# SAT-to-SAT: Declarative Extension of SAT Solvers with New Propagators

**Tomi Janhunen** and **Shahab Tasharrofi**
{tomi.janhunen,shahab.tasharrofi}@aalto.fi
Helsinki Institute for Information Technology HIIT
Department of Computer Science
Aalto University, FI-00076 AALTO, Finland

**Eugenia Ternovska**
ter@cs.sfu.ca
Computing Science Department
Simon Fraser University, Burnaby, BC, Canada

## Abstract

Special-purpose propagators speed up solving logic programs by inferring facts that are hard to deduce otherwise. However, implementing special-purpose propagators is a non-trivial task and requires expert knowledge of solvers.

This paper proposes a novel approach in logic programming that allows (1) *logical specification of both the problem itself and its propagators* and (2) *automatic incorporation of such propagators into the solving process*. We call our proposed language $P[R]$ and our solver SAT-to-SAT because it facilitates communication between several SAT solvers.

Using our proposal, non-specialists can specify new reasoning methods (propagators) in a declarative fashion and obtain a solver that benefits from both state-of-the-art techniques implemented in SAT solvers as well as problem-specific reasoning methods that depend on the problem's structure. We implement our proposal and show that it outperforms the existing approach that only allows modeling a problem but does not allow modeling the reasoning methods for that problem.

## Introduction

**Motivation.** *Propagation mechanisms*, i.e., reasoning methods that produce new information about a problem, are at the heart of modern solver architectures. Different scientific communities that develop state-of-the-art solvers such as SAT, ASP and CP solvers (Een and Sörensson 2005; Gebser, Kaufmann, and Schaub 2012; de la Banda et al. 2006); have long understood the need for methods to extend the existing propagation mechanisms of their solvers. For example, modern SAT solvers such as MiniSAT and its descendants (Een and Sörensson 2005; Audemard and Simon 2009) can be easily extended with new special-purpose propagation mechanisms (Gebser, Janhunen, and Rintanen 2014c). As another example, the ASP solver clasp (Gebser, Kaufmann, and Schaub 2012) provides a "post propagator" class designed to implement new inference mechanisms.

Due to the success of propagators in modern solvers, recently, many new *domain-specific propagation mechanisms* have been proposed. That is, existing solvers are extended with new mechanisms that are applicable only when the problem has a specific structure. In this sense, such extended solvers are *less general* (they apply only to problems with certain structures) but *more efficient* (for such problems).

To emphasize the importance of domain-specific propagation mechanisms, observe a few of such recent works: (1) In (Bjørner 2012), the author extends Z3 Satisfiability Modulo Theories (SMT) solver (De Moura and Bjørner 2008) with several new domain-specific reasoning mechanisms (called theories in SMT solving community). (2) In (Liang et al. 2014), the authors extend CVC4 SMT solver (Barrett et al. 2011) (or any other DPLL(T) solver (Nieuwenhuis, Oliveras, and Tinelli 2004) in general) with reasoning mechanisms and propagation methods for the theory of strings and regular expressions. (3) In (Gebser, Janhunen, and Rintanen 2014b; 2014c), the authors introduce several propagation methods to avoid cyclic graphs. They show that their reasoning mechanisms hugely boost the performance of SAT solvers on the problems whose solutions are acyclic graphs. (4) Finally, in (Gebser, Janhunen, and Rintanen 2014a), the authors use their acyclicity reasoning methods to solve answer set programs (ASP) using SAT solvers.

**Problem.** Despite the effectiveness of special-purpose propagators, developing new propagators still requires expert knowledge of how a general-purpose solver works and how one can extend such solvers with new reasoning methods. Therefore, currently, when solving a declaratively modeled problem, one is limited only to the general-purpose inference mechanisms provided by the solver and *it is impossible to extend solvers with complex reasoning techniques that take advantage of the problem's structure*.

**Goals.** In this paper, we introduce a framework to (1) *declaratively specify new reasoning methods* and (2) *automatically extend modern solvers with such reasoning methods* so as to obtain fast problem-specific solvers.

Since SAT solvers form the core of many different state-of-the-art logic programming frameworks such as (Aavani et al. 2012; Wittocx, Marién, and Denecker 2008; De Moura and Bjørner 2008; Barrett et al. 2011), this paper focuses on specifying new reasoning methods for modern conflict-driven clause learning (CDCL) SAT solvers. However, this paper's methods also apply to other modern solvers such as ASP solvers. Extending our results to other logic programming frameworks such as Clingo (Gebser et al. 2008) is a subject of future research.

**Contributions.** *New Paradigm in Logic Programming:* We

introduce the syntax and semantics needed to declaratively specify new domain-specific reasoning methods as well as the transformations required to incorporate such new reasoning methods into the solving process. This way, we introduce *a new paradigm in logic programming in which logical statements are used to describe both the problem and the reasoning methods available to solve that problem.*
*Automatic Generation of Problem-specific Solvers:* We provide all the algorithms required to extend existing CDCL SAT solvers with new declaratively specified reasoning methods. We also prove the correctness of our algorithms.
*Complexity and Expressiveness:* We study the expressiveness of our framework and show that we can express and solve all $\Sigma_2^P$ problems (2nd level of polynomial hierarchy).
*Experimental Results:* We implement our methods on top of Glucose (Audemard and Simon 2009) and show that introducing new reasoning mechanisms hugely boosts the solving time of a problem. Thus, we show that our new paradigm to solve problems by also describing their reasoning methods outperforms the existing ground and solve paradigm.

## Background

This section briefly reviews the background material that is used in this paper.
**Structures and Partial Structures.** Let $\tau$ be a set of vocabulary symbols. A *$\tau$-structure $\mathcal{A}$* consists of a domain, denoted by $dom(\mathcal{A})$, plus interpretations of vocabulary symbols $R \in \tau$, denoted by $R^{\mathcal{A}}$. Also, for $\sigma$-structure $\mathcal{A}$ and vocabulary symbols $\sigma' \subseteq \sigma$, *restriction of $\mathcal{A}$ to vocabulary* $\sigma'$, denoted by $\mathcal{A}|_{\sigma'}$, is obtained by discarding interpretation of vocabulary symbols in $\sigma \setminus \sigma'$. Now, $(\sigma \cup \varepsilon)$*-structure $\mathcal{B}$ expands $\sigma$-structure $\mathcal{A}$* if $\mathcal{B}|_\sigma = \mathcal{A}$.

*Partial $\tau$-structures* are similar to $\tau$-structures except that they may allow values in their interpretations. That is, partial interpretation $\mathcal{R}$ might assign tuple $t$ as: (1) true (denoted by $t \in \mathcal{R}$), (2) false (denoted by $t \notin \mathcal{R}$), or (3) unknown (denoted by $t \overset{?}{\in} \mathcal{R}$). Partial structures without unknown values are structures. We also use $\mathcal{S}(\sigma, \varepsilon)$ to denote all partial $(\sigma \cup \varepsilon)$-structures $\mathcal{B}$ with $\mathcal{B}|_\sigma$ not being partial (i.e., $\mathcal{B}|_\sigma$ interprets all tuples as true or false and not as unknown).

Now, for partial $\tau$-structures $\mathcal{B}$ and $\mathcal{B}'$, we say that $\mathcal{B}'$ *extends* $\mathcal{B}$, denoted by $\mathcal{B} \sqsubseteq \mathcal{B}'$, if $\mathcal{B}'$ and $\mathcal{B}$ agree on all known values in $\mathcal{B}$. That is, $t \in R^{\mathcal{B}}$ implies $t \in R^{\mathcal{B}'}$ and $t \notin R^{\mathcal{B}}$ implies $t \notin R^{\mathcal{B}'}$. Moreover, for $\sigma$-structure $\mathcal{A}$, partial $(\sigma \cup \varepsilon)$-structure $\mathcal{B}$ is an *empty expansion of* $\mathcal{A}$ if $\mathcal{B}$ expands $\mathcal{A}$ but has no new information. That is, $t \overset{?}{\in} R^{\mathcal{B}}$ if $R \in \varepsilon$.
**Model Expansion.** For logic $\mathcal{L}$, the model expansion (MX) task is as follows (Mitchell and Ternovska 2005):

**Definition 1** ($\mathcal{L}$-MX: Model Expansion for $\mathcal{L}$ (Mitchell and Ternovska 2005))**.** *For logic $\mathcal{L}$, formula $\phi \in \mathcal{L}$ over vocabulary $\sigma \cup \varepsilon$, and $\sigma$-structure $\mathcal{A}$, the* task of model expansion (MX) for formula $\phi$ and structure $\mathcal{A}$ *is to find expansion $\mathcal{B}$ of $\mathcal{A}$ satisfying $\phi$, i.e., $\mathcal{B}|_\sigma = \mathcal{A}$ and $\mathcal{B} \models_{\mathcal{L}} \phi$.*

Model expansion task usually represents search problems. There, a formula $\phi$ is fixed to specify a particular computational problem and $\mathcal{A}$ serves as the input to $\phi$ while $\mathcal{B}$ is meant to represent the answer to problem $\phi$ with input $\mathcal{A}$. Hence, $\mathcal{A}$ is known as *instance (input) structure* and $\mathcal{B}$ as *expansion (output) structure*. Similarly $\sigma$ and $\varepsilon$ are respectively known as instance (input) and expansion (output) vocabularies of problem $\phi$. This paper always uses $\sigma$ for input and $\varepsilon$ for output vocabulary.

The following example illustrates these concepts:

**Example 1** (Hamiltonian Path)**.** *In Hamiltonian Path problem, you are given a directed graph $\mathcal{G} := (N; A)$ and asked to find a path $P$ in the graph that passes all nodes exactly once. This paper focuses on a version of Hamiltonian path where the starting point $s$ of the path is also given. In FO-MX following formulas specify Hamiltonian path problem:*

$$\forall x \forall y \; (P(x,y) \rightarrow A(x,y)) \tag{1}$$

$$\left.\begin{array}{c}[\forall x \forall y \forall y' \; (P(x,y) \wedge P(x,y') \rightarrow y = y')] \; \wedge \\ [\forall x \forall x' \forall y \; (P(x,y) \wedge P(x',y) \rightarrow x = x')] \; \wedge \\ \forall x \; (\neg P(x,s))\end{array}\right\} \tag{2}$$

$$\left.\begin{array}{c}\left[\forall x \; (R_0(x) \vee \cdots \vee R_{|N|}(x))\right] \; \wedge \\ \left[\forall x \; (R_0(x) \leftrightarrow x = s)\right] \; \wedge \\ \bigwedge_{0 \leq n < |N|} \left[\forall y \; (R_{n+1}(y) \leftrightarrow \exists x \; R_n(x) \wedge P(x,y))\right]\end{array}\right\} \tag{3}$$

*Here, Equation* (1) *asserts path $P$ is a subset of arcs from $A$. Equation* (2) *asserts that $P$ contains each node at most once and that starting node $s$ cannot have incoming arcs. Finally, Equation* (3) *asserts that all nodes are reachable from $s$ via $P$. Here, instance vocabulary is $\sigma := \{V, A, s\}$ and expansion vocabulary is $\varepsilon := \{P, R_0, \cdots, R_{|N|}\}$.*
**CDCL SAT Solver.** Conflict-driven clause-learning (CDCL) SAT solvers (Biere et al. 2009) add clause learning to DPLL so as to test satisfiability of CNF formulas. Implementing ideas in this paper is done by extending one such CDCL SAT solver: Glucose (Audemard and Simon 2009).
**Grounding and CNF conversion procedures.** In order to solve a problem defined in a high-level language such as first-order logic, one has to use a *grounder*. ENFRAGMO (Aavani et al. 2012), IDP (Wittocx, Mariën, and Denecker 2008) and SATGRND (Gebser et al. 2015) are typical off-the-shelf grounders for SAT. This paper uses $\mathrm{Gnd}(\phi; \mathcal{A})$, for a formula $\phi$ and finite structure $\mathcal{A}$, to denote the grounding procedure of such tools. In our implementation, we use SATGRND (Gebser et al. 2015) as our grounder of choice.

## $P[R]$: Defining Problems + Reasoning Methods

While existing logic programming frameworks only allow users to specify the constraints that define the solution space of a problem, our proposal allows users to axiomatize both the problem and its reasoning methods. This section introduces the syntax and semantics of $P[R]$, our language for declaratively specifying a problem plus its reasoning methods. We also give the grounding transformations used to incorporate such declaratively specified reasoning methods into our new solver, SAT-to-SAT.

**Definition 2** ($P[R]$: Syntax and Semantics)**.** *A specification in $P[R]$ has the form $(\phi, \{\psi_1, \ldots, \psi_n\})$ with $\phi, \psi_1, \ldots, \psi_n$ being FO sentences. Here, $\phi$ is known as the* main problem *and $\psi_1, \ldots, \psi_n$ as $\phi$'s* propagators (reasoning methods).

*Also, for propagator $\psi_i$, we define* propagator vocabulary*, denoted by* $\text{p-voc}(\psi_i)$*, as part of $\psi_i$'s vocabulary that is not shared with the main problem $\phi$, i.e.,* $\text{p-voc}(\psi_i) = vocab(\psi_i) \setminus vocab(\phi)$*. Moreover, if non-ambiguous, we use $\tau_i$ to denote propagator vocabulary* $\text{p-voc}(\psi_i)$.

*Finally, $P[R]$ specification $(\phi, \{\psi_1, \dots, \psi_n\})$ is satisfied by $(\sigma \cup \varepsilon)$-structure $\mathcal{B}$ if and only if $\mathcal{B}$ satisfies the second-order formula below:*

$$\phi \wedge \bigwedge_{1 \le i \le n} (\neg \exists \tau_i \; \psi_i) \tag{4}$$

*Here, assuming $\tau_i = \{R_1, \cdots, R_k\}$, formula $\exists \tau_i \; \psi_i$ abbreviates a second order quantification over vocabulary symbols $R_1, \dots, R_k$, i.e., $\exists \tau_i \; \psi_i := (\exists R_1 \exists R_2 \cdots \exists R_k \; \psi_i)$.*

The following example describes Hamiltonian path problem in the language of $P[R]$:

**Example 2.** *$P[R]$ can specify Hamiltonian path problem by tuple $(\phi, \{\psi\})$ where $\phi$ is the conjunction of formulas in Equations* (1) *and* (2)*, and $\psi$ is as below:*

$$\psi := \left\{ \begin{array}{c} C(s) \wedge [\exists x \; \neg C(x)] \wedge \\ [\forall x \forall y \; (P(x,y) \wedge C(x) \rightarrow C(y))] \end{array} \right. \tag{5}$$

*Here, main problem $\phi$ asserts all conditions on Hamiltonian paths except the reachability of every node from the starting node $s$ that is ensured by $\psi$: our only propagator in this encoding. Propagator $\psi$ uses symbols $P$, $C$ and $s$ and, hence, its vocabulary $\tau$ is $\{C\}$ because $C$ (standing for "Cut") is the only symbol in $vocab(\psi) \setminus vocab(\phi)$.*

*Now, according to $P[R]$'s semantics, propagator $\psi$ asserts the non-existence of a cut $C$ satisfying Equation* (5)*. Informally, it means that all cuts $C$ that include $s$ and all nodes reachable from $s$ (through path $P$) should include all nodes $u$ of the graph. Now, since non-existence of cut separating $s$ from some node $u$ means that all nodes $u$ are reachable from $s$, the second-order formula $\phi \wedge (\neg \exists C \; \psi)$ axiomatizes Hamiltonian path problem (as required).*

Example 2 shows that $P[R]$ can axiomatize Hamiltonian path problem using a propagator to guarantee reachability instead of directly encoding reachability in first-order logic (as was done in Example 1). However, other propagators can also be used to solve Hamiltonian path problem. For example, in (Gebser, Janhunen, and Rintanen 2014c), the authors use an acyclicty propagator to solve the same problem.

The most important property of $P[R]$ is replacing special-purpose propagator implementations with their logical description. The following example describes acyclicity propagator of (Gebser, Janhunen, and Rintanen 2014c) in $P[R]$:

**Example 3.** *Hamiltonian path problem can also be specified in $P[R]$ using pair $(\phi', \{\psi'\})$ with $\phi'$ and $\psi'$ as follows:*

$$\phi' := \left\{ \begin{array}{l} \forall x \forall y \; (P(x,y) \rightarrow A(x,y)). \\ \forall x \forall y \forall y' \; (P(x,y) \wedge P(x,y') \rightarrow y = y'). \\ \forall y \; (y \neq s \rightarrow \exists x \; (P(x,y))). \end{array} \right. \tag{6}$$

$$\psi' := \left\{ \begin{array}{l} \forall y \; (C(y) \rightarrow \exists x \; (C(x) \wedge P(x,y))). \\ \exists x \; C(x). \end{array} \right. \tag{7}$$

*Here, main problem $\phi'$ asserts, as before, that (1) path $P$ only contains arcs in the graph, (2) it contains at most one outgoing arc per node of the graph, and, (3) except starting node $s$, all nodes have at least one incoming arc.*

*Also, intuitively, propagator $\psi'$ (with vocabulary $\{C\}$ standing for "cycle"), asserts that $C$ is a cycle formed from arcs in $P$. It does so by ensuring that $C$ is non-empty and that $P$ connects every node in $C$ to some other node in $C$.*

*Therefore, according to $P[R]$'s semantics, structure $\mathcal{B}$ satisfies $(\phi', \{\psi'\})$ iff $\mathcal{B}$ satisfies second-order formula $\phi' \wedge (\neg \exists C \; \psi')$. This means that $P^{\mathcal{B}}$ is a Hamiltonian path because it is an acyclic collection of arcs with size $|N| - 1$ such that out-degree of all nodes is at most one and in-degree of all nodes except $s$ is exactly one.*

Example 3 suggests that $P[R]$ can describe many different propagators for the same problem. Later, we show that this is not a coincidence and $P[R]$ can indeed describe all possible propagators in NP.

Examples 2 and 3 show two different propagators for the same Hamiltonian path problem and, hence, one could also envisage specification $(\phi \wedge \phi', \{\psi, \psi'\})$ in $P[R]$ using both propagators $\psi$ and $\psi'$. Our framework translates such a $P[R]$-specification into a SAT solver that avoids cycles and unreachable nodes. That is, a SAT solver that backjumps whenever its partial solution is cyclic or makes some node non-reachable from $s$, it would backjump.

After describing the syntax and semantics of $P[R]$, let us now describe the grounding process of $P[R]$ specifications:

**Definition 3** (Grounding $P[R]$)**.** *For $P[R]$-specification $\alpha = (\phi, \{\psi_1, \cdots, \psi_n\})$ and finite $\sigma$-structure $\mathcal{A}$,* grounding of $\alpha$ w.r.t. $\mathcal{A}$*, denoted by $\text{Gnd}(\alpha; \mathcal{A})$, is a pair $(\phi', \{(L_1, U_1, \psi'_1), \cdots, (L_n, U_n, \psi'_n)\})$ such that:*

1. *$\phi' := \text{Gnd}(\phi; \mathcal{A})$.*
2. *For each $1 \le i \le n$, we have $\psi'_i = \text{Gnd}([\![\psi_i]\!]^{\varepsilon}; \mathcal{A})$ where $[\![\psi_i]\!]^{\varepsilon}$ is the same FO formula as $\psi_i$ except that all $\psi_i$'s positive occurrences of $R \in \varepsilon$ are replaced by a new predicate symbol $R_l$ and all $\psi_i$'s negative occurrences of $R \in \varepsilon$ are replaced by another new predicate symbol $R_u$.*
3. *Finally, $L_i$ and $U_i$ map atoms $R(\bar{t})$ to, respectively, lowerbound atoms $R_l(\bar{t})$ and upperbound atoms $R_u(\bar{t})$ if they appear in $\psi'_i$.*

As Definition 3 shows, the main difference between $P[R]$'s grounding procedure and FO's grounding procedure is the transformation $[\![\psi]\!]^{\varepsilon}$ that replaces positive/negative occurrences of $R \in \varepsilon$ with $R_l/R_u$. The new symbols $R_l$ and $R_u$ (for $R \in \varepsilon$) are intended to respectively represent the *lowerbound* and the *upperbound* of a partial interpretation for $R$. That is, if $R(\bar{t})$ is either true or false then $R_l(\bar{t}) = R_u(\bar{t}) = R(\bar{t})$ but if $R(\bar{t})$ is unknown then $R_l(\bar{t})$ is false and $R_u(\bar{t})$ is true.

**Definition 4** (2-Valued Representation)**.** *Let $\mathcal{B} \in \mathcal{S}(\sigma, \varepsilon)$, $\varepsilon_l = \{R_l \mid R \in \varepsilon\}$ and $\varepsilon_u = \{R_u \mid R \in \varepsilon\}$. Then, $(\sigma \cup \varepsilon_l \cup \varepsilon_u)$-structure $\mathcal{B}'$* is the 2-valued representation of $\mathcal{B}$ *if $\mathcal{B}'|_{\sigma} = \mathcal{B}|_{\sigma}$ and, for all $R \in \varepsilon$, $R_l^{\mathcal{B}'}$ is the lower-bound of $R^{\mathcal{B}}$ and $R_u^{\mathcal{B}'}$ is the upper-bound of $R^{\mathcal{B}}$.*

Now, the intuition behind transformation $[\![\psi]\!]^{\varepsilon}$ in Definition 3 can be described by the following theorem:

**Theorem 1.** *Let $(\phi, \{\psi_1, \cdots, \psi_n\})$ be a $P[R]$ specification, $\mathcal{B} \in \mathcal{S}(\sigma, \varepsilon)$ be a partial solution to $\phi$, and $\mathcal{B}'$ be the 2-valued representation of $\mathcal{B}$. Now, if $\mathcal{B}' \models \exists \tau_i [\![\psi_i]\!]^\varepsilon$ (for some $i$). Then, all $(\sigma \cup \varepsilon)$-structures $\mathcal{B}''$ extending $\mathcal{B}$ satisfy $\exists \tau_i\ \psi_i$.*

As this paper shows later, Theorem 1 is the foundation for our SAT-to-SAT solving algorithm because it provides a sufficient condition to discontinue a search branch in the solver. Following corollary formalizes this point:

**Corollary 1.** *Let $\alpha = (\phi, \{\psi_1, \cdots, \psi_n\})$ be $P[R]$ specification, let $\mathcal{B} \in \mathcal{S}(\sigma, \varepsilon)$ be a partial structure, and let $\mathcal{B}'$ be the 2-valued representation of $\mathcal{B}$. If $\mathcal{B}' \models \exists \tau_i\ [\![\psi_i]\!]^\varepsilon$ then $\mathcal{B}$ cannot be extended to satisfy $\alpha$.*

The following example describes these ideas:

**Example 4.** *Consider propagators $\psi$ and $\psi'$ of Examples 2 and 3. By Definition 3, $[\![\psi]\!]^\varepsilon$ and $[\![\psi']\!]^\varepsilon$ are as follows:*

$$[\![\psi]\!]^\varepsilon := \left\{ \begin{array}{l} C(s) \wedge \exists x\ (\neg C(x)) \wedge \\ {[\forall x \forall y\ (P_u(x,y) \wedge C(x) \to C(y))]} \end{array} \right. \tag{8}$$

$$[\![\psi']\!]^\varepsilon := \left\{ \begin{array}{l} {[\forall y\ (C(y) \to \exists x\ (C(x) \wedge P_l(x,y)))]} \wedge \\ {[\exists x\ C(x)]} \end{array} \right. \tag{9}$$

*Remember that, $P_u(x,y)$ is true whenever $P(x,y)$ is true or undefined (and false otherwise) while $P_l(x,y)$ is false whenever $P(x,y)$ is false or undefined (and true otherwise). Hence, second-order formula $\exists C\ ([\![\psi]\!]^\varepsilon)$ is true whenever enough $P_u(x,y)$'s are made false so that a cut $C$ separating $s$ from some node $x$ is found even if all the currently unassigned values in $P$ are assigned true. Therefore, formula $\neg\exists C\ ([\![\psi]\!]^\varepsilon)$ holds if no such cut exists, i.e., if all nodes are reachable from $s$ using $P$.*

*Similarly, formula $\exists C\ ([\![\psi']\!]^\varepsilon)$ is true whenever enough $P_l(x,y)$'s are made true so that a cycle $C$ (that is present in all possible extensions of current partial structure) is found. Thus, second-order formula $\neg\exists C\ ([\![\psi']\!]^\varepsilon)$ guarantees the non-existence of such cycles and, hence, the acyclicity of $P$.*

## Expressiveness and Complexity of $P[R]$

As Definition 2 shows, semantically, $P[R]$ specification $\alpha := (\phi, \{\psi_1, \cdots, \psi_n\})$ is equivalent to second-order formula $\phi \wedge (\neg\exists\tau_1\psi_1) \wedge \cdots \wedge (\neg\exists\tau_n\psi_n)$. Using this semantics, we now investigate the complexity of model expansion task for $P[R]$ as well as its expressiveness in terms of problems $\alpha$ and reasoning methods $\psi_i$ that $P[R]$ can express.

We use Fagin's theorem (Fagin 1974), to characterize NP with existential second-order ($\exists$SO) logic, as well as results from (Kolokolova et al. 2010) to relate model expansion task and $\exists$SO to prove the following expressiveness results.

**Theorem 2.** *If $\mathcal{K}$ is a class of finite structures that is closed under isomorphism, then:*
*(1) $P[R]$ can axiomatize $\mathcal{K}$ iff $\mathcal{K}$ is $\Sigma_2^P$-decidable.*
*(2) $P[R]$ propagators can axiomatize $\mathcal{K}$ iff $\mathcal{K}$ is NP-decidable.*

Theorem 2 shows that $P[R]$ specifications can describe all problems in $\Sigma_2^P$. In particular, $P[R]$ can also describe any $\Sigma_2^P$-complete problem. Hence, we have:

**Corollary 2.** *Let $\alpha = (\phi, \{\psi_1, \cdots, \psi_n\})$ be a fixed $P[R]$ specification. Given a $\sigma$-structure $\mathcal{A}$, deciding the existence of an expansion $\mathcal{B}$ of $\mathcal{A}$ that satisfies $\alpha$ is $\Sigma_2^P$-complete.*

Note that, as Corollary 2 shows, solving $P[R]$-MX is $\Sigma_2^P$-hard. Hence, unless NP= $\Sigma_2^P$, no polynomial time reduction from $P[R]$ to SAT exists. Thus, a new solver is needed to solve $P[R]$ specifications. Next section proposes our SAT-to-SAT algorithm for that purpose.

## SAT-to-SAT: Solving $P[R]$ Specifications

This section shows how ground $P[R]$ specifications, as in Definition 3, can be solved using our solver, SAT-to-SAT.

Algorithm 1 shows that SAT-to-SAT extends CDCL by introducing a new propagation handling procedure "S2S-prop" in the common CDCL search method. Procedure "S2S-prop" takes a list of recently propagated literals and checks if some reasoning method $\psi_i$ can find a conflict. If so, a conflict clause is generated, it is added to learnt clauses of CDCL solver, and the conflict analysis and backjumping procedures are initiated.

As Algorithm 1 shows, given a grounding of $P[R]$ specification $(\phi, \{\psi_1, \cdots, \psi_n\})$, SAT-to-SAT initializes $n+1$ SAT solver instances $S_0, S_1, \cdots, S_n$ where $S_0$ is tasked with solving the main problem $\phi$ while each $S_i$ $(1 \leq i \leq n)$ is tasked to help $S_0$ by finding conflicts according to $\psi_i$.

Solving a $P[R]$ specification using SAT-to-SAT is done collaboratively so that main solver $S_0$ communicates with helping solvers $S_1, \cdots, S_n$. This communication is facilitated by method "S2S-prop" that takes $S_0$'s current partial interpretation $I$ plus the list $L$ of $S_0$'s recently propagated literals and checks if some helping solver $S_i$ can generate some new conflict clause $C$. If so, $C$ is added to $S_0$'s clauses and $S_0$'s conflict analysis procedure is initialized on $C$. If not, $S_0$ continues exploring its current search branch.

**Generating Conflict Clauses.** SAT-to-SAT uses method "S2S-apply" to check if the new reasoning method $\psi_i$ can generate a new conflict clause. This is done by calling $S_i$'s search method with assumptions that correspond to the under approximation of $S_0$'s current partial interpretation. If $S_i$ returns a model, by Corollary 1, $S_0$ cannot extend its current partial structure to one that satisfies $P[R]$ specification $(\phi, \{\psi_1, \cdots, \psi_n\})$. Hence, a conflict clause is generated so as to prevent $S_0$ from following its current search branch.

In SAT-to-SAT, such a conflict clause is generated by method "S2S-reason" that takes a satisfying interpretation $I$ of CNF formula $F$ and generates clause $C$ that is falsified by the current partial interpretation of $S_0$. Conflict clause $C$ is generated by (1) iterating over all clauses $C' \in F$ that are satisfied by $S_0$'s current partial solution, and then (2) negating one of literals $l \in C'$ that satisfies $C'$. The following theorem proves the correctness of "S2S-reason".

**Theorem 3.** *When running SAT-to-SAT on input $\mathrm{Gnd}(\alpha; \mathcal{A})$ with $\alpha := (\phi, \{\psi_1, \cdots, \psi_n\})$ and $\mathcal{A}$ a $\sigma$-structure, every clause $C$ generated by "S2S-reason" is guaranteed to:*
*(a) be true in all expansions $\mathcal{B}$ of $\mathcal{A}$ that satisfy $\alpha$, and,*
*(b) be false in $S_0$'s current partial interpretation.*

Theorem 3 shows that clauses $C$ generated by "S2S-reason" cause solver $S_0$ to backtrack.

**Example 5.** *Let $(\phi, \{\psi\})$ be as in Example 2. Then, all clauses generated by "S2S-reason" have the form of $P(s_1,t_1) \vee \cdots \vee P(s_k,t_k)$ with $P(s_i,t_i)$'s being all the*

**Algorithm 1** SAT-to-SAT algorithm to solve $P[R]$ specs

```
1:  procedure SAT-TO-SAT($F_\phi$, $(L_i, U_i, F_{\psi_i}) : 1 \le i \le n$)
2:      Initiate $n+1$ SAT solvers $S_0, S_1, \cdots, S_n$
3:      Add clauses of $F_\phi$ to $S_0$
4:      for all $i \in \{1, \cdots, n\}$ do
5:          Add clauses of $F_{\psi_i}$ to $S_i$
6:          $T[i] := \emptyset$                    ▷ $i$-th propagator's trigger literals
7:      Add propagator handler "S2S-PROP" to $S_0$
8:      Add backjump handler "S2S-BACKJUMP" to $S_0$
9:      TrigsInitialized := false
10:     return $S_0$.SEARCH( )                    ▷ Solving main problem
11: procedure S2S-PROP($I$, $\{l_1, \cdots, l_k\}$)
12:     for all $i \in \{1, \cdots, n\}$ do
13:         if ¬TrigsInitialized or $\exists l_j$ s.t. $l_j \in T[i]$ then
14:             if S2S-APPLY($S_i$,$L_i$,$U_i$,$I$) = (false, $C$) then
15:                 $S_0$.LEARN-CLAUSE($C$)
16:                 $S_0$.ANALYZE-CONFLICT($C$)
17:                 return
18:     TrigsInitialized := true
19: procedure S2S-BACKJUMP
20:     TrigsInitialized := false
21: procedure S2S-APPLY($S_i$, $L$, $U$, $I$)
22:     $A := \{y|(x,y) \in L_i, x \in I\} \cup \{\neg y|(x,y) \in L_i, x \notin I\}$
23:         $\cup \{\neg y|(x,y) \in U_i, \neg x \in I\} \cup \{y|(x,y) \in U_i, \neg x \notin I\}$
24:     $R := S_i$.SEARCH($A$)          ▷ Solve $\psi_i$ under assumptions $A$
25:     if $R = (\text{SAT}, I')$ then              ▷ Generate conflict clause
26:         Let $F$ be all clauses in SAT Solver $S_i$
27:         $C :=$ S2S-REASON($F$, $L \cup U$, $I'$)
28:         return (false, $C$)
29:     else if $R = (\text{UNSAT}, C)$ then        ▷ Find trigger conditions
30:         $T[i] := \emptyset$
31:         for all $l \in C$ do
32:             Let $y := var(l)$ (the Boolean variable inside $l$)
33:             Let $x$ be such that $(x,y) \in L \cup U$
34:             if $l$ is positive then $T[i] := T[i] \cup \{\neg x\}$
35:             else $T[i] := T[i] \cup \{x\}$
36: procedure S2S-REASON($F$, $M$, $I$)
37:     $C := \emptyset$
38:     $Y := \{y \mid (x,y) \in M\}$
39:     for all clauses $l_1 \vee \cdots \vee l_k$ in $F$ do
40:         if $\forall l_i$ either $I \not\models l_i$ or $var(l_i) \in Y$ then
41:             Let $j \in \{1, \cdots, k\}$ be such that $I \models l_j$
42:             Let $y := var(l_j)$ and $x$ be s.t. $(x,y) \in M$
43:             if $l_j$ is positive then $C := C \cup \{\neg y\}$
44:             else $C := C \cup \{y\}$
45:     return clause $\bigvee_{l \in C} l$
```

*arcs from inside cut $C$ to outside it. Moreover, all $P(s_i,t_i)$'s are falsified in the current partial interpretation $I$ of $S_0$. Hence, no matter how $I$ is extended, nodes inside $C$ (including $s$) cannot be connected to nodes outside $C$. Thus, the new clause correctly forces $S_0$ to backtrack.*

**Trigger Conditions.** Since running method "S2S-apply" involves the heavy operation of solving another satisfiability problem, method "S2S-prop" refrains from calling "S2S-apply" unless a *trigger condition* holds for propagator $\psi_i$.

Trigger conditions for a propagator $\psi_i$ are obtained dynamically whenever the $S_i$'s search fails, i.e., when SAT solver $S_i$ cannot find a new conflict in $S_0$'s partial interpretation. In such cases, $S_i$ returns $(\text{UNSAT}, C)$ which informally means that, as long as all assumptions in $C$ are true, $S_i$ remains unsatisfiable. Hence, we run $S_i$ again only when at least one assumption in $C$ becomes false. The following theorem guarantees the correctness of this method.

**Theorem 4.** *In method "S2S-prop" of Algorithm 1, if variable "TrigsInitialized" is true and $L \cap T[i] = \emptyset$, i.e., none of the trigger literals are recently propagated, then $\psi_i$ cannot generate a conflict clause under partial interpretation $I$.*

The following example illustrates how triggers look like in the case of reachability propagator for Hamiltonian path:

**Example 6.** *Let $(\phi, \{\psi\})$ be as in Example 2. Then, the trigger $T[1]$ generated by $S_1$ always has the following form $\{\neg P(u_1,v_1), \cdots, \neg P(u_k,v_k)\}$ where (1) $P(u_i,v_i)$ is either true or unknown according to current partial interpretation $I$, and, (2) $(u_1,v_1), \cdots, (u_k,v_k)$ form a rooted (and directed) spanning tree of the graph with root $s$. It is thus clear that as long as none of arcs $P(u_i,v_i)$ in such a spanning tree become false, every node remains reachable from $s$ (at least through the spanning tree). Thus, in such cases, calling solver $S_1$ is futile because all nodes are reachable.*

Looking at Examples 5 and 6, it is noteworthy that SAT solver $S_1$ or methods "S2S-prop" and "S2S-apply" are not aware of the fact that they are trying to find a cut to prove non-reachability of a node and that generating conflict clauses, and finding trigger conditions are all done automatically without knowing what problem is being solved. Yet, as these examples show, the notions of conflict clauses and trigger conditions take the form of known graph-theoretical concepts with close ties to the problem that is being solved, i.e., reachability. This is a surprising consequence of SAT-to-SAT solving method and stands to show that our definitions of conflict clauses and trigger conditions are indeed natural.

Finally, also note that, while reachability is a well-studied graph-theoretical property, our $P[R]$ language and our SAT-to-SAT solver can describe and solve many other reasoning methods that are not well-studied. This generality makes $P[R]$ an extremely suitable framework to describe less well-studied reasoning methods $R$ because one can readily benefit from notions of conflict clauses and trigger conditions that may not be known about $R$ beforehand. We conclude this section by a theorem about SAT-to-SAT's correctness.

**Theorem 5.** *Algorithm 1 is correct. That is, given $\text{Gnd}(\alpha; \mathcal{A})$ with $\alpha := (\phi, \{\psi_1, \cdots, \psi_n\})$ and $\mathcal{A}$ being a $\sigma$-structure, SAT-to-SAT returns $(SAT, \mathcal{B})$ if an expansion $\mathcal{B}$ of $\mathcal{A}$ exists that satisfies $\alpha$, and it returns UNSAT otherwise.*

## Experiments

This section includes our experimental results for our running example of Hamiltonian path and shows that, using $P[R]$ results in a huge boost in the solving time of Hamiltonian path instances. These preliminary results show the usefulness of $P[R]$'s approach to logic programming. More experiments will be presented in a journal version of this paper.

We have implemented SAT-to-SAT by extending GLUCOSE 3.0 (Audemard and Simon 2009). Our Glucose extension closely matches Algorithm 1 except for using fast data

| Hamiltonian Path Instances (15m time limit) | | | | | | | |
|---|---|---|---|---|---|---|---|
| Size | Total Inst. | Glucose Enc. of Ex. 1 | | SAT-to-SAT Enc. of Ex. 2 | | SAT-to-SAT Enc. of Ex. 3 | |
| | | # | Time | # | Time | # | Time |
| 50 | 20 | 20 | 4.85s | **20** | **0.02s** | **20** | **0.02s** |
| 100 | 20 | 4 | 390s | **20** | **0.13s** | 20 | 0.63s |
| 150 | 20 | 0 | — | **20** | **1.14s** | 20 | 7.52s |
| 200 | 20 | 0 | — | **20** | **9.00s** | 20 | 74.0s |
| 250 | 20 | 0 | — | **20** | **82.3s** | 18 | 283s |
| 300 | 20 | 0 | — | **9** | **288s** | 5 | 639s |

Table 1: Solving Hamiltonian path using SAT-to-SAT on two different encodings plus using Glucose on a direct encoding.

structures such as "BitSets" to expedite problem solving. We also use SATGRND (Gebser et al. 2015) as our grounder.

The instances of Hamiltonian path are randomly generated using a random planar graph generator that is developed as part of LEDA (Mehlhorn and Näher 1992) library. These instances are then solved in three different ways: Using GLUCOSE to solve a direct encoding of Hamiltonian path as in Example 1, and using SAT-TO-SAT to solve Hamiltonian path once as encoded in Example 2, and, another time as encoded in Example 3.

Table 1 summarizes the results of running our experiments on an Ubuntu 14.04 Linux desktop with kernel version 3.13.0-57, an Intel(R) Core(TM) i5-4590 CPU running at 3.30GHz, plus 16GB of memory. The time limit for each instance is set at 15 minutes and the "#" columns in Table 1 show the number of instances that were solved using a particular encoding in 15 minutes. Moreover, columns with title "Time" show the average solving time for solved instances.

As Table 1 shows, both of the encodings for SAT-TO-SAT (which is based on GLUCOSE) hugely outperform direct encodings on GLUCOSE by at least two orders of magnitude. We attribute this efficiency *to SAT-to-SAT's ability to model propagators in addition to the main problem*. This way of modeling problems is not possible in existing modeling frameworks and is the main novelty of current paper.

## Related Work

This section summarizes the most relevant works to the current paper and discusses the similarities and/or differences between those works and ours.

In (Bayless et al. 2013), the authors propose a SAT modulo SAT approach to solve the partitioned Boolean satisfiability problem. From a technical viewpoint, their methods are similar to ours because they also instantiate several SAT solvers to solve a problem. However, their conceptual viewpoint is different because the semantics of internal solvers is not changed. This difference can be also observed in terms of the complexity of the overall problem. While $P[R]$ can express $\Sigma_2^P$-complete problems, the SAT modulo SAT approach cannot solve anything beyond NP. Informally, it means that SAT modulo SAT approach cannot describe propagators because, as shown in this paper, describing propagators requires universal second-order quantification (e.g., quantifying over all possible cuts as in Example 2) which is not doable by SAT modulo SAT.

In (Bayless et al. 2015), the authors strive for the same goals as us (i.e., lowering the cost of implementing new propagators) but they focus less on automatizability of their approach. In fact, they require known algorithms to be implemented in non-declarative languages such as C++ and be used in correspondence with underlying SAT solvers. This is while our SAT-to-SAT approach uses helping SAT solvers to realize new reasoning mechanisms. As shown in this paper, our approach yields automatic generation of conflict clauses as well as dynamic detection of trigger conditions: two concepts that cannot be easily automatized in their approach. Furthermore, their approach only applies to monotonic theories while ours can describe all propagators in NP.

In (Abío et al. 2013), the authors discuss the potential disadvantage of using new propagators which may result in creating an exponential number of learnt clauses. Instead, they propose using propagators that are aware of the compact encodings of a problem and that generate their conflict clauses with respect to those compact encodings. While their criticisms do not directly apply to $P[R]$ language and SAT-TO-SAT solver, their proposal shows an interesting potential extension of our approach in which declaratively specified propagators generate conflict clauses with respect to some compact encoding of that propagator.

## Conclusion and Future Works

This paper introduced a new logic programming language $P[R]$ plus a new solver SAT-TO-SAT to solve $P[R]$ specifications. We showed how $P[R]$ models both the problem itself and the reasoning methods that help solving that problem. We also showed how SAT-TO-SAT uses several communicating SAT solvers to solve a ground $P[R]$ specification. We studied the expressiveness and complexity of $P[R]$ language and showed that SAT-TO-SAT hugely outperforms GLUCOSE (SAT-TO-SAT's underlying SAT solver) when solving the Hamiltonian path problem. Some of the future directions of our approach are listed below:

**Solving QBF using an Extension of SAT-to-SAT.** Quantified Boolean Formulas (QBF) are an important extension of Satisfiability problem with applications in verification and solving modal logic formulas. QBF is a PSPACE-complete problem and, thus, cannot be solved using either off-the-shelf SAT solvers or SAT-TO-SAT. However, one can easily observe that, if $P[R]$ allowed propagators to be any $P[R]$ specification (instead of a first-order sentence), then it could have expressed all problems in PSPACE. The main obstacle, however, is to extend SAT-TO-SAT so that it can solve such complex instances. We believe that such an extension is possible and, thus, we aim to extend SAT-TO-SAT as such.

**Different Conflict Clause Generation Methods.** SAT-TO-SAT uses method "S2S-reason" to generate new conflict clauses by taking a literal from each clause of $\psi_i$'s definition. While our experiments show that this strategy works well, one could envisage many other strategies to generate conflict clauses. Studying the effect of such different strategies is one of our future research directions.

**SAT-to-SAT as a $\Sigma_2^P$ solver.** SAT-TO-SAT can solve $\Sigma_2^P$ problems. Thus, we intend to compare its performance against existing $\Sigma_2^P$ solvers. We also intend to study how SAT-TO-SAT fits into the DPLL(T) framework of (Nieuwenhuis, Oliveras, and Tinelli 2004).

## Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Mathematics of Information Technology and Complex Systems (MITACS), LogicBlox Inc., and the Academy of Finland (Finnish Centre of Excellence in Computational Inference Research COIN, 251170).

## 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 *LPAR 2012*, 15–22.

Abío, I.; Nieuwenhuis, R.; Oliveras, A.; Rodríguez-Carbonell, E.; and Stuckey, P. 2013. To encode or to propagate? the best choice for each constraint in SAT. In Schulte, C., ed., *Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 97–106.

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern SAT solvers. In Boutilier, C., ed., *IJCAI 2009*, 399–404.

Barrett, C.; Conway, C. L.; Deters, M.; Hadarean, L.; Jovanovic, D.; King, T.; Reynolds, A.; and Tinelli, C. 2011. CVC4. In Gopalakrishnan, G., and Qadeer, S., eds., *CAV 2011*, volume 6806 of *Lecture Notes in Computer Science*, 171–177. Springer.

Bayless, S.; Val, C. G.; Ball, T.; Hoos, H. H.; and Hu, A. J. 2013. Efficient modular SAT solving for IC3. In *Formal Methods in Computer-Aided Design (FMCAD), 2013*, 149–156. IEEE.

Bayless, S.; Bayless, N.; Hoos, H. H.; and Hu, A. J. 2015. SAT modulo monotonic theories. In Bonet, B., and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, 3702–3709. AAAI Press.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T. 2009. Conflict-driven clause learning SAT solvers. *Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications* 131–153.

Bjørner, N. 2012. Engineering theories with Z3. In Korovin, K.; Schulz, S.; and Ternovska, E., eds., *IWIL 2012*, volume 22 of *EPiC Series*, 1–2. EasyChair.

de la Banda, M. J. G.; Marriott, K.; Rafeh, R.; and Wallace, M. 2006. The modelling language Zinc. *Principles and Practice of Constraint Programming (CP 2006)* 700–705.

De Moura, L., and Bjørner, N. 2008. Z3: An efficient SMT solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, 337–340. Berlin, Heidelberg: Springer-Verlag.

Een, N., and Sörensson, N. 2005. MiniSat v1.13 - a SAT solver with conflict-clause minimization, system description for the SAT competition.

Fagin, R. 1974. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of computation, SIAM-AMC proceedings* 7:43–73.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Thiele, S. 2008. *A User's Guide to gringo, clasp, clingo, and iclingo*. http://potassco.sourceforge.net/.

Gebser, M.; Janhunen, T.; Kaminski, R.; Schaub, T.; and Tasharrofi, S. 2015. Writing declarative specifications for clauses. In *3rd International Workshop on Grounding, Transforming, and Modularizing Theories with Variables*. http://research.ics.aalto.fi/software/asp/satgrnd/.

Gebser, M.; Janhunen, T.; and Rintanen, J. 2014a. Answer set programming as SAT modulo acyclicity. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *ECAI 2014*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, 351–356. IOS Press.

Gebser, M.; Janhunen, T.; and Rintanen, J. 2014b. ASP encodings of acyclicity properties. In Baral, C.; Giacomo, G. D.; and Eiter, T., eds., *KR 2014*. AAAI Press.

Gebser, M.; Janhunen, T.; and Rintanen, J. 2014c. SAT modulo graphs: Acyclicity. In Fermé, E., and Leite, J., eds., *JELIA 2014*, volume 8761 of *Lecture Notes in Computer Science*, 137–151. Springer.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence* 187-188:52–89.

Kolokolova, A.; Liu, Y.; Mitchell, D.; and Ternovska, E. 2010. On the complexity of model expansion. In *Proc., 17th Int'l Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-17)*, 447–458. Springer. LNCS 6397.

Liang, T.; Reynolds, A.; Tinelli, C.; Barrett, C.; and Deters, M. 2014. A DPLL(T) theory solver for a theory of strings and regular expressions. In Biere, A., and Bloem, R., eds., *CAV 2014*, volume 8559 of *Lecture Notes in Computer Science*, 646–662. Springer.

Mehlhorn, K., and Näher, S. 1992. Algorithm design and software libraries: Recent developments in the LEDA project. In *In Proceedings of IFIP: 12th World Computer Congress*, 493–505. Elsevier.

Mitchell, D. G., and Ternovska, E. 2005. A framework for representing and solving NP search problems. In *Proc. AAAI'05*, 430–435.

Nieuwenhuis, R.; Oliveras, A.; and Tinelli, C. 2004. Abstract DPLL and abstract DPLL modulo theories. In Baader, F., and Voronkov, A., eds., *Proceedings of 11th International Conferemce on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'04)*, Lecture Notes in Computer Science (LNCS), 36–50. Springer.

Wittocx, J.; Mariën, M.; and Denecker, M. 2008. The IDP system: A model expansion system for an extension of classical logic. In *Proceedings of the 2nd Workshop on Logic and Search*, 153–165.