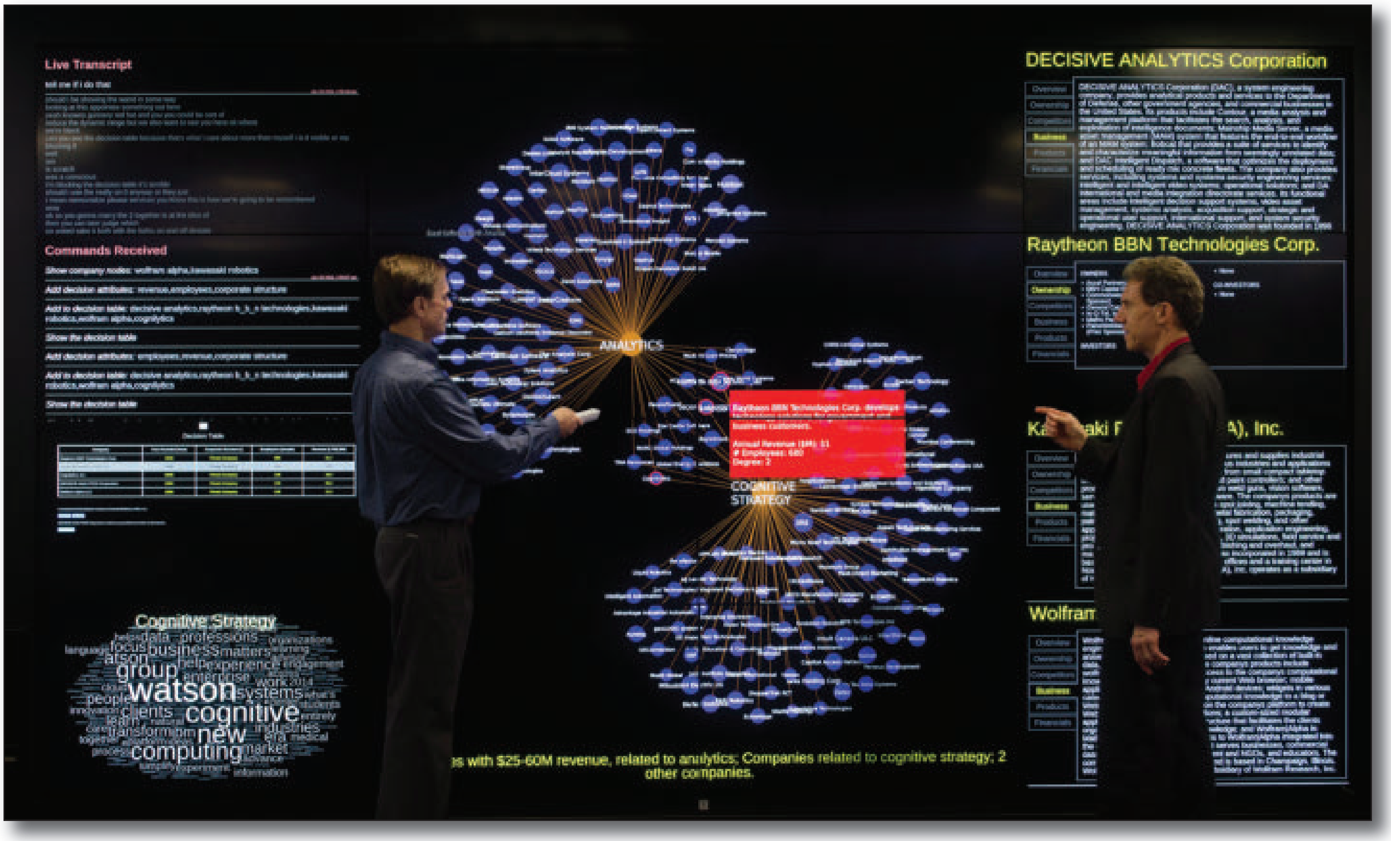


Figure 2. The Mergers and Acquisitions Prototype Application.

People working with one another and with Celia to discover companies that match desired criteria obtain detailed information about likely candidates and winnow the chosen companies down to a small number that are most suitable.

ment. We explain each of these functions in the sections that follow.

Command Interpretation

The system enables speech and gesture to be used together as one natural method of communication. Utterances are captured by microphones, rendered into text by speech recognition engines, and published to a "transcript" message channel managed by the message broker. The message broker supports high-performance asynchronous messaging suitable for the real-time concurrent communication in the cognitive environment.

We have tested and customized a variety of speech-recognition engines for the cognitive environment. The default engine is IBM Attila (Soltau, Saon, and Kingsbury 2010). It has two modes: a first that renders the transcription of an utterance once a break is detected on the speech channel (for example, half a second of silence), and a second that renders a word-by-word transcription without waiting for a pause. In the former mode there is some probability that subsequent words will alter the assessment of earlier words. We run both modes in parallel to enable agents to read and publish partial interpretations immediately.

Position and motion tracking uses output from the position and motion sensors in combination with visual object recognition using input from the cameras to locate, identify, and follow physical objects in the environment. The user identity tracking agent maps recognized people to unique users using acoustic speaker identification, verbal introduction ("Celia, I am Bob"), facial recognition upon entry, or other methods. The speaker's identity, if known, is added to each message on the transcript channel, making it possible to interleave dialogues to some degree, but further research is needed to handle complex multiuser dialogues. The persistent session information includes persistent identities for users, including name, title, and other information collected during and across sessions.

The natural language parsing agent subscribes to the transcript channel, processes text transcriptions of utterances containing an attention word (for example, "Celia" or "Watson") into a semantic representation that captures the type of command and any relevant parameters, and publishes the representation to a command channel. Our default parser is

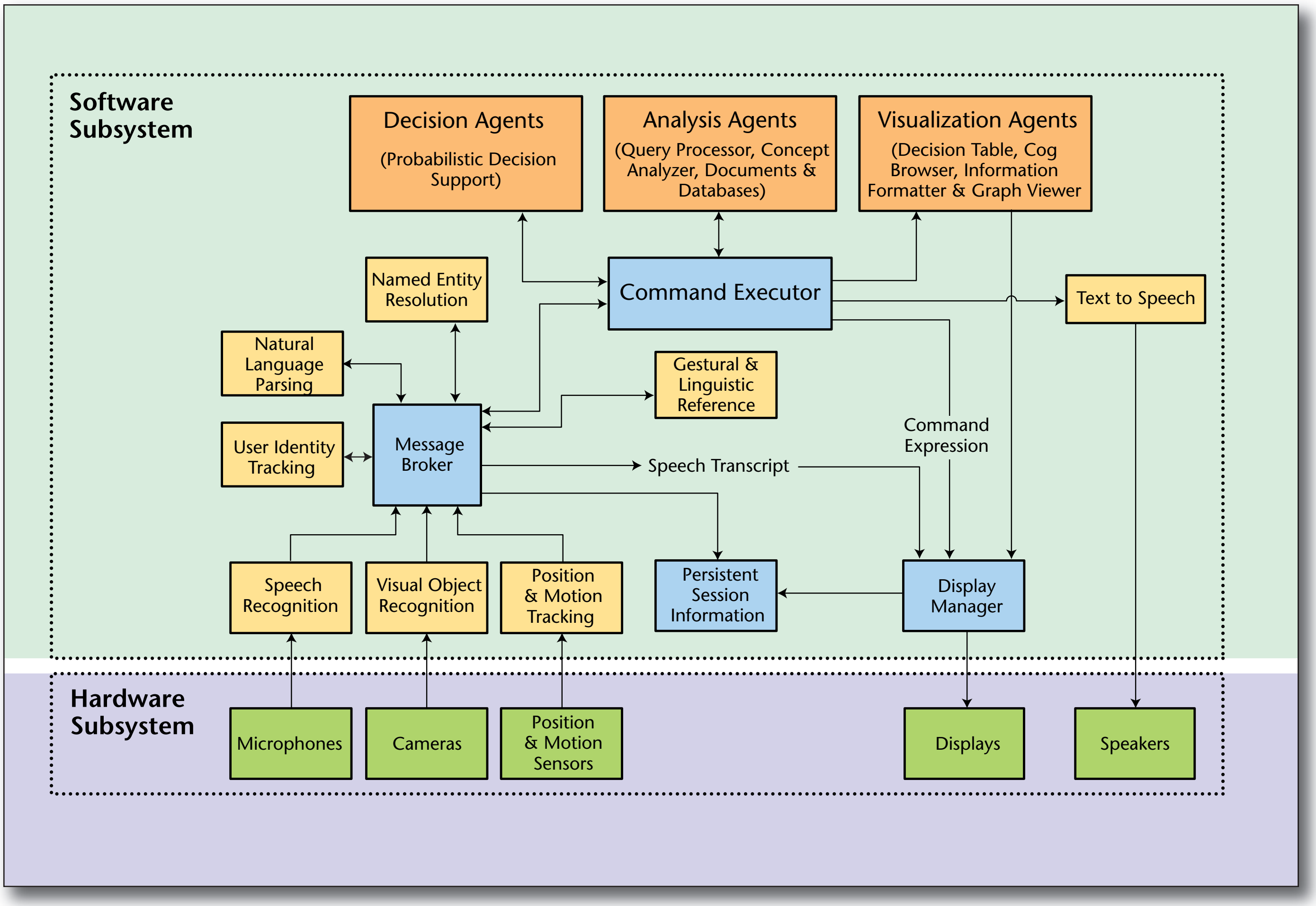

Software Subsystem
Decision Agents (Probabilistic Decision Support)
Analysis Agents (Query Processor, Concept Analyzer, Documents & Databases)
Visualization Agents (Decision Table, Cog Browser, Information Formatter & Graph Viewer
Named Entity Resolution
Command Executor
Text to Speech
Natural Language Parsing
Gestural & Linguistic Reference
User Identity Tracking
Message Broker
Speech Transcript
Command Expression
Speech Recognition
Visual Object Recognition
Position & Motion Tracking
Persistent Session Information
Display Manager
Hardware Subsystem
Microphones
Cameras
Position & Motion Sensors
Displays
Speakers

Figure 3. Architecture of the Mergers and Acquisitions Prototype.

based on regular expression matching and template filling. It uses a hierarchical, composable set of functions that match tokens in the text to grammatical patterns and outputs a semantic representation that can be passed on to the command executor. Most sentences have a subject-verb-object structure, with "Celia" as the subject, a verb that corresponds to a primitive function of one of the agents or a more complex domain-specific procedure, and an object that corresponds to one or more named entities, as resolved by the named entity resolution agent, or with modifiers that operate on primitive types such as numbers or dates. Another parser we have running in the cognitive environment uses a semantic grammar that decomposes top-level commands into terminal tokens or unconstrained dictation of up to five words. The resulting parse tree is then transformed into commands, each with a set of slots and fillers (Connell 2014). A third parser using Slot Grammar (SG) is from the Watson System (McCord, Murdock, and Boguraev 2012). It is a deep parser, producing both syntactic structure and semantic annotations on input sentences. Interrogatives can be identified by the SG parser and routed directly to a version of the IBM Watson system, with the highest confidence answer generated back to the user through text-to-speech.

An important issue that arose early in the development of the prototype was the imperfection of speech transcription. We targeted a command completion rate of over 90 percent for experienced users, but with word-recognition accuracies in the low to mid 90 percent range and commands ranging from 5 to 20 or more words in length, we were not achieving this target. To address this deficiency, we developed several mechanisms to ensure that speech-based communication between humans and Celia would work acceptably in practice. First, using a half hour of utterances captured from user interaction with the prototype, we trained a speech model and enhanced this with a domain-specific language model using a database of 8500 company names extracted from both structured and unstructured sources. The named entity resolution agent was extended to resolve acronyms and abbreviations and to match both phonetically and lexicographically. To provide better feedback to users, we added a speech transcript. Celia's utterances are labeled with "Celia," and a command expression is also shown to reflect the command that the system processed. If the system doesn't respond as expected, users can see whether Celia misinterpreted their request, and if so, reissue it. Finally, we implemented an undo feature to support restoration of the immediately prior state when the system misinterprets the previous command.

The gestural and linguistic reference agent is responsible for fusing inputs from multiple modes into a single command. It maintains persistent referents for recently displayed visual elements and recently mentioned named entities for the duration of a session. When Celia or a user mentions a particular company or other named entity, this agent creates a referent to the entity. Likewise, when the display manager shows a particular company or other visual element at the request of a user or Celia, a referent is generated. Using the referents, this agent can find relevant entities for users or Celia based on linguistic references such as pronouns, gestures such as pointing, or both. Typically the referent is either something the speaker is pointing toward or is something that has recently been mentioned. In the event that the pronoun refers to something being pointed at, the gestural and linguistic reference agent may need to use the visual object recognition's output, or if the item being pointed at resides on a screen, the agent can request the virtual object from the appropriate agent.

Command Execution

The command executor agent subscribes to the command channel and oversees command execution. Some of its functions may be domain-specific. The command executor agent communicates over HTTP web services to the decision agents, analysis agents, and visualization agents. Often, the command executor serves as an orchestrator, calling a first agent, receiving the response, and reformulating that response to another agent. Thus, the command executor maintains state during the utterance. A single request from a user may trigger a cascade of agent-to-agent communication throughout the command execution part of the flow, eventually culminating in activity on the displays and/or synthesized speech being played over the speakers.

Decision Making

In the mergers and acquisitions application, people have high-level goals that they want the system to help them achieve. A common approach from decision theory is to specify a utility function (Walsh et al. 2004). Given a utility function defined in terms of attributes that are of concern to the user, the system's objective is to take actions or adjust controls so as to reach a feasible state that yields the highest possible utility. A particularly attractive property of this approach is that utility can be used to propagate objectives through a system from one agent to another. However, experience with our symbiotic cognitive computing prototype suggests that this traditional approach misses something vital: people start with inexact notions of what they want and use computational tools to explore the options. It is only during this sometimes-serendipitous exploration process that they come to understand their goals better. Certain companies appeal to us, sometimes before we even know why, and it can take a serious introspection effort (an effort that may be assisted by the cognitive system) to discover which attributes matter most to us or to realize we may be biased. This realization prompted us to design a probabilistic decision support agent (Bhattacharjya and Kephart 2014). This agent starts with a set of candidate solutions to the decision support problem and attributes, and a highly uncertain model of user preferences. As the user accepts or rejects its recommendations, or answers questions about trade-offs, the agent progressively sharpens its understanding of the users' objectives, which it models as a probability distribution of weights in the space of possible utility functions. The agent is able to recommend filtering actions, such as removing companies or removing attributes, to help users converge on a small number of targets for mergers and acquisitions.

Text and Data Analysis

The concept analyzer agent provides additional data for decision making by extracting concepts and relationships from documents. While IBM Watson was trained for general question answering using primarily open web sources such as Wikipedia, we anticipate that most applications will also involve harnessing data from private databases and third-party services. For the mergers and acquisitions application, we developed a large database of company annual reports. Concepts extracted from the reports can be linked to the companies displayed in the graph viewer and when a user asks for companies similar to a selected company, the system is able to retrieve companies through the concepts and relationships. The query processor agent provides a query interface to a database of company financial information, such as annual revenue, price-earnings ratio, and income.

Persistent Session Information

We have added the ability for the cognitive environment to capture the state of the interaction between users and agents either as needed or at the end of a session. It does this by creating a snapshot of what agents are active, what is being displayed, what has been said, and what commands have been completed. The snapshots are saved and accessible from any device thus enabling users to take products of the work session outside of the cognitive environment.

The session capture feature allows users to review past decisions and can support case-based reasoning (Leake 1996). It also provides continuity because users can stop a decision-making process and continue later. Finally, it allows for some degree of portability across multiple cognitive environments.

Text-to-Speech

The text-to-speech agent converts text into spoken voice. The current system has a choice of two voices: a North American English female voice or a North American English male voice (which was used by the IBM Watson system). In order to keep interruptions to a minimum, the speech output is used sparingly and usually in conjunction with the visual display.

Visualization

The visualization agents work in the cognitive environment's multiuser multiscreen networked environment. Celia places content on the 25 displays and either the visualization agents or the users then manipulate the content in three dimensions. We designed the M&A application visualizations to work together in the same visual space using common styles and behaviors. The display manager coordinates content placement and rendering using placement defaults and constraints. The cog browser enables people to find and learn about the available cogs in the cognitive environment. It displays a cloud of icons representing the society of cogs. The icons can be expanded to reveal information about each cog and how they are invoked.

Many agents have functions that can be addressed through speech commands or through gesture. For example, the information formatter can display company information (on the right in figure 2) and allows a user to say "products" or select the products tab to get more information about the company's products. In addition, many of the agents that provide building blocks for Celia's decision-making functions are cogs that are also independently addressable through visual interfaces and speech commands. For example, the graph viewer can be used by Celia to recommend companies but is also available to users for visualizing companies meeting various criteria.

Agent Management

We have implemented several management modules that operate in parallel with the other parts of the system to allow agents to find instances of other agents that are running and thereby enable discovery and communication. When an agent is first launched, it registers itself to a life-cycle manager module to advertise its REST interfaces and types of messages it publishes over specific publish-and-subscribe channels. When an agent requires the services of a second agent, it can locate an instance of the second agent by querying the lifecycle manager, thereby avoiding the need to know details of the second agent's running instance.

Exchange 1

Brian: Celia, this is Brian. I need help with acquisitions.

Celia: Hello Brian, how can I help you with mergers and acquisitions?

Exchange 2

Brian: Celia, show me companies with revenue between $25 million and $50 million and between 100 and 500 employees, pertaining to analytics.

Celia: Here is a graph showing 96 companies pertaining to biotechnology *(Celia displays the graph).*

Exchange 3

Brian: Celia, place the companies named brain science, lintolin, and tata, in a decision table.

Celia: Ok. (*Celia shows a table with the 3 companies, one per row, and with columns for the name of the company, the revenue, and number of employees).*

Celia: I suggest removing Lyntolin. Brain Sciences, Incorporated has greater revenue and greater number of employees (*Celia highlights Brain Sciences and Lyntolin).*

Figure 4. A Sample Dialogue Processed by the Mergers and Acquisitions Prototype.

The agents used in the M&A application and others are available as cogs in the CEL and work as one intelligent agent to provide access to cognitive computing services. Taken together, the agents provide a completely new computing experience for business users facing tough decisions.

A sample dialogue is shown in figure 4. To handle exchange 1, the system processes the first sentence using the speech recognition engine and sends a message with the transcribed text to the message broker on the transcript channel. The natural language parser listens on the transcript channel and publishes a semantic representation based on a dependency parse of the input that identifies the name Brian in the object role. The user identity tracking agent resolves the name against the persistent identifier for Brian and publishes a command on the command channel with a set user identifier. The command executor then requests the speech recognition agent to start using the speaker's speech model. The next sentence is processed similarly, but the gestural and linguistic reference agent resolves "I" to "Brian." The verb "help" is recognized as the main action and the actor is the persistent identifier for Brian. The command executor recognizes the representation as the initialization of the mergers and acquisitions application using a pattern rule. It calls the display manager, which saves and hides the state of the displays. The command executor then generates the response that is sent to the text to speech agent. This requires querying the user identity tracking agent to map the persistent identifier for Brian to his name.

To handle exchange 2, a similar flow happens, except the command executor calls the query processor agent to find companies matching the revenue and company size criteria. Upon receiving the response, the command executor calls the graph viewer, which adds nodes representing each matching biotechnology company to a force-directed graph on the display (in the center in figure 2). The command executor also calls the text-to-speech agent to play an acknowledgement over the speakers. The user can then manipulate the graph using gestures or issue further speech commands.

For exchange 3, the command executor first calls the named entity resolver three times to resolve the exact names of the companies referred to by the user; for example it might resolve "brain science" into "Brain Sciences, Incorporated." Upon receiving these responses, the command executor calls the query processor agent to obtain company information, which it then sends to the probabilistic decision support agent. This agent must interact with the decision table agent, which in turn uses the display manager to display the output to the user. While all of this is happening, the executor also calls the text-to-speech agent to acknowledge the user's request. Coordination between speech and display thus happens in the command executor. During this interaction, the transcript displayer and command displayer display the utterance and the interpreted command.

In the next section, we discuss our work to date on the cognitive environment in terms of both prior work and our original symbiotic cognitive computing principles.

Discussion

Prior intelligent decision-making environments focus primarily on sensor fusion, but fall short of demonstrating an intelligent meeting participant (Ramos et al 2010). The New EasyLiving Project attempted to create a coherent user experience, but was focused on integrating I/O devices (Brumitt et al. 2000). The NIST Meeting Room has more than 280 microphones, seven HD cameras, a smart whiteboard, and a locator system for the meeting attendees (Stanford et al. 2003), but little in the way of intelligent decision support. The CALO (Cognitive Assistant that Learns and Organizes) DARPA project includes a meeting assistant that captures speech, pen, and other meeting data and produces an automated transcript, segmented by topic, and performs shallow discourse understanding to produce a list of probable action items (Voss and Ehlen 2007), but it does not focus on multimodal interaction.

The experience of building and using the M&A application has been valuable in several respects. First, while we haven't yet run a formal evaluation, we've found that the concept of a cognitive environment for decision making resonates well with business users. To date we have now had more than 50 groups of industry executives see a demonstration and provide feedback. We are now working closely with the mergers and acquisitions specialists at IBM to bring aspects of the prototype into everyday use. Second, the prototype has helped us to refine and at least partially realize the symbiotic cognitive computing principles defined in this article, and to gain a better understanding of the nature of the research challenges. Here we assess our work on the prototype in terms of those principles.

First, how much of the symbiosis is grounded in the physical and cognitive circumstances? We have just started to explore the use of physical and linguistic context to shape the symbiosis. Some aspects of maintaining presence are implemented. For example, motion tracking and visual recognition are used to capture the presence of people in the room. However, endowing intelligent agents with the ability to effectively exploit information about individual people and their activities and capabilities remains a significant research challenge. Multiple people in the cognitive environment can interact with the system, but the system's ability to associate activities with individual users is limited. The session capture agent tracks both human commands and agent actions, but additional work is required to reflect the activity of people and agents in the environment back to participants. The linguistic and gestural reference agent maintains some useful context for switching between applications, but additional research is needed to exploit this context to enable an extended dialogue.

Second, how much does the cognitive environment support the connection principle, enabling people and intelligent agents to engage with one another? We feel that the architecture and implementation support all of the requirements, at least to some degree. First, barriers between human intention and system execution are reduced by multimodal interactions that allow users to converse with the system almost as if it were a human partner rather than having to deal with the cumbersome conventions of typical user interfaces, but the lack of affordances in speech-based interaction remains a challenge. The cog browser provides users with some understanding of the capabilities of various cogs but the system does not offer assistance. The system supports interactions across multiple environments; cogs can in effect follow the user to different physical spaces, marshaling the input and output resources that they find there — thereby reducing the time required to initiate system computations and actions when moving across cognitive environments. The cognitive environment cooperates with users in multiple ways: decision agents use elicitation techniques to develop an understanding of user goals and trade-offs and then to guide users toward decisions that best realize them, Celia listens for commands and the command executor responds only when adequate information is available for a response. Because multiple users can see the same display and Celia has access to displayed objects through the display manager, Celia can track and respond to their coordinated actions.

Does the cognitive environment support the representation principle? The CEL and its agents maintain representations that are the basis for communication between participants and with Celia. The identity and location of users in the room, the developing conversation with Celia and recognized commands, and the state of ongoing decisions are all captured, externalized, and shared outside the environment, providing common ground between both people in the physical environment and those connected to the environment only remotely or periodically. In future work, we would like to recognize individual differences in representation preferences, and be able to conduct "private" interactions with individual users through the media of their choice.

We have realized the modularity requirements of self-description, composition, and intercomponent communication by implementing the cognitive environment as a multiagent system. Some agents are strongly human centered, providing services such as speech and gesture transcription, speech synthesis, or visualization. Others mainly serve the needs of other agents. For example, the life-cycle manager facilitates agent communication and composition by enabling agents to advertise their capabilities to one another and use one another's services. An important research challenge is to create deeper semantic descriptions of services to allow users to select and compose services as needed through natural language dialogue.

How does the cognitive environment support adaptation, improving with time? Currently most of the system's improvement is offline, not during the dialogue. For example, we trained speech models and extended the language model with custom dictionaries. Users' gestural vocabularies could also be modeled and interpreted, or we could develop individualized models of combinations of speech, gesture, and larger movements. We currently capture useful data during interactions with users that can be used to improve interactions in the future. For example, the system captures linguistic and gestural context, which can in principle be mined by other agents seeking to detect patterns that might be used to better anticipate user needs. Ultimately we would like cognitive systems to adapt to users' goals and capabilities, available I/O resources, and available cogs to maximize the effectiveness of the entire session and improve the symbiotic relationship between users and the system.

Despite our successes with engineering a symbiotic cognitive computing experience, practical applications continue to be a challenge: speech commands in a noisy room are often misrecognized and we cannot reliably identify individual speakers, the tracking of gestures is still error prone with both wands and hand gestures, and natural language inputs require domain-specific natural language engineering to map commands to the proper software services and invoke decision, analysis, and visualization agents. Despite these challenges, the cognitive environment provides a valuable test bed for integrating a variety of IBM Research cognitive computing technologies into new scenarios and business applications.

Conclusions

This article introduced our work on symbiotic cognitive computing. We outlined five principles: context, connection, representation, modularity, and adaptation, and we showed how key requirements that flow from these principles could be realized in a cognitive environment. We have started to apply this environment to real business problems, including strategic decision making for corporate mergers and acquisitions.

Reaching a true symbiosis between cognitive computing and human cognition is a significant multiyear challenge. The IBM Watson question-answering system and other intelligent agents can be embedded in physical spaces, but additional research is needed to create cognitive computing systems that can truly sense the world around them and fully interact with people to solve difficult problems.

Our future directions include detection and understanding of emotion, cognitive computing in virtual and mixed reality, simultaneous speech and gesture understanding, integration of uncertain data from sensors into real-time interaction, and machine learning to improve decisions over time. We are also interested in exploring tasks such as information discovery, situational assessment, and product design where difficult decisions require bringing together people who have complementary skills and experience and providing them with large amounts of structured and unstructured data in one collaborative multisensory multimodal environment.

The IBM Watson *Jeopardy!* system demonstrated the ability of machines to achieve a high level of performance at a task normally considered to require human intelligence. We see symbiotic cognitive computing as the next natural step in the evolution of intelligent machines: creating machines that are embedded in the world and integrate with every aspect of life.

Acknowledgements

Thanks also to Wendy Kellogg, Werner Geyer, Casey Dugan, Felicity Spowart, Bonnie John, Vinay Venkataraman, Shang Gao, Mishal Dholakia, Tomas Beren, Yedendra Shirnivasan, Mark Podlaseck, Lisa Amini, Hui Su, and Guru Banavar.

References

Anderson, J. 1983. *The Architecture of Cognition.* Mahwah, NJ: Lawrence Erlbaum Associates.

Anderson, J.; Farrell, R.; and Sauers, R. 1984. Learning to Program in LISP. *Cognitive Science* 8(2): 87–129. dx.doi.org/10.1207/s15516709cog0802_1

Bhattacharjya, D., and Kephart, J. O. 2014. Bayesian Interactive Decision Support for Multi-Attribute Problems with Even Swaps. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence,* 72–81. Seattle, WA: AUAI Press.

Bolt, R. A. 1980. Put-That-There. In *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques: SIGGRAPH '80,* 262–270. New York: Association for Computing Machinery. dx.doi.org/10.1145/800250.807503

Brumitt, B. L.; Meyers, B.; Krumm, J.; Kern, A.; and Shafer, S. 2000. EasyLiving: Technologies for Intelligent Environments. In *Handheld and Ubiquitous Computing, 2nd International Symposium,* September, Lecture Notes in Computer science Volume 1927, ed P. Thomas and H.-W. Gellersen, 12–27. Berlin: Springer.

Carbini, S.; Delphin-Poulat, L.; Perron, L.; and Viallet, J. 2006. From a Wizard of Oz Experiment to a Real Time Speech and Gesture Multimodal Interface. *Signal Processing* 86(12): 3559-3577. dx.doi.org/10.1016/j.sigpro.2006.04.001.

Connell, J. 2014. Extensible Grounding of Speech for Robot Instruction. In *Robots That Talk and Listen,* ed. J. Markowitz,175–201. Berlin: Walter de Gruyter GmbH and Co.

Ferrucci, D.; Brown, E.; Chu-Carroll, J.; Fan, J.; Gondek, D.; Kalyanpur, A. A.; Lally, A.; Murdock, J. W.; Nyberg, E.; Prager, J.; Schlaefer, N.; and Welty, C. 2010. Building Watson: An Overview of the DeepQA Project. *AI Magazine* 31(3): 59–79.

Genesereth, M. R., and Ketchpel, S. P. 1994. Software Agents. *Communications of the ACM* 37(7), 48–53. dx.doi.org/10.1145/176789.176794

Grice, P. 1975. Logic and Conversation. In *Syntax and Semantics. 3: Speech Acts,* ed. P. Cole and J. Morgan, 41–58. New York: Academic Press.

Hutchins, E. 1995. *Cognition in the Wild.* Cambridge, MA: The MIT Press.

Kelly, J., and Hamm, S. 2013. *Smart Machines: IBM's Watson and the Era of Cognitive Computing.* New York: Columbia University Press.

Krum, D.; Omoteso, O.; Ribarsky, W.; Starner, T.; and Hodges, L. 2002. Speech and Gesture Multimodal Control of a Whole Earth 3D Visualization Environment. In *Proceedings of the 2002 Joint Eurographics and IEEE TCVG Symposium on Visualization,* 195-200. Goslar, Germany: Eurographics Association.

Laird, J.; Newell, A.; and Rosenbloom, P. 1987. SOAR: An Architecture for General Intelligence. *Artificial Intelligence* 33(1): 1–64. dx.doi.org/10.1016/0004-3702(87)90050-6

Langley, P.; Laird, J. E.; and Rogers, S. 2009. Cognitive Architectures: Research Issues and Challenges. *Cognitive Systems Research* 10(2): 141–160. dx.doi.org/10.1016/j.cogsys.2006.07.004

Leake, D. B. 1996. *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. Menlo Park, CA: AAAI Press.

Licklider, J. C. R. 1960. Man-Computer Symbiosis. *IRE Transactions on Human Factors in Electronics* Volume HFE-1(1): 4–11. www.dx.doi.org/10.1109/THFE2.1960.4503259

McCord, M. C.; Murdock, J. W.; and Boguraev, B. K. 2012. Deep Parsing in Watson. *IBM Journal of Research and Development* 56(3.4): 3:1–3:15.

Minsky, M. 1988. *The Society of Mind.* New York: Simon and Schuster.

Modha, D. S.; Ananthanarayanan, R.; Esser, S. K.; Ndirango, A.; Sherbondy, A. J.; and Singh, R. 2011. Cognitive Computing. *Communications of the ACM* 54(8): 62–71. dx.doi.org/10.1145/1978542.1978559

Nardi, B. A. 1996. Activity Theory and Human-Computer Interaction. In *Context and Consciousness: Activity Theory and Human-Computer Interaction,* ed. B. Nard, 7–16. Cambridge, MA: The MIT Press.

Oviatt, S.; and Cohen, P. 2000. Perceptual User Interfaces: Multimodal Interfaces that Process what Comes Naturally. *Communications of the ACM* 43(3): 45-53. dx.doi.org/10.1145/330534.330538Ramos, C.; Marreiros, G.; Santos, R.; and Freitas, C. F. 2010. Smart Offices and Intelligent Decision Rooms. In *Handbook of Ambient Intelligence and Smart Environments,* ed. H. Nakashima, H. Aghajan, and J. C. Augusto, 851–880. Berlin: Springer. dx.doi.org/10.1007/978-0-387-93808-0_32

Romano, N. C., and Nunamaker, J. F. 2001. Meeting Analysis: Findings from Research and Practice. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*. Los Alamitos, CA: IEEE Computer Society.

Sharma, R.; Yeasin, M.; Krahnstoever, N.; Rauschert, I.; Cai, G.; Brewer, I.; MacEachren, A.; Sengupta, K. 2003. Speech-Gesture Driven Multimodal Interfaces for Crisis Management. Proceedings of the IEEE 91(9): 1327–1354. dx.doi.org/10.1109/JPROC.2003.817145

Shrobe, H.; Coen, M.; Wilson, K.; Weisman, L.; Thomas, K.; Groh, M.; Phillips, B.; Peters, S.; Warshawsky, N.; and Finin, P. 2001. The Intelligent Room. MIT AI Laboratory AFRL-IF-RS-TR-2001-168 Final Technical Report. Rome, New York: Air Force Research Laboratory.

Soltau, H.; Saon, G.; and Kingsbury, B. 2010. The IBM Attila Speech Recognition Toolkit. In 2010 IEEE Workshop on Spoken Language Technology, SLT 2010 — Proceedings, 97–102. Piscataway, NJ: Institute for Electrical and Electronics Engineers. dx.doi.org/10.1109/slt.2010.5700829

Stanford, V.; Garofolo, J.; Galibert, O.; Michel, M; and Laprun, C. 2003. The (NIST) Smart Space and Meeting Room Projects: Signals, Acquisition Annotation, and Metrics. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '03),* 4, 6–10. Piscataway, NJ: Institute for Electrical and Electronics Engineers. dx.doi.org/10.1109/icassp.2003.1202748

Voss, L. L., and Ehlen, P. 2007. The CALO Meeting Assistant. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. Stroudsberg, PA: Association for Computational Linguistics 2007. dx.doi.org/10.3115/1614164.1614173

Walsh, W. E.; Tesauro, G.; Kephart, J. O.; and Das, R. 2004. Utility Functions in Autonomic Systems. In *Proceedings of the 1st International Conference on Autonomic Computing (ICAC 2004),* 70–77. Piscataway, NJ: Institute for Electrical and Electronics Engineers. dx.doi.org/10.1109/ICAC.2004.1301349

Weiser, M., and Brown, J. S. 1996. Designing Calm Technology. *PowerGrid Journal* 1.01, 1–5.

Robert Farrell is a research staff member at the IBM T. J. Watson Research Center in Yorktown Heights, NY, USA. He has a long-term research interest in the cognitive processes of human learning, knowledge representation, reasoning, and language understanding. His past work includes cognitive models, intelligent tutoring systems, and social computing applications. He is currently working on software to extract knowledge from unstructured information sources.

Jonathan Lenchner is chief scientist at IBM Research-Africa. Previously he was one of the founders of the IBM Cognitive Environments Lab in Yorktown Heights, NY. His research interests include computational geometry, robotics, AI, and game theory. His recent work includes research on humanoid robots and development of an immersive environment to help a professional sports team with trades and draft picks.

Jeffrey Kephart is a distinguished research staff member at IBM T. J. Watson Research Center, and a Fellow of the IEEE. He is known for his work on computer virus epidemiology and immune systems, self-managing computing systems, electronic commerce, and data center energy management. Presently, he serves as a principal investigator on a cognitive computing research project with a large energy company and leads work on applying intelligent agent technologies to corporate mergers and acquisitions.

Alan Webb is a senior software engineer at the IBM T. J. Watson Research Center. His present research interests are focused upon applying the principles of distributed cognition as an inspiration for pervasive cognitive environments. He is currently working on a generalized system architecture for the cognitive environment and development of the mergers and acquisitions application.

Michael Muller is a research staff member in the Cognitive User Experience group at IBM Research in Cambridge, MA. His research areas have included collaboration in health care, metrics and analytics for enterprise social software, participatory design, and organizational crowdfunding. His current work focuses on employee experiences in the workplace.

Thomas Erickson is a social scientist and interaction designer at the IBM T. J. Watson Research Center. His research has to do with designing systems that enable groups of people to interact coherently and productively in both virtual and real environments.

David Melville is a research staff member at IBM T. J. Watson Research Center. His research interests include immersive data spaces, spatial computing, adaptive physical architecture, and symbiotic experience design.

Rachel Bellamy is a principal research staff member and group manager at IBM T. J. Watson Research Center and heads the Research Design Center. Her general area of research is human-computer interaction and her current work focuses on the user experience of symbiotic cognitive computing.

Daniel Gruen is a cognitive scientist in the Cognitive User Experience group at IBM Research in Cambridge, MA. He is interested in the design of systems that let strategic decision makers seamlessly incorporate insights from cognitive computing in their ongoing deliberations and creative thinking. He is currently working with a variety of companies to understand how such systems could enhance the work they do.

Jonathan Connell is a research staff member at IBM T. J. Watson Research Center. His research interests include computer vision, machine learning, natural language, robotics, and biometrics. He is currently working on a speech-driven reactive reasoning system for multimodal instructional dialogue.

Danny Soroker is a research staff member at IBM T. J. Watson Research Center. His research interests include intelligent computation, human-computer interaction, algorithms, visualization, and software design. He is currently working on agents to support problem solving and decision making for corporate mergers and acquisitions.

Andy Aaron is a research staff member at IBM T. J. Watson Research Center. He was on the speech team for the IBM Watson Jeopardy! match. Along with his work in speech synthesis and speech recognition, he has done sound design for feature films and produced and directed TV and films.

Shari Trewin is a research staff member at IBM T. J. Watson Research Center. Her current research interests include multimodal human-computer interaction and accessibility of computer systems. She is currently working on knowledge extraction from scientific literature and interaction designs for professionals working with this extracted knowledge.

Maryam Ashoori is a design researcher at IBM T. J. Watson Research Center. She has a passion for exploring the intersection between art and computer science. Her work has resulted in several novel applications for the Cognitive Environments Laboratory, including a ”Zen Garden” for unwinding after a long day and a service for sparking creativity for inventors.

Jason Ellis is a research staff member at IBM T. J. Watson Research Center. His research interests include social computing and usability. He is currently working on collaborative user interfaces for cognitive systems.

Brian Gaucher is senior manager of the Cognitive Environments Laboratory at IBM T. J. Watson Research Center. He leads teams specializing in user experience design and physical infrastructure of cognitive computing environments. His work focuses on the creation of highly interactive physical spaces designed to improve decision making through always-on ambient intelligence.

Dario Gil is the vice president of science and technology for IBM Research. As director of the Symbiotic Cognitive Systems department, he brought together researchers in artificial intelligence, multiagent systems, robotics, machine vision, natural language processing, speech technologies, human-computer interaction, social computing, user experience, and interaction design to create symbiotic cognitive computing technology, services, and applications for business.

Remembering Marvin Minsky

Kenneth D. Forbus, Benjamin Kuipers,
Henry Lieberman

■ *Marvin Minsky, one of the pioneers of artificial intelligence and a renowned mathematicial and computer scientist, died on Sunday, 24 January 2016, of a cerebral hemmorhage. He was 88. In this article, AI scientists Kenneth D. Forbus (Northwestern University), Benjamin Kuipers (University of Michigan), and Henry Lieberman (Massachusetts Institute of Technology) recall their interactions with Minksy and briefly recount the impact he had on their lives and their research. A remembrance of Marvin Minsky was held at the AAAI Spring Symposium at Stanford University on March 22. Video remembrances of Minsky by Danny Bobrow, Benjamin Kuipers, Ray Kurzweil, Richard Waldinger, and others can be on the sentient webpage*[1] *or on youtube.com.*

The photographs in this article were taken at the AAAI-05 conference in Pittsburgh, Pennsylvania.

Kenneth D. Forbus

In his 1960 essay, Steps Toward Artificial Intelligence, Marvin wrote:

> A visitor to our planet might be puzzled about the role of computers in our technology. On the one hand, he would read and hear all about wonderful "mechanical brains" baffling their creators with prodigious intellectual performance. And he (or it) would be warned that these machines must be restrained, lest they overwhelm us by might, persuasion, or even by the revelation of truths too terrible to be borne. On the other hand, our visitor would find the machines being denounced on all sides for their slavish obedience, unimaginative literal interpretations, and incapacity for innovation or initiative; in short, for their inhuman dullness.

Similar conversations are being held today. But the balance has shifted. In the 1960s through the 1980s many philosophers and physicists made public comments that AI was impossible, and its seeming achievements illusory. Today, of course, physicists and philosophers are more likely to issue dire warnings about the dangers of AI. That shows just how far this young scientific enterprise has come since its founding, by Marvin Minsky, John McCarthy, Allen Newell, Herb Simon, and others, only 60 years ago.

Marvin was a cognitive scientist before the term was invented. He looked to psychology, neuroscience, and biology for clues to how minds worked, pulling together disparate ideas through the lens of computation. While all of his work has been influential, his final book, *The Emotion Machine,* provides a grand synthesis that is worthwhile reading for anyone interested in AI.

In popular histories, Marvin has often been caricatured as an enemy of neural models. Nothing could be further from

the truth. His interest in modeling brains as well as minds was clear to all who knew him. For example, it was Marvin who invited a freshly minted Ph.D., David Marr, whose thesis was a mathematical model of the cerebellum, to join the AI Lab as a research scientist. What Marvin could not abide were people who did not clearly understand the limitations of their models. Minsky and Papert's analysis of perceptrons provided needed clarity, delineating important theoretical limitations on them. In his preface to the second edition of *Perceptrons,* Marvin pointed out that Rumelhart and McClelland's own data showed that their networks could not learn exclusive-or without exponential amounts of data, as he predicted. Similarly, Marvin was impatient with the primitive neuroscience techniques such as single-cell recordings available for much of his life. The widespread adoption of imaging techniques by neuroscientists suggests that many modern researchers agree as to the power of analyzing larger-scale neural systems.

I was lucky enough to start working at the MIT AI Lab from the second week of my freshman year at MIT in 1973, through the end of my Ph.D. work in 1984. Staying at the same institution was more common in those days. AI labs were very few in number — most universities didn't have them. Such labs had access to resources, like computers and the ARPANET, that most students didn't have access to, even on the MIT campus. Marvin had created a vibrant, thriving laboratory. He loved to play with ideas, with anyone who was interested. He could often be found in the Playroom, a large open space, riffing about AI or just about any other aspect of science or engineering (or science fiction).

Like many brilliant people, Marvin could be eccentric. For example, his basement was legendary for being a kind of Sargasso Sea of papers and artifacts. In fact, I saw a talk by a historian in the Playroom on the history of Lisp that used it as a resource. The historian dated events in Lisp's creation by treating Marvin's basement as an archeological dig site, dating documents based on their position within layers of other papers. Marvin's desk was similarly cluttered. This was a challenge for students who needed to get a copy of their thesis to him to read, since the document would quickly vanish into the morass. (People didn't read on screen back then, there weren't enough computers around.) Students came up with various ways to ensure that their document attracted Marvin's attention. My technique was to use a telephone cord to hang the thesis from the ceiling so that it dangled over the middle of his desk, at eye level, bobbing gently. But his comments, once you got his attention, were always insightful.

Artificial intelligence has come a long way in just 60 years. The scientific and engineering triumphs

have already changed our lives in many ways, and this process will continue for the foreseeable future. And in doing so, we will continue that voyage that Marvin and others started, to understand minds by building them.

Benjamin Kuipers

Marvin Minsky was my advisor, and he was very important in my life.

My story starts back when I was in high school, and my dad and I would take long walks, talking about math, science, and the nature of the mind. We agreed that this was one of the great scientific problems of all time. I couldn't wait to get to college, so I could take a psychology course, and learn about the science of the mind.

Well, I got to college. And I took a psychology course. And it was a crashing disappointment. The interesting parts weren't rigorous, and the rigorous parts weren't interesting. So I decided I had guessed wrong about psychology, and majored in mathematics. After college, I went to MIT for graduate school in pure math. I was planning to be pure as the driven snow, committed to crystalline mathematical beauty.

But in the spring of 1973, just as a lark, I took the graduate Introduction to Artificial Intelligence course offered by Marvin Minsky and Seymour Papert. The skies opened! I realized that this was what I had been looking for all along! Their computational methods demonstrated ways to model interesting and important properties of the mind. Like calculus and differential equations had transformed physics more than three centuries before, symbolic knowledge representation and inference methods were beginning a historical transformation of the science of the mind. Like calculus and differential equations had grown and evolved over centuries to become more and more powerful tools for doing physics (and so much else), I knew that these methods would grow and evolve over the decades, and perhaps the centuries, to become powerful enough to describe the mind. I knew that this was what I wanted to spend my life on.

I became a phantom grad student in the Math Department, with a picture on the bulletin board in Building 2, but physically at home in the AI Lab in Tech Square. I was inspired by Marvin's essays, by his students' theses in *Semantic Information Processing,* by class meetings in the iconic Minsky living room, and by discussions in the AI Lab Playroom. Most of all, I was inspired by circulating drafts of the famous "frames paper." There, in a precursor to *The Society of Mind,* Marvin laid out his vision for the organization of knowledge as rich descriptions of complex objects,

situations, and events, rather than at the finer granularity of logical sentences.

Marvin agreed to be my thesis advisor, as I undertook a project to understand knowledge of space, specifically the large-scale space of the "cognitive map," describing the structure of buildings and cities at a larger scale than can be observed all at once. He was a wonderful doctoral advisor for me, respecting my ideas, and giving me full freedom to pursue them. My priorities and my direction were deeply inspired by his thinking, but he never pushed me to do things his way. Someone quoted his advice on advising students: "Make sure the students believe that all the good ideas are their own."

Every month or so, as I worked on my thesis research, I would sit down in his office and tell him, "I think I need some advising." He would ask what I was up to, and I would explain my progress. Then he would tell me what seemed like a completely random story, like a time that he and Seymour got lost in Buenos Aires, while they were there for a conference. I would leave thinking, "What was that all about?" Then I would think about it, and think about it, and eventually, I would realize that there was a really great idea hidden in there, and it would contribute to my thesis. But I never knew where the idea came from.

At a rough time in my career, he was an enormous help, and the situation resolved very much for the better. I am very grateful, and I try to pay it forward. He also gave brief and pointed praise, that I still treasure, for coming up with an idea that he really liked.

Marvin had many different accomplishments, but chief among them was being a founder of the field of AI. He and his colleagues started the process of creating the tools and ideas that are revolutionizing the science of the mind. He was disappointed that we haven't come closer to achieving the goals of AI during his lifetime.

Centuries ago, Newton and Leibniz created tools and ideas that revolutionized the problem of physics, and the work of understanding physics is still not done. Likewise, the problem of the mind is a problem for the centuries, not merely for the decades. Marvin's contributions have made lasting changes to our understanding of the problem of the mind.

Henry Lieberman

Whenever we encountered each other, Marvin's eyes would light up, he'd smile, and, instead of "Hello" or "How are you?", his greeting was always: "What's new?" I'd tell him what was new with me, or some topic I was thinking about. He'd always react in a way that was surprising, amusing, and profound.

Marvin would plant time bombs in your head.

He'd say something, and you would have a hard time deciding, was that serious? What did that mean? Did he really believe that? What kind of point was he trying to make? Three months later, you'd be walking down the street, and then suddenly it would hit you. "Wait a second! Why *do* we like fun?"

Marvin was a no-nonsense person. He didn't care about money, power, or status. Many times, we would visit some university where he'd be fawned over by the tenured professors. He'd walk right past them, find a grad student in front of a computer and cold coffee and cold pizza. He'd tap the student on the shoulder and say, "What are you thinking about?"

He didn't even really care about computers. He cared about understanding how the mind works. If you wanted to think about it with him, he always had time for you. The human mind is the most complicated thing in existence. What could be more fun than trying to figure out how it works?

But what's the right level to describe it? I think the answer to that question was Marvin's greatest contribution to computer science — and to psychology.

We're just bags of chemicals. Can intelligence be explained at the chemical level? We're just strings of neurons. We can map the 302 neurons of *C. elegans,* but does that tell us how the worm thinks? The mind must work by electrical impulses flowing around the brain. Will looking at the signals tell us how it works? Different parts of the brain do different things. Does the geography of the brain explain thinking? Other people will tell you the mind is all math, is all biology, is all social interaction. And they'd be right each time.

The brain is hardware. The mind is the software it runs. Marvin explained the workings of the mind as components of software in a high-level programming language. What we think of as concepts can be reified as knowledge representations, as Marvin taught us in the Frame paper. The activity of thinking can be represented as heuristic procedures—trans-frames, micronemes, ways to think, K-lines, A-brains, and B-brains.

Marvin wasn't afraid to push people's hot buttons, especially concerning those aspects of human thought that people cherished, but obstinately refused to try to explain: consciousness, emotion, religion.

Marvin worked in every aspect of AI, from neural nets to cognitive science. The controversies that ravage the field are often just silly arguments between top-down and bottom-up approaches. Marvin advocated coexistence among reactive, reflective, and deliberate layers. What Marvin thought was important was the architecture for managing what he called the Society of Mind, showing how components of the mind both cooperate and compete. His Causal-Diversity Matrix classified the diversity of methods according to what they were good for.

I was once at a graduate seminar with Marvin, and everybody went around the room introducing themselves. "Hi, I'm Robert, I'm a first-year grad student," "Hi, I'm Rebecca, I'm a second year grad student," and so on. Then Marvin's turn: "Hi, I'm Marvin. I'm a 60th year grad student"!

I can only claim to be a 40th year grad student myself. Right now, I'm trying to help launch the Minsky Center for Artifical Intelligence at MIT's CSAIL, the modern incarnation of the Lab he founded. My fondest hope, and the best way to honor his memory, is to try to help create more 60-year graduate students.

Note

1. www.sentient.ai/minsky.

Kenneth D. Forbus is the Walter P. Murphy professor of computer science and a professor of education at Northwestern University.

Benjamin Kuipers is a professor of computer science and engineering at the University of Michigan.

Henry Lieberman is a research scientist at the Massachusetts Institute of Technology's Computer Science and Artificial Intelligence Lab (CSAIL).

Reports of the 2016 AAAI Workshop Program

Stefano V. Albrecht, Bruno Bouchard, John S. Brownstein, David L. Buckeridge, Cornelia Caragea, Kevin M. Carter, Adnan Darwiche, Blaz Fortuna, Yannick Francillette, Sébastien Gaboury, C. Lee Giles, Marko Grobelnik, Estevam Hruschka, Jeffrey O. Kephart, Parisa Kordjamshidi, Viliam Lisy, Daniele Magazzeni, Joao Marques-Silva, Pierre Marquis, David Martinez, Martin Michalowski, Zeinab Noorian, Enrico Pontelli, Alex Rogers, Stephanie Rosenthal, Dan Roth, Scott Sanner, Arash Shaban-Nejad, Arunesh Sinha, Tran Cao Son, William Streilein, Sylvie Thiebaux, Byron C. Wallace, Toby Walsh, Michael Witbroc, Jie Zhang

■ *The Workshop Program of the Association for the Advancement of Artificial Intelligence's Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16) was held at the beginning of the conference, February 12–13, 2016. Workshop participants met and discussed issues with a selected focus, and the workshop provided an informal setting for active exchange among researchers, developers, and users on topics of current interest. The AAAI-16 workshops were an excellent forum for exploring emerging approaches and task areas, for bridging the gaps between AI and other fields or between subfields of AI, for elucidating the results of exploratory research, or for critiquing existing approaches. The 15 workshops held at AAAI-16 were Artificial Intelligence Applied to Assistive Technologies and Smart Environments (WS-16-01), AI, Ethics, and Society (WS-16-02), Artificial Intelligence for Cyber Security (WS-16-03), Artificial Intelligence for Smart Grids and Smart Buildings (WS-16-04), Beyond NP (WS-16-05), Computer Poker and Imperfect Information Games (WS-16-06), Declarative Learning Based Programming (WS-16-07), Expanding the Boundaries of Health Informatics Using AI (WS-16-08), Incentives and Trust in Electronic Communities (WS-16-09), Knowledge Extraction from Text (WS-16-10), Multiagent Interaction Without Prior Coordination (WS-16-11), Planning for Hybrid Systems (WS-16-12), Scholarly Big Data: AI Perspectives, Challenges, and Ideas (WS-16-13), Symbiotic Cognitive Systems (WS-16-14), and World Wide Web and Population Health Intelligence (WS-16-15).*

Artificial Intelligence Applied to Assistive Technologies and Smart Environments

Smart environments have the potential to enhance the quality of life of people by giving assistance in the activities of daily life. These systems are particularly interesting for impaired and frail people because they can improve their autonomy and reduce the need for caregivers. The development of technologies to improve the building of assistive systems has increased this last decade. However, despite this growing interest toward these technologies, no real adoption tendency has been observed yet. Indeed, impairments and particularities of users are so diverse that implementing solutions that are mandatory for users' well-being represent one of the major challengees in terms of universal design. The goal of this workshop was to investigate new solutions to scientific problems occurring in various topics related to artificial intelligence applied in the domain of impaired people assistance.

The 2016 AAAI workshop brought together academic and industrial researchers from several subfields of AI. One main theme of papers presented at the workshop was activity recognition. In fact, this theme is one major challenge in providing good assistance. Indeed, assistive systems need to get accurate information about monitored people. This can be considered as the first step. Several papers provided models and analysis about performing an efficiency detection from different kinds of data.

Another major theme was actions performed by assistive devices. We can divide the papers on this theme into three categories. Some papers presented methods to improve the efficiency of the decision process. Others focused on adaptation of actions to address the needs of users. Finally, some papers studied the ethical issues in autonomous systems.

The workshop participants discussed how assistive technologies can benefit from innovation in domains such as the internet of things. However, participants shared the objective of developing methods to reduce the costs and increase the efficiency of assistive technology and agreed that they would like to attend a future edition of the workshop.

Bruno Bouchard and Sébastien Gaboury served as cochairs of this workshop. The papers of the workshop were published as technical report WS-16-01 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence*.

AI, Ethics, and Society

This workshop focused on the ethical and societal implications of building AI systems. It is a response to the increasing appetite from both within and outside the AI research community for such discussions.

The topics addressed within the workshop included the future of AI; AI as a threat to or savior for humanity; mechanisms to ensure moral behaviors in AI systems; safeguards necessary within AI research; and the impact of AI on work and other aspects of our lives. The workshop was divided into two parts. In the morning, there were seven contributed talks and six posters. The titles of the talks give a good idea of the range of topics covered: Human-like Morality and Ethics for Robots; Patience Is Not a Virtue: AI and the Design of Ethical Systems; Quantilizers: A Safer Alternative to Maximizers for Limited Optimization; Why the Technological Singularity May Never Happen; Modeling Progress in AI; Taxonomy of Pathways to Dangerous Artificial Intelligence; and Reinforcement Learning as a Framework for Ethical Decision Making.

In the afternoon, there were a dozen or so short talks from recipients of the Future of Life grants on AI safety. These grants are funded by a $10 million donation from Elon Musk to promote safe and beneficial AI research. The workshop finished with a panel discussion on the most promising research directions for keeping AI beneficial. The panelists were Stuart Russell, Vincent Conitzer, David Parkes, Percy Liang, Stefano Ermon, and Benjamin Rubinstein.

Special thanks go to Richard Mallah who, in his capacity as part of the organizing committee, put together the program for the afternoon. Toby Walsh served as chair of the workshop. The papers of the workshop were published as technical report WS-16-02 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence* (Palo Alto, CA: AAAI Press).

Artificial Intelligence for Cyber Security

This workshop focused on research and applications of artificial intelligence to cyber security, including machine learning, game theory, natural language processing, knowledge representation, and automated and assistive reasoning. Talks throughout the day emphasized cyber systems and research on techniques to enable resilience in cyber security systems augmented by human-machine interactions. The workshop began with a keynote speech by George Cybenko (Dartmouth College) on Cyber Security Challenges Amenable to AI. Cybenko suggested a need to address the AI challenges across the full spectrum, including deterrence, protection, intrusion detection, adaptation, and recovery. AI can help, since the growth in cyber vulnerabilities is, in many cases, exploiting user behavior (for example, phishing as an entry point). The role of AI is in predicting patterns of behavior to address potential vulnerabilities.

On the topic of malware, papers presented included techniques employing semantic meaning to identify clusters of malware families, showing good true positives versus low false positives in identifying clusters by using integer linear programming. Additionally, an approach was presented using Bayesian networks to determine malware lineage present in a directed acyclic graph.

Two papers followed focusing on human-machine interaction (HMI). One paper integrated prior knowledge of actor behavior and leveraged machine-learning techniques to work on a reduced space. This enabled the application of rules to represent the actor behavior and better accuracy in final cyber attribution. The second paper leveraged the Levenshtein distance to measure cognitive burden during password creation, usage, and recollection, in order to develop strong password recommendation tools.

There were three position papers within the workshop that were shorter presentations on a topic of interest, including the need to augment cyber security tools by incorporating learning from human behavior; recommendations for a multiarmed bandit approach to quickly structure and configure honeypots; and using game theory for incentivizing participants in sharing information while incorporating timely updates to improve the incentives.

The afternoon keynote from Robert Laddaga (Vanderbilt University) was titled "AI and the Future of Cyber Security." Laddaga argued that the use of AI technologies in cyber security will inevitably improve both defenders and attackers. He stressed the need for active methods to detect adversary activity, protect cyber systems, and that sensors should be deployed on systems themselves in addition to the network. He remarked that more needs to be done and called for terrain shaping, in which the cyber landscape is altered strategically to disadvantage attackers, not just to confuse them. Laddaga closed with a reminder that the need for AI to protect systems will increase as the cost of damage will move from data and machine damage to loss of human life.

The afternoon session of the workshop featured two interesting talks related to AI and its impact on cyber security. The first presented a call for a unified cyber ontology to help with information sharing. Such a mechanism represents concepts to share information automatically and could support deeper analytical investigation. The second talk presented a new model for effective cyber defense based upon active perception — deploying new sensors to understand the environment in the wake of a cyber event — to test hypotheses and reduce uncertainty prior to alerting an analyst.

The workshop concluded with a panel discussion on the subject AI and Cyber Operations: Challenges of Community Acceptance, including panelists Richard Lippmann (MIT Lincoln Laboratory), Vern Paxson (University of California at Berkeley), Benjamin Rubinstein (University of Melbourne), and Milind Tambe (University of Southern California). Panelists discussed the need to leverage natural language processing, game theory, and other AI capabilities as a toolbox within the specified domain of cyber security. The biggest challenge identified was that cyber is a rapidly changing domain, and it is extremely difficult to model the nonstationarity. Paxson stressed that his views on the limitations of machine learning in cyber security were specific to intrusion detection, which suffers from these challenges. Final conclusions from the panel focused on the need to leverage AI as a force multiplier and decision support tool for cyber defenders, focusing on human-in-the-loop rather than full automation.

This was the first AI for Cyber Security workshop. It was cochaired by David Martinez, William Streilein, Kevin M. Carter, and Arunesh Sinha, who also wrote this report. The papers of the workshop were published as technical report WS-16-03 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence* (Palo Alto, CA: AAAI Press).

Artificial Intelligence for Smart Grids and Smart Buildings

The proliferation of intelligent devices and the availability of electric monitoring facilities, broadband communication networks, computational intelligence, and customer-driven electricity storage and generation capabilities have posed the foundations for the next generation of power grids and buildings: smart grids and smart buildings. Three key aspects distinguish the smart grid from the more traditional electric grid: (1) producers and consumers have access to information (for example, production costs, customers' electricity needs, time distribution of demands); (2) continuous access to information and communication is possible (for example, producers and consumers can negotiate prices); and (3) energy can be produced not only by power plants, but also by customers (for example, through renewable sources, which can be intermittent) and stored for later use (or redistributed through the electric grid). In general terms, a smart grid enables the distributed generation and two-directional flow of electricity, within an integrated system.

Smart buildings form an important component of the smart grid, where technology enables buildings to provide common services (for example, illumination, thermal comfort, air quality, sanitation) in a sustainable fashion and at low environmental impact. AI plays a key role in the smart grid and in smart buildings; the infrastructure provides information to support automated decision making on how to autonomously adapt production and consumption of energy, optimize costs, waste, and environmental impact, and ensure safe, secure, and efficient operation.

The goal of this workshop was to bring together researchers and practitioners from different areas of AI, to explore both established and novel applications of AI techniques to solve problems related to the design, implementation, and deployment of smart grids and smart buildings.

The workshop was well attended and conducive of extensive discussions and interactions between speakers and the audience. The workshop was opened by a team-delivered invited presentation by Mario Berges (Carnegie Mellon University) and Henning Lange (Aalto University), exploring the role of AI in solving problems related to electricity disaggregation — that is, the problem of providing estimates of the consumption of individual electrical appliances in a building from measurements of voltage and/or current at select locations in the facility. This initial presentation was followed by a related contributed presentation, by the team from the University of Minnesota (Mark Valovage and Maria Gini), that focused on the use of label-correction techniques and prioritization methods in enhancing classification of individual appliances in a household during electricity disaggregation.

After a short break, the workshop continued with three contributed talks focused on the use of planning and optimization techniques in energy management, ranging from the use of planning with uncertainty to handle electric vehicles, to the use of MDP techniques in building agents for the Power TAC competition, to the exploration of how distributed constraint optimization can be adapted to handle the dynamicity of energy delivery models.

The afternoon session placed emphasis on the role of machine learning and classification in handling smart grids and smart buildings, with particular emphasis on reduced consumption prediction (in the presence of demand response events), prediction of adverse events and modeling of energy consumption in wireless sensor networks. The final session of the workshop explored personal preferences and individual comfort, with applications to support smart house buying (accounting for energy costs) and prediction of thermal comfort in smart buildings.

The workshop was cochaired by Enrico Pontelli and Tran Cao Son (New Mexico State University), Alex Rogers (University of Oxford), and Sylvie Thiebaux (NICTA and Australian National University), who also coauthored this report. The papers of the workshop were published as technical report WS-16-04 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence* (Palo Alto, CA: AAAI Press).

Beyond NP

A new computational paradigm has emerged in computer science over the past few decades, which is exemplified by the use of SAT solvers to tackle problems in the complexity class NP. According to this paradigm, a significant research and engineering investment is made toward developing highly efficient solvers for a prototypical problem (for example, SAT), that is representative of a broader class of problems (for example, NP). The cost of this investment is then amortized as these solvers are applied to a broader class of problems through reductions (in contrast to developing dedicated algorithms for each encountered problem). SAT solvers, for example, are now routinely used to solve problems in many domains, including diagnosis, planning, and software and hardware verification.

Motivated by the success of this computational paradigm, both in theory and practice, the goal of this workshop was to consolidate and promote research that advances this paradigm more broadly, while focusing on solvers that reach beyond NP. The workshop brought together researchers that work on a variety of problems in beyond NP complexity classes and corresponding solvers, including propositional and first-order probabilistic reasoning, planning, quantified Boolean formulas, function and optimization problems such as Max-SAT, knowledge compilation, and model counting. The workshop included six invited talks that provided a perspective on a wide spectrum of beyond NP solvers and problems. It also included 15 papers on these subjects, ranging from ones that included new technical contributions to ones that provided surveys and perspectives on the state of the art.

A number of themes emerged throughout the workshop and its talks. One theme concerned the status of competitions and evaluations for beyond NP solvers, the existence of standard formats and benchmarks for beyond NP problems, and the impact this may have on future progress. Another theme was the extent to which solvers for a particular problem have converged on a standard approach just like SAT solvers have converged on a standard approach for solving satisfiability. A major theme of the workshop related to the two facets of the PSPACE complexity class, which are exemplified by the two prototypical problems corresponding to quantified Boolean formulas and planning.

The workshop participants discussed some next steps for advancing the reduce-then-solve computational paradigm targeted by the workshop. This included a categorization and explanation of prototypical problems that are complete for various beyond NP complexity classes, including the PP, NP^{PP}, and PP^{PP} complexity classes, which include key problems that arise in probabilistic reasoning. There was recognition that more community awareness is needed about beyond NP complexity classes and that existing and well-researched problems fall into these classes. It was also noted that the practice of reducing problems, which is now prevalent for problems in NP, can benefit from some tutorials and illustrative examples relating to beyond NP problems, especially ones that are PSPACE-complete. A third discussion point related to establishing standard formats and benchmarks for the more recent beyond NP solvers, including those for first-order model counting and knowledge compilation. The suggestion was to implement these next steps by augmenting the BeyondNP.org community website with additional material to serve these objectives.

Adnan Darwiche, Joao Marques-Silva, and Pierre Marquis served as cochairs of this workshop and wrote this report. The papers of the workshop were published as technical report WS-16-05 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence* (Palo Alto, CA: AAAI Press). The slides for invited talks are posted at the workshop website (beyondnp.org/workshop16).

Computer Poker and Imperfect Information Games

Recent years have brought substantial progress in research on imperfect information games. There is an active community of researchers focusing on computer poker, which recently computed near-optimal strategy for the smallest poker variant commonly played by people and achieved human-level performance even in more complex variants of this game. Game-theoretic models with all sorts of uncertainty and imperfect information have been applied in security domains ranging from protecting critical infrastructure and wildlife to cyber security. Computer agents able to play previously unknown imperfect information games based only on a formal description of a game's dynamics have been developed.

In this workshop, we aimed to create a forum where researchers studying theoretical and practical aspects of imperfect information games can meet, present their recent results, and discuss their new ideas. Moreover, we tried to facilitate interaction between distinct communities studying various aspect and focusing on various domains in imperfect information games.

The workshop program was composed of 12 technical paper presentations, announcement of the results of the Annual Computer Poker Competition (ACPC), and two discussion sessions. The presentations covered topics from computing Nash and Stackelberg equilibria in large extensive form games and automated creation of game abstractions through opponent modeling and exploitation, heuristic search methods for imperfect information games, and taxonomy of different subclasses of imperfect information games, to applications of game theory in urban crime prevention and wildlife protection.

The winner of the total bankroll part of the 2016 computer poker competition was a program by Noam Brown and Tuomas Sandholm from Carnegie Mellon University. The winner of the bankroll instant run-off was implemented by an independent researcher, Eric Jackson.

The first discussion was on the future of the workshop. The attendees agreed that they are most interested in running a full-scale, six player No-Limit Texas Hold'em competition and that the total bankroll track of the competition should be modified to better motivate creating adaptive agents. Furthermore, the present competitors agreed that they do not mind using complicated variance reduction techniques in the evaluation of the algorithms and that it is important to attract more competitors to the competition. Interesting options to do that would be to include a track with a very small limit for the size of the agent, or a track that cannot by entered by anyone who has already participated in multiple past competitions.

The topic of the second discussion was how to make the communities of researchers in poker, search in games, general game playing, and security games collaborate more efficiently. Everybody agreed that the problems the communities try to solve have many similar aspects and that they would benefit from closer interaction. The main drawback is that all the communities are quite productive and use slightly different terminology; therefore, it is difficult to follow the latest developments in the related fields. A proposed solution is to repeat events similar to this workshop that bring the communities together, and to present intensive advanced tutorials on the individual subfields. Unlike traditional tutorials, these could assume basic knowledge of key concepts of game theory and focus on the most important recent results and key challenges that the subfields face.

Viliam Lisy, Michael Thielscher, and Thanh Nguyen served as cochairs of this workshop. This report was written by Viliam Lisy. The papers of the workshop were published as technical report WS-16-06 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence* (Palo Alto, CA: AAAI Press).

Declarative Learning Based Programming

Declarative Learning Based Programming (DeLBP) aims at facilitating and simplifying the design and development of intelligent real-world applications that use machine learning and reasoning by addressing the following commonly observed challenges: interacting with messy, naturally occurring data; specifying the requirements of the application at a high abstraction level; dealing with uncertainty in data and knowledge in various layers of the application program; using representations that support flexible relational feature engineering; using representations that support flexible reasoning and structure learning; integrating a range of learning and inference algorithms; and finally addressing these issues in one unified programming environment.

Conventional programming languages offer no help to application programmers that attempt to design and develop applications that make use of real-world data and reason about it in a way that involves learning interdependent concepts from data, incorporating existing models, and reasoning about existing and trained models and their parametrization. Over the last few years the research community has tried to address these problems from multiple perspectives, most notably various approaches based on probabilistic programming, logical programming, and the integrated paradigms. The goal of this workshop was to present and discuss the current lines or research in these directions and the ways various challenges have been addressed. We attempted to motivate the need for further research toward a unified framework in this area, building on some of the key existing paradigms, including those of probabilistic programing (PP), probabilistic logical programming (PLP), database management systems (DBMS), and statistical relational learning (SRL), and place these ideas in the context of learning based programming.

This workshop brought together researchers from the areas of probabilistic logical programming, statistical relational learning, lifted inference, constraint programming, probabilistic programming, and data mining. The workshop started with an introductory talk given by Dan Roth that highlighted different perspectives and avenues for this research that lead to learning based programming and included two invited talks on these topics. Kristian Kersting talked on the democratization of optimization, focusing on declarative programing for lifted inference and efficient optimization in relational domains, exploiting the rich logical structure underlying many AI and data mining problems. Lise Getoor's talk introduced probabilistic soft logic (PSL), a declarative probabilistic programming language that can capture rich structure and scales well, and emphasized the PSL's mathematical framework, hinge-loss Markov random fields (HL-MRFs).

We ended the workshop with a panel and discussed the difficulties of developing and popularizing the use of these kinds of languages, the type of audience that the we may want to target, interesting applications that can be addressed, and the ways different research communities can collaborate to make progress in this research direction.

Parisa Kordjamshidi served as the chair of this workshop. This report was written by Parisa Kordjamshidi and Dan Roth. The papers of the workshop were published as technical report WS-16-06 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence* (Palo Alto, CA: AAAI Press).

Expanding the Boundaries of Health Informatics Using AI

The 20th century laid a foundation of evidence-based medicine that relied on populations and large groups of patients to derive generalized results and observations that were applied to (mostly passive) patients. Yet, the 21st century is shaping up as a time where the patient and personalized health data drive health care innovation and delivery. The availability of this vast amount of personalized data allows for care tailored to a specific patient, an approach coined *personalized medicine*. Moreover, the availability of this data allows for the constant monitoring and discovery of deviations from patient-specific averages (possibly different from population-based averages). These deviations may signal developing problems, and their early detection allows for more effective treatment leading to proactive medicine. Finally, patients are no longer passive recipients of (personalized) treatments and therapies, but they actively participate as a decision maker in their development, customization, and application. This shift has led to the emergence of participatory medicine.

To tackle issues that arise in proactive, personalized, and participatory medicine information technology will need to evolve to improve communication, collaboration, and teamwork among patients, their families, health-care communities, and care teams involving practitioners from different fields and specialties. All of these changes require novel solutions and the AI community is well positioned to provide both theoretical- and application-based methods and frameworks. The goal of this workshop is to focus on creating and refining AI-based approaches that (1) process personalized data, (2) help patients (and families) participate in the care process, (3) improve patient participation, (4) help physicians utilize this participation in order to provide high quality and efficient personalized care, and (5) connect patients with information beyond those available within their care setting. The extraction, representation, and sharing of health data, patient preference elicitation, personalization of "generic" therapy plans, adaptation to care environments and available health expertise, and making medical information accessible to patients are some of the relevant problems in need of AI-based solutions.

This year's workshop built on the very successful AAAI-13 workshop and AAAI 2014 fall symposium on the same topic. The workshop received a large number of submissions that were divided into two main themes. Learning and prediction focused on novel methods for mining semantics from patterns over electrocardiogram data, an adaptive ensemble learning approach for personalized diagnosis, predictive analytics using smartphone sensors for depressive episodes, and a learning-based approach to predicting the 30-day risk and cost of hospital readmissions. The information integration theme saw papers describing how to combine multiple concurrent physiological streams to assess a patient's condition, and an approach to simultaneous influencing and mapping social networks to improve interventions for the homeless community in Los Angeles.

In addition to these themes, two invited speakers provided crucial insights into and directions for health informatics research. The first invited talk by Niels Peek (University of Manchester) was titled "Analytical Challenges for Smarter Health Systems." Peek described the current opportunity to create smarter, "learning" health systems by utilizing the information infrastructure provided by electronic health record (EHR) systems. Examples included using the data that is collected through this infrastructure to develop predictive models for risk stratification and to compare the effectiveness of different treatments in real-world populations. He also positioned EHR systems to be used as a delivery platform to give feedback and advice to clinicians at the point of care. Peek also challenged the audience to develop new tools to make sense of these data that reach beyond traditional analytical concepts and to improve clinical computerized decision support systems that have mostly led to alert fatigue rather than improving care.

The second talk, given by John H. Holmes (University of Pennsylvania), was titled "Data Driven Clinical Research: If Only It Were So Simple." Holmes presented both the opportunities and challenges posed by the availability of ever-increasing amounts of highly heterogeneous clinical data for the data scientist and clinical researcher. The opportunity for enhanced clinical research is manifested in the expanding data and information ecosystem. The challenges are more subtly detected, but present nonetheless. Merging heterogeneous data into an analyzable whole, understanding the clinical context of an image or waveform without their semantic integration with clinical observation data, ecologic fallacy, and data quality were some of the challenges discussed.

Martin Michalowski served as the workshop chair and authored this report. Szymon Wilk and Jay M. Tenenbaum and served as cochairs. The papers of the workshop were published as technical report WS-16-08 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence* (Palo Alto, CA: AAAI Press).

Incentives and Trust in Electronic Communities

The area of trust and reputation modeling has experienced rapid growth in the past decade. With the growing prevalence of social interaction through electronic means, trust, reputation, and privacy, become considerably important. Many computational and theoretical models of trust and reputation mechanisms have been recently developed well-suited for variety of domains such as e-commerce, social network, blogs, ad hoc networks, and others. They present trust as a multifaceted concept that operates at many levels and plays important roles in ensuring reliable interactions. Although trust-enabled systems allow people to act under uncertainty and mitigate the risk of negative consequences, still socio-technical attacks often succeed by exploiting loopholes in the design of trust and security policies. Besides, the diversity of participants in the continuously growing electronic communities encourages cheating and opportunistic behaviors as it is more difficult in such environments to detect and punish fraudulent users. Many techniques have been developed to discourage deception and fraud in e-communities and stabilize trust between participants. These techniques are designed to promote trusting relationships, honesty behaviors, and create incentive for participants to contribute truthful opinions.

Trust and incentive have bidirectional relationships. As trustworthiness measures are used as part of incentive mechanisms to promote honesty in electronic communities, incentive mechanisms motivate participants to contribute their truthful opinions, which are useful for trust modeling. Hence, trust and reputation systems should not only provide a means to detect and prevent malicious activities but also design a mechanism to discourage dishonesty attitudes among participants.

The primary objective of this workshop is to bring together researchers in both the area of game theory for designing incentive mechanisms and the area of trust and reputation modeling, toward the design of more effective trust, reputation, and incentive mechanisms for creating safe electronic communities.

This year, our workshop accepted seven papers. There were also two invited talks, one given by Jiliang Tang (Yahoo Labs, USA) and another by Yan

Wang (Macquarie University, Australia). The workshop participants discussed the importance of designing an incentive-enabled system in electronic communities to foster honest participation among community members and how the workshop was useful in bringing together researchers from different fields such as game theory, user modeling, and trust modeling and encouraging their contributions in creating safe electronic communities.

Jie Zhang, Zeinab Noorian, and Stephen Marsh served as cochairs of this workshop. This report was written by Jie Zhang and Zeinab Noorian. The papers of the workshop were published as technical report WS-16-09 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence* (Palo Alto, CA: AAAI Press).

Knowledge Extraction from Text

Text understanding is an old, yet-unsolved, AI problem consisting of a number of nontrivial steps. The critical step in solving the understanding problem is knowledge acquisition from text, that is, a transition from a nonformalized text, explicitly or implicitly, into a formalized actionable language (that is, capable of supporting automated reasoning). Other steps in the text-understanding pipeline include linguistic processing, reasoning, text generation, search, question answering, and others. These are more or less solved to a degree that would support composition of a text understanding service. However, we know that knowledge acquisition, the key bottleneck, can be done by humans, even though automation of the process is still out of reach in its full breadth.

After failed attempts in the past (due to a lack of both theoretical and technological prerequisites), in recent years the interest in text understanding and knowledge acquisition form text has been growing. There are numerous AI research groups studying various aspects of the problem in the areas of computational linguistics, machine learning, probabilistic and logical reasoning, and semantic web. The commonality among all the newer approaches is the use of recent advances in machine learning to deal with representational change on the level of words, sentences, concepts, and so on.

The workshop brought together researchers from a variety of different approaches for extracting knowledge from text, in addition to researchers in fields that provide empirical or theoretical foundations to the main topic of the workshop. The oral presentations (including the keynote given by Peter Clark, from AI2) revealed a set of interesting and innovative ideas, as well as topics that should help to guide the future of the research community. The workshop participants discussed how knowledge extraction approaches should go beyond named entity recognition (NER) and relation-extraction tasks. The scientific debates focused on new types of knowledge that can be extracted from text, such as relations among events instead of relations among entities, as well as what the community has learned about extracting knowledge from specific target domain texts, such as mars literature. The challenge of extracting knowledge from texts written in different languages (other than English), such as Chinese and Portuguese, was also addressed, focusing on different domains such as never-ending learning to read the web, commonsense relations extraction, as well as automatic email answering agents. The workshop also presented discussions on how parallel computing can help in the scaling-up of topic model algorithms.

Blaz Fortuna, Marko Grobelnik, Estevam Hruschka, and Michael Witbrock served as cochairs of this workshop and wrote this report. The papers of the workshop were published as technical report WS-16-10 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence* (Palo Alto, CA: AAAI Press).

Multiagent Interaction Without Prior Coordination

Interaction between agents is the defining attribute of multiagent systems, encompassing problems such as planning in a decentralized setting, learning other agent models, composing teams with high task performance, and selected resource-bounded communication and coordination. While there is significant variety in methodologies used to solve such problems, many of these methods depend on some form of prior coordination. For example, learning algorithms may assume that all agents share a common learning method and prior beliefs, distributed optimization methods may assume specific structural constraints regarding the partition of state space and cost/rewards, and symbolic methods often make strong assumptions regarding norms and protocols. However, such assumptions are easily violated in realistic problems. Thus, there is a need for new models and algorithms that specifically address the case of ad hoc interactions.

The purpose of this workshop was to discuss the role of such predefined knowledge and coordination in multiagent systems, and to provide a venue for research on novel models and algorithms that specifically address multiagent interaction without prior coordination (MIPC). There were a total of six accepted papers, addressing diverse topics such as ad hoc coalitions in human-robot societies, identifying and tracking nonstationary opponents, and policy communication for coordination with unknown teammates. A continuing trend in several of the workshop papers was the use of beliefs over a set of hypothesized policy types. In addition to the paper presentations, there were invited talks by Michael Bowling from the University of Alberta on "Adventures in Implicit Agent Modeling," and Gal Kaminka from Bar-Ilan University on "Teams, Swarms, Crowds, and Collectives: Special Cases?"

This was the third meeting in this workshop series. The community has grown steadily since the first workshop took place in 2014, reflecting a growing awareness of issues relating to prior coordination in multiagent systems. The workshop organizers intend to continue the workshop series. The workshop was chaired by Stefano Albrecht, Katie Genter, and Somchaya Liemhetcharat. The chairs would like to thank the workshop participants, the invited speakers, the program and advisory committee, and the AAAI staff for making the workshop a success.

This report was written by Stefano Albrecht. The papers of the workshop were published as The papers of the workshop were published as technical report WS-16-12 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence* (Palo Alto, CA: AAAI Press).

Planning for Hybrid Systems

The purpose of this workshop was to explore and promote new approaches to planning with hybrid models. Hybrid systems are systems with both continuous control variables and discrete logical modes. Many interesting real problems are indeed hybrid systems, including oil refinery management, mission planning for autonomous vehicles, supply management and disaster recovery, and applications in control of smart cities. Planning in these domains requires rich models to capture the interaction between discrete and continuous change and methods for reasoning with temporal, spatial, and continuous constraints.

This workshop follows on two previous highly successful hybrid planning events held at the International Conference on Automated Planning and Scheduling (ICAPS). ICAPS 2012 ran a special track on continuous planning, attracting a range of excellent talks on the topic of integrated planning and control. This track brought together planning experts, roboticists, experts in model-based reasoning, experts in runtime verification, and control engineers. This continued as a well-attended workshop at ICAPS 2013. The workshop was opened up to the wider AI community (and applications of interest to this broader community) by the workshop at AAAI-16, which was attended by researchers from planning, robotics, machine learning, hybrid system control and verification, and model-based reasoning and led to a fascinating range of papers and talks. Key topical areas for paper submission included architectures for hybrid systems, planning with SMT (satisfiability modulo theories), temporal logic in hybrid systems, PDDL+ planning, and a fascinating range of applications spanning quantum computing for Mars Lander activity scheduling, urban traffic signal control, autonomous marine vehicles, airport surface operations, and robotics. In short, the exceptional attendance and energetic discussion of this workshop demonstrate the growing influence and applications relevance of hybrid systems planning in AI along with its appeal to a diverse range of researchers and research fields.

This workshop was organized and cochaired by Daniele Magazzeni (King's College London, UK), Scott Sanner (Oregon State University), and Sylvie Thiebaux (Australian National University and NICTA/Data61) and held on February 13, 2016, in Phoenix. Daniele Magazzeni, Scott Sanner, and Sylvie Thiebaux also wrote this report. The papers of the workshop were published as technical report WS-16-12 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence* (Palo Alto, CA: AAAI Press).

Scholarly Big Data: AI Perspectives, Challenges, and Ideas

Academics and researchers worldwide continue to produce large numbers of scholarly documents including papers, books, technical reports, and associated data such as tutorials, proposals, and course materials. For example, PubMed has over 20 million documents, 10 million unique names, and 70 million name mentions. Google Scholar has many millions more, it is believed. Understanding how at scale research topics emerge, evolve, or disappear, what is a good measure of quality of published works, what are the most promising areas of research, how authors connect and influence each other, who are the experts in a field, and who funds a particular research topic are some of the major foci of the rapidly emerging field of scholarly big data.

The primary goals and objectives of the workshop were to promote both theoretical results and practical applications for scholarly big data, and address challenges faced by today's researchers, decision makers, and funding agencies as well as well-known technological companies such as Microsoft and Google, repositories, and publishers such as Elsevier.

Papers presented at the workshop covered a variety of topics including the presentation of a test collection for citation recommendation; the design of evaluation data sets for document similarity models in large scholarly retrieval systems; an analysis of NIH funding patterns over time; data extraction from scientific charts and summary generation; an approach to extend research footprints using disparate sources; improving public access to nonopen biomedical literature; topic-level academic influence of scientific literature; lineage encoding in scholarly articles; and improving discoverability of research papers by augmenting their titles with more terms.

The workshop also included two invited talks and an introduction to the CiteSeerX digital library. The first talk, given by Douglas Downey (Northwestern University), focused on mining topics and key phrases from scientific documents and the application of keyphrase extraction within the semantic scholar scientific search engine. The second talk, given by Alex Wade (Microsoft Research Redmond), focused on academic knowledge: new research opportunities with the Microsoft Academic Graph (MAG), which is a freely available data set that includes information about academic publications and citations, researchers, venues, and topics. MAG is a heterogeneous graph that can be used to study the influential nodes of various types, including authors, affiliations, and venues.

Cornelia Caragea, C. Lee Giles, Alex Wade, and Irwin King served as cochairs of this workshop. This report was written by Cornelia Caragea and C. Lee Giles. The papers of the workshop were published as technical report WS-16-13 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence* (Palo Alto, CA: AAAI Press).

Symbiotic Cognitive Systems

In his 1960 article "Man-Machine Symbiosis," J. C. R. Licklider predicted a time when "the main intellectual advances will be made by men and computers working together in intimate association." While much of the emphasis within the AI community over the ensuing half century was placed upon tools for automation such as speech recognition or surpassing humans at challenging intellectual tasks such as chess or *Jeopardy!* the last few years have witnessed a resurgent interest in symbiotic cognitive systems: collectives of humans and intelligent agents that collaborate to accomplish cognitive tasks better than either can alone.

The objective of this workshop was to synthesize a new vision and research agenda for symbiotic cognitive systems and to try to establish a community that might have a continued existence at future AI workshops and conferences. For that purpose, the workshop brought together researchers working on technologies, architectures, applications, and visions of symbiotic cognitive computing in application domains that spanned robotics, cognitive environments, and cognitive objects. Presentations at the workshop fell into three main categories, with one session devoted to each, with ample time devoted at the end of each session to discussing, summarizing and abstracting the presented material.

In the session on patterns of symbiotic systems, presenters discussed general issues that apply broadly across many different instantiations of symbiotic cognitive systems. Several discussed aspects of knowledge representation, including its role in enabling distributed cognition, in semantically mapping elements of the physical world to that representation, and in teaching agents about relationships that need to be included in that representation. One insight that emerged during the discussion was the idea of confusion or uncertainty. The ability of a system to sense its own uncertainty is necessary in order to generate behaviors that are likely to improve the system's representation of the world or the humans and agents in the environment. The role that attention can play in reorienting the priorities of a cognitive system was discussed, as was the relationship of attention to representation.

The session on symbiotic interactions focused on aspects of interactions between machines and humans or the world that they inhabit. One thread of discussion concerned verbal communication of needs, intents and plans. Methods of collecting verbal communication methods included crowdsourcing to collect a variety of language, profiling human users using social norms and elicitation dialogs, and virtual reality. A second thread of discussion concerned nonverbal communication of information pertaining to the physiological, mental, or emotional state of humans, possibly using cognitive objects (such as chairs) as sensors, while a third concerned systems that could be taught to interpret and execute commands that involved recognizing and manipulating objects in the physical world.

The third session was on learning about and from humans. On the topic of learning about humans, work was presented and discussed about inferring fatigue or other physiological attributes from eye tracking, and learning models of human reaction to advertising by observing behavior and analyzing facial expressions. The use of human demonstrations, situated dialog, text understanding, virtual words, and storytelling were all discussed as methods to learn from humans. The third session was concluded by an interactive poster session where authors could present and discuss their work in further detail.

The workshop concluded with a lively discussion among all participants that focused on questions such as "What are the essential common characteristics and themes of symbiotic cognitive systems?" "What distinguishes them from other interactive systems?" and "Is it worthwhile to create a symbiotic community, and if so how should we go about it?" It is fair to say that, while an absolute consensus was not reached regarding the first two questions, there was great enthusiasm for continuing the discussion at another venue in the near future, perhaps at AAMAS, IJCAI, or another AI conference.

This report was written by Jeffrey O. Kephart and Stephanie Rosenthal. Jeffrey O. Kephart, Stephanie Rosenthal, and Manuela Veloso served as cochairs of the workshop. The papers of the workshop were published as technical report WS-16-14 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence* (Palo Alto, CA: AAAI Press).

World Wide Web and Population Health Intelligence

Population health intelligence includes a set of activities to extract, capture, and analyze multidimensional socioeconomic, behavioral, environmental, and health data to support decision making to improve the health of different populations. Advances in artificial intelligence tools and techniques and Internet technologies are dramatically changing the ways that scientists collect data and how people interact with each other and with their environment. Moreover, the Internet is increasingly used to collect, analyze, and monitor health-related reports and activities and to facilitate health-promotion programs and preventive interventions.

This workshop follows the success of previous AAAI workshops on the same topic held in 2014 in Quebec, Canada, and in 2015 in Austin, Texas, USA. This workshop brought together computer scientists, biomedical and health informaticians, researchers, students, industry professionals, and representatives of national and international public health agencies. Participants were interested in the theory and practice of computational models of web-based public health intelligence. The papers and demonstrations presented at the workshop covered a broad range of disciplines within artificial intelligence including knowledge representation, machine learning, natural language processing, and online social media analytics. From an application perspective, presentations addressed topics in epidemiology, environmental and public health informatics, disease surveillance, health behavior monitoring, and disaster management.

One of the main themes of this workshop was the exploration and monitoring of online social media (for example, twitter) to analyze behavioral patterns. Models of behavior were used to enhance forecasting, guide decision making, enable situational awareness, and inform response strategies. The workshop also included three invited talks. Sudha Ram (professor and director of INSITE Center for Business Intelligence and Analytics, University of Arizona) gave a presentation on using big data for predictive analytics in population and personalized health care. She demonstrated examples from developing predictive models using streaming sensor and social media data sets combined with health-care records. Soon Ae Chun (professor and director of the Information Systems Informatics program at the City University of New York, College of Staten Island) described values and risks of using patient-generated social health data in health care. Damon Centola (associate professor of communication at the Annenberg School for Communication at the University of Pennsylvania) presented his findings from a series of novel experiments designed to study the dynamics of behavioral diffusion in large social networks. His results showed a striking effect of network topology on the diffusion of health behavior, contrary to the expectations of classical network theory.

To promote open debate and exchange of opinion among participants, the workshop was concluded with a panel discussion moderated by Arash Shaban-Nejad and including Sudha Ram, Soon Ae Chun, and Damon Centola. The major theme of the panel was to discuss the future of online surveillance and interventions for changing individual health behaviors.

Arash Shaban-Nejad, David L. Buckeridge, Byron C. Wallace, and John S. Brownstein served as cochairs of this workshop and wrote this report. The papers of the workshop were published as technical report WS-16-15 in *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence* (Palo Alto, CA: AAAI Press).

Stefano Albrecht is a postdoctoral fellow at the University of Texas at Austin.

Bruno Bouchard is assistant professor at Université du Québec à Chicoutimi .

John S. Brownstein is a researcher for Boston Children's Hospital and a professor at Harvard University.

David L. Buckeridge, MD, is an associate professor at McGill University.

Cornelia Caragea is an assistant professor at University of North Texas.

Kevin M. Carter is an associate group leader at MIT Lincoln Laboratory.

Adnan Darwiche is a professor at the University of California, Los Angeles.

Blaz Fortuna is a researcher consultant for Bloomberg L.P. and Jozef Stefan Institute.

Yannick Francillette is from Université du Québec à Chicoutimi

Sébastien Gaboury is assistant professor at Université du Québec à Chicoutimi.

C. Lee Giles is a professor at the Pennsylvania State University.

Marko Grobelnik is a research team leader at Jozef Stefan Institute.

Estevam R. Hruschka Jr. is an associate professor at the Federal University of São Carlos.

Jeffrey O. Kephart is a member of the research staff at IBM Thomas J. Watson Research Center.

Parisa Kordjamshidi is a postdoctoral researcher at the University of Illinois at Urbana-Champaign.

Viliam Lisy is a postdoctoral fellow at the Department of Computing Science, University of Alberta.

Daniele Magazzeni is a lecturer in computer science at King's College London.

Joao Marques-Silva is a professor at the University of Lisbon.

Pierre Marquis is a professor at Université d'Artois.

David Martinez is an associate division head at MIT Lincoln Laboratory.

Martin Michalowski is a senior principal research scientist at Adventium Labs.

Zeinab Noorian is a postdoctoral research fellow at Ryerson University.

Enrico Pontelli is a professor at New Mexico State University.

Alex Rogers is a professor at the University of Oxford.

Stephanie Rosenthal is a research scientist at Carnegie Mellon University.

Dan Roth is a professor at the University of Illinois at Urbana-Champaign.

Scott Sanner is an assistant professor at the University of Toronto.

Arash Shaban-Nejad is affiliated with the School of Public Health, University of California, Berkeley.

Arunesh Sinha is a postdoctoral researcher at the University of Southern California.

Tran Cao Son is a professor at New Mexico State University.

William Streilein is a Group Leader at MIT Lincoln Laboratory.

Sylvie Thiebaux is a professor at the Australian National University and a research leader at the National ICT Australia.

Byron C. Wallace is an assistant professor at the University of Texas at Austin.

Toby Walsh is a professor at the University of New South Wales and a group leader at Data61.

Michael Witbrock is a consulting computer scientist at Lucid AI.

Jie Zhang is an associate professor at Nanyang Technological University.

The International Competition of Distributed and Multiagent Planners (CoDMAP)

Antonín Komenda, Michal Štolba, Daniel L. Kovacs

■ *This article reports on the first international Competition of Distributed and Multiagent Planners (CoDMAP). The competition focused on cooperative domain-independent planners compatible with a minimal multiagent extension of the classical planning model. The motivations for the competition were manifold: to standardize the problem description language with a common set of benchmarks, to promote development of multiagent planners both inside and outside of the multiagent research community, and to serve as a prototype for future multiagent planning competitions. The article provides an overview of cooperative multiagent planning, describes a novel variant of standardized input language for encoding mutliagent planning problems, and summarizes the key points of organization, competing planners, and results of the competition.*

Automated planning, as a subfield of artificial intelligence, proposes possibly tractable, heuristic, and algorithmic solutions to computationally hard combinatorial problems of sequential decision making. In the case of domain-independent planning, the input of a planner does not contain only the planning problem instance but also its domain, compactly describing the mechanics of the environment.

The history of competitions of domain-independent automated planning began in 1998, with the first International Planning Competition (IPC)[1] organized by Drew McDermott (chair), Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. IPC flourished, and during the next 17 years IPC grew into various tracks comparing a variety of extensions of the classical planning model based on STRIPS (Fikes and Nilsson 1971) and Action Description Language (ADL) (Pednault 1989), using the Planning Domain Description Language (PDDL) (McDermott et al. 1998) as the de facto standard problem-description (input) language. Although IPC focused only on single-agent planning, the competition did add new tracks over the years, beyond the initial deterministic track. For example the probabilistic track, organized by Blai Bonet and Bob Givan, was begun in 2006. Currently this track focuses on planning problems defined as Markov decision processes (MDPs) and partially observable MDPs (POMDPs) suitable for modeling uncertainty in case of single-agent planning, and to some limited extent in case of multiagent planning. For the planning community, IPC became not only a standard way to compare the performance of planners, but also a source of a wide variety of benchmarks motivated by both real-world problems and challenging fundamental features of the planners.

In contrast to multiagent uncertainty planning modeled as POMDPs, Ronen Brafman and Carmel Domshlak (2008) proposed a model for (cooperative) domain-independent multiagent planning for discrete and deterministic environments. A special form of partial observability was defined in terms of privacy, where agents should not know, observe, or use private knowledge of other agents. The motivation was to create a minimal extension of the classical planning model towards multiagent planning. The result was MA-STRIPS — extended STRIPS by partitioning of possible actions according to the particular agents and by defining what facts about the state of the world should be treated as public or private knowledge.

MA-STRIPS planning agents solve one common planning problem, which is partitioned to several subproblems. Partitioning according to agents is related to splitting one large planning problem into smaller parts by factoring, which can radically lower complexity in some cases. MA-STRIPS by its privacy requirements also implies that the private parts of the problem have to be solved by their respective owner planning agents.

The real-world motivation for MA-STRIPS spans over a wide variety of problems (Nissim and Brafman 2014), similarly to classical planning. Be it a consortium of cooperating logistic companies with common transportation tasks, but private know-how about local transport possibilities; or a team of spatially separated gas station inspectors with a common goal to analyze quality of gasoline in the whole country, but with private knowledge about the particular gas stations; or a heterogeneous fleet of satellites and rovers surveying a distant planet, for which keeping local information private is the only feasible way not to overload the communication network. In these motivational cases, MA-STRIPS problem partitioning would be defined over trucks, inspectors, rovers, and satellites. The know-how of mentioned corporations, knowledge of inspectors, and local information of robots would define MA-STRIPS private knowledge and obviate sharing all of the information freely among the agents.

After the MA-STRIPS model was introduced, a substantial number of multiagent planners were proposed. However, the differences in the used input languages and absence of a common set of benchmarks rendered their comparison impractical. In this article, we report on the first competition of distributed and multiagent planners compatible with the MA-STRIPS model. Our goal is to improve the situation by preparing a common language and benchmarks and providing a common ground for comparison. The competing planners were either centralized or distributed, competing in two separate tracks. Each of them performed planning in advance (offline) for cooperative agents with common and publicly known goal(s). The agents acted in a shared deterministic environment, and in the distributed track, they were required to keep parts of their planning problems private, that is, not share it with other agents. All actions were discrete time and nondurative. We have organized the competition as part of the workshop on Distributed and Multiagent Planning at the International Conference on Automated Planning and Scheduling during winter and spring 2015.

Input Language

Following the minimalistic extension of STRIPS to MA-STRIPS by Brafman and Domshlak in 2008, we wanted a simple extension of PDDL toward multiagent planning, also compatible with MA-STRIPS. We chose MA-PDDL (Kovacs 2012) and extended it with a partitioning definition and a definition of privacy of objects and predicates (and thus implicitly of the privacy of actions). The extension allowed defining agents in various ways: as objects, constants, or not explicitly at all. This variability allowed us to reuse many interesting classical planning benchmarks.

The definition of privacy in MA-STRIPS is implicit and follows a simple rule, which says that a fact is public if it is required or modified by two or more actions of different agents. An action is public if it requires or modifies at least one public fact. Based on a review of literature, and a conducted precompetition poll,[2] we found that such a definition could be too rigid, especially for future versions of the competition. We slightly relaxed the MA-STRIPS notion of privacy and declared it explicitly in the MA-PDDL description. Our privacy definition follows MA-STRIPS in the sense that facts and actions can be private to particular agents or public among all agents, however what facts and actions are private and public is determined by a process coined maximally concealing grounding (MCG), which is based on three rules:

(1) A public predicate definition grounded with public objects / constants is a *public fact.*

(2) A public predicate definition grounded with at least one object / constant private to agent α is a *private fact* of agent α (grounding a single predicate definition with objects private to different agents is not allowed).

(3) A private predicate grounds to a *private fact* regardless of privacy of the objects used for grounding.

By convention, a PDDL object representing an agent was private to that given agent. If it was not, other agents of the same PDDL type would be able to ground and use the other agent's actions.

We have defined two ways how to encode[3] multiagent planning problems: either as unfactored MA-PDDL or as factored MA-PDDL. With regard to information, the two representations are equivalent. The difference is in the information separation. As for distributed multiagent planning, it is important to provide the respective agents only with information allowed to them by the privacy requirements. Description of the MA-PDDL variants follows.

Unfactored MA-PDDL

Unfactored MA-PDDL stems naturally from classical PDDL. It uses a pair of files for all agents. One file contains the domain information and the other the specification of a problem instance within that domain.

Unfactored MA-PDDL is defined in terms of two extensions of classical PDDL. The first informs the planner that action definitions are annotated with an additional specification of the agent owning the grounding of the action. Using this extension over all actions unambiguously defines action partitioning. The other extension, with a help of the previously defined MCG rules, unambiguously defines what facts are private and public to given agents and, using the MA-STRIPS definition, what actions are private and public.

Factored MA-PDDL

Factored MA-PDDL results straightforwardly from the distributed nature of multiagent systems. Each separate planning agent uses its own pair of domain and problem description files (denoted as a MA-PDDL factor). Each pair defines information relevant only to that particular agent.

Action partitioning ensues directly from factorization of the input. As each planning agent's factor contains only relevant objects, constants, and actions, there is an unambiguous grounding of them. Objects and constants that were common for more than one agent were by convention bound over the same names. The grounding semantics of privacy using the MCG rules is the same as in the unfactored variant with respect to the partitioning by the MA-PDDL factors.

Competition Tracks

The success of a planning competition is determined to a large extent by the number of contestants. Because there was no historical experience from previous multiagent competitions, we wanted to open the competition to the widest possible audience. A survey of literature on multiagent planners together with the precompetition poll provided enough information to set the rules for the competition so that an ample amount of already existing multiagent planners could compete and still the key motivations of the competition remained satisfied.

Technically, the fundamental discriminator of current multiagent planners is whether they can work distributively on multiple interconnected physical machines or not. Running planners in such a distributed setup is incomparable to running planners centrally on one machine with a shared memory space. To accommodate planners running in either mode, the competition was split into two tracks: centralized and distributed (see figure 1). The following paragraphs describe the requirements and emphasize the differences between them.

Centralized Track

The centralized track aimed at running multiagent planners on one physical multicore machine with one shared memory space that allowed use of any means of communication among its agents (see figure 1, top). This included a setup with only one agent, which is typical for classical planners. It was up to the planner whether it ran on one or more machine cores and which type of communication it used, if any. In contrast to IPC, in the centralized CoDMAP track the planners had to read the input in MA-PDDL, either factored or unfactored. The MA-STRIPS partitioning and privacy definitions were indicated in the input MA-PDDL files, but the planners were allowed to ignore them. The required output was a sound (valid) sequential plan solving the provided planning problem. As a result of the above-described openness, the centralized track allowed comparison of a wide spectrum of multiagent planners. The track was therefore annotated as transitional and highly compatible.

Distributed Track

The distributed track removed the compatibility compromises of the centralized track. All competing multiagent planners had to run in a distributed fashion (as several planning agents) on a cluster of interconnected multicore machines, where each machine was dedicated only for one planning agent (see figure 1, bottom).

The input was limited to factored MA-PDDL and the partitioning matched the physical machines. Privacy followed the factored MA-PDDL definition and the MCG rules. Planning agents could communicate only public information over the TCP/IP network. Competition rules forbid explicitly exchanging any private information among the planning agents. The output was a set of plans: one plan per planning agent using only actions defined by its respective MA-PDDL factor. The soundness of plans was tested after their linearization. Concurrent actions of different agents at any given time did not have to be mutually exclusive. The distributed track was unique and novel in comparison to the IPC tracks.

Competition Domains

The planners were evaluated over a set of 12 benchmark domains. The domains were motivated by important and interesting real-world problems or by problems exposing and testing theoretical features of the planners. We used domains from literature on multiagent planning that are in most cases multiagent variants of the classical IPC domains: BLOCKSWORLD, DEPOT, DRIVERLOG, ELEVATORS08, LOGISTICS00, ROVERS, SATELLITES, SOKOBAN, WOODWORKING, and ZENOTRAVEL. Each domain had 20 problem instances, with varying size, number of objects, constants, agents, and thus

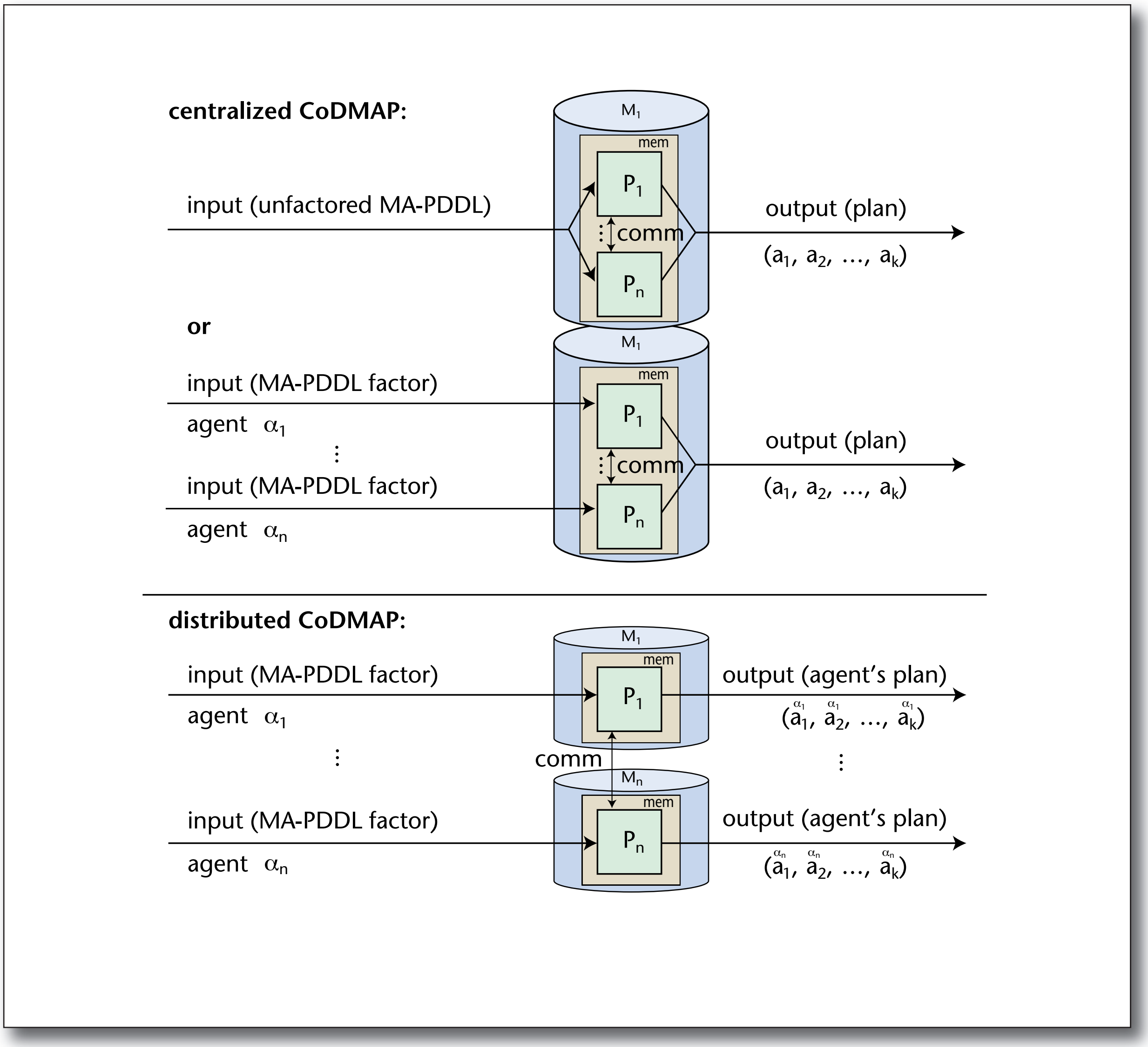


Figure 1. Comparison of IPC and CoDMAP Tracks.

complexity. The biggest problems had 10 agents and about 100 objects/constants. Additionally, we added two novel domains inspired by well-known multiagent problems, not modeled in MA-STRIPS or MA-PDDL previously: TAXI and WIRELESS. The taxi domain can be considered a multiagent variation on the logistic domain, in which monotonous (relaxed) heuristic planning benefits. On the contrary, the wireless domain is modeled such that monotonous planning is deceived by a concept of circulating messages among the sensors causing the relaxation planners to think the communication is free of charge. Description of the novel domains follows.

TAXI

TAXI problems model on-demand transport in a city (see figure 2, left). Two types of agents represent taxis and passengers. Each taxi and passenger is at a particular location. A location can be free of taxis and two locations can be directly connected. Connected locations form a topology of the city. Each taxi can transport only one passenger from the location it stays at and only to a free drop-off location (a location containing no other taxis). A taxi can move only between connected locations.

WIRELESS

WIRELESS problems model distributed gathering of data by a group of smart sensors to a base station (see figure 2, right). The base and sensors are represented by agents, where some of them are neighbors (they are in range of their radios). The neighbor relation defines the topology of an ad hoc radio network among the sensors and the base. Sensor agents have

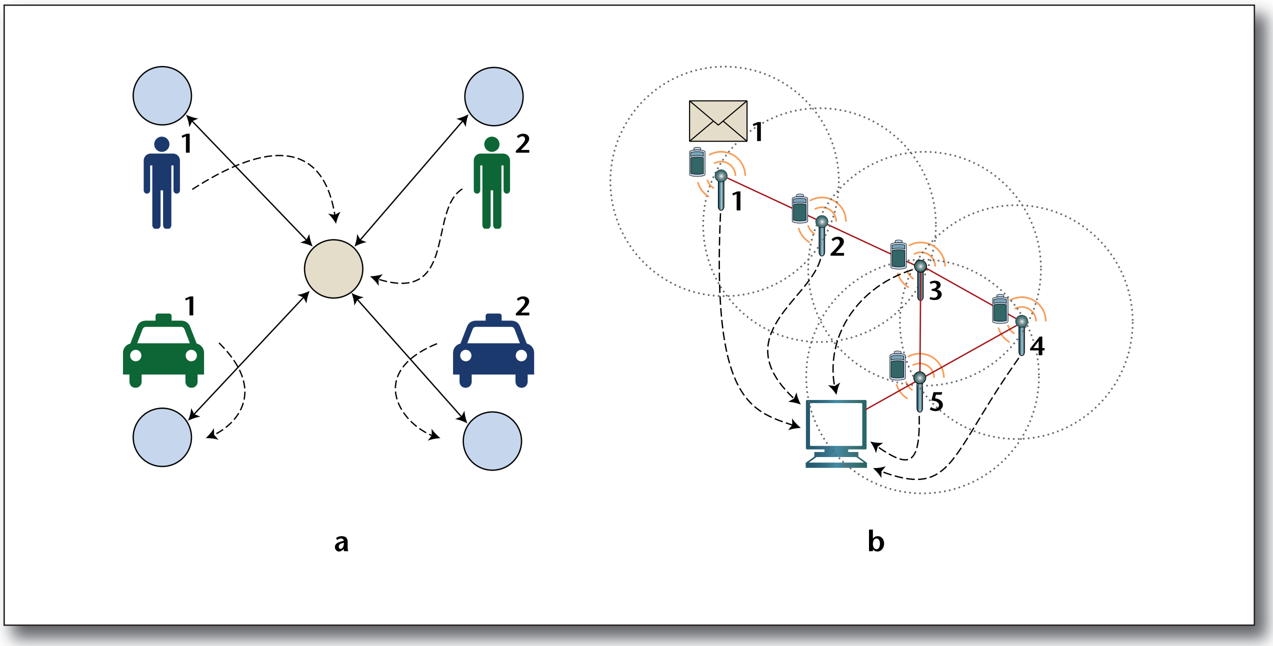


Figure 2. Example Problem Instances of the Two Novel CoDMAP Domains.

TAXI (left). WIRELESS (right). The figures represent initial states of the easiest instances of the domains. Dashed arrows show the goals of individual agents. In the TAXI problem, both passengers want to be transported to the central location and the taxi drivers want to end at the same locations they started from (the garage). In the WIRELESS problem, all five sensor nodes are initially at normal battery level, and there is only one allowed message in the system represented by an envelope initially at sensor 1. Data from all sensors has to be gathered by the base station represented by a computer.

four possible private energy levels. A sensor with more than zero energy can do its measurement and generate measurement data, which decreases its energy by one level. A sensor can add measurement data (possibly of other sensors) to a message if it has the data and the message is in its memory. A sensor with more then zero energy can send a message to a neighboring sensor or base, which decreases its energy by one level. Receiving a message as well as extracting measurement data from a message does not change the energy level. The number of messages usable in parallel is limited; however, they can be reused sequentially. The goal is to gather the measurement data of selected sensors at the base station.

Validation and Evaluation

Each run of a planner in the competition was restricted to 30 minutes and 8 gigabytes of RAM per physical machine (in the centralized track per problem, while in the distributed track per one agent, that is, a factor of a problem) on quad-core machines at 3.9 GHz. The machines were for the distributed track interconnected into an IP subnet with one 10 Gbps switch and 1 Gbps Ethernet cards.

The metrics used to compare the planners were coverage (number) of solved problems, IPC score over the plan quality (ratio to the optimal solution), and IPC score over the planning time (ratio to the fastest solution). In the distributed track, the plan quality was evaluated both in terms of total cost (sum of costs of all used actions) and makespan (the maximum time step of the plan if executed in parallel). The validity and quality of plans was evaluated using the VAL tool,[4] which can handle parallel plans and also performs checks of mutually exclusive actions.

The intentionally weak rules of the centralized track attracted a number of (classical) planners adapted to process multiagent input in MA-PDDL. This resulted in a wide spectrum of planners in terms of the way of partitioning and privacy preservation. To fairly compare the efficiency of these planners, competition rules required submission of a short paper for each submitted planner with a list of required items, which can be used by the community to select only those results that are relevant for their research. The tighter rules of the distributed track allowed a much easier comparison of competing planners.

Competing Planners and Selected Results

Some of the competing planners were submitted in several configurations. For the centralized track, we received 12 planners in 17 configurations prepared by 8 teams. For the distributed track 6 configurations of 3 planners by 3 teams were received. All teams

Centralized track			Distributed track		
1.–2.	ADP	222	1.	PSM	180
3.	MAP-LAPKT	216	2.	MAPlan	174
4.	CMAP	210	3.	MH-FMAP	107

Table 1. Best Performing Planners in the Metrics of Solved Problems (Coverage) Out of Overall 240 Benchmarks.

were from the research community of automated planning and multiagent systems. A selection of the results is listed in table 1. Complete, detailed, and interactive results can be found on the official CoDMAP webpage.[5] In the following paragraphs we summarize the key principles of the best performing planners.

The winner of the centralized track, ADP, by Crosby, Rovatsos, and Petrick (2013), was based on the idea of automatic decomposition of a planning problem to agents (however, it ignored the partitioning and the privacy predefined in MA-PDDL) using a graph of causal dependencies among actions. The planning process itself interleaved the subgoal calculation phase and the search phase by the FastDownward planner (Helmert 2006). A similar principle was used in the CMAP planner by Borrajo (2013), where the planner did not plan for subgoals but adapted and merged partial plans of different agents. Additionally, the subproblems were obfuscated such that privacy was preserved. The MAP-LAPKT planner by Muise, Lipovetzky, and Ramirez (2015) compiled the multiagent planning problems to classical problems respecting the predefined partitioning and emulating the partial observability resulting from the privacy.

The winner of the distributed track represented sets of possible local plans of the agents as finite state machines. The structures, coined *planning state machines,* giving the planner name PSM (Tozicka, Jakubuv, and Komenda 2014), were projected to a public part of the problem and merged. Provided that a merger of all public projections of agents' PSMs was nonempty, a coordination plan was found and was extended to the global solution. The PSM planner preserved privacy as the PSMs were kept local and the merging process communicated only the public projections. MAPlan by Fišer, Štolba, and Komenda (2015) and MH-FMAP by Torreño, Onaindia, and Sapena (2014) were distributed multiheuristic forward-chaining search planners, in the former case, in the space of states and in the latter, in the space of partial-ordered plans by a distributed variant of the best first search algorithm. MAPlan and MAP-LAPKT were the only optionally optimal planners in the competition.

Conclusions and Future Directions

The first international Competition of Distributed and Multiagent Planners became a thorough and nearly complete comparison of existing multiagent planning systems compatible with the MA-STRIPS model. It served as a successful proof-of-concept prototype of a multiagent competition showing good direction and viability similarly to first IPC 17 years ago. We are highly confident that a new track on multiagent planning can become a valuable addition to the next International Planning Competition.

Future directions for the competition can take advantage of the extensibility of the MA-PDDL language. An obvious direction is to use the looser privacy definition allowed by MA-PDDL and MCG and propose planning problems with complex privacy requirements like private goals. A partitioning related extension is to allow joint actions, which have to be performed by two or more agents at the same time.

Acknowledgments

This work was partially supported by the Czech Science Foundation (grant no. 15-20433Y), Israel Science Foundation, and the International Conference on Automated Planning & Scheduling 2015. We thank the ICAPS 2015 conference chairs Ronen Brafman and Carmel Domshlak for valuable initial input about the competition and Peter Benda from Czech Technical University in Prague for IT support. We would like to thank all participants in the competition; without their interest and hard work, the competition would be impossible.

Notes

1. See ipc98.icaps-conference.org/.
2. See the precompetition poll, bit.ly/1IsNoqY.
3. The extended BNF of MA-PDDL can be found at agents.fel.cvut.cz/ codmap/MA-PDDL-BNF.pdf.
4. See www.inf.kcl.ac.uk/research/groups/ planning.
5. See agents.fel.cvut.cz/codmap.

References

Borrajo, D. 2013. Plan Sharing for Multi-Agent Planning. Presented at the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13). (icaps13.icaps-conference.org/wp-content/uploads/2013/05/ dmap13-proceedings.pdf)

Brafman, R. I., and Domshlak, C. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS-08),* 28–35 Menlo Park, CA: AAAI Press.

Crosby, M.; Rovatsos, M.; and Petrick, R. 2013. Automated Agent Decomposition for Classical Planning. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS-13),* 46–54 Menlo Park, CA: AAAI Press.

Fikes, R., and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI-71),* 608–620 San Francisco: William Kaufmann.

Fišer, D.; Štolba, M.; and Komenda, A. 2015. MAPlan. Paper presented at the Competition of Distributed and Multi-Agent Planners (CoDMAP-15). (agents.fel.cvut.cz/codmap/results/CoDMAP15-proceedings.pdf.)

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26: 191–246.

Kovacs, D. L. 2012. A Multi-Agent Extension of PDDL3.1. Paper presented at the 3rd Workshop on the International Planning Competition (IPC. (icaps12.icaps-conference.org/workshops/ipc2012-proceedings.pdf)

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL — The Planning Domain Definition Language. Technical Report TR-98-003, Yale Center for Computational Vision and Control. New Haven, CT, Yale University.

Muise, C.; Lipovetzky, N.; and Ramirez, M. 2015. MAP-LAP-KT: Omnipotent Multi-Agent Planning via Compilation to Classical Planning. Paper presented at the Competition of Distributed and Multi-Agent Planners (CoDMAP-15). (agents.fel.cvut.cz/ codmap/results/CoDMAP15-proceedings. pdf.)

Nissim, R., and Brafman, R. 2014. Distributed Heuristic Forward Search for Multi-Agent Planning. Journal of Artificial Intelligence Research 51: 293–332.

Pednault, E. P. D. 1989. ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR-89),* 324–332 San Francisco, CA: Morgan Kaufmann.

Torreño, A.; Onaindia, E.; and Sapena, O. 2014. FMAP: Distributed Cooperative Multi-Agent Planning. *Applied Intelligence* 41(2): 606–626.

Tozicka, J.; Jakubuv, J.; and Komenda, A. 2014. Generating Multi-Agent Plans by Distributed Intersection of Finite State Machines. In P*roceedings of the 21st European Conference on Artificial Intelligence (ECAI-14),* 1111–1112 Setúbal, Portugal: SciTePress.

Antonín Komenda is a research fellow at the Agent Technology Center (ATG) at the Faculty of Electrical Engineering (FEE), Czech Technical University in Prague. His research and project work focuses on domain-independent planning, both classical and multiagent in deterministic and uncertain planning models. Komenda earned his Ph.D. from Czech Technical University in Prague and was a postdoctoral fellow at Technion – Israel Institute of Technology from 2013 to 2014.

Michal Štolba is a Ph.D. student at the Agent Technology Center (ATG) at the Faculty of Electrical Engineering (FEE), Czech Technical University in Prague (CTU). His research focuses mainly on heuristic search for domain-independent multiagent planning and distributed heuristic computation. He holds a master's degree in automated planning from the Strathclyde University in Glasgow, UK.

Daniel L. Kovacs is an external lecturer in the Department of Measurement and Information Systems at the Budapest University of Technology and Economics, where he received his B.Sc. and M.Sc. in computer science in 2003 and completed his Ph.D studies in 2006. His current research is focused on multiagent planning, bounded rationality, and game theory. He was an organizer of the first three ICAPS-DMAP workshops, 2013–15.

VANDERBILT UNIVERSITY

THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE (EECS) AT VANDERBILT UNIVERSITY is seeking candidates for two tenured/tenure-track (T/TK) faculty positions. Appointments at all ranks will be considered; we prefer to fill at least one position at the assistant professor of computer science rank. Successful candidates are expected to teach at the undergraduate and graduate levels and to develop and grow vigorous programs of externally funded research. Areas of focus for this search are: (1) Computer Assisted Surgery and Interventions. We seek world-class expertise in medical image and signal analysis, computer vision, and medical robotics. The successful candidate will expand the CS curriculum in these areas and have a strong interest in translational research, collaboration with the Vanderbilt University Medical Center, and will be expected to engage with the Vanderbilt Institute in Surgery and Engineering (VISE: http://www.vanderbilt.edu/vise). VISE's mission is the creation, development, implementation, clinical evaluation and translation of methods, devices, algorithms, and systems designed to facilitate surgical and interventional processes and their outcome. (2) Big Data/Data Science/AI. We seek world-class expertise in broadly defined areas of data science, machine learning, data mining, visualization, computer vision, and/or artificial intelligence. The Vanderbilt CS program provides a unique, collaborative, and interdisciplinary research environment. New trans-institutional programs are creating opportunities for research on issues of broad significance that create and extend collaborations across multiple fields.

Vanderbilt University is a private, internationally renowned research university located in vibrant Nashville, Tennessee. Its 10 schools share a single cohesive campus that nurtures interdisciplinary activities. The School of Engineering is on a strong upward trajectory in national and international stature and prominence, and has built infrastructure to support a significant expansion in faculty size. In the 2015 rankings of graduate engineering programs by U.S. News & World Report, the School ranks third among programs with fewer than 100 faculty members. 5-year average T/TK faculty funding in the EECS Department is nearly $1M per year. All junior faculty members hired during the past 15 years have received prestigious young investigator awards, such as NSF CAREER and DARPA CSSG.

With a metro population of approximately 1.5 million people, Nashville has been named one of the 15 best U.S. cities for work and family by Fortune magazine, was ranked as the #1 most popular U.S. city for corporate relocations by Expansion Management magazine, and was named by Forbes magazine as one of the 25 cities most likely to have the country's highest job growth over the coming five years. Major industries include tourism, printing and publishing, manufacturing technology, music production, higher education, finance, insurance, automobile production and health care management.

Vanderbilt University is an equal-opportunity, affirmative-action employer that aspires to become a leader among peer institutions in making meaningful and lasting progress in responding to the needs and concerns of women and members of under-represented minority groups. Applications should be submitted on-line at:
https://academicjobsonline.org/ajo/jobs/7736.

For more information, please visit our web site: http://engineering.vanderbilt.edu/eecs. Applications will be reviewed on a rolling basis beginning November 1, 2016 with telephone interviews beginning December 1, 2016. The final application deadline is January 15, 2017.

Automated Process Planning for CNC Machining

Christian Fritz

■ *This article describes an application of AI planning to the problem of automated process planning for machining parts, given raw stock and a CAD file describing the desired part geometry. Researchers at PARC have found that existing planners from the AI community fall short on several requirements, most importantly regarding the expressivity of state and action representations and the ability to exploit domain-specific knowledge to prune the search space. In this article I describe the requirements for this application and what kind of results from the planning community helped most. Overall, in this project as well as others, I found that even significant results from domain-independent planning may not be relevant in practice.*

"Unsurprisingly, AI planning in industry is domain-specific."

A large portion of today's industrial manufacturing relies on subtractive machining, a process in which a fast-spinning tool successively removes material from raw stock, for example, a block of aluminum, in order to arrive at a part geometry as specified by a design engineer in a computer-aided design (CAD) file. Planning for this process typically involves identifying the sequence of orientations in which the work piece needs to be fixed, identifying the sequence of tools to be used in each orientation, identifying the part of the volume to be removed in each step of each orientation, and identifying the machine to use for each step of each orientation. Until now, this planning process was not automated but done by humans.

The Defense Advanced Research Projects Agency (DARPA) recognized that this lack of automation was a source of delays in the design and production of new vehicles and requested proposals to address this issue and to provide automated manufacturing feedback back to designers. At Palo Alto Research Center (PARC), researchers recognized the potential business value to designers as well as manufacturers, and this value proposition was validated during project execution by presenting early prototypes of the software to potential users. The objective of PARC's uFab project hence was to create a software tool that, given just a CAD file and a representation of available machines and tools, generates a process plan in real time. While work in this area had been done in the 1980s under the name computer-aided process planning (CAPP) (Alting and Zhang 1989), none of the approaches that were pursued then resulted in a fully automated solution. A major shortcoming of these systems was their reliance on features, recognizable configurations of faces on a part such as pockets, slots, and holes, in order to represent states and actions. In these approaches, planning amounted merely to selecting from a set of predefined operations to make each of the recognized features. This reliance on feature-based representations hindered their broad applicability to parts that could not be easily described as a combination of features or where feature recognition was difficult, ambiguous, or error prone.

Feature-Free Process Planning

My group therefore went with an entirely different representation. Enabled by much more computational power and larger memory than in the 1980s, the approach uses a hybrid representation of faces and voxels — three-dimensional pixels. The CAD geometry to be analyzed is discretized into a large number of voxels (on the order of 200 per coordinate axis). Given this representation, actions are characterized by a tool and an orientation, and their effect is described by the removal volumes, that is, the sets of voxels that can be reached by the tool in the given orientation, given its length and diameter, without colliding with the desired geometry. Computing these removal volumes requires geometric reasoning, and is much slower than one would want in an inner loop of a planner (ca. 200 ms). For this reason we precompute all maximal removal volumes ahead of planning, and then only intersect and union these volumes during planning.

Taken together, these properties of the representation rendered Planning Domain Definition Language (PDDL)–based planners unusable for purposes or this research. These planners do not have the representational expressivity to capture geometric operations, nor are they well suited to computing large intersections (millions of logical conjunctions) and unions (millions of logical disjunctions). Nor did they need to. Given that my colleagues and I worked on this problem for multiple years, which is not uncommon in industry, it was entirely acceptable to design and implement a domain-specific planner that uses fast C-implementations for required set operations, the z-buffer algorithm (Rossignac and Requicha 1986) for geometric reasoning, and intelligent precomputation of action effects where possible. Relative to these domain-specific needs, implementing the actual search used for planning was the easy part. My colleagues and I used a combination of beam and weighted A* search, and exploited some insights from solutions for the set-cover problem. Since the value-proposition was based on an interactive experience, the challenge was to balance optimality with response time for the user, but in addition to the usual search parameters like beam width and heuristic weight, my colleagues and I had broad domain-specific knowledge at our disposal, which proved most powerful.

As a result, PARC is now able to offer a web-based service for automated process planning that can save machine shop owners several hours of manual work each day, resulting in clear monetary value. This benefit is most pronounced for machine shops that specialize on "high-mix, low-volume" business, that is, where only a relatively low quantity of each newly quoted part is cut.

Lessons Learned

Catalyzed by the international planning competition, the AI planning community has accomplished great speedups in domain-independent planning for a number of planning categories. However, in my experience with this industrial application as well as others, domain-independence rarely matters in practical applications. In industry, almost by definition, people have industry-specific, that is, domain-specific, planning needs. If the planning community would like to help industry with these needs, then a shift in focus may be advisable. The type of research that I believe industry would most benefit from involves more expressive representations, including metalanguages for creating domain-specific languages to use for state and action space representation, hybrid approaches between declarative planning and procedural programming, such as TLPlan (Bacchus and Kabanza 1998), Golog (Levesque et al. 1997), or even procedural attachments, as well as fundamental insights about combinatorial search that are independent of the representation. Any advances that are specific to domain-independent planning in PDDL, such as the powerful deletion heuristic, are unlikely to move the scale in practice, at least for me. The competition is simply too high: dealing with the NP or PSPACE hardness of planning depends on the ability to exploit structure, and knowing the domain simply puts domain-specific planners in an (exponentially) unfair advantage over domain-independent approaches. As a result, the business model for domain-independent planners is questionable, and I believe aspiring graduate students are ill advised to focus too much on them if they care about the practical long-term impact of their work.

Acknowledgements

The work on this article was performed while the author was at Palo Alto Research Center, Palo Alto, CA, USA.

References

Alting, L., and Zhang, H.-C. 1989. Computer Aided Process Planning: The State-of-the-Art Survey. *International Journal of Production Research.* 04/1989; 27(4): 553–585. dx.doi.org/10.1080/00207548908942569

Bacchus, F., and Kabanza, F. 1998. Planning for Temporally Extended Goals. *Annals of Mathematics and Artificial Intelligence* 22(1–2):5–27, 1998. dx.doi.org/10.1023/A:1018985923441

Levesque, H. J.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming* 31(1–3): 59–83. dx.doi.org/10.1016/S0743-1066(96)00121-5

Rossignac, J. R., and Requicha, A. A. G. 1986. Depth-Buffering Display Techniques for Constructive Solid Geometry. *IEEE Computer Graphics and Applications* 6(9) (September): 29–39. dx.doi.org/10.1109/MCG.1986.276544

Christian Fritz until recently led the Representation and Planning area at PARC. Prior to assuming that role he was a research scientist, working on various applied planning projects. Prior to PARC he was a postdoctoral researcher at USC ISI. Fritz earned his Ph.D. in computer science from the Knowledge Representation group at the University of Toronto, Canada, and his MS and BS degrees in computer science from RWTH Aachen, Germany.

Worldwide AI

■ *This column provides an encounter with the artificial intelligence research community in the state of Israel. The first section introduces this community and its special attributes. The second section provides an overview of some recent research projects done in Israel. The author serves as the chair of the Israeli Association for Artificial Intelligence.*

Ariel Felner

The Israeli AI Community

Israel is a young and a small country consisting of only approximately 20,000 square kilometers in area and a population of approximately 8 million. Since its establishment in 1948, The Israeli government has placed great importance on establishing excellent research institutions and universities.

As a consequence, there are eight universities in Israel, as well as a handful of research institutions and numerous colleges, and Israel has excelled in numerous fields of research. A clear sign of this is that Israel has produced eight Nobel laureates in the past 15 years, out of 154 worldwide. Computer science research in Israel dates back to the country's founding, and five Turing Award winners (out of 62) are Israelis. AI research in Israel has been firmly established since the 1980s, and there are currently quite a few AI research groups and labs in Israeli universities.

This column introduces the Israeli AI community and many of its unique attributes. It also covers a number of recent research projects in the field of AI that are done in different institutions within the country.

The Israeli Association for AI

The Israeli Association for Artificial Intelligence (IAAI),[1] a member of the European Association for Artifical Intelligence (EURAI), is an umbrella organization for AI researchers in Israel. The primary goals of the organization are to promote the study and research of AI in Israel, to encourage cooperation between Israeli AI researchers, and to promote collaboration with AI researchers worldwide.

Israelis are known to be very friendly and they like to socialize. In addition, Israel's small size means that its two most distant universities — Technion in Haifa (the north) and Ben-Gurion University of the Negev (the south) — are only 187 kilometers apart. As a result, many members of IAAI have strong personal and research relations throughout the country. Very often, Israeli AI researchers from throughout the country will organize mutual visits during which they hold research meetings, give talks at seminars, or participate in M.Sc. and Ph.D thesis defenses. Because of the country's small size, these visits can often be done in a day, without requiring an overnight stay or taking a flight.

Israeli AI conferences and workshops have a very important role in creating the unique atmosphere of the Israeli AI community. They are organized by IAAI in an attempt to strengthen the interaction and collaboration between the various research groups. IAAI symposia are held once or twice a year in various research institutions in the country, where the different AI groups present their recent achievements to the entire Israeli AI community and have valuable time to meet and collaborate. Special attention is given to students and to young faculty members, thus affording them an opportunity to introduce themselves to the entire community. A special slot is reserved for young students who had papers accepted to one of the top tier international AI conferences. Such students then have the opportunity to present their talk to the smaller domestic audience. Roughly 100 researchers participate in these events, enabling them to familiarize themselves with the latest achievements of their peers from around the country, and enjoy the very warm atmosphere the IAAI events afford. Figure 1 was taken during the last IAAI symposium at Ben-Gurion University, on January 25, 2016. The focus of the symposium was Israeli AI and cyber security.

Ever since 1989, IAAI has also held the biannual Bar-Ilan Symposium on the Foundation of Artificial intelligence (BISFAI), which is traditionally hosted and organized by the devoted AI group of Bar-Ilan University. BISFAI usually lasts for a couple of days in June. As a mini AI conference, BISFAI holds sessions on all areas of AI, including poster sessions, panels, tutorials, and invited speakers both from inside and outside of Israel. The program committees for all these events select high-quality papers for presentation, but the papers are not archived. As a result, most of them later appear in top tier AI venues.

Relations with the International AI Community

Many Israeli AI researchers have strong collaborative relationships with other research groups and colleagues abroad; they coauthor many papers and jointly work on many research projects. Many Israelis spend the summer or at least a few weeks per year with their collaborators abroad. Naturally, the international colleagues always receive a warm invitation to visit the respective Israeli group; many of them indeed do come for a visit, often also taking a side visit to one or more of the other universities and enjoying a day of sightseeing.

As any other healthy AI community, Israeli AI researchers exercise important responsibilities in International AI organizations, in reviewing papers and in organizing and participating in all levels of program committees of conferences.

The Latest International Conferences in Israel

In summer 2015, the Israeli AI community had the honor of hosting two international conferences: ICAPS-2015, which took place on June 7–11, 2015, in downtown Jerusalem, and SoCS-2015, which was held on June 11–13 in Ein-Gedi, an oasis on the shore of the Dead Sea. The two conferences had a joint session in Jerusalem on June 11, which included selected papers from both conferences and a great keynote talk by Stuart Russell on effective decision making. Aside from the great technical programs, attendees of these conferences were also able to also enjoy a number of social events. The 163 ICAPS participants were treated to a reception at the Tower of David in the old city followed by a spectacular sound and lights show, a walking tour of the old city of Jerusalem, and a banquet in a unique restaurant overlooking the old city of Jerusalem. The 62 SoCS participants were taken on a guided tour of Masada—an ancient fortification situated on top of an isolated mesa—which dates back to the first century BCE. Some of the attendees even climbed up by foot instead of taking the cable car (figure 2).

Thirty-three individuals attended both SoCS and ICAPS and had a full week of activities. This is great evidence for the strong connection between the search and planning communities.

Based on the strong success of these two conferences, IAAI is considering placing a bid to host larger AI conferences.

The Latest Israeli AI Achievements

There are many AI groups in Israel. This section overviews a few representative AI projects that took place in Israel recently, primarily focusing on the projects of young Israeli scientists.

Voting Systems for Bounded Rational Agents

Reshef Meir (Technion) and Yaakov (Kobi) Gal (Ben-Gurion University) are designing and analyzing voting systems for bounded rational agents. They combine tools from computational social choice with human-computer decision making to understand the voting behavior of human participants interacting with these systems.

Together with Maor Tal, a graduate student, they have implemented voting games that replicate two common real-world scenarios of group decision making (Tal, Meir, and Gal 2015). In the first, a single voter votes once after seeing a large preelection poll. In the second game, several voters play simultaneously and change their votes as the game progresses, as in small committees. The payment for participants is determined based on the candidate who has the most

Figure 1. The IAAI Smposium at Ben-Gurion University, January 25, 2016.

Moshe Vardi from Rice University in his invited talk on SAT solvers. *Inset:* Coffee break.

votes in the end. Thus far the voting behavior of hundreds of participants was recorded, both in the lab and through Amazon Mechanical Turk.

Analysis of the data reveals that people can be classified into at least three groups, two of which are not engaged in any strategic behavior. The third and largest group tends to select the natural default action when there is no clear strategic alternative. When an active strategic decision can be made that improves their immediate payoff, people usually choose that strategic alternative. The study provides insight for multiagent system designers in uncovering patterns that provide reasonable predictions of voters behaviors, which may facilitate the design of agents that support people or act autonomously in voting systems.

A chief contribution in this work is the release of VoteLib,[2] a database that contains all the collected data that is freely available to the research community. VoteLib allows researchers to test their own theories and train their models without incurring the overhead of collecting the data and will advance research in AI and computational social choice. The database is currently being extended to support large-scale experiments including potentially thousands of voters.

Bitcoin Research at the Hebrew University

Aviv Zohar and Jeffrey S. Rosenschein from the Hebrew University in Jerusalem have recently explored several issues related to the Bitcoin network. Bitcoin is supported by a P2P network of nodes that are spread around the globe. Each node is an economically driven agent that works with its peers together to authorize transactions in the currency and to maintain the infrastructure required for the system's operation. The Bitcoin protocol awards these participants (which are often called miners) with rewards in exchange for investing computational effort in securing the system and processing transactions.

Work in the group on this topic has focused on the incentives of miners to act according to the rules of the system (Lewenberg et al. 2015), on mechanisms that utilize incentives to increase the transaction throughput of Bitcoin (Sapirshtein, Sompolinsky, and Zohar 2015), and on the use of cooperative game-theoretic models to analyze which coalitions of miners will form and how they will distribute rewards (Lewenberg, Sompolinsky, and Zohar 2015).

The underlying consensus mechanisms of Bitcoins can in many ways be explained as a voting process through which miners coordinate their actions and agree on the same set of accepted transactions. The group is currently exploring connections between computational social choice and cryptocurrencies. This research direction has interesting implications: it yields alternative mechanisms to the ones used by Bitcoin that can allow the system to scale to reach higher transaction volumes and faster processing times.

Additional work is focused on the creation of the P2P overlay network itself. Bitcoin's network must allow its nodes to broadcast messages to their peers, but the lack of strong identities in the system implies that it is difficult for nodes to tell if they are indeed connected to other honest participants or if they are only connected to attackers that have created multi-

Figure 2. SoCS-2015 Participants on a Guided Tour of Masada.
The intrepid participants climbed up to the top of Masada. *Inset:* Those who took the cable car waited for them.

ple fake identities. Indeed, Bitcoin is vulnerable to attacks on its overlay infrastructure (Heilman et al. 2015), and the team is exploring ways to model network formation as a game that is played between nodes and their attacker.

Teams of Robots at Bar-Ilan University

A primary strength of the AI group at Bar Ilan University (BIU) is intelligent robotics, in particular the study of teamwork in multirobot and multiagent systems (Gal Kaminka), adversarial robotics (Noa Agmon), and human interaction with multiple agents and robots (David Sarne, Sarit Kraus). Kaminka and Agmon's research on multirobot patrolling, exploration, and formations was highly publicized; some of it resulted in patents and technology transfer projects, as well as two spin-off startups.

Over the last six years the BIU team have combined their expertise into a project that allows a single human operator to interact effectively with a team of robots to conduct an urban search-and-rescue operation. Specifically, the goal is to combine the best technology in robotics, teamwork, agent-based, and intelligent interfaces to alleviate the cognitive load on the operator in this important task. Unlike competing projects outside of Israel, the assumption is of a quick setup, with only a single operator working from a laptop.

Initially, the project focused on intelligent user interfaces and multirobot collaboration in the mapping process. One specific challenge in the user interface rose because the initial requirement was for the human operator to do the visual search — the robots were there just to provide videos. This meant that an operator was still required to watch all imagery taken by all the robots. Kosti, Kaminka, and Sarne (2014) developed a novel user interface that allowed the user to view images in context of their location on the map, automatically selecting the best image showing a given location.

The focus shifted, however, to how a mediating agent can assist in the visual search process, drawing the operator's attention to robots requiring assistance, and to locations suspect of containing potential victims. Here, the agent uses learning to optimize what advice to provide to the operator, and when (Rosenfeld et al. 2015).

Automated Negotiation at Ariel University

One of the recent research efforts in the newly founded Ariel University was on the problem of automated negotiation with the development of DoNA, a domain-based automated negotiator that won second place out of 21 teams in the 2015 Automated Negotiation Agent Competition (ANAC) (Erez and Zuckerman 2016). ANAC is a yearly competition that pits automated negotiation agents against one another in a series of negotiation sessions with various parameters and rules.

DoNA is extremely simple: while most agents in the competition today are using complex strategies, learning algorithms, and opponent modeling techniques, DoNA looks only at two domain parameters: the reservation value and the time discount factor (hence, the name of DoNA is domain-based agent). DoNA uses these two values in order to decide between four heuristics that are based on cognitive behavior in negotiations. DoNA does not do any optimization or learning nor any form of opponent modeling, yet it managed to beat all the opponents in the 2013 and 2014 competition. DoNA was also enrolled in the 2015 competition (which also had nonlinear domains for the first time), and it managed to attain the second place in the individual utility category, with statistically insignificant difference from the first place.

A joint team from Ariel, Bar-Ilan, and the Jerusalem College of Technology (JCT) led by Inon Zuckerman, Noa Agmon, and Avi Rosenfed developed NegoChat, the first chat-based automated negotiation agent (Rosenfeld et al. 2014). The project started with an effort to adopt the state-of-the-art human-agent automated negotiator from a menu-based interface to a chat-based interface. This required a new negotiation strategy as humans who are using a chat interface tend to negotiate in a step-by-step manner and not deal on the complete set of issues (Zuckerman et al. 2015). With that in mind, the group developed NegoChat (Rosenfeld et al. 2014), and later an extension by the name of NegoChat-A (Rosenfeld et al. 2016), which deviates from the rational Pareto-optimal, and offers strategy to a new cluster-based strategy that constructs the offers in a stepwise manner inside its current cluster of possible offers. This leads to offers that are more successful when negotiating with humans who are using chat-based interfaces.

Summary

This column gave a brief encounter with the Israeli AI community and some of the research that was conducted recently. The Israeli AI community hopes to continue with its traditions and events and hopes to continue to be a solid member in the international AI community. On top of all, the Israeli AI community hopes to continue and conduct high-quality research in the field of AI.

Notes

1. www.ise.bgu.ac.il/iaai/.
2. www.votelib.org.

References

Erez, E. S., and Zuckerman, I. 2016. Dona — A Domain-Based Negotiation Agent. In *Next Frontier in Agent-Based Complex Automated Negotiation,* Studies in Computational Intelligence, volume 638, ed. N. Fukuta, T. Ito, M. Zhang, K. Fujita, and V. Robu. Berlin: Springer.

Heilman, E.; Kendler, A.; Zohar, A.; and Goldberg, S. 2015. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. Paper presented at the 24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12–14.

Kosti, S.; Kaminka, G. A.; and Sarne, D. 2014. A Novel User-Guided Interface for Robot Search. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems.* Piscataway, NJ: Institute for Electrical and Electronics Engineers. dx.doi.org/10.1109/IROS.2014.6943022

Lewenberg, Y.; Bachrach, Y.; Sompolinsky, Y.; Zohar, A.; and Rosenschein, J. S. 2015. Bitcoin Mining Pools: A Cooperative Game Theoretic Analysis. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015,* 919–927. New York: Association for Computing Machinery.

Lewenberg, Y.; Sompolinsky, Y.; and Zohar, A. 2015. Inclusive Block Chain Protocols. In *Financial Cryptography and Data Security — 19th International Conference, FC 2015, Revised Selected Papers,* 528–547. Berlin: Springer. dx.doi.org/10.1007/978-3-662-47854-7_33

Rosenfeld, A.; Agmon, N.; Maksimov, O.; Azaria, A.; and Kraus, S. 2015. Intelligent Agent Supporting Human-Multirobot Team Collaboration. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015,* 1902–1908. Palo Alto, CA: AAAI Press.

Rosenfeld, A.; Zuckerman, I.; Segal-Halevi, E.; Drein, O.; and Kraus, S. 2014. Negochat: A Chat-Based Negotiation Agent. In *International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS-14,* ed. A. Bazzan, M. Huhns, A. Lomuscio, and P. Scerri, 525–532. New York: Association for Computing Machinery.

Rosenfeld, A.; Zuckerman, I.; Segal-Halevi, E.; Drein, O.; and Kraus, S. 2016. Negochat-A: A Chat-Based Negotiation Agent with Bounded Rationality. *Autonomous Agents and Multi-Agent Systems* 30(1): 60–81. dx.doi.org/10.1007/s10458-015-9281-9

Sapirshtein, A.; Sompolinsky, Y.; and Zohar, A. 2015. Optimal Selfish Mining Strategies in Bitcoin. In *Financial Cryptography and Data Security — 20th International Conference, FC 2016, Revised Selected Papers.* Berlin: Springer.

Tal, M.; Meir, R.; and Gal, Y. K. 2015. A Study of Human Behavior in Online Voting. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015,* 665–673. New York: Association for Computing Machinery.

Zuckerman, I.; Segal-Halevi, E.; Rosenfeld, A.; and Kraus, S. 2015. First Steps in Chat-Based Negotiating Agents. In *Next Frontier in Agent-Based Complex Automated Negotiation,* Studies in Computational Intelligence Volume 596. 89–109. Berlin: Springer. dx.doi.org/10.1007/978-4-431-55525-4_6

Ariel Felner received his Ph.D. in 2002 from Bar-Ilan University, Israel, and is now an associate professor at the Information Systems Engineering Department, Ben-Gurion University, Israel. He is the chair of the Israeli Association for Artificial Intelligence (IAAI) and a council member of the Symposium of Combinatorial Search (SoCS). He is interested in all aspects of heuristic search in AI.

AAAI News

Fall News from the Association for the Advancement of Artificial Intelligence

AAAI Returns to the City by the Bay in 2017!

The Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17) and the Twenty-Ninth Conference on Innovative Applications of Artificial Intelligence (IAAI-17) will be held February 4–9 at the Hilton San Francisco Union Square in San Francisco, California, USA. San Francisco is a world-class city full of delights for every visitor. From the iconic Golden Gate bridge to its renowned art galleries, science museums, and picturesque neighborhoods, the City takes pride in its unrivaled attractions and treasures. The Hilton Hotel is located near Union Square and very close to shops, restaurants, and entertainment venues, as well as public transportation. The conference location is a great starting point to explore the City's tremendous ethnic and cultural diversity and its wide variety of offerings. San Francisco is also perfectly positioned to explore the entire Bay Area, whether for recreation or business. The Napa Valley, Pacific Coast, Silicon Valley, and East Bay are all a short distance away. As an unrivaled center for AI and tech professionals, San Francisco promises to be a vibrant home for AAAI-17. For local information, please visit the San Francisco travel site at www.sftravel.com.

Please note these upcoming deadlines for AAAI-17:

Doctoral Consortium

September 21: Applications due

The Doctoral Consortium (DC) provides an opportunity for a group of Ph.D. students to discuss and explore their research interests and career objectives with a panel of established researchers in artificial intelligence. The consortium's objectives are to provide a setting for mutual feedback on participants' current research and guidance on future research directions; to develop a supportive community of scholars and a spirit of collaborative research; to support a new generation of researchers by offering advice about academic, research, industrial, and non-traditional career paths; and, finally, to contribute to the overall conference goals through interaction with other researchers and participation in conference events.

Student Abstract and Poster Program

September 30: Abstracts due

The goal of the AAAI-17 Student Abstract and Poster program is to provide a forum in which students can present and discuss their work during its early stages, meet some of their peers who have related interests, and introduce themselves to more senior members of the field. The program is open to all students at the undergraduate, masters, and doctoral levels.

Workshop Program

October 21: Submissions due

The AAAI-17 workshop program includes approximately 20 workshops covering a wide range of topics in artificial intelligence. The AAAI-17 Workshop Call for Participation is now available at www.aaai.org/Workshops/ws17.php. Workshop submissions are due October 21, unless otherwise noted at the individual workshop website. Submission requirements vary for each workshop. Please consult the individual workshop description for complete information about where to submit your paper and a link to the workshop supplementary website, where more detailed information will be available.

Demonstrations Program

October 21: Short papers and video due

The AAAI-17 Demonstrations Program is intended to foster discussion and exchange of ideas among researchers and practitioners from academe and industry by presenting software and hardware systems and research prototypes of such systems, including their capabilities and workings. Accepted demonstrations will be allocated one time slot during one of the main conference evening poster programs, and will have a short paper included in the proceedings. Submissions from everyone, including authors of paper submissions to AAAI, IAAI, and AAAI-17 workshops, are encouraged. Work submitted to other tracks (such as the Robotics Exhibition and the Videos Competition) can also be submitted to the Demonstrations Program, according to the guidelines in the Call for Videos.

AI Video Competition

November 15: Submissions Due

The goal of the AAAI-17 Video Competition is to show the world how much fun AI is by documenting exciting artificial intelligence advances in research, education, and application. View previous entries and award winners at the

AI Videos Past Competitions page (www.aaai videos.org/past_competitions).

For More AAAI-17 Information

Website: www.aaai.org/aaai17

For complete information about AAAI-17, please visit the AAAI-17 website at or write to us at aaai17@aaai.org. The AAAI-17 Program Cochairs are Shaul Markovitch (Technion, Israel) and Satinder Singh (University of Michigan, USA).

The IAAI-17 Conference Chair and Cochair are James Crawford (Orbital Insight, USA) and G. Michael Youngblood (PARC, a Xerox Company, USA).

We hope to see you in San Francisco in February!

AAAI-17 Student Scholar and Volunteer Program

AAAI is pleased to announce the continuation of its Student Scholarship Program for 2017, which is sponsored by the *AI Journal*. The Student Scholar Program provides partial travel support for students who are full-time undergraduate or graduate students at colleges and universities; are members of AAAI; submit papers to the conference program or letters of recommendation from their faculty advisor; and submit scholarship applications to AAAI by November 18, 2016. In addition, repeat scholarship applicants must have fulfilled the volunteer and reporting requirements for previous awards. In the event that scholarship applications exceed available funds, preference will be given to students who have an accepted technical paper, and then to students who are actively participating in the conference in some way. However, all eligible students are encouraged to apply.

The Student Volunteer Program is an essential part of the conference and student participation is a valuable contribution. Volunteers will support AAAI organizers in San Francisco, California. In 2017, a limited number of complimentary technical program registrations will be available for students who volunteer during the conference. Preference will be given to participating students for the volunteer positions. Local students or students not requiring travel assistance can apply for the Volunteer Program if openings are available. AAAI membership is required for eligibility. The deadline for volunteer applications is November 18, 2016.

For further information about the Scholarship Program or the Volunteer Program, or to obtain an application, please contact AAAI at scholarships17@aaai.org.

Join Us for AIIDE in October!

The Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-16) will be held at the Embassy Suites by Hilton San Francisco Airport – Burlingame, in the San Francisco Bay Area, October 8-12, 2016. AIIDE is the definitive point of interaction between entertainment software developers interested in AI and academic and industrial AI researchers. Sponsored by AAAI, the conference is targeted at both the research and commercial communities, promoting AI research and practice in the context of interactive digital entertainment systems with an emphasis on commercial computer and video games.

The tentative line-up of invited speakers for AIIDE-16 includes Katja Hofmann (Microsoft Research), Jonathan Blow (Number ONone), Kevin Dill (Lockheed Martin), and the AIIDE-16/MIG-16 Joint Invited Talk will feature Simon Clavet (UbiSoft Montreal). The program will also include technical paper presentations, a poster and demo session, and the annual Starcraft AI Competition. The Playable Experiences track will continue this year with four systems with articulable innovation in the use of AI directly affecting the user's experience, including *Rogue Process, Elsinore, Conceptually Blended Levels in a Unity Engine,* and *Bad News*. Finally, the workshop program, to be held October 8 and 9, will include the following three workshops: Artificial Intelligence in Adversarial Real-Time Games (W1); Experimental AI in Games (W2); and Player Analytics (W3).

The full conference program and registration information is available at aiide.org. The late registration deadline is September 9. Onsite rates will be in effect after that date. Preregistration is strongly encouraged. The online registration form is available at www.regonline.com/aiide16, and will be open through the conference period. Onsite registration will be held in the foyer of the Embassy Ballroom on the Main Level of the hotel. For more information about registration or hotels in the area, please consult www.aiide.org, or write to aiide16@aaai.org.

HCOMP 2016 Registration Open

Please join us for the Fourth AAAI Conference on Human Computation and Crowdsourcing (HCOMP 2016), to be held October 30 – November 3 at the AT&T Executive Conference Center on the campus at the University of Texas at Austin, Austin, Texas, USA. HCOMP is the premier venue for disseminating the latest research findings on crowdsourcing and human computation. While artificial intelligence (AI) and human-computer interaction (HCI) represent traditional mainstays of the conference, HCOMP believes strongly in inviting, fostering, and promoting broad, interdisciplinary research. This field is particularly unique in the diversity of disciplines it draws upon, and contributes to, ranging from human-centered qualitative studies and HCI design, to computer science and artificial intelligence, economics and the social sciences, all the way to cultural heritage, digital humanities, ethics, and policy. The HCOMP conference is aimed at promoting the exchange of advances in human computation and crowdsourcing among not only researchers, but also engineers and practitioners, to encourage dialogue across a spectrum of disciplines and communities of practice.

The theme for HCOMP 2016 is *interaction* — interaction between people and technology that is foundational to human computation; interaction between theoretical foundations, experimental work, and engineering; interaction between the computational, scientific, and social applications of crowdsourcing; and interaction between diverse disciplines and perspectives, within our community and beyond.

The program will include keynote addresses by Iyad Rahwan (MIT), Nathan Schneider (University of Colorado Boulder), and Ashish Goel (Stanford University). In addition, on the first day of the conference, HCOMP-16 will feature a Doctoral Consortium and a tutorial on Crowdsourced Data Processing: Industry and Academic Perspectives, as well as three workshops: CrowdCamp; Human Computation for Image and Video Analysis (GroupSight); and Mathematical Foundations of Human Computation.

The main technical program has been expanded to include two new tracks on Industry and Practice and Encore Papers. Past tracks on works-in-progress and demos will also continue.

Complete registration and hotel information is available at www.humancomputation.com/2016. The late registration deadline is October 7, 2016. The deadline for hotel reservations at the AT&T Conference Center is October 3, 2016. The discounted room rate of $185 per night (single/double) may not be available after this date.

ICWSM 2017 Comes to Canada

Please join us for the Eleventh International AAAI Conference on Weblogs and Social Media, to be held in Montréal, Québec Canada, May 15–18. 2017. This interdisciplinary conference is a forum for researchers in computer science and social science to come together to share knowledge, discuss ideas, exchange information, and learn about cutting-edge research in diverse fields with the common theme of online social media. This overall theme includes research in new perspectives in social theories, as well as computational algorithms for analyzing social media. ICWSM is a singularly fitting venue for research that blends social science and computational approaches to answer important and challenging questions about human social behavior through social media while advancing computational tools for vast and unstructured data. Full conference details will be posted at www.icwsm.org/2017 as they become available.

AAAI 2016 Fall Symposium Registration

The Association for the Advancement of Artificial Intelligence's 2016 Fall Symposium Series will be held Thursday through Saturday, November 17-19 at the Westin Arlington Gateway, Arlington Virginia, adjacent to Washington, DC. The six symposia are (1) Accelerating Science: A Grand Challenge for AI; (2) Artificial Intelligence for Human-Robot Interaction; (3) Cognitive Assistance in Government and Public Sector Applications;(4) Cross-Disciplinary Challenges for Autonomous Systems; (5) Privacy and Language Technologies; and (6) Shared Autonomy in Research and Practice

An informal reception will be held on Thursday, November 17. A general plenary session, in which the highlights of each symposium will be presented, will be held on Friday, November 18. Symposia generally range from 40–75 participants each. Participation will be open to active participants as well as other interested individuals on a first-come, first-served basis. Each participant will be expected to attend a single symposium. AAAI technical reports will be distributed to participants in electronic format and posted in the AAAI digital library

The final deadline for registration is October 21, 2016. For registration information, please contact AAAI at fss16@aaai.org or visit AAAI's web site at www.aaai.org/Symposia/Fall/fss16.php. A hotel room block has been reserved at the Westin. The cut-off date for reservations is October 17, 2016 at 5:00 pm ET. Please call +1-888-627-7076 (reference AAAI) for reservations, or reserve a room online via the URL liste earlier.

AAAI 2017 Spring Symposium Series

AAAI is pleased to present the 2017 Spring Symposium Series, to be held Monday – Wednesday, March 27–29, 2017, at Stanford University. The eight symposia are as follows:

AI for the Social Good

Eric Horvitz (Microsoft Research), Barbara Grosz (Harvard University), Amy Greenwald (Brown University), David Parkes (Harvard University), Carla Gomes (Cornell University), Stephen Smith (Carnegie Mellon University), Gregory Hager (Johns Hopkins University), Ann W. Drobnis (Computing Research Association), Nicole Sintov (Ohio State University), Milind Tambe (University of Southern California), Amulya Yadav (University of Southern California), Fei Fang (Harvard Universit), Bryan Wilder (University of Southern California)

Computational Construction Grammar and Natural Language Understanding

Luc Steels (Universitat Pompeu Fabra, Spain), Jerome Feldman (International Computer Science Institute)

Computational Context: Why It's Important, What It Means, and Can It Be Computed?

Ranjeev Mittu (Naval Research Laboratory); W.F. Lawless (Paine College); Don Sofge (Naval Research Institute); David Aha (Naval Research Institute)

Designing the User Experience of Machine Learning Systems

Mike Kuniavsky (PARC), Elizabeth Churchill (Google), Molly Wright Steenson (Carnegie Mellon University)

Interactive Multisensory Object Perception for Embodied Agents

Vivian Chu (Georgia Institute of Technology), Jivko Sinapov (University of Texas at Austin), Jeannette Bohg (MPI for Intelligent Systems), Sonia Chernova (Georgia Institute of Technology), Andrea L. Thomaz (University of Texas at Austin)

Learning from Observation of Humans

Santiago Ontañón (Drexel University), Avelino J. González (University of Central Florida), José L. Montaña (University of Cantabria, Spain)

Science of Intelligence: Computational Principles of Natural and Artificial Intelligence

Gemma Roig (Massachusetts Institute of Technology), Xavier Boix (National University of Singapore)

Well-Being AI: From Machine Learning to Subjective Oriented Computing

Takashi Kido (Rikengenesis, Japan), Keiki Takadama, (The University of Electro-Communications, Japan)

For additional information, and links to the supplementary websites for each symposium, please see www.aaai.org/Symposia/Spring/ sss16.php. Submissions for the symposia are due on October 28, 2016. Notification of acceptance will be given by November 29, 2016. Material to be included in the technical reports of the symposium must be received by January 25, 2017. Registration information will be available by December 15, 2016. Please contact AAAI at sss17@aaai.org with any questions.

AAAI Fellows Nominations Solicited

The 2017 Fellows Selection Committee is currently accepting nominations for AAAI Fellow. The AAAI Fellows program is designed to recognize people who have made significant, sustained contributions to the field of artificial intelligence over at least a ten-year period. All regular members in good standing are encouraged to consider nominating a candidate. At least two references must accompany nominations. The nominator or one of the references must be a AAAI Fellow who is a current member of AAAI. For further information about the Fellows Program or to receive nomination and reference forms, please contact AAAI at 650-328-3123; by fax at 650-321-4457; or by email at fellows17@aaai.org. Nomination materials are also available on the AAAI web site at www.aaai.org/Awards/fellows.php. The deadline for nominations is September 30, 2016.

The AAAI Feigenbaum Prize

The Feigenbaum Prize Selection Committee is currently accepting nominations for the 2017 Feigenbaum Prize. The AAAI Feigenbaum Prize is awarded biennially to recognize and encourage outstanding Artificial Intelligence research advances that are made by using experimental methods of computer science. The "laboratories" for the experimental work are real-world domains, and the power of the research results are demonstrated in those domains. The Feigenbaum Prize may be given for a sustained record of high-impact seminal contributions to experimental AI research; or it may be given to reward singular remarkable innovation and achievement in experimental AI research. The prize is $10,000 and is provided by the Feigenbaum Nii Foundation and administered by AAAI. All nominations must be accompanied by at least one letter of support. For further information about the AAAI Feigenbaum Prize and to access the electronic nomination form, please visit www.aaai.org/Awards/feigenbaum.php The deadline for nominations is September 30, 2016.

2017 AAAI Special Award Nominations

AAAI is pleased to announce the continuation of its two special awards in 2017, and is currently seeking nominations for the 2017 AAAI Classic Paper Award, and the AAAI Distinguished Service Award. The 2017 AAAI Classic Paper Award will be given to the author of the most influential paper(s) from the Sixteenth National Conference on Artificial Intelligence, held in 1999 in Orlando, Florida. The 2017 AAAI Distinguished Service Award will recognize one individual for extraordinary service to the AI community. Awards will be presented at AAAI-17 in San Francisco, California, USA. Complete nomination information, including nomination forms, is available at www.aaai.org/Awards/fellows.php. The deadline for nominations is September 30, 2016. For additional inquiries, please contact Carol Hamilton at hamilton@aaai.org.

2017 AAAI/EAAI Outstanding Educator Award

The AAAI/EAAI Outstanding Educator award was created in 2016 to honor a person (or group of people) who has made major contributions to AI education that provide long-lasting benefits to the AI community. Examples include innovating teaching methods, providing service to the AI education community, generating pedagogical resources, designing curricula, and educating students outside of higher education venues (or the general public) about AI. The award includes a $1,000 honorarium, complimentary one-year AAAI membership (new or renewal), and a complimentary conference registration to the 2017 EAAI/AAAI conferences, where the award will be conferred.

Complete nomination information is available at www.aaai.org/Awards/eaai-call.php. The deadline for nominations is October 17, 2016. For additional inquiries, please contact avid Poole (eaaiaward@aaai.org).

AAAI Senior Member Grade of Membership

AAAI is now taking applications from regular members for the AAAI Senior Member grade of membership. This status is designed to recognize members who have achieved significant accomplishments within the field of Artificial Intelligence. To be eligible for nomination for Senior Member, candidates must be consecutive members of AAAI for at least five years and have been active in the professional arena for at least ten years. Applications should include information that details the candidate's scholarship, leadership, and professional service. At least two references, one of which must be written by a AAAI Fellow or a current AAAI Senior Member must accompany the senior member application. References should be submitted by colleagues who know the candidate, and are familiar with their work and accomplishments.

Each year a maximum of 25 members will be elected to the Senior status. All applications and references must conform to the requirements listed on the form, and must be received by September 30, 2016.

For complete details and an application form, please see www.aaai.org/Awards/senior.php, or contact Carol Hamilton at seniormember16@aaai.org.

AAAI Elects New President-Elect and Councilors

Incoming AAAI President Subbarao Kambhampati (left) is presented with his new gavel by outgoing AAAI president Thomas G. Dietterich (right) at the July 2017 AAAI Executive Council Meeting in New York City.

AAAI is pleased to announce and congratulate the continuing slate of officers and new councilors for the AAAI Executive Council.

President: Subbarao Kambhampati (Arizona State University, USA)
Past President: Thomas G. Dietterich (Oregon State University, USA)
President-Elect: Yolanda Gil (USC Information Sciences Institute, USA)
Secretary-Treasurer: Ted Senator

Incoming Councilors (through 2019)

Blai Bonet (Universidad Simón Bolívar, Venezuela)
Mausam (Indian Institute of Technology Delhi, India)
Michela Milano (Università di Bologna, Italy)
Qiang Yang (Hong Kong University of Science and Technology, Hong Kong)

Eight existing Executive Council members, elected in 2015 and 2016, will continue their terms of service during the coming year. For a complete list, please refer to www.aaai.org/Organization/officers.php.

AAAI also thanks the four retiring Councilors, and retiring Past President, Manuela Veloso (Carnegie Mellon University, USA), for their dedicated service and generous donations of time. The four retiring officers are Sven Koenig (University of Southern California, USA), Sylvie Thiebaux (NICTA, Australia), Francesca Rossi (University of Padua, Italy), and Brian Williams (Massachusetts Institute of Technology).

First Call for Nominations for Executive Councilors

Every year four new councilors are elected to serve three-year terms on the AAAI Executive Council. All elected councilors are expected to attend at least two council meetings per year, and actively participate in AAAI activities. Nominees must be current members of AAAI. The Nominating Committee encourages all regular members in good standing to place an individual's name before them for consideration. (Student and library members are not eligible to submit candidates' names.) The Nominating Committee, in turn, will nominate eight candidates for councilor in the spring. In addition to members' recommendations, the committee will actively recruit individuals in order to provide a balanced slate of candidates. AAAI members will vote in the late spring.

To submit a candidate's name for consideration, please send the following information to
Carol Hamilton, Executive Director, AAAI, 2275 East Bayshore Road, Suite 160, Palo Alto, CA 94303; by fax to 650/321-4457; or by email to hamilton@aaai.org:

Name
Affiliation
City, State or Province, Country
Email address
URL
Year of membership in AAAI
Approximate number of AAAI publications
At least two sentences describing the candidate and why he would be a good candidate

Please include any additional information or recommendations that would be helpful to the Nominating Committee. Nominators should contact candidates prior to submitting their names to verify that they are willing to serve, should they be elected. The deadline for nominations is December 15, 2016.

AAAI Conferences Calendar

This page includes forthcoming AAAI sponsored conferences, conferences presented by AAAI Affiliates, and conferences held in cooperation with AAAI. AI Magazine *also maintains a calendar listing that includes nonaffiliated conferences at www.aaai.org/Magazine/calendar.php.*

AAAI Sponsored Conferences

Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. AIIDE-16 will be held October 8–12 in San Francisco, California, USA.

URL: aiide.org

Fourth AAAI Conference on Human Computation and Crowdsourcing. HCOMP-16 will be held October 30–November 3 in Austin, Texas, USA.

URL: humancomputation.com

AAAI Fall Symposium. The AAAI Fall Symposium Series will be held November 17–19 in Arlington, Virginia, adjacent to Washington DC USA.

URL: www.aaai.org/Symposia/Fall/fss16.php

Thirty-First AAAI Conference on Artificial Intelligence. AAAI-17 will be held February 4–9 in San Francisco, California USA.

URL: www.aaai.org/aaai17

Twenty-Ninth Innovative Applications of Artificial Intelligence Conference. IAAI-17 will be held February 4–9 in San Francisco, California USA.

URL: www.aaai.org/iaai17

AAAI Spring Symposium Series. The AAAI 2017 Spring Symposium Series will be held March 27–29, 2017, at Stanford University adjacent to Palo Alto, CA USA.

URL: www.aaai.org/Symposia/Spring/sss17.php

Twelfth International AAAI Conference on Web and Social Media. ICWSM-17 will be held May15–18 in Montréal, Québec, Canada.

URL: www.icwsm.org/2017

The Thirty-Second AAAI Conference on Artificial Intelligence. AAAI-18 will be held February 4–10 at the Hilton New Orleans Riverside Hotel, New Orleans, Louisiana USA.

URL: www.aaai.org/aaai18

Thirtieth Innovative Applications of Artificial Intelligence Conference. The IAAI-18 Conference will be held February 4–10 at the Hilton New Orleans Riverside Hotel, New Orleans, Louisiana USA.

URL: www.aaai.org/iaai18.php

Conferences Held by AAAI Affiliates

The 16th International Conference on Autonomous Agents and Multiagent Systems. AAMAS 2017 will be held May 8–12, 2017 in São Paulo, Brazil.

URL: aamas2017.org

Thirtieth International Florida AI Research Society Conference. FLAIRS-2017 will be held May 22–24, 2017 inMarco Island, Florida, USA.

URL: www.flairs-30.info

The 27th International Conference on Automated Planning and Scheduling. ICAPS-17 will be held June 18–23, 2017 in Pittsburgh, PA USA.

URL: icaps17.icaps-conference.org

Conferences Held in Cooperation with AAAI

2016 IEEE International Conference on Systems, Man and Cybernetics. SMC2016 will be held October 9–12, 2016 in Budapest, Hungary.

URL: smc2016.org

8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management. IC3K 2016 will be held 9–11 November 2016 in Porto, Portugal
www.ic3k.org

Eighth International Joint Conference on Computational Intelligence. IJCCI 2016 will be held 9-11 November, 2016 in Porto, Portugal

URL: www.ijcci.org

Human, Machines, and the Future of Work. De Lange Conference X will be held December 5–6, 2016 at Rice University in Houston, Texas, USA

URL: delange.rice.edu/conference_X

9th International Conference on Agents and Artificial Intelligence. ICAART 2017 will be held 24-26 February, 2017 in Porto, Portugal

URL: www.icaart.org

6th International Conference on Pattern Recognition Applications and Methods. ICPRAM 2017 will be held 24-26 February, 2017 in Porto, Portugal

URL: www.icpram.org

The 16th International Conference on Artificial Intelligence and Law. ICAIL 2017 will be held 12-16 June, 2017 in London, UK

URL: nms.kcl.ac.uk/icail2017

Thirtieth International Conference on Industrial, Engineering, and Other Applications of Applied Intelligent Systems. IEA/AIE-2017 will be held June 17–21, 2017 in Arras, France.

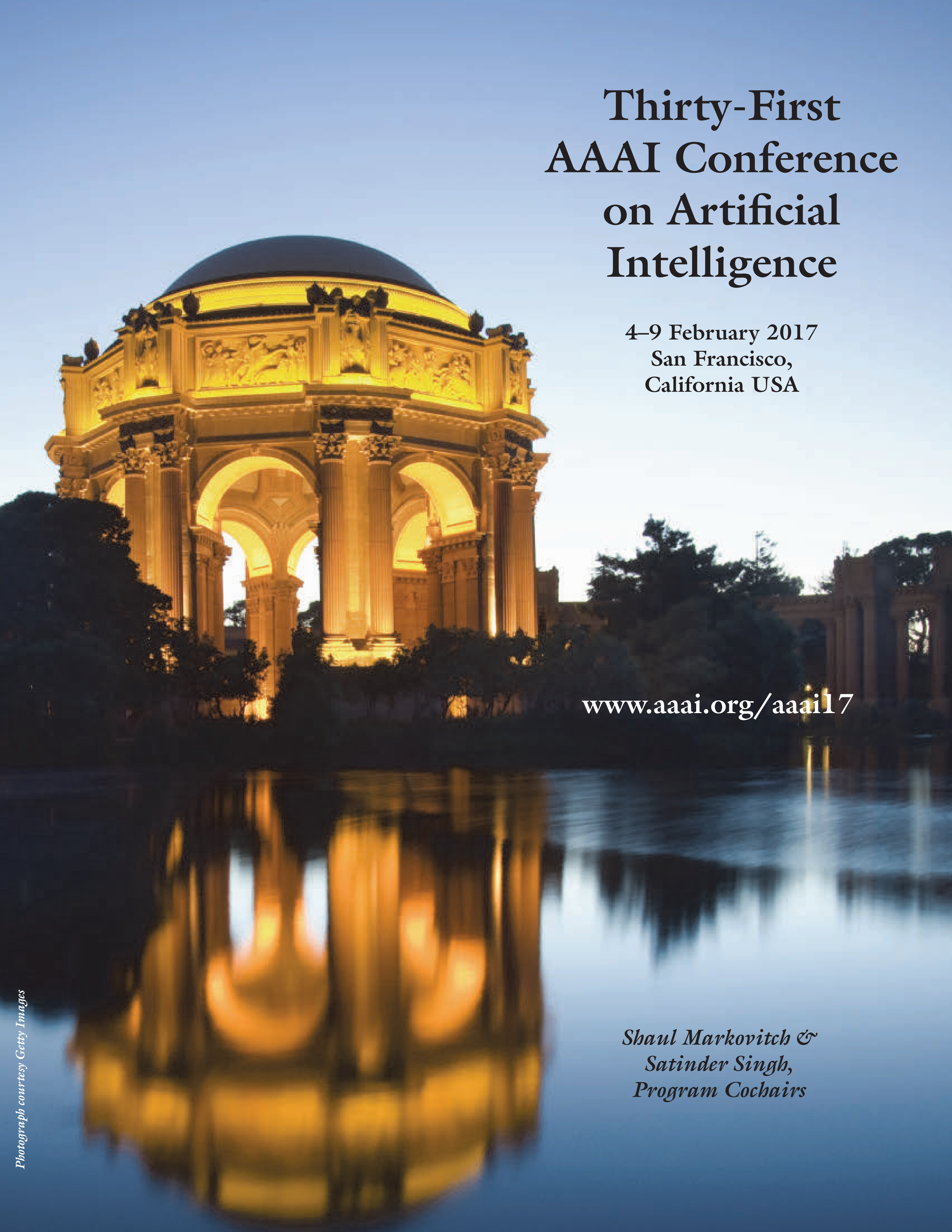
URL: www.ieaaie2017.org

Please Join Us for the Fourth AAAI Conference on Human Computation and Crowdsourcing

October 30–November 3 2016

Austin Texas, USA

www.humancomputation.com/2016/



Thirty-First AAAI Conference on Artificial Intelligence

4–9 February 2017
San Francisco,
California USA

www.aaai.org/aaai17

Shaul Markovitch &
Satinder Singh,
Program Cochairs

Photograph courtesy Getty Images