

Supported Semantics for Modular Systems

Shahab Tasharrofi and Eugenia Ternovska
{sta44,ter}@cs.sfu.ca

Simon Fraser University

Abstract. Modular systems offer a language-independent declarative framework in which modules from different languages are combined to describe a conceptually and/or computationally complex problem. The original semantics developed to combine modules in a modular system is a model-theoretical semantics with close ties to relational algebraic operations.

In this paper, we introduce a new semantics for modular systems, called supported model semantics, that extends the original model-theoretic semantics of modular systems and captures non-monotonic reasoning on the level of modular system (rather than the level of individual modules). Moreover, we also use supported model semantics for modular systems to compare the expressiveness of modular systems with that of heterogeneous non-monotonic multi-context systems [2]. We show that, under very few conditions, all multi-context systems can be translated into equivalent modular systems. Our result is true for both major semantics of multi-context systems, i.e., the equilibrium semantics and the grounded equilibrium semantics. Thus, we show that modular systems under supported model semantics generalize multi-context systems under either of its major semantics.

1 Introduction

Tasharrofi and Ternovska introduced modular systems [7] as a language-independent framework that combines modules in order to solve a more complex problem. The modules in this framework represent individual and usually standalone knowledge bases with (possibly) their own unique syntax and semantics. Thus, using the modular system framework, one is able to represent and solve problems that involve heterogeneous knowledge bases.

In this paper, we add a new and important feature to the framework of modular systems in order to extend the ability of interaction between the knowledge bases of different individual modules. That is, we allow modules in a modular system to not only share their models (which was also possible before) but to also share the reasoning and the inter-dependencies inside these models. We achieve this goal by extending the framework of modular systems with a new and powerful mechanism called supports. Using supports, modules of a modular system would be able to reason about a situation together.

Intuitively speaking, supports add the ability for individual modules to say why something is considered true. Therefore, the new semantics, that we introduce here for modular systems with support, can trace back the justifications of each atom and disallow models that are self-justified. The following example shows how supports play a role in everyday situations:

Example 1 (Shopping). Consider John who needs to buy a set of matching clothes for a formal event he would be attending. He has to choose his type of clothing (e.g. between a tie and a bowtie), the color of his suit and a matching shirt. For example, if he buys a dark suit, he will have to buy a white shirt. An intended solution for this problem is a solution in which, after shopping, John has a fully matching set of clothes.

If we represent this problem using modular systems and each department of a clothing store is represented by a different module, the previous semantics of modular systems fails to reject the non-intended model in which John buys everything in the store. Although this solution satisfies John's goal of having a set of matching clothes, it is still an unreasonable solution because its extravagance is unjustified. According to the problem statement, John does not have any reason to support buying both a dark suit and a light one. However, the previous semantics of modular systems is unable to capture this intuition.

The problem in Example 1 is that, up to now, modular systems do not provide any means for accompanying beliefs with their justifications. Under such a semantics, there is no easy way to guarantee that John of Example 1 buys only one set of matching clothes. This type of problem happens in many other situations as well. One of the examples that is frequently referenced in the literature is the connectivity example that follows.

Example 2 (Connectivity). You are given a set V of vertices and a set E of edges in a graph and you are asked to find all pairs (u, v) of vertices such that u and v are connected through edges in E . We know that this problem is polytime computable but if we try to formalize it as a fixpoint computation, we would have something as follows: find the fixpoint of function $C_E(R) := E \cup \{(u, u) \mid u \in V\} \cup \{(u, v) \mid \exists w ((u, w) \in E \wedge (w, v) \in R)\}$. However, we know that the only interesting fixpoint for this function is its least fixpoint (which characterizes the solution to our connectivity problem). Moreover, we also know that the connectivity problem cannot be characterized as the fixpoint of any function that uses only first order sentences for constructing sets.

Example 2 showcases a situation where not only is the notion of justification needed to obtain a reasonable semantics but also a situation where justifications can provide meaningful information about connectivity of two nodes. The following example shows how justifications can be used to identify paths between connected nodes.

Example 3 (Connectivity: Justifications as Paths). Consider the connectivity problem of Example 2. Assume that, for each pair $(u, v) \in R$ of connected nodes, we denote the justification for connectivity of (u, v) using $j_R(u, v)$ which satisfies the following conditions: (1) $j_R(u, u) = \{\}$ for all u , (2) for $u \neq v$, if $(u, v) \in E$ then $j_R(u, v) = \{v\}$, and, (3) otherwise, $j_R(u, v) = \{w\}$ where $(u, w) \in E$ and $(w, v) \in R$.

Note that, for all fixpoints R of C_E (i.e., if $R = C_E(R)$), at least one function $j_R(u, v)$ exists that satisfies the required conditions because, if R is a fixpoint of C_E and $(u, v) \in R$ then at least one of the conditions (1) $u = v$, (2) $(u, v) \in E$, or, (3) $\exists w ((u, w) \in E \wedge (w, v) \in R)$ is true. Moreover, if R is the least fixpoint of equation $R = C_E(R)$, we can define function $j_R(u, v)$ such that $j_R(u, v)$ contains the first node visited in a path from u to v .

Such a justification function carries a very useful piece of information: the path that starts at u and ends at v . This path can be computed as follows:

$$path_j(u, v) := \begin{cases} [v] & \text{if } j(u, v) = \{\}, \\ (u : path_j(w, v)) & \text{if } j(u, v) = \{w\}. \end{cases}$$

where $[x]$ and $(x : xs)$ respectively denote a list containing only one element x , and a list that starts with element x and is followed by list xs (as is customary in functional programming languages).

Examples 1 and 2 define two cases when we are interested only in justifiable solutions (not all the solutions). Moreover, Example 3 shows how certain justifications can contain useful information about a model. It remains to see how exactly such useful justifications can be characterized. Following example demonstrates the essential property that characterizes useful justifications in the context of connectivity problem.

Example 4 (Connectivity: Well-founded Justifications). Again, consider the problem of connectivity as in Example 2. Also, let R be a fixpoint of equation $R = C_E(R)$ and j_R be a justification function for R that satisfies the conditions of Example 3. Moreover, let us take our measure of usefulness for a justification j_R to be the finitude of lists created by function $path_j$. That is, a j_R is considered useful if $path_j(u, v)$ is finite (for all $(u, v) \in R$). Note that, the finitude of $path_j(u, v)$ implies the connectivity of u and v (because v can be reached from u in finitely many steps). Thus, a useful justification j_R exists if and only if R is the set of connected vertices.

So, now, the question is that, given j , if finitude of $path_j(u, v)$ can be characterized in terms of j itself. It turns out that, $path_j(u, v)$ is finite for all (u, v) if and only if, for all vertices v , a strict well-ordering $<_v$ exists such that: (1) $u <_u v$ for all $u \neq v$, and, (2) $w <_u v$ for all $w \in j_R(u, v)$.

Example 4 shows how the well-founded justifications characterize the right solutions of the connectivity problem. Interestingly, well-founded justifications characterize many other useful justification functions as well. For example, both the semantics of Horn formulas and stable model semantics can be characterized as sets of “reasonable” fixpoints with well-founded justifications [6]. Example below demonstrates this correspondence in the case of stable model semantics for normal propositional logic programs.

Example 5 (Stable Model Semantics: Well-justified). Consider a normal propositional logic program P , i.e., P is a set of rules of form $h \leftarrow b_1, \dots, b_k, \text{not } b_{k+1}, \dots, \text{not } b_n$ where h and $b_1, \dots, b_n$ are propositional atoms. Then, by [6], we know that for all sets S of propositional atoms, S is a stable model of P if and only if a well-founded ordering $<$ on S exists so that, for all $a \in S$, we have a rule $a \leftarrow b_1, \dots, b_k, \text{not } b_{k+1}, \dots, b_n$ in P such that: (1) for all $1 \leq i \leq k$, we have $b_i \in S$ and $b_i < a$, and, (2) for all $k+1 \leq i \leq n$, we have $b_i \notin S$.

Together, Examples 1 – 5 show the importance of non-circular and well-founded justifications in characterizing intended models of a system. In this paper, we want to augment modular systems with a similar notion of justifications. As we show in this paper, adding justifications to the semantics of modular system extends the expressiveness of our modular system framework.

Contributions

The following summarizes our contributions in this paper.

Justification and Support for Models of Primitive Modules: We augment primitive models of a modular system with the notions of justification and support functions for their models. In essence, each justification function justifies one of the models of a primitive module. Also, support functions assign possible justification functions to a model, i.e., a single model might be associated with several different justification functions.

Extending Support to Combined Modular Systems: We recursively define meaningful support (and justification) functions for combinations of modular systems by combining the support functions of combined modules. We show the usefulness of our definitions through several natural examples.

Defining Supported Model Semantics for Modular Systems: We define supported model semantics of modular systems to accept exactly those models of a modular system that are supported. We show that this definition captures the idea of non-circular and well-founded justifications and that many non-trivial semantics can readily be defined in terms of supported semantics for modular systems. We also show that supported model semantics generalizes the previous model-theoretical semantics for modular systems.

Expressing Equilibria of Multi-context Systems using Supported Model Semantics: We show that, the class of multi-context systems with structural belief sets, can be naturally translated into a modular system under supported model semantics. We show that our translation is robust enough to be extended to multi-context systems with variables in their bridge rules.

Expressing Grounded Equilibria of Multi-context Systems using Supported Model Semantics: Similar to the previous case, we also translate multi-context systems under grounded equilibrium semantics to modular systems under supported model semantics. Interestingly, the translation we use for both these multi-context systems is similar to each other and the only difference between these two translations is the justification functions of the primitive modules in these two translations. Therefore, supported model semantics effectively generalizes the two different semantics of normal and grounded equilibria for multi-context systems.

2 Background

Atoms in Structures Recall that a structure $\mathcal{A}$ is a domain (a set of abstract elements denoted by $dom(\mathcal{A})$) plus an interpretation of its vocabulary symbols (denoted by $vocab(\mathcal{A})$). While this definition satisfactorily represents structures, in this paper, we represent structures in a different form that is best suited to the goals of this paper.

Let $\mathcal{A}$ be a structure and let S_r and S_f be, respectively, the set of relational and functional vocabulary symbols of $\mathcal{A}$ (constants are zero-ary functions). Then, we represent $\mathcal{A}$ by its domain plus the set of its truth assignments as follows:

1. For each n -ary relational vocabulary symbol $R \in S_r$ and for each n -ary tuple $t \in R^{\mathcal{A}}$, the truth assignments of $\mathcal{A}$ include atoms of form R_t .

2. For each n -ary functional vocabulary symbol $f \in S_f$ and for each n -ary tuple $t \in [dom(\mathcal{A})]^n$, the truth assignments of $\mathcal{A}$ include atoms of form $f_{t \mapsto a}$ where $a \in dom(\mathcal{A})$ is such that $f^{\mathcal{A}}(t) = a$.
3. Truth of assignments of $\mathcal{A}$ does not contain anything else.

In other words, the set of $\mathcal{A}$'s truth assignments is the following set:

$$\{R_t \mid R \in vocab(\mathcal{A}), R \text{ is a relational symbol, and } t \in R^{\mathcal{A}}\} \cup \{f_{t \mapsto a} \mid t \in [dom(\mathcal{A})]^n, f \in vocab(\mathcal{A}), f \text{ is a functional symbol, and } f^{\mathcal{A}}(t) = a\}.$$

Moreover, although tuples are normally encapsulated in angled brackets $\langle \cdot \rangle$, for unary tuples, we sometimes drop these brackets and write R_a and $f_{a \mapsto b}$ instead of, respectively, $R_{\langle a \rangle}$ and $f_{\langle a \rangle \mapsto b}$. Also, for zero-ary vocabulary symbols, we may use R or f_a to respectively denote $R_{\langle \rangle}$ or $f_{\langle \rangle \mapsto a}$. Furthermore, we use the term *atom* (or *true atom*) of a structure $\mathcal{A}$ to denote a member of $\mathcal{A}$'s truth assignments. Also, we denote truth assignments of $\mathcal{A}$ by $at(\mathcal{A})$ (read atoms of $\mathcal{A}$).

Modular Systems *Each modular system abstractly represents an MX task, i.e., a set (or class) of structures over some instance (input) and expansion (output) vocabulary. Intuitively, a modular system is described as a set of primitive modules (individual MX tasks) combined some algebraic operations similar to Codd's relational algebraic operations (but working on sets of structures instead of relational tables). The algebra of modular systems is formally defined recursively starting from primitive modules:*

Definition 1 (Primitive Module). *A primitive module M is a model expansion task (or, equivalently, a class of structures) with distinct instance (input) vocabulary σ and expansion (output) vocabulary ε .*

A primitive module M can be given, for example, by a decision procedure D_M , a first- or second-order formula ϕ , or an ASP program. Before recursively defining our algebraic language, we have to define composable and independent modules [5]:

Definition 2 (Composable, Independent). *Modules M_1 and M_2 are composable if $\varepsilon_{M_1} \cap \varepsilon_{M_2} = \emptyset$ (no output interference). Module M_2 is independent from M_1 if $\sigma_{M_2} \cap \varepsilon_{M_1} = \emptyset$ (no cyclic module dependencies).*

Definition 3 (Well-Formed Modular Systems $MS(\sigma, \varepsilon)$). *The set of all well-formed modular systems $MS(\sigma, \varepsilon)$ for a given input, σ , and output, ε , vocabularies is defined as follows.*

Base Case, Primitive Modules: *If M is a primitive module with instance (input) vocabulary σ and expansion (output) vocabulary ε , then $M \in MS(\sigma, \varepsilon)$.*

Projection *If $M \in MS(\sigma, \varepsilon)$ and $\tau \subseteq \sigma \cup \varepsilon$, then $\pi_\tau(M) \in MS(\sigma \cap \tau, \varepsilon \cap \tau)$.*

Sequential Composition: *If $M \in MS(\sigma, \varepsilon)$, $M' \in MS(\sigma', \varepsilon')$, M is composable (no output interference) with M' , and M is independent from M' (no cyclic dependencies) then $(M \triangleright M') \in MS(\sigma \cup (\sigma' \setminus \varepsilon), \varepsilon \cup \varepsilon')$.*

Union: *If $M \in MS(\sigma, \varepsilon)$, $M' \in MS(\sigma', \varepsilon')$, M is independent from M' , and M' is also independent from M then $(M \cup M') \in MS(\sigma \cup \sigma', \varepsilon \cup \varepsilon')$.*

Feedback: If $M \in MS(\sigma, \varepsilon)$, $R \in \sigma$, $S \in \varepsilon$, and R and S are symbols of the same type and arity, then $M[R = S] \in MS(\sigma \setminus \{R\}, \varepsilon \cup \{R\})$.

Nothing else is in the set $MS(\sigma, \varepsilon)$.

The input-output vocabulary of module M is denoted $vocab(M)$. The description of a modular system (as in Definition 3) gives an algebraic formula representing a system. *Subsystems* of a modular system M are sub-formulas of the formula that represents M . Clearly, each subsystem of a modular system is a modular system itself.

3 Modular Systems Extended

This section generalizes modular systems with supported semantics. Informally speaking, supported semantics augments each model with some possible justification functions for that model. While the model itself carries the membership information for tuples, the justification functions carry some information in addition to these membership information: *they partially reason about why each tuple is present in a model*.

Recall that one of the great properties of modular systems is their language-independence that was achieved through using a model-theoretic semantics. Therefore, in this section, we also define supported semantics model-theoretically to preserve the language-independence property. We begin our study by defining justifications of models in a modular system.

Definition 4 (Justification for Models). Let $M \in MS(\sigma, \varepsilon)$ be a modular system and let $\mathcal{B} \in M$ be a model of M . Then, a function $j : at(\mathcal{B}|_\varepsilon) \mapsto \mathcal{P}(at(\mathcal{B}))$ that maps each true atom of $\mathcal{B}$'s expansion vocabulary to a subset of $\mathcal{B}$'s true atoms is called a justification function if and only if a well-founded ordering $<$ on $at(\mathcal{B}|_\varepsilon)$ exists so that:

for all $R_t \in at(\mathcal{B}|_\varepsilon)$ and for all $S_{t'} \in (j(R_t) \cap at(\mathcal{B}|_\varepsilon))$ we have $S_{t'} < R_t$.

Note that Definition 4 disallows circular justifications as desired and in accordance with motivations in Section 1. This is because, if something depends on itself, by transitivity of ordering relations, we should have $R_t < R_t$ (for some $R_t \in at(\mathcal{B}|_\varepsilon)$) which is impossible in all orderings. Now, let us apply this concept to our example of connectivity in graphs.

Example 6 (Connectivity Module). Consider modular system $M_C \in MS(\{E\}, \{R\})$ that computes connected pairs R of vertices from the set E of a graph's edges. Also, let $\mathcal{B} \in M_C$ be a model of M_C and define sets S^n (for $n \in \mathbb{N}$) as follows:

$$\begin{aligned} S^0 &:= \{(u, u) \mid u \text{ is a vertex}\}, \\ S^{n+1} &:= S^n \cup \{(u, v) \mid \text{vertex } w \text{ exists s.t. } (u, w) \in E^{\mathcal{B}} \text{ and } (w, v) \in S^n\}. \end{aligned}$$

By definition of connectivity, we know that $R^{\mathcal{B}} := \bigcup_{n \in \mathbb{N}} S^n$. Now, define $rank(u, v)$ to be the smallest n such that $(u, v) \in S^n$. Then, a function j is a justification function for $\mathcal{B}$ if it satisfies the following:

$$j(R_{\langle u, v \rangle}) := \begin{cases} \{\} & \text{if } rank(u, v) = 0, \\ \{E_{\langle u, w \rangle}, R_{\langle w, v \rangle}\} & \text{if } (u, w) \in E^{\mathcal{B}} \text{ and } rank(u, v) > rank(w, v). \end{cases}$$

Note that, in Definition 4 and, thus, also in Example 6, a justification function is defined only on true atoms of output vocabulary, i.e., true atoms of $\mathcal{B}|_\varepsilon$. This is because each module is only responsible for what it generates (i.e., the outputs of the module) and not what it is given (i.e., module's inputs) and, thus, the only reasonable justification one can expect from a module is justification of its outputs. For example, in the connectivity problem, it is reasonable to ask M_C to justify the truth of $R(a, b)$ (because $R(a, b)$ is generated by M_C) but it is unreasonable to ask M_C to justify the existence of an edge between vertices a and b (because M_C only receives edges in its input). The justification for truth of $E(a, b)$ should be requested from the module that has generated interpretation of E or the user of a modular system that has given E as an input.

Informally speaking, justification functions justify true atoms of a structure based on its other (better founded) true atoms. For example, if we have $j(R_a) = \{S_a, S_b\}$, it means that the presence of tuple $\langle a \rangle$ in interpretation of $R^{\mathcal{B}}$ is justified by S_a and S_b . However, justifications usually refer to incomplete conditionals and should not be misunderstood with logical implication. That is, the fact that R_a in $\mathcal{B}$ is justified by S_a and S_b does not logically imply that whenever S_a and S_b are true atoms of a model then R_a is also a true atom of that model. For example, M might have another model $\mathcal{B}'$ such that $S_a, S_b \in at(\mathcal{B}')$ but $R_a \notin at(\mathcal{B}')$. So, a true atom being justified by a set of other true atoms should not be mistaken with a true atom being implied by that other set of true atoms. Let us now define support functions for modular systems.

Definition 5 (Support Functions for Modular Systems). *Let $M \in MS(\sigma, \varepsilon)$ be a modular system. A support function for M , denoted by Sup_M is a function that associates each model $\mathcal{B}$ of M with a set of justification functions for $\mathcal{B}$, i.e., we have:*

if $\mathcal{B} \in M$ and $j \in Sup_M(\mathcal{B})$ then j is a justification function for $\mathcal{B}$.

Based on Definition 5, $Sup_M(\mathcal{B})$ is a collection of different possible justifications for structure $\mathcal{B}$. The following example shows how to define a support function for our connectivity module.

Example 7 (Connectivity Module's Support Function). Consider module M_C from Example 6. For model $\mathcal{B} \in M_C$, define $Sup_{M_C}(\mathcal{B})$ to be the set of all justification functions for structure $\mathcal{B}$ that satisfy the conditions of Example 6.

Note that, based on Definition 5, some model $\mathcal{B}$ of a modular system M might be unjustified, i.e., $Sup_M(\mathcal{B}) := \{\}$. Moreover, as we showed in Examples 1 – 3, using non-circular and well-founded justifications, we can model many problems more naturally. Therefore, we define supported model semantics as follows.

Definition 6 (Supported Model Semantics for Modular Systems). *Let $M \in MS(\sigma, \varepsilon)$ be a modular system, Sup_M be a support function for M and $\mathcal{B}$ be a $(\sigma \cup \varepsilon)$ -structure. Then, $\mathcal{B}$ is a supported model of M if (1) $\mathcal{B} \in M$, i.e., $\mathcal{B}$ is a model of M , and (2) $Sup_M(\mathcal{B}) \neq \emptyset$, i.e., $\mathcal{B}$ is supported. Also, supported semantics of modular systems is defined to be a semantics for modular systems whose intended models are exactly the set of supported models of M . Moreover, we use $Sup[M] := \{\mathcal{B} \mid \mathcal{B} \in M \text{ and } Sup_M(\mathcal{B}) \neq \emptyset\}$ denotes the set of supported models of M .*

Definition 6 distinguishes supported models simply as models with at least one possible justification. Next, we want to specify how support functions of complex modules can be obtained using support functions of their constituents, e.g., how $Sup_{M_1 \triangleright M_2}$ is defined in terms of Sup_{M_1} and Sup_{M_2} . Definitions 7–10 specify how support functions are combined together.

Definition 7 (Support of Composition $M_1 \triangleright M_2$). Let $M := M_1 \triangleright M_2$ be a well-formed modular system and let Sup_1 and Sup_2 respectively denote support functions of modular systems M_1 and M_2 . Then, support function Sup_M for composition of M_1 and M_2 is defined as follows. For each model $\mathcal{B} \in M$, we define $Sup_M(\mathcal{B})$ to contain exactly those justification functions $j : at(\mathcal{B}|_{\varepsilon_M}) \mapsto 2^{at(\mathcal{B})}$ that are obtained by combining some $j_1 \in Sup_1(\mathcal{B}|_{vocab(M_1)})$ and some $j_2 \in Sup_2(\mathcal{B}|_{vocab(M_2)})$ as below:

$$j(R_t) := \begin{cases} j_1(R_t) & \text{if } R \in \varepsilon_{M_1}, \\ j_2(R_t) & \text{if } R \in \varepsilon_{M_2}. \end{cases}$$

More intuitively, every justification function $j \in Sup_{M_1 \triangleright M_2}(\mathcal{B})$ behaves as a justification function of M_1 when applied on output vocabulary symbols of M_1 and as some other justification of M_2 when applied on output vocabulary symbols of M_2 . Proposition below states that the function defined by Definition 7 is indeed a support function according to Definition 5.

Proposition 1. Let $M := M_1 \triangleright M_2$ be a well-formed modular system. Then, Sup_M as in Definition 7 is well-defined.

Proof. First, since M is well-formed, outputs of M_1 and M_2 do not interfere, i.e., $\varepsilon_{M_1} \cap \varepsilon_{M_2} = \emptyset$. Therefore, the two cases in Definition 7 are mutually exclusive. Also, since $\varepsilon_M = \varepsilon_{M_1} \cup \varepsilon_{M_2}$, the two cases in Definition 7 cover all possible cases.

Second, every $j \in Sup_M(\mathcal{B})$ defines a non-circular and well-justified justification function. To prove this, let $j_i \in Sup_i(\mathcal{B}|_{vocab(M_i)})$ (for $i \in \{1, 2\}$) be the two justifications functions that are combined to form function j . Also, let $<_1$ and $<_2$ be two well-founded orderings that witness non-circularity of j_1 and j_2 respectively. Then, it is easy to check that well-founded ordering $<$ defined below witnesses the non-circularity of justification function j . For $R_t, S_{t'} \in at(\mathcal{B}|_{\varepsilon_M})$ we have:

$$R_t < S_{t'} \Leftrightarrow \text{one of } \left\{ \begin{array}{l} R, S \in \varepsilon_{M_1} \text{ and } R_t <_1 S_{t'}, \\ R, S \in \varepsilon_{M_2} \text{ and } R_t <_2 S_{t'}, \text{ or,} \\ R \in \varepsilon_{M_1} \text{ and } S \in \varepsilon_{M_2} \end{array} \right\} \text{ holds.}$$

Let us now define how support functions change under the projection operator in modular systems. This is much more involved than the previous case because projection hides some of the true atoms that might have been used to justify other atoms. Therefore, in order to find a justification function after a projection, we need a process of “unwinding” that we define below.

Definition 8 (τ -Unwinding). Let $M \in MS(\sigma, \varepsilon)$ be a modular system, $\mathcal{B} \in M$ be a model of M , j be a justification function for $\mathcal{B}$, and let $\tau \subseteq \varepsilon$ be a subset of output vocabulary symbols. Then, the result of unwinding j according to τ is the limit of function

series f^n (for $n \in \mathbb{N}$) defined below. All functions f^n map $at(\mathcal{B}|_\varepsilon)$ to $2^{at(\mathcal{B})}$:

$$\begin{aligned} f^0(R_t) &= j(R_t), \\ f^{n+1}(R_t) &= \{S_{t'} \mid S_{t'} \in f^n(R_t) \text{ and } S \notin \tau\} \cup \{f^n(S_{t'}) \mid S_{t'} \in f^n(R_t) \text{ and } S \in \tau\}. \end{aligned}$$

Note that Definition 8 is well-defining because the function series f^n always has a limit. This is because, by definition, justification functions are well-founded and non-circular. Therefore, there is no infinite descending chain of true atoms and, hence, for each atom $R_t \in \mathcal{B}|_{(\varepsilon \setminus \tau)}$, there exists a natural number n such that, for all $n' > n$, $f^{n'}(R_t) = f^n(R_t)$. Using the notion of unwinding, we can define the support functions of the projection operator as follows.

Definition 9 (Support of Projection). Let $M' := \pi_\tau(M)$ be a well-formed modular system, $\mathcal{B}' \in M'$ be a model of M' , and Sup_M be a support function for M . Then, $Sup_{M'}(\mathcal{B}')$ is the set of functions $j' : at(\mathcal{B}'|_{\varepsilon_{M'}}) \mapsto at(\mathcal{B}')$ for which structure $\mathcal{B}$ and function j exist such that (1) $\mathcal{B} \in M$, (2) $\mathcal{B}|_\tau = \mathcal{B}'$, (3) $j \in Sup_M(\mathcal{B})$, and, (4) unwinding j according to vocabulary $(vocab(M) \setminus \tau)$ produces a function f such that $j'(R_t) = f(R_t)$ for all $R_t \in at(\mathcal{B}'|_{\varepsilon_{M'}})$.

Informally speaking, Definition 9 says that a model $\mathcal{B}'$ of M' is justified if $\mathcal{B}'$ can be expanded to a model $\mathcal{B}$ of M (the underlying not-projected modular system) such that justifications of atoms in $at(\mathcal{B}|_\varepsilon)$ does not depend on the choice of atoms in $at(\mathcal{B}') \setminus at(\mathcal{B})$.

The following definition shows how support functions are defined after applying feedback operations. Remember that each feedback operation changes one input vocabulary symbol to an output vocabulary symbol. Therefore, the new justification function should also justify the atoms of the new output vocabulary symbol.

Definition 10 (Support of Feedback $M[P = Q]$). Let $M' := M[P = Q]$ be a well-formed modular system with $P \in \sigma_M$ and $Q \in \varepsilon_M$. Then, $Sup_{M'}(\mathcal{B})$ is the set of all functions $j' : at(\mathcal{B}|_{(\varepsilon_M \cup \{P\})}) \mapsto 2^{at(\mathcal{B})}$ such that:

- (1) $j'(P_t) = \{Q_t\}$,
- (2) justification function $j \in Sup_M(\mathcal{B})$ exists so that $j'(R_t) = j(R_t)$ for all $R_t \in at(\mathcal{B}|_{\varepsilon_M})$, and,
- (3) j' is well-founded and non-circular, i.e., well-founded ordering $<$ on $at(\mathcal{B}|_{(\varepsilon_M \cup \{P\})})$ exists such that, for all $R_t, S_{t'} \in at(\mathcal{B}|_{(\varepsilon_M \cup \{P\})})$, if $S_{t'} \in j'(R_t)$ then $S_{t'} < R_t$.

Informally speaking, a feedback operator is saying that all atoms of $\mathcal{B}|_{\varepsilon_M}$ are justified as before and the atoms of the new output vocabulary symbol P are simply justified through the atoms of Q (because P 's interpretation is exactly Q 's interpretation). However, feedbacks can generate self-justifying loops because atom Q_t might have been justified (either directly or indirectly) by P_t in M . In such a situation, adding feedback creates circular justifications because Q_t is justified by P_t as before, and P_t is now justified by Q_t . In order to disallow such circular justifications, the third condition of Definition 10 is added to guarantee that j' is a well-founded and non-circular justification function.

Definition 11 (Support of Union). Let $M := M_1 \cup M_2$ be a well-formed modular system, $\mathcal{B} \in M$ be a model of M , and Sup_1, Sup_2 be support functions for M_1 and M_2 respectively. Then $Sup_M(\mathcal{B})$ is the set of justification functions $j : at(\mathcal{B}|_{\varepsilon_M}) \mapsto 2^{at(\mathcal{B})}$ such that the following conditions is true for either $i = 1$ or $i = 2$: $\mathcal{B}|_{vocab(M_i)} \in M_i$ and justification function $j' \in Sup_i(\mathcal{B}|_{vocab(M_i)})$ exists such that (a) $j(R_t) = j'(R_t)$ for all $R_t \in at(\mathcal{B}|_{\varepsilon_{M_i}})$, and, (b) $j(R_t) = \{\}$ for all $R_t \in (at(\mathcal{B}|_{\varepsilon_M}) \setminus at(\mathcal{B}|_{\varepsilon_{M_i}}))$.

Now that we have defined all support functions for all operations of a modular system, we can state some of the properties of supported model semantics for modular systems. The first property that follows states that supported model semantics generalizes the model-theoretical semantics. We obtain this result through using a naive justification function known as *empty justification* that is defined below.

Definition 12 (Empty Justification). Let $M \in MS(\sigma, \varepsilon)$ be a modular system and $\mathcal{B} \in M$ be a model of M . Then, the justification function $e : at(\mathcal{B}|_{\varepsilon}) \mapsto 2^{at(\mathcal{B})}$ where $e(R_t) := \{\}$ for all $R_t \in at(\mathcal{B}|_{\varepsilon})$ is called the empty justification for $\mathcal{B}$. Also, the support function $Sup_{\emptyset}$ of M that associates all $\mathcal{B} \in M$ to singular set $\{e\}$ is called the empty support function of M .

Theorem 1. Let $M \in MS(\sigma, \varepsilon)$ be a well-formed modular system so that all primitive modules of M are supported by the empty support function $Sup_{\emptyset}$. Then, for all $(\sigma \cup \varepsilon)$ -structures $\mathcal{B}$, we have that $\mathcal{B} \in M$ if and only if $\mathcal{B}$ is a supported model of M .

Theorem 1 states that supported model semantics naturally generalizes the model-theoretic semantics for modular systems. Moreover, it states that empty justification functions define the essence of the model-theoretic semantics for modular systems. That is, when justification functions do not provide any extra information about possible reasons for believing in something, every model is as reasonable as every other model. Of course, when we use some non-trivial justification functions except the empty justification function, some models (i.e., supported models) become more reasonable than other models (i.e., non-supported models).

In the next section, we show that supported model semantics for modular systems is expressive enough to include both the equilibrium semantics of multi-context systems and the grounded equilibrium semantics of multi-context systems (when contexts use structural belief sets).

4 From Multi-Context to Modular Systems

This section establishes a formal connection between the expressiveness of multi-context systems and that of modular systems. We show that, in the presence of some very basic modules, multi-context systems can be encoded as modular systems. In order to do so, we give a natural translation T from MCSs to modular systems such that changing the support functions of primitive modules of the resulting modular system, we can characterize both the equilibrium semantics and the grounded equilibrium semantics of MCSs in terms of supported models of the resulting modular system. Therefore, we prove that supported model semantics of modular systems generalizes and unifies both types of equilibrium semantics (grounded or not) of multi-context systems.

4.1 Encoding MCSs and Equilibria

We first translate multi-context systems to modular systems and then talk about two possible choices as the support functions of the primitive modules used in our translation. As modular systems do not support bridge rules, we encode bridge rules using modules that perform relational algebraic operations. In our translation, we assume that each such operation is modeled by a primitive module. We use $M_{\bowtie}$ for performing relational algebraic join, $M_{\cup}$ for performing relational algebraic union, M_{Π} for performing relational algebraic projection, and M_{Compl} for performing relational algebraic complementation. We put a lot of emphasize on the term “relational algebraic” to make sure that the reader understands the difference between, e.g., primitive module $M_{\cup}$ and modular system operation $\cup$: the former is a primitive module in modular system that performs the operation of union on its input vocabulary and generates some output while the latter is not a primitive module but is an operation that combines two modular systems.

Model-theoretic assumptions on MCS For simplicity of the exposition, we focus on multi-context systems for which the belief sets BS_{L_i} of the logic L_i of each context $C_i = (L_i, kb_i, br_i)$ is a class of (relational) structures over some fixed vocabulary τ_i , where each structure corresponds to a belief set¹. The vocabulary τ_i contains, in particular, all symbols that are associated with context C_i in some bridge rule of the multi-context system. Without loss of generality, we assume that $\tau_i \cap \tau_j = \emptyset$ for $i \neq j$. For a multi-context system $MCS = (C_1, \dots, C_n)$, its vocabulary is the union of the vocabularies of its contexts, $\tau := \bigcup_{i=1}^n \tau_i$.

Translation T (from MCS to modular systems): Let $MCS := (C_1, \dots, C_n)$. For each vocabulary symbol P of MCS , introduce a new symbol P' of the same type and arity. We use these new symbols to create loops and simulate information propagation through bridge rules. Also, let us denote pairwise equalities on primed and unprimed symbols in C_i by $\overline{P'_i} = \overline{P_i}$. Then,

$$T[MCS] := (T^C[C_1] \cap \dots \cap T^C[C_n])[\overline{P'_1} = \overline{P_1}] \dots [\overline{P'_n} = \overline{P_n}],$$

Translation T^C of contexts: Translation of $C_i := \langle L_i, kb_i, br_i \rangle$ passes information from the translation of bridge rules to the translation of contexts:

$$T^C[C_i] := T^{br}[br_i] \triangleright T^L[kb_i; C_i].$$

Translation T^L of knowledge bases: Let $C_i := \langle L_i, kb_i, br_i \rangle$ and let τ_i be the vocabulary of C_i . Also, let $H_1, \dots, H_m$ be new symbols that represent heads of rules in br_i . Now, $T^L[kb_i; C_i]$ is a primitive module M with $\sigma_M = \{H_1, \dots, H_m\}$, $\varepsilon_M = \tau_i$ and:

$$\mathcal{B} \in M \iff \mathcal{B}|_{\tau_i} \in ACC_L(kb \cup \{H_i(\bar{a}) \mid \bar{a} \in H_i^{\mathcal{B}}\}).$$

Translation T^{br} of bridge rules: Bridge rules are translated to model their computation using relational algebra, i.e., (1) complement the interpretation of negated literals

¹ Non-structural belief sets can be encoded as structures for the purpose of this translation but we are not concerned with this encoding here.

using M_{Compl} , (2) join the interpretation of positive literals with the complemented interpretation of negative literals using $M_{\bowtie}$, (3) project the joint interpretation according to variables present in the head using M_{π} , and, (4) finally, use $M_{\cup}$ to compute the union of all projected interpretations for rules with the same symbol in their head. This is done by first partitioning br to subsets $br_1, \dots, br_m$ based on the head of rules, i.e., $r, r' \in br_i \Leftrightarrow hd(r) = hd(r')$. Then, we have:

$$\begin{aligned} T^{br}[br] &:= T^p[br_1] \cap \dots \cap T^p[br_m], \\ T^p[\{r_1, \dots, r_k\}] &:= (T^r[r_1] \cap \dots \cap T^r[r_k]) \triangleright M_{\cup}^p, \end{aligned}$$

where $T^r[r]$ translates one bridge rule and $M_{\cup}^p$ performs the union operation². The output vocabulary of module $M_{\cup}^p$ is the symbol H that appeared before. The input vocabulary of $M_{\cup}^p$ is $\{P_{r_1}, \dots, P_{r_k}\}$ with each P_{r_i} denoting output of one rule construction.

Translation of rule r , $T^r[r]$, is obtained by composing $T^{body}[r]$ (translation of r 's body) and M_{π}^r as follows:

$$\begin{aligned} T^r[r] &:= T^{body}[r] \triangleright M_{\pi}^r, \\ T^{body}[r] &:= (\bigcap_{P \in body^-(r)} M_{Compl}^P) \triangleright M_{\bowtie}^r, \end{aligned}$$

Translation of r 's body, as before, is computed by joining either the predicates themselves or their complement (through M_{Compl}^P).

4.2 Support Functions of Primitive Modules in Translation T

Translation T that we gave in the previous section defined the structure of a modular system. However, in order to be able to use supported model semantics for this modular system, we should define a support function for every primitive module used in this translation. The primitive modules we used in translation T were divided into two categories: the primitive modules that performed relational algebraic operations, and the primitive modules that represented a context.

Here, we specify the support functions of all relational algebraic primitive modules and leave the specification of support functions for context modules to the next sections:

Complement: $Sup_{\neg}(\mathcal{B}) = \{e\}$ where e is the empty justification function as in Definition 12.

Join: $Sup_{\bowtie}(\mathcal{B}) = \{j\}$ where $j(R_t)$ is the set of atoms $S_{t'}$ with $S \in \sigma_{M_{\bowtie}}$, $t' \in S^{\mathcal{B}}$, and $S(t') \in body^+(r)$ where r is the bridge rule represented by $M_{\bowtie}$.

Projection: Let $\sigma = \{S\}$ and $\varepsilon = \{R\}$ be the input and output vocabulary of the projection module. Then, $Sup_{\pi}(\mathcal{B})$ is the set of all mappings that take R_t to a set $\{S_{t'}\}$ where $t' \in S^{\mathcal{B}}$ and t' matches t on all projected columns (according to M_{π}).

Union: If $\sigma_{M_{\cup}} = \{S^1, \dots, S^n\}$ and $\varepsilon_{M_{\cup}} = \{R\}$, then $Sup_{\cup}(\mathcal{B})$ is the set of all different mappings that take R_t to the singleton set $\{S_{t'}^i\}$ with $1 \leq i \leq n$ and $t \in S^{i^{\mathcal{B}}}$.

² $M_{\cup}^p$ performs a standard relational algebraic operation and should not be confused with the operation that combines modules.

Intuitively, each justification function for an algebraic relational operation says why each tuple is present in the result of that operation according to the operator's semantics. For example, the semantics of algebraic operation π_1 is to keep just the first element of each tuple and discard the rest. Therefore, justification $j_{\pi_1}(R_a)$ can be either $\{S_{\langle a,b \rangle}\}$ or $\{S_{\langle a,b \rangle}\}$ but not $\{S_{\langle b,a \rangle}\}$.

4.3 Expressing Equilibrium Semantics of MCSs using Supported Model Semantics

In this part, we use the empty support function for supporting primitive modules that represented contexts in the translation given by T before. Now, we want to prove that the supported models of modular system obtained by $T[MCS]$ correctly represents MCS (under equilibria semantics). We use $vocab(S_i)$ to refer to the vocabulary of structure S_i .

Definition 13. Consider belief state $S := (S_1, \dots, S_n)$ such that $vocab(S_i) \cap vocab(S_j) = \emptyset$ (for $i \neq j$). Also, let $D_1, \dots, D_n$ be n new unary predicate symbols. Structure $\mathcal{B}$ over vocabulary $\{D_1, \dots, D_n\} \cup \bigcup_{i \in \{1, \dots, n\}} vocab(S_i)$ represents belief state S if: (1) $dom(\mathcal{B}) = dom(S_1) \cup \dots \cup dom(S_n)$, (2) $D_i^{\mathcal{B}} = dom(S_i)$ for $i \in \{1, \dots, n\}$, and, (3) $R^{\mathcal{B}} = R^{S_i}$ if $R \in vocab(S_i)$.

Definition 14. A modular system M correctly represents multi-context system $MCS = (C_1, \dots, C_n)$ under equilibria semantics if for all belief states $S = (S_1, \dots, S_n)$ and its corresponding structure $\mathcal{B}$, $\mathcal{B} \in M$ iff S is an equilibrium of MCS .

Theorem 2. Let MCS be a multi-context system and $M := T[MCS]$. Also, let the relational algebraic primitive modules of M use support functions as in Section 4.2 and contextual primitive modules of M use the empty support function. Then, M correctly represents MCS under equilibrium semantics.

Generalizing to rules with variables The original definition of multi-context systems disallows bridge rules with variables, but when limited to contexts with structural belief sets (as in our case), the original definition can easily be generalized to accommodate variables under the usual assumptions of multi-sorted logics (that are needed because belief sets might have different domains). Such an extension of multi-context system is proposed in [4]. Theorem 2 is still correct under such extensions (i.e., if variables are present and usual assumptions on multi-sorted logics are guaranteed). This further proves the robustness of supported model semantics for modular systems.

4.4 Translating Grounded Equilibria

Another semantics for multi-context systems is the grounded equilibrium semantics that is defined for the *reducible* subset of multi-context systems. Here, we show that grounded equilibria can also be represented in the extended modular system semantics. Moreover, we use exactly the same translation T as in Section 4.1 except that, now, we use different support functions for the primitive modules that represent contexts in the

translation T . Note that, here, we also use the same support functions for other relational algebraic primitive modules as given in Section 4.2.

So, it only remains to define support function of context modules. Since contexts are reducible here, we use the reducibility condition to define the proper support function. By definition of T^L , $\sigma_M := \{H^1, \dots, H^n\}$ (symbols occurring in the head of rules in br) and $\varepsilon_M := \{R^1, \dots, R^m\}$ (symbols from context C that occur in the body of some bridge rule in the multi-context system). Now, define $Sup_M(\mathcal{B})$ to be the set of all mappings from R_a^i to a minimal set H' satisfying the following conditions:

$$H' \subseteq H^{1^B} \cup \dots \cup H^{n^B} \text{ and,} \\ \{S\} = ACC(red_L(kb \cup H', \mathcal{B})) \Rightarrow \bigwedge_{i \in \{1, \dots, m\}} R^{i^B} \subseteq S.$$

Now, we want to show that, using these support functions for primitive modules, supported models of $T[MCS]$ uniquely correspond to grounded equilibria of MCS . We first prove this property for the case of MCS being a definite multi-context system and then extend it to the general case of all reducible multi-context systems.

Proposition 2. *Let MCS be a definite multi-context system and $M := T[MCS]$. Also, let relational algebraic primitive modules of M be supported by functions given in Section 4.2 and context modules of M be supported by the minimality-based function above. Then, M has a unique supported model $\mathcal{B}$ that represents the unique minimal equilibrium of MCS .*

Definition 15. *A supported modular system M correctly represents a multi-context system $MCS = (C_1, C_2, \dots, C_n)$ under grounded equilibrium semantics if for all belief states $S = (S_1, \dots, S_n)$ and its corresponding structure $\mathcal{B}$, we have $\mathcal{B}$ is a sup. model of $M \Leftrightarrow S$ is a grounded equilibrium of MCS .*

Theorem 3. *Let MCS be a reducible multi-context system and $M := T[MCS]$ be such that relational algebraic primitive modules of M are supported by functions given in Section 4.2 and context modules of M are supported by minimality-based support functions defined in this section. Then, M correctly represents MCS under grounded equilibrium semantics.*

Together, Theorems 2 and 3 show that both MCSs under equilibrium semantics and MCSs under grounded equilibrium semantics can be naturally translates into a modular system under supported model semantics. Thus, we showed that supported model semantics generalizes and unified the two different semantics of multi-context systems: its equilibrium semantics and its grounded equilibrium semantics.

5 Conclusion and Future Directions

In this paper, we showed that the concept of support in modular systems is useful for naturally representing many sets of intended models. We also showed that modular system operators that combine more primitive modules can be easily and naturally extended to allow justification functions and support functions. We also showed that our

new supported model semantics for modular systems generalizes our previous model-theoretical semantics for modular systems. In this paper, we defined supported model semantics using a model-theoretical viewpoint.

Moreover, in this paper, we showed that supported model semantics for modular systems is expressive enough to unify two different semantics that have previously been defined for multi-context systems. Therefore, we now have a formal comparison between the expressive power of these modular systems and the expressive power of multi-context systems. Secondly, we showed that capturing the two different semantics of multi-context systems is doable by just changing the support function for the modules that represent a context. In this paper, we studied only two of these support functions for context modules. It is left to study what other interesting and meaningful support functions can be used for the underlying primitive modules to obtain other useful semantics for multi-context systems.

More importantly, through relating modular systems to multi-context systems, we have provided the necessary means to cross-fertilize these two different but related frameworks for interlinking knowledge bases. For example, a consequence of Theorem 2 is that the search for equilibria in multi-context systems can be performed using the algorithm we developed in [8] for finding solutions to modular systems. An interesting future research direction is to extend that algorithm to handle modular systems with supported vocabularies, and to apply it for finding grounded equilibria of multi-context systems.

Finally, another future direction that uses the results of this paper is to develop diagnosis procedures similar to [1, 3] for modular systems under supported model semantics and thus helping users of modular system framework to find and fix bugs in a complicated modular system.

References

1. Markus Bögl, Thomas Eiter, Michael Fink, and Peter Schüller. The MCS-IE system for explaining inconsistency in multi-context systems. In Tomi Janhunen and Ilkka Niemelä, editors, *Logics in Artificial Intelligence - 12th European Conference, JELIA 2010, Helsinki, Finland, September 13-15, 2010. Proceedings*, volume 6341 of *Lecture Notes in Computer Science*, pages 356–359. Springer, 2010.
2. Gerhard Brewka and Thomas Eiter. Equilibria in heterogeneous nonmonotonic multi-context systems. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI'07) - Volume 1*, pages 385–390. AAAI Press, 2007.
3. Thomas Eiter, Michael Fink, Peter Schüller, and Antonius Weinzierl. Finding explanations of inconsistency in multi-context systems. In Fangzhen Lin, Ulrike Sattler, and Mirosław Truszczyński, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010*. AAAI Press, 2010.
4. Michael Fink, Lucantonio Ghionna, and Antonius Weinzierl. Relational information exchange and aggregation in multi-context systems. In James P. Delgrande and Wolfgang Faber, editors, *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, volume 6645 of *Lecture Notes in Computer Science*, pages 120–133. Springer, 2011.

5. Matti Järvisalo, Emilia Oikarinen, Tomi Janhunen, and Ilkka Niemelä. A module-based framework for multi-language constraint modeling. In *Proceedings of the 10th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Computer Science (LNCS)*, pages 155–168. Springer-Verlag, 2009.
6. Vladimir Lifschitz. Thirteen definitions of a stable model. In Andreas Blass, Nachum Dershowitz, and Wolfgang Reisig, editors, *Fields of logic and computation*, pages 488–503. Springer-Verlag, Berlin, Heidelberg, 2010.
7. S. Tasharrofi and E. Ternovska. A semantic account for modularity in multi-language modelling of search problems. In *Proceedings of the 8th International Symposium on Frontiers of Combining Systems (FroCoS)*, pages 259–274, October 2011.
8. Shahab Tasharrofi, Xiongnan Newman Wu, and Eugenia Ternovska. Solving modular model expansion tasks. In *Proceedings of the 25th International Workshop on Logic Programming (WLP'11)*, volume abs/1109.0583. Computing Research Repository (CoRR), September 2011.