Distributed Nonmonotonic Multi-Context Systems

Minh Dao-Tran and Thomas Eiter and Michael Fink and Thomas Krennwallner
Institute of Information Systems, Vienna University of Technology
Favoritenstrasse 9–11, A-1040 Vienna, Austria
\{dao,eiter,fink,tkren\}@kr.tuwien.ac.at

Abstract
We present a distributed algorithm for computing equilibria of heterogeneous nonmonotonic multi-context systems (MCS).

Introduction
In the last years, there has been increasing interest in systems comprising multiple knowledge bases. The rise of distributed systems and the World Wide Web fostered this development, and to date, several formalisms are available that accommodate multiple, possibly distributed knowledge bases. One of them are Multi-Context Systems (MCS), which consist of several theories (the contexts) that are interlinked with bridge rules that allow to add knowledge into a context depending on knowledge in other contexts. For instance, the bridge rule \( a \leftarrow (2 : b) \) of a context \( C_1 \) means that \( C_1 \) should conclude \( a \) if context \( C_2 \) believes \( b \). MCS have applications in various areas, such as argumentation, data integration, or multi-agent systems. Here, each context may model the beliefs of an agent while the bridge rules model an agent’s perception of the environment, i.e., other contexts. Data integration from different knowledge sources is in general a driving force for MCS applications. One possibility is the example given in (Eiter et al. 2010) for finding explanations of inconsistencies in MCS. Another real life application is project costs and time management, which typically includes many constraints like working laws, holiday restrictions, working time per week, etc. Here, regulations are kept in different knowledge bases like an ontology of personal costs, some rules that can compute the work amount for work packages, personal timekeeping, central administration data, local preferences, and so on. Bridge rules may act to build up a system whose models describe consistent states of such a management system. Moreover, multi-context systems allow to model bidirectional input and output for external knowledge sources that goes beyond the import interface of hex-programs (Eiter et al. 2005).

Roughly, MCS can be divided into monotonic and nonmonotonic MCS. Examples of the former kind are (McCarthy 1993; Giunchiglia and Serafini 1994; Ghidini and Giunchiglia 2001) (see also (Serafini and Bouquet 2004) for a comparison of these approaches), while (Brewka, Roelofsen, and Serafini 2007; Brewka and Eiter 2007) are nonmonotonic MCS. The general MCS framework of Brewka and Eiter (2007) is of special interest, as it generalizes previous approaches in contextual reasoning and allows for heterogeneous and nonmonotonic MCS, i.e., a system may have different, possibly nonmonotonic logics in its contexts (thus furthering heterogeneity), and bridge rules may use default negation (to deal, e.g., with incomplete information). Hence, nonmonotonic MCS interlinking monotonic context logics are possible. Although virtually all formalizations of MCS are inherently targeted for distributed, autonomous systems, no distributed algorithms for MCS exist. Closest to one is an algorithm for checking satisfiability of homogeneous, monotonic MCS in (Roelofsen, Serafini, and Cimatti 2004), where a centralized control accesses contexts in parallel. The lack of distributed algorithms is due to several obstacles: (i) the semantic abstraction of contexts to belief sets hinders interference with the local evaluation processes in contexts; (ii) information hiding and security aspects disable access to the context theories themselves, merely interfaces to the belief sets are provided; (iii) the complete system topology might be unknown to a context, which hinders decomposed evaluation; and (iv) the bridge rules of two contexts may refer to each other, thus creating cyclic systems that must be handled with care.

Motivated by these defiance, we contribute the following.

- As a stepping stone, we introduce partial equilibria for Brewka and Eiter-style MCS, which enable us to build
We recall some basic notions of heterogeneous nonmonotonic logic $C$, where $A$ follows below).

- To incorporate bridge rules into the local evaluations, we define loop formulas, inspired by similar compilations of logic programs into SAT theories (Lin and Zhao 2004).

We report on a DMCS prototype implementation for the setting above and show initial experimental results. To our knowledge, this is the first implementation of such MCS.

**Preliminaries**

We recall some basic notions of heterogeneous nonmonotonic multi-context systems (Brewka and Eiter 2007) and disjunctive logic programs under answer set semantics.

**Multi-Context Systems**

A logic is, viewed abstractly, a tuple $L = (\text{KB}_L, \text{BS}_L, \text{ACC}_L)$, where

- $\text{KB}_L$ is a set of well-formed knowledge bases, each being a set (of formulas),
- $\text{BS}_L$ is a set of possible belief sets, each being a set (of formulas), and
- $\text{ACC}_L$: $\text{KB}_L \rightarrow 2^{\text{BS}_L}$ assigns each $kb \in \text{KB}_L$ a set of acceptable belief sets.

This covers many (non-)monotonic KR formalisms like description logics, default logic, answer set programs, etc.

For example, a (propositional) ASP logic $L$ may be such that $\text{KB}_L$ is the set of answer set programs over a (propositional) alphabet $A$, the set of possible belief sets $\text{BS}_L = 2^A$ contains all subsets of atoms, and $\text{ACC}_L$ assigns each $kb \in \text{KB}_L$ the set of all its answer sets (more details about ASP follow below).

**Definition 1** A multi-context system (MCS) $M = (C_1, \ldots, C_n)$ consists of contexts $C_i = (L_i, kb_i, br_i), 1 \leq i \leq n$, where $L_i = (\text{KB}_i, \text{BS}_i, \text{ACC}_i)$ is a logic, $kb_i \in \text{KB}_i$ is a knowledge base, and $br_i$ is a set of $L_i$-bridge rules of the form

$$s \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), 
\text{not } (c_{j+1} : p_{j+1}), \ldots, \text{not } (c_m : p_m)$$

where $1 \leq c_k \leq n$, $p_k$ is an element of some belief set of $L_{c_k}$, $1 \leq k \leq m$, and $kb_i \cup \{s\}$ in $\text{KB}_i$ for each $kb_i \in \text{KB}_i$.

Informally, bridge rules allow to modify the knowledge base by adding $s$, depending on the beliefs in other contexts.

The semantics of an MCS $M$ is defined in terms of particular belief states, which are sequences $S = (S_1, \ldots, S_n)$ of belief sets $S_i \in \text{BS}_i$. Intuitively, $S_i$ should be a belief set of the knowledge base $kb_i$; however, also the bridge rules $br_i$ must be respected. To this end, $kb_i$ is augmented with the conclusions of all $r \in br_i$ that are applicable.

Formally, $r$ of form (1) is applicable in $S_i$ if $p_k \in S_{c_k}$ for $1 \leq i \leq j$, and $p_k \notin S_{c_j}$ for $j + 1 \leq k \leq m$. Let $\text{app}(R, S)$ denote the set of all bridge rules $r \in R$ that are applicable in $S$ and head($r$) the part $s$ of any $r$ of form (1).

**Definition 2** A belief state $S = (S_1, \ldots, S_n)$ of a multi-context system $M$ is an equilibrium iff for all $1 \leq i \leq n$, $S_i \in \text{ACC}_i(kb_i \cup \{\text{head}(r) \mid r \in \text{app}(br_i, S_i)\})$.

**Example 1** Let $M = (C_1, C_2, C_3, C_4)$ be an MCS such that all $L_i$ are ASP logics, with alphabets $A_1 = \{a\}, A_2 = \{b\}, A_3 = \{c, d, e\}, A_4 = \{f, g\}$. Suppose

- $kb_1 = \emptyset, br_1 = \{a \leftarrow (2 : b), (3 : c)\}$;
- $kb_2 = \emptyset, br_2 = \{b \leftarrow (4 : g)\}$;
- $kb_3 = \{c \leftarrow d; d \leftarrow e\}, br_3 = \{c \leftarrow e \leftarrow \text{not } (4 : f)\}$;
- $kb_4 = \{f \leftarrow g \leftarrow\}, br_4 = \emptyset$.

One can check that $S = (\{a\}, \{b\}, \{c, d, \neg e\}, \{\neg f, g\})$ is an equilibrium of $M$.

In the rest of this paper, we assume that contexts $C_i$ have finite belief sets $S_i$ that are represented by truth assignments $v_{S_i}: \Sigma_i \rightarrow \{0, 1\}$ to a finite set $\Sigma_i$ of propositional atoms such that $p \in S_i$ iff $v_{S_i}(p) = 1$ (as in Brewka and Eiter, 2007, such $S_i$ may serve as kernels that correspond one-to-one to infinite belief sets). To improve readability of our examples, we will include $\neg p$ in belief sets $S_i$ to highlight false atoms $p$. Furthermore, we assume that the $\Sigma_i$ are pairwise disjoint and that $\Sigma = \bigcup_i \Sigma_i$.

**Answer Set Programs**

Let $A$ be a finite alphabet of atomic propositions. A disjunctive rule $r$ is of the form

$$a_1 \lor \cdots \lor a_k \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n, \ (2)$$

for $k \geq 0$ and $n \geq m \geq 0$, where $a_1, \ldots, a_k, b_1, \ldots, b_n$ are atoms from alphabet $A$. We let $H(r) = \{a_1, \ldots, a_k\}$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{b_1, \ldots, b_m\}$ and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$. An answer set program $P$ is a finite set of rules $r$ of form (2).

An interpretation for $P$ is any subset $I \subseteq A$. It satisfies a rule $r$, if $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. $I$ is a model of $P$, if it satisfies each $r \in P$.

The GL-reduct (Gelfond and Lifschitz 1991) $P^I$ of $P$ relative to $I$ is the program obtained from $P$ by deleting (i) every rule $r \not\in P$ such that $B^- (r) \cap I \neq \emptyset$, and (ii) all not $b_j$, for $b_j \in B^-(r)$, from every remaining rule $r$.

An interpretation $I$ of a program $P$ is called an answer set of $P$ iff $I$ is a $\subseteq$-minimal model of $P^I$.

For example, the program $P = \{a \lor b \leftarrow; d \leftarrow a, \text{not } c; e \leftarrow b\}$ has two answer sets, viz. $\{a, d\}$ and $\{b, e\}$.

**Generic Evaluation of Nonmonotonic MCS**

The aim of this section is to provide a distributed algorithm for computing equilibria of an MCS. Taking a local stance, we consider a context $C_k$ and compute those parts of (potential)
equilibria of the system which contain coherent information from all contexts that are ‘reachable’ from $C_k$.

Let us start defining some concepts required. The notion of import closure formally captures what is ‘reachable’.

**Definition 3 (Import Closure)** Let $M = (C_1, \ldots, C_n)$ be an MCS. The import neighborhood of a context $C_k$ is the set

$$In(k) = \{c_i \mid (c_i : p_i) \in B(r), r \in br_k\}.$$  

Moreover, the import closure $IC(k)$ of $C_k$ is the smallest set $S$ such that (i) $k \in S$ and (ii) for all $i \in S$, $In(i) \subseteq S$.

Alternatively, we can constructively characterize

$$IC(k) = \{k\} \cup \bigcup_{j \geq 2} IC^j(k),$$

where $IC^0(k) = In(k)$, and $IC^{j+1}(k) = \bigcup_{i \in IC^j(k)} In(i)$. Note that the import closure of any context is finite, i.e., for an MCS $M = (C_1, \ldots, C_n)$ and $C_k$ from $M$, $|IC(k)| \leq n$.

**Example 2** Consider $M$ in Example 1. Then $In(1) = \{2, 3\}$, $In(2) = In(3) = \{4\}$, and $In(4) = \emptyset$; the import closure of $C_1$ is $IC(1) = \{1, 2, 3, 4\}$ (see Figure 1(a)).

Based on the import closure we define partial equilibria.

**Definition 4 (Partial Belief States and Equilibria)** Let $M = (C_1, \ldots, C_n)$ be an MCS, and let $\epsilon \notin \bigcup_{i=1}^{n} BS_i$. A partial belief state of $M$ is a sequence $S = (S_1, \ldots, S_n)$, such that $S_i \in BS_i \cup \{\epsilon\}$, for $1 \leq i \leq n$.

A partial belief state $S = (S_1, \ldots, S_n)$ of $M$ is a partial equilibrium of $M$ w.r.t. a context $C_k$ if $i \in IC(k)$ implies $S_i \in ACC_k(\{\text{head}(r) \mid r \in \text{app}(br_i, S)\})$, and if $i \notin IC(k)$, then $S_i = \epsilon$, for all $1 \leq i \leq n$.

As an aside, $IC(k)$ essentially defines a subsystem $M'$ of $M$ that is connected by bridge rules. We use partial equilibria of $M'$ instead of equilibria of $M'$ to keep the original MCS intact. Our view is similar to unnamed attributes in a relational database; essentially, we reference contexts in an MCS by position as in standard equilibria. Alternative representations of equilibria for subsystems are possible but would prohibit to easily talk about the initial $M$ without additional mappings from $M'$ to $M$. Thus, for the purpose of this paper it is more convenient to use the reference-by-position approach.

For combining partial belief states $S = (S_1, \ldots, S_n)$ and $T = (T_1, \ldots, T_n)$, we define their join $S \bowtie T$ as the partial belief state $(U_1, \ldots, U_n)$ with

(i) $U_i = S_i$ if $T_i = \epsilon$ or $S_i = T_i$,

(ii) $U_i = T_i$, if $T_i \neq \epsilon$ and $S_i = \epsilon$,

for all $1 \leq i \leq n$. Note that $S \bowtie T$ is void, if some $S_i = T_i$ are from $BS_i$, but different. The join of two sets $S$ and $T$ of partial belief states is then naturally defined as $S \bowtie T = \{S \bowtie T \mid S \in S, T \in T\}$.

**Example 3** Consider two sets of partial belief states:

$$S = \{(\epsilon, \{b\}, \epsilon, \{-f, g\}), (\epsilon, \{-b\}, \epsilon, \{f, -g\})\}$$

and

$$T = \{(\epsilon, \epsilon, \{-e, -d, e\}, \{-f, g\}), (\epsilon, \{e, d, -\epsilon\}, \{-f, g\}), (\epsilon, \{-e, -d, -\epsilon\}, \{f, -g\})\}.$$  

Their join is given by

$$S \bowtie T = \{(\epsilon, \{b\}, \{-e, -d, e\}, \{-f, g\}), (\epsilon, \{-b\}, \{-e, -d, -\epsilon\}, \{f, -g\})\}.$$  

**Distributed Algorithm**

Given an MCS $M$ and a starting context $C_k$, we aim at finding all partial equilibria of $M$ w.r.t. $C_k$ in a distributed way. To this end, we design an algorithm DMCS, whose instances run independently at each context node and communicate with other instances for exchanging sets of partial belief states. This provides a method for distributed model building, and the DMCS algorithm can be applied to any MCS such that appropriate solvers for the respective context logics are available. As a main feature of DMCS, it can also compute projected partial equilibria, i.e., partial equilibria projected to a relevant portion of the signature of the import closure of the starting context. This can be exploited for specific tasks like, e.g., local query answering or consistency checking. When computing projected partial equilibria, the information communicated between contexts is minimized, keeping communication cost low.

In the sequel, we present a basic version of the algorithm, abstracting from low-level implementation issues. Moreover, it is assumed that the topology of the overall MCS is not known at context nodes (at the end of this section, we discuss potential enhancements given topology information). The idea is as follows: starting from context $C_k$, we visit the import closure of $C_k$ by expanding the import neighborhood at each context like in a depth-first search, maintaining the set of visited contexts in a set $hist$, until a leaf context is reached, or a cycle is detected (by noticing the presence of the current context in $hist$). A leaf context simply computes its local belief sets, transforms all belief sets into partial belief states, and returns this result to its parent (invoking context). In case of a cycle, the context detecting the cycle, say $C_i$, must also break it, by (i) guessing belief sets for the “export” interface of $C_i$, (ii) transforming the guesses into partial belief states, and (iii) returning them to the invoking context.

The results of intermediate contexts are partial belief states, which can be joined, i.e., consistently combined, with partial belief states from their neighbors; a context $C_k$ returns its local belief sets, joined with the results from its neighbors, as final result.

For computing projected partial equilibria, the algorithm offers a parameter $V$, the relevant interface. Given a (partial) belief state $S$ and set $V \subseteq \Sigma$ of variables, the restriction of $S$ to $V$, denoted $S|_V$, is given by the (partial) belief state $S' = (S_1|_V, \ldots, S_n|_V)$, where $S_i|_V = S_i \cap V$ if $S_i \neq \epsilon$, and $\epsilon|_V = \epsilon$; the restriction of a set of (partial) belief states $S$ to $V$ is $S|_V = \{S' \mid S' \in S\}$.

Let $V(k) = \{p_i \mid (c_i : p_i) \in B(r), r \in br_k\}$ denote the import interface of context $C_k$. By $V^*(k) = \bigcup_{i \in IC(k)} V(i)$, the recursive import interface of $C_k$, we refer to the interface of the import closure of $C_k$.

Given a context $C_k$, we have two extreme cases: 1. $V = V^*(k)$ and 2. $V = \Sigma$. In Case 1, DMCS basically checks for consistency on the import closure of $C_k$ by computing partial
Algorithm 1: DMCS($V, \text{hist}$) at $C_k = (L_k, kb_k, br_k)$

Input: $V$: relevant interface, $\text{hist}$: visited contexts  
Data: $c(k)$: static cache  
Output: set of accumulated partial belief states  
(a) if $c(k)$ is not empty then return $c(k)$  
(b) if $k \in \text{hist}$ then  
\hspace{1em} // cyclic: guess local beliefs w.r.t $V$  
\hspace{2em} $S := \text{guess}(V, C_k)$  
\hspace{1em} else // acyclic: collect neighbor beliefs and add local ones  
\hspace{2em} $T := \{(\epsilon, \ldots, \epsilon)\}$ and $\text{hist} := \text{hist} \cup \{k\}$  
\hspace{1em} foreach $i \in \ln(k)$ do  
\hspace{3em} if for some $T \in T$, $T_i = \epsilon$ then  
\hspace{4em} $T := T \bowtie C_i, \text{DMCS}(V, \text{hist})$  
\hspace{1em} foreach $T \in T$ do  
\hspace{2em} $S := S \cup \text{lsolve}(T)$  
\hspace{1em} $(c(k) := S|_{V}$  
\hspace{1em} return $S|_{V}$  

Algorithm 2: lsolve($S$) at $C_k = (L_k, kb_k, br_k)$

Input: $S$: partial belief state  
Output: set of locally acceptable partial belief states  
$T := \text{ACC}(kb_k \cup \{\text{head}(r) | r \in \text{app}(br_k, S)\})$  
return $\{(S_1, \ldots, S_{k-1}, T_k, S_{k+1}, \ldots, S_n) \mid T_k \in T\}$

Algorithm 1 uses the following primitives:

- function guess($V, C_k$): guesses all possible truth assignments for the relevant interface w.r.t. $C_k$, i.e., for $\Sigma_k \cap V$.

Algorithm 2 utilizes the following steps:

1. Check the cache for an appropriate partial belief state.
2. Check for a cycle.
3. If a cycle is detected, then guess partial belief states of the relevant interface of the context running DMCS.
4. If no cycle is detected, but import from neighbor contexts is needed, then request partial belief states from all neighbors and join them.
5. Compute local belief states given the imported partial belief states collected from neighbors.
6. Cache the current (projected) partial belief state.

The next examples illustrate the evaluation runs of DMCS for finding all partial equilibria with different MCS. We start with an acyclic run.

Example 4 Reconsider $M$ from Example 1. Suppose the user invokes $C_1.\text{DMCS}(V, \emptyset)$, where $V = \{a, b, c, f, g\}$, to trigger the evaluation process. Next, $C_1$ forwards in (d) requests to $C_2$ and $C_3$, which both call $C_4$. When called for the first time, $C_4$ calculates in (e) its own belief sets and assembles the set of partial belief states

$$S_4 = \{(\epsilon, \epsilon, \epsilon, \{f, -g\}), \{\epsilon, \epsilon, \{f, -g\}\}$$

After caching $S_4|_{V}$ in (f), $C_4$ returns $S_4|_{V}$ to one of the contexts $C_2, C_3$ whose request arrived first. On second call, $C_4$ simply returns to the other context $S_4|_{V}$ from the cache.

$C_2$ and $C_3$ next call lsolve (in (e)) two times each, which results in $S_2 = S$ resp. $S_3 = T$ with $S, T$ from Example 3.

$C_1$, after computing in (d) $S_2|_{V} \bowtie S_3|_{V}$ ($= (S \bowtie T)|_{V}$ in Example 3), calls lsolve in (e) thrice to compute the final result:

$$S_1|_{V} = \left\{ \begin{array}{ll}
\{(a), \{b\}, \{c\}, \{f, -g\}\} \\
\{\{\neg a\}, \{b\}, \{\neg c\}, \{f, -g\}\} \\
\{\{\neg a\}, \{b\}, \{\neg c\}, \{f, -g\}\} \\
\end{array} \right. \right.$$  

The next example illustrates the run of DMCS on a cyclic topology.

Example 5 Let $M = (C_1, C_2, C_3)$ be an MCS such that each $L_i$ is an ASP logic, and

- $kb_1 = \emptyset$, $br_1 = \{a \leftarrow \neg (2 : b)\}$;  
- $kb_2 = \emptyset$, $br_2 = \{b \leftarrow (3 : c)\}$; and  
- $kb_3 = \emptyset$, $br_3 = \{c \leftarrow d \leftarrow \neg (1 : a)\}$.

Figure 1(b) shows the cyclic topology of $M$. Suppose that the user sends a request to $C_1$ by calling $C_1.\text{DMCS}(V, \emptyset)$ with $V = \{a, b, c\}$.

In step (d) of Algorithm 1, $C_1$ calls $C_2.\text{DMCS}(V, \{\})$, then context $C_2$ calls $C_3.\text{DMCS}(V, \{1, 2\})$, thus $C_3$ invokes $C_1.\text{DMCS}(V, \{1, 2, 3\})$. At this point, the instance of DMCS at $C_1$ detects a cycle in (b) and guesses the partial belief states

$$S_1' = \{(\{a\}, \epsilon, \epsilon), (\{\neg a\}, \epsilon, \epsilon)\}$$

for $\Sigma_1 \cap V$. Then, following the dotted lines in Figure 1(b), the set $S_2'|_{V} = S_1'$ is the return value for the request from $C_3$.

1In order to relate variables to context signatures, $V$ can either be a vector of sets, or variables in $V$ are prefixed with context ids; for simplicity, we kept $V$ as a set without further assumptions.
who joins it with \( T \) and then calls lsolve(\( T \)) for each \( T \in T \)
in \( e \), resulting in
\[
S_3 = \bigl\{ (\{ -a \}, c, \{ c, -d \}), \\
(\{ -a \}, c, \{ -c, d \}), \\
(\{ a \}, c, \{ -a, -d \}) \bigr\}.
\]

The next step of \( C_3 \) is to return \( S_3|_V \) back to \( C_2 \),
which will proceed as \( C_3 \) before, joins \( S_3|_V \) with \( T \),
and calls lsolve(\( T \)) for each \( T \in T \) in \( e \). The result is the set
of belief states
\[
S_2 = \bigl\{ (\{ -a \}, \{ b \}, \{ c \}), \\
(\{ -a \}, \{ -b \}, \{ c \}), \\
(\{ a \}, \{ -b \}, \{ -c \}) \bigr\},
\]
which will be sent back to \( C_1 \) as \( S_2|_V \). Notice that belief
state \( (\{ -a \}, \{ -b \}, \{ c \}) \) is inconsistent with \( C_1 \), but will be
eventually eliminated once \( C_1 \) evaluates \( S_2|_V \) with lsolve.

Next, \( C_1 \) will join \( S_2|_V \) with \( T \). In \( e \), \( C_1 \) uses the results
from \( C_2 \) to call lsolve, and the union gives us
\[
S_1 = \bigl\{ (\{ -a \}, \{ b \}, \{ c \}), \\
(\{ a \}, \{ -b \}, \{ -c \}) \bigr\},
\]
which is also sent back to the user as final result.

Given an MCS \( M = (C_1, \ldots, C_n) \) and a context \( C_k \),
using the recursive import interface of \( C_k \), i.e., \( V^*(k) \), as the
relevant interface is a safe (lower) bound for the correctness
of Algorithm 1. In what follows, let \( M, C_k \), and \( V^*(k) \) as
above.

**Theorem 1** For all \( V \triangleright V^*(k) \), \( S' \in C_k.\text{DMCS}(V, \emptyset) \) iff
there exists a partial equilibrium \( S' \) of \( M \) w.r.t. \( C_k \) such
that \( S' = S|_V \).

We can compute partial equilibria at \( C_k \) if we use \( V_k \).

**Corollary 2** \( S \) is a partial equilibrium of \( M \) w.r.t. \( C_k \) iff \( S \in C_k.\text{DMCS}(V_k, \emptyset) \).

Under the assumption that \( M \) has a single root context \( C_1 \),
i.e., such that \( i \in IC(1) \) for all \( 2 \leq i \leq n \), DMCS
computes equilibria. (Disconnected contexts in \( M \) can be always
connected to a new root context using simple bridge rules.)

**Corollary 3** \( S \) is an equilibrium of the MCS \( M \) iff \( S \in C_1.\text{DMCS}(V_2, \emptyset) \) for a single root context \( C_1 \).

An analysis of the algorithm yields the following upper
bound on the communication activity.

**Proposition 4** In a run of DMCS, the number of mes-
sages exchanged between contexts \( C_i, \) where \( i \in IC(k) \),
is bounded by \( 2 \cdot |E(k)| \), where \( E(k) = \{(i, c_j) \mid i \in IC(k), r \in br_i, (c_j : p_j) \in B(r)\} \).

**Discussion**

Algorithm DMCS naturally proceeds “forward” in the im-
port direction of context \( C_k \). Thus, starting from there, it
computes partial equilibria which cover \( C_k \) and contexts in
its import closure. All other contexts will be ignored; in fact,
they are unknown to all contexts in the closure. While partial
equilibria may exist for \( C_k \) and its import closure, the whole
MCS could have no equilibrium, because, e.g., (P1) contexts
that access beliefs from \( C_k \) or its closure get inconsistent, or
(P2) an isolated context or subsystem is inconsistent.

Enhancements of DMCS may deal with such situations:
As for (P1), the context neighborhood may include both
importing and supporting contexts. Intuitively, if \( C_i \) imports
from \( C_j \), then \( C_i \) must register to \( C_j \). By carefully adapting
DMCS, we can then solve (P1). However, (P2) remains;
this needs knowledge about the global system topology.

A suitable assumption is the existence of a manager \( M \)
that is reachable from every context \( C_i \) in the system,
which can ask \( M \) whether some isolated inconsistent context or
subsystem exists. If \( M \) affirms, \( C_i \)’s DMCS simply returns \( \emptyset \),
eliminating all partial equilibria.

We can weaken the manager assumption by introducing
routers (improving decentralization and information encapsu-
lation). Instead of asking the manager, a context \( C_i \) queries
an assigned router \( R \), which collects the necessary topology
information for \( C_i \) or makes a cache look-up. The informa-
tion exchange between \( C_i \) and \( R \) is flexible, depending on
the system setting, and could contain contexts that import
information from \( C_i \), or isolated and inconsistent contexts.

A further advantage of topological information is that \( C_i \)
can recognize cyclic and acyclic branches upfront, and the
invocation order of the neighborhood can then be optimized,
by starting with all acyclic branches before entering cyclic
subsystems. The caching mechanism can be adapted for
acyclic branches, as intermediate results are complete and
the cache is meaningful even across different evaluation sessions.

In our setting, we are safe assuming that \( V^*(k) \subseteq V \). But
this is not needed if \( M \) resp. the import closure of \( C_k \) has no
join-contexts, i.e., contexts which have at least two parents. If
we have access to path information in $M$ at each context, we could calculate $V$ on the fly and change it accordingly during MCS traversal. In particular, for tree-shaped or ring topology, we can restrict $V$ to the local shared interface between $C_k$ and its import neighbors. In absence of join-contexts, $V$ must be made “big enough,” e.g., using path information (see Bairakkad et al., 2010). Furthermore, join-contexts may be eliminated by virtually splitting them, if orthogonal parts of the contexts are accessed. This way, scalability to many contexts can be achieved.

Loop Formulas for Multi-Context Systems

Algorithm DMCS incorporates in step (e) via Isolve the bridge rules $br_k$ into the local knowledge base $kb_k$, given belief input from a belief state $T$, and then computes the belief sets; this is done for all $T \in T$.

In certain settings, it is possible to compile $br_k$ into $kb_k$, yielding some $kb'_k$, such that the belief sets of $kb'_k$ are precisely the possible belief sets $T_k$ in the return value of any Isolve($S$); hence, the for-loop in step (e) can be replaced by a single join $S := T \bowtie B_{k_i} \mid v$, where $B_k$ are the acceptable belief sets of $kb'_k$, properly converted to partial belief states.

For example, this is possible for classical logics $L_k$ (assuming that contexts are not self-referential), or ASP logics. This is because there are well-known transformations of ASP programs $P$ into equivalent classical theories $\phi(P)$, such that the answer sets of $P$ are given by the classical models of $\phi(P)$, which hinge on loop formulas (Lin and Zhao 2004; Lee and Lifschitz 2003). Informally, such formulas ensure that cyclic rules like $a \leftarrow b$, $b \leftarrow a$ are satisfied, viewed as classical implications, by assigning $a$ and $b$ true only if there is support for the truth of $a$ or $b$ by some other rule.

In this section, we develop loop formulas for MCS’s, by which bridge rules can be compiled into a local classical theory. In fact, we combine this with a loop formula transformation of ASP programs into classical theories; this enables us to obtain particular equilibria satisfying groundedness. Roughly, we adapt support such that also bridge rules have an effect on loops (e.g., $a \leftarrow (1 : c)$ on the loop above), but we distinguish local support and bridge support.

We assume that in $M = (C_1, \ldots, C_n)$, all logics $L_i$ are ASP logics with $\Sigma_i = \mathcal{A}_i$, and $\Sigma = \mathcal{A}$. Furthermore, we assume that all heads of bridge rules are (disjunctive) facts (this is no loss of generality). This allows us to adapt disjunctive loop formulas to encode bridge rules as classical theories.

Let $\neg A = \{\neg a \mid a \in A\}$, and, as usual, $\bigvee F = \bigvee_{f \in F} f$ and $\bigwedge F = \bigwedge_{f \in F} f$ (note that $\bigwedge \emptyset = \bot$ and $\bigvee \emptyset = \top$).

For any ASP rule $r$, we then define

$$\kappa(r) = \bigwedge B^+(r) \land \bigwedge \neg B^-(r) \supset \bigvee H(r),$$

and for any set $R$ of ASP rules, $\kappa(R) = \bigwedge_{r \in R} \kappa(r)$.

The support formula of a set $A \subseteq \mathcal{A}$ w.r.t. an ASP rule $r$ is

$$\varepsilon(A, r) = \bigwedge B^+(r) \land \bigwedge \neg B^-(r) \land \bigwedge \neg (H(r) \land A),$$

and w.r.t. any set $R$ of ASP rules, $\varepsilon(A, R) = \bigvee_{r \in R} \varepsilon(A, r)$.

To build support formulas w.r.t. a bridge rule $r$ of form (1), we convert it to an ASP rule $\ell(r)$ by replacing $(c_k : p_k)$ with $p_k$, $1 \leq k \leq m$; for any bridge rule set $R$, we let $\ell(R) = \{\ell(r) \mid r \in R\}$.

We identify the support rules and the external support rules of a set of ASP rules $R$ w.r.t. a set $A \subseteq \mathcal{A}$ as

$$SR(A, R) = \{r \in R \mid H(r) \cap A \neq \emptyset\} \text{ and}$$

$$ER(A, R) = \{r \in R \mid H(r) \cap A \neq \emptyset, B^+(r) \cap A = \emptyset\},$$

respectively (note the $SR(A, R)$ are not minimizing).

We next define necessary dependency relations. In a set of ASP rules $R$, we say that $a$ depends on $b$, denoted $a \rightarrow b$, if $a \in H(r) \land b \in B^+(r)$ for some rule $r \in R$. The set of dependencies in context $C_i$ is then the set of all pairs $a \rightarrow b$ such that $a \rightarrow b$ in $kb_i \cup \ell(br_i)$.

Based on this, we define the dependency graph of an MCS and loops for contexts and MCS’s.

**Definition 5 (Dependency Graph and Loops)** The dependency graph of an MCS $M = (C_1, \ldots, C_n)$ is the directed graph $G = (\mathcal{A}, \bigcup_{1 \leq i \leq n} \rightarrow_i)$.

A loop of $C_i$ (resp., $M$) is an any set $L \subseteq A_i$ (resp., $L \subseteq \mathcal{A}$) of atoms iff the subgraph of $G$ induced by $L$ is strongly connected.

Note that each singleton $\{a\}$ is a loop.

**Example 6** Consider $M = (C_1)$ and $M' = (C_1')$, where

$$kb_1 = \{a \leftarrow b \}, \quad br_1 = \{a \leftarrow (1 : b)\},$$

$$kb'_1 = \{c \leftarrow d \}, \quad br'_1 = \{c \leftarrow (1 : c)\}.$$}

The loops of $C_1$ are $\{a\}$, $\{b\}$, and $\{a, b\}$, and those of $C_1'$ are $\{c\}$, $\{d\}$, and $\{c, d\}$.

Next we define the local loop formula of a context.

**Definition 6 (Local Loop Formulas)** Let $L$ be a loop of context $C_i$. Then the loop formula for $L$ w.r.t. $C_i$ is

$$\lambda(L, C_i) = \left(\bigvee L \supset \varepsilon(L, ER(L, kb_i) \cup \bigwedge \text{SR}(L, \ell(br_i)))\right).$$

Furthermore, the loop formula of context $C_i$ is the conjunction $\lambda(C_i) = \bigwedge L \lambda(L, C_i)$ of all loops $L \subseteq C_i$.

**Example 7** Continuing Example 6, $L_1 = \{a, b\}$ has the loop formula $i) \lambda(L_1, C_1) = a \lor b \lor b \lor a$. Indeed, both rules of $br_1$ are support rules of $C_1$ w.r.t. $L_1$, which has no external support rules in $kb_1$; thus $\lambda(L_1, C_1) = a \lor b \lor \varepsilon(L_1, \ell(br_1))$, and $\varepsilon(L_1, \ell(br_1)) = b \lor a$. Similarly, $L_1' = \{c, d\}$ has the loop formula (ii) $\lambda(L_1', C_1') = c \lor d \lor \neg d \lor \neg c$.

Experts on loop formulas will notice that (i) is weaker than the loop formula $a \lor b \lor \supset$ of the program $\{a \leftarrow b ; b \leftarrow a\}$, and admits $\{a, b\}$ as a model of the translation; this complies with the MCS semantics. Similarly, (ii) is weaker than $c \lor d \lor \supset$ but can eliminate the model $\{c, d\}$. Finally, we have $\lambda(\{a\}, C_1) = a \lor b$, $\lambda(\{b\}, C_1) = b \lor a$, $\lambda(\{c\}, C_1') = c \lor d \lor \neg d$, and $\lambda(\{d\}, C_1') = d \lor c \lor \neg c$.

We now can transform the whole MCS into a formula.
Applying then valuations of the atoms in bridge rule bodies; Janhunen et al. 2009). Furthermore, we have developed
\[ \pi(C_i) \] and \[ \lambda(C_i) \] \(1 \leq i \leq n\), and
\[ \pi(M) = \bigwedge_{i=1}^{n} \pi(C_i). \]

Here, \(\pi(C_i)\) describes the belief sets of \(C_i\), depending on valuations of the atoms in bridge rule bodies; \(\pi(M)\) just aligns descriptions. The next result shows that this correctly characterize the equilibria of \(M\).

**Theorem 5** The equilibria of any \(M\) with ASP logics correspond one-to-one to the models of the formula \(\pi(M)\).

Note that by this theorem, consistency of \(M\) (i.e., existence of some equilibrium) maps to a distributed SAT problem.

**Example 8** For \(M'\) from Example 6, we have
\[ \pi(M) = (b \lor a) \land (a \lor b) \land (a \lor b \lor a \lor b), \] and
\[ \pi(M') = (c \lor d) \land (d \lor c) \land (c \land d) \land (\neg d \land c) \land \]
\[ (c \lor d \land d) \land (d \lor c \land \neg d) \land (\neg d \land c \land \neg d). \]

Clearly, \(\pi(M)\) has the models \(\emptyset\) and \(\{a, b\}\), while it can be checked that \(\pi(M')\) has no model.

**Example 9** Let us reconsider \(M\) from Example 1. We have
\[ \pi(C_1) = \lambda(C_1) = b \land c \land a \land \lambda(C_1) = a \land b \land c \land \lambda(C_2) = b \land g \land \lambda(C_3) = d \land c \land \lambda(C_3) = d \land c \land \lambda(C_4) = f \lor g \land \lambda(C_4) = f \land g \land \lambda(C_4). \]

The formula \(\pi(M) = \pi(C_1) \land \ldots \land \pi(C_4)\) has three models, namely \(\{a, b, c, d\}, \{b, e, g\}\), and \(\{f\}\). They correspond to the three projected equilibria of \(M\) shown in Example 4.

We can adapt the algorithm DMCS for \(\pi\) easily: after step (a), we insert \(S := \text{Isolve}(\epsilon(e_1, \ldots, e_n)); \) moreover, we delete in step (c) the for-loop, and replace in step (e) the for-loop with \(S := (\forall v < T)\). Furthermore, we replace in \(\mathcal{M}\) each context \(C_i\) with \(C'_i = (L'_i, \pi(C'_i), b^{r'}),\) where \(L'_i\) is propositional logic, \(\pi(C'_i)\) is a renaming of \(\pi(C_i)\) such that variables in different contexts are disjoint, and \(b^{r'}\) contains \(\{a_{i_1} \leftarrow (j : a_{j_1}); \neg a_{i_1} \leftarrow (j : \neg a_{j_1})\}\) for every renamed original atom \(a\) occurring in both \(\pi(C_i)\) and \(\pi(C'_i)\), \(i \neq j\). Applying then DMCS to the \(M\)’s, we obtain the equilibria of \(M\).

**Theorem 5** may be generalized to contexts with extensions of ASP logics that have loop formula characterizations, like those in (Ferraris, Lee, and Lifschitz 2006; Janhunen et al. 2009). Furthermore, we have developed such a characterization for modal logic programs (Dao-Tran et al. 2009), which feature modules akin to imperative programs and have increased expressiveness.

Note that this loop formula characterization of equilibria may lead in the worst case to an exponential blow-up in the size of the MCS. This is not surprising, as standard loop formulas (Ferraris, Lee, and Lifschitz 2006; Lin and Zhao 2004;

Lee and Lifschitz 2003) also face this situation, and Lifschitz and Razborov (2006) show that this unavoidable, under the widely believed assumption from computational complexity theory that polynomial time computations cannot be simulated with small propositional formulas. A remedy would be to encode bridge rules in answer set programs. Note however, that some ASP solvers like ASSAT rely internally on loop formulas and SAT solving techniques for model search; thus the expected performance gain from a short ASP encoding might not always surface in practice.

**Loop Formulas for Grounded Equilibria**

Equilibria lack groundedness in general, as cyclic bridge rules might be applied unfoundedly (e.g., \(a \leftarrow (1 : b)\) and \(b \leftarrow (a)\) in Example 6). To overcome this, grounded equilibria were proposed in (Brewka and Eiter 2007) for certain MCS’s, in which bridge rules intuitively act under ASP semantics.

Our transformation to capture grounded equilibria. We restrict here to normal ASP logics \(L_i\), i.e., \(KB_i\) is the set of all normal (disjunction-free) ASP programs (this ensures a technical reducibility condition for \(L_i\)).

Adapting Definitions 6 and 7, we define global loop formulas.

**Definition 8** (Global Loop Formulas) Let \(\mathcal{L}\) be a loop of \(MCS = (C_1, \ldots, C_n)\). The loop formula for \(\mathcal{L}\) w.r.t. \(M\) is
\[ \lambda(\mathcal{L}, M) = \bigvee_{i=1}^{n} \varepsilon(\mathcal{L}, \bigcup_{i=1}^{n} \mathcal{E}(\mathcal{L}, \lambda(kb_i \cup \ell(br_i)))). \]

and the loop formula of \(M\) is the conjunction \(\lambda(M) = \bigwedge_{\mathcal{L}} \lambda(\mathcal{L}, M)\) for all loops \(\mathcal{L}\) of \(M\). Furthermore, we let
\[ \pi_{GE}(M) = \lambda(M) \land \bigwedge_{i=1}^{n} (\varepsilon(kb_i) \land \varepsilon(\ell(br_i))). \]

We then can show that \(\pi_{GE}\) captures grounded equilibria.

**Theorem 6** The grounded equilibria of any \(M\) with normal ASP logics correspond one-to-one to the models of \(\pi_{GE}(M)\).

**Example 10** The MCS \(M\) in Example 6 has
\[ \pi_{GE}(M) = (a \lor b) \land (b \lor a) \land (a \lor b \lor \bot) \lor \] as the external support rules \(\mathcal{E}(\{a, b\}, kb_1 \cup \ell(br_1)) = \emptyset\). The only model of \(\pi_{GE}(M)\) is \(\emptyset\), which corresponds to the grounded equilibrium of \(M\).

Note that DMCS cannot be run straightforwardly on \(\pi_{GE}(M)\), as the formulas in \(\pi_{GE}(M)\) are intermingled and prohibit a clear context separation. Intuitively, this is due to the encoded groundedness check. We can overcome this simply by extending \(M\) for \(\pi(M)\) above to \(M'' = (C_0, C_1, \ldots, C_n)\), where \(C_0 = (\lambda_0, \pi_{GE}(M), br_0)\) has propositional logic \(\lambda_0\) and \(br_0 = \bigcup_{i=1}^{n} \{a \leftarrow (i : a) | a \in A_i\}\); intuitively, \(C_0\) filters out grounded equilibria. We then run DMCS on \(M''\) at \(C_0\).

**Implementation and Experimental Results**

We present initial results for a SAT-solver based prototype implementation of DMCS under Linux, written in C++.\(^3\)
We used the development version of clasp (2010-01-31) as a SAT solver, which accepts DIMACS CNF input (Gebser et al. 2007). Specifically, all $\pi(C_k)$ are encoded as CNF clauses and clasp builds all models at a context $C_k$. Conceptually, any SAT solver that enumerates all models would serve the same purpose. Every context has its own server process running on the machine.

Distributed model computation for nonmonotonic systems is new, and there are no implementations that would serve as a basis for comparison with our approach. We do not plan to use large systems with thousands of contexts, and pay attention to systems that do not have too many contexts (up to several dozens). For this, we randomly created small MCS’s and report initial findings on scalability issues.

For initial experimentation, we created random MCS instances of fixed topologies generalizing the diamond and ring in Figure 1(a) and 1(b) (stacking $m$ diamonds in a tower of $3m + 1$ nodes). A parameter setting $P = (n, s, b, r)$ specifies (i) the number $n$ of contexts, (ii) the local alphabet size $|\Sigma_i| = s$ (each $C_i$ has a random ASP program on $s$ atoms with $2^k$ answer sets, $0 \leq k \leq s/2$), (iii) the maximum interface size $b$ (number of atoms exported), (iv) and the maximum number $r$ of bridge rules per context, each having at most 2 body literals.

Table 1 shows some experimental results for $P_1 = (7, 8, 4, 4)$ and $P_2 = (10, 12, 6, 6)$. Each row $D_j$ (resp., $R_j$) displays pure computation time (no output) for sequential diamonds (resp., a ring), where the # columns show the numbers of projected partial equilibria computed at $C_1$ (initiated by sending the request $V^{{\pi}}(1)$ to $C_1$). The response time varies a lot, especially for $P_2$, as well as the result size, due to the particular instances. All works quite fast for small interfaces; increasing $s$ should, modulo local solving and projection time, not lead to an overall increase. Stacking multiple diamonds in a tower ($m = 3$ for $P_2$) models hard instances with many joins. This is reflected in the ratio of running time to result size, compared to the ring. There is a lot of room for improvement; e.g. with the local shared interface optimization for ring, thus fewer results, we gain a significant speedup for $P_2$ (<1.4 secs in $R_1$, <0.8 secs in all other cases). Figure 2(a) and 2(b) show a graphical representation of the runtime behavior for the tests above.

### Table 1: Runtime (secs) on Pentium M 1.73GHz, 1GB RAM

<table>
<thead>
<tr>
<th>topology</th>
<th>$P_1 = (7, 8, 4, 4)$</th>
<th>#</th>
<th>$P_2 = (10, 12, 6, 6)$</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td>0.248</td>
<td>1014</td>
<td>19.520</td>
<td>1944</td>
</tr>
<tr>
<td>$D_2$</td>
<td>0.129</td>
<td>56</td>
<td>3.361</td>
<td>5240</td>
</tr>
<tr>
<td>$D_3$</td>
<td>0.159</td>
<td>32</td>
<td>18.948</td>
<td>15088</td>
</tr>
<tr>
<td>$D_4$</td>
<td>0.121</td>
<td>348</td>
<td>46.249</td>
<td>14596</td>
</tr>
<tr>
<td>$D_5$</td>
<td>0.085</td>
<td>12</td>
<td>11.289</td>
<td>5220</td>
</tr>
<tr>
<td>$R_1$</td>
<td>0.109</td>
<td>176</td>
<td>33.020</td>
<td>120736</td>
</tr>
<tr>
<td>$R_2$</td>
<td>0.106</td>
<td>208</td>
<td>2.285</td>
<td>10548</td>
</tr>
<tr>
<td>$R_3$</td>
<td>0.185</td>
<td>80</td>
<td>0.637</td>
<td>2200</td>
</tr>
<tr>
<td>$R_4$</td>
<td>0.135</td>
<td>111</td>
<td>12.176</td>
<td>98656</td>
</tr>
<tr>
<td>$R_5$</td>
<td>0.320</td>
<td>2119</td>
<td>6.870</td>
<td>19388</td>
</tr>
</tbody>
</table>

**Related Work**

Roelofsen, Serafini, and Cimatti (2004) described evaluation of monotone MCS with classical theories using SAT solvers for the contexts in parallel. They used a (co-inductive) fixpoint strategy to check MCS satisfiability, where a centralized process iteratively combines results of the SAT solvers. Apart from being not truly distributed, an extension to nonmonotonic MCS is non-obvious; also, no caching was used.

Serafini and Tamilin (2005) and Serafini, Borgida, and Tamilin (2005) defined distributed tableau algorithms for reasoning in distributed ontologies, which are akin to MCS. Distributed ontologies enforce a stronger coupling of the contexts as they provide an alignment of the vocabulary. Thus, the abstract view of MCS is not possible. The algorithms can be used to decide consistency of distributed description logic knowledge bases, provided that the distributed TBox is acyclic. The DRAGO system implements this approach.

Hirayama and Yokoo (2005) dealt with distributed SAT (DisSAT), where the local theories (agents) are propositional.
We have presented a basic distributed algorithm, while computing multiple (or all) equilibria is not supported. Proof

We can now prove Theorem 1. 

\[ S^i = C_i \cdot \text{DMCS}(V, \text{hist} \cup \{k\}), \]

\[ \vdots \]

\[ S^{i_m} = C_{i_m} \cdot \text{DMCS}(V, \text{hist} \cup \{k\}), \]

such that for every \( S^{i_j} \in S^i \), there exists a partial equilibrium \( S^{i_j} \) of \( M \) w.r.t. \( C_i \) such that \( S^{i_j} \mid V = S^{i_j} \).

Since \( \text{In}(k) \neq \emptyset \), step (d) is executed; let

\[ T = S^{i_1} \supset \cdots \supset S^{i_m} \]

be the result of calling all DMCS at \( C_{i_1}, \ldots, C_{i_m} \). Furthermore, let \( S = \bigcup \{\text{Isolve}(S') \mid S \in T\} \) be the result of executing step (e). Eventually, \( S' \in S\mid V \). Since every DMCS at \( C_{i_1}, \ldots, C_{i_m} \) returns its partial equilibria w.r.t. \( C_i \) projected to \( V \), we have that every \( T \in T \) is a partial equilibrium w.r.t. \( C_i \) projected to \( V \). \( M \) is acyclic and we have visited all contexts from \( \text{In}(k) \), thus by Lemma 7 we get that for \( T \in T \), \( \text{app}(br_k,T) \) gives us all applicable bridge rules \( r \) regardless of \( T_j = \epsilon \) in \( T \), as \( j \notin \text{In}(k) \). Hence, for all \( T \in T \), \( \text{Isolve}(T) \) returns only partial belief states, where each component is projected to \( V \) except the 4th component. As every \( T \in T \) preserves applicability of the rules by Lemma 7, we get that for every \( S' \in S\mid V \), there exists a partial equilibrium \( S \) of \( M \) w.r.t. \( C_i \) such that \( S' = S\mid V \).

The proof for cyclic \( M \) works similarly. In a run we eventually end up in a context \( C_i \) such that \( i \in \text{hist} \) again and guess in step (c). Thus, all possible projected partial belief states \( (\epsilon, \ldots, \epsilon, \epsilon, \ldots, \epsilon) \) on \( V \) will be returned and the calling context can proceed with the result. The result is then propagated throughout the system and will be returned to the first invoked instance of DMCS at \( C_i \), which is the first context of the cycle. Here, another call to \( \text{Isolve} \) then assures that only partial equilibria survive and will be returned to the calling context, or if \( C_i = C_k \), to the user.

\( (\Rightarrow) \) We give now a proof sketch for soundness of DMCS. Let \( S' \in C_k \cdot \text{DMCS}(V, \emptyset) \) such that \( V \supseteq V^{*}(k) \). We show now that there is a partial equilibrium \( S \) of \( M \) w.r.t. \( C_k \) such that \( S' = S|_{V} \). We proceed by structural induction on the topology of an MCS, and start with acyclic MCS \( M \).

Base case: \( C_k \) is a leaf with \( \text{In}(k) = \emptyset \) and \( br_k = \emptyset \) and \( k \notin \text{hist} \). This means that (d) is not executed, hence, in (e), \( \text{Isolve} \) runs exactly once on \((\epsilon, \ldots, \epsilon)\), and we get as result the set of all belief states \( S = \text{lsolve}(\epsilon, \ldots, \epsilon) = \{(\epsilon, \ldots, \epsilon, T_k, \epsilon, \ldots, \epsilon) \mid T_k \in \text{ACC}_k(kb)\} \). We further get that \( S' \subseteq S|_{V} \). Towards a contradiction, assume that there is no partial equilibrium \( S = (S_1, \ldots, S_n) \) of \( M \) w.r.t. \( C_k \) such that \( S' = S|_{V} \). From \( \text{In}(k) = \emptyset \), we get that \( IC(k) = \{k\} \), thus the partial belief state \((\epsilon, \ldots, T_k, \epsilon, \ldots, \epsilon) \in S \) is a partial equilibrium of \( M \) w.r.t. \( C_k \). Contradiction.

Induction step: assume that the import neighborhood of context \( C_k \) is \( \text{In}(k) = \{i_1, \ldots, i_m\} \) and

\[ S^{i_1} = C_{i_1} \cdot \text{DMCS}(V, \text{hist} \cup \{k\}), \]

\[ \vdots \]

\[ S^{i_m} = C_{i_m} \cdot \text{DMCS}(V, \text{hist} \cup \{k\}), \]

Regarding possible applications in the Web field, distributed MCS could serve as a host language for distributed SPARQL reasoning (Schenk and Staab 2008). For that, a non-ground semantics must be developed that relies on grounding as customary in logic programming.

Appendix

Proof Sketch for Theorem 1

Lemma 7 For any context \( C_i \) and partial belief state \( S \) of an MCS \( M = (C_1, \ldots, C_n) \), \( \text{app}(br_k, S) = \text{app}(br_k, S|_{V}) \) for all \( V \supseteq V^{*}(k) \).

Proof Follows from

\[ r \in \text{app}(br_k, S) \Leftrightarrow \forall (r_i : p_i) \in B^+(r) : p_i \in S_{r_i} \land \forall (r_k : p_k) \in B^-(r) : p_k \notin S_{r_k} \]

\[ \Leftrightarrow \forall (r_i : p_i) \in B^+(r) : S_{r_i} \subseteq S|_{V} \land \forall (r_k : p_k) \in B^-(r) : S_{r_k} = S|_{V} \]

\[ \Rightarrow S \supseteq V^{*}(k) \land S|_{V} \subseteq S|_{V} \]

\[ \Rightarrow r \in \text{app}(br_k, S|_{V}). \]

We can now prove Theorem 1.

\[ \Rightarrow \] We start by showing soundness of DMCS. Let \( S' \in C_k \cdot \text{DMCS}(V, \emptyset) \) such that \( V \supseteq V^{*}(k) \). We show now that
have to show that $S'_i = S_i \downarrow V$. Since $V \supseteq V^*(k)$, we get by Lemma 7 that the applicability of bridge rules in $br_i$ is preserved. Thus, at each $C_i$, the return value $S_i \downarrow V$ contains $S'$ and so $S'_i$.

**Proof Sketch for Proposition 4**

The proof sketch is as follows: for a context $C_k$ of an MCS $M = (C_1, \ldots, C_n)$, the set $E(k)$ contains all dependencies from contexts $C_i$ for $i \in IC(k)$. Since we visit all $(i, j) \in E(k)$ exactly twice during DFS-traversal of $M$ (once when calling $C_j$, DMCs($V$, hist) at $C_i$, and once when retrieving $S_i \downarrow V$ from $C_j$ in $C_i$), the claim follows.

**Proof Sketch of Theorem 5**

$(\Rightarrow)$ Let $S = (S_1, \ldots, S_n)$ be an equilibrium of the MCS $M$. We have that $S_i$ is an answer set of

$$R_i = kb_i \cup \{head(r) \mid r \in app(br_i, S)\}.$$  

We show now that $S = \bigcup_{1 \leq i \leq n} S_i$ is a model of $\pi(M)$ by showing that, for all $1 \leq i \leq n$,

(i) $S_i \models \kappa(kb_i)$,

(ii) $S_i \models \kappa(\ell(br_i))$, and

(iii) $S_i \models \lambda(C_i)$

Part (i): From $S_i$ being an answer set of $R_i$, we get $S_i \models r$ for all $r \in kb_i$. It follows that $S_i \models \kappa(r)$, hence $S_i \models \kappa(kb_i)$, and finally, we get that $S \models \kappa(kb_i)$.

Part (ii): Take an arbitrary bridge rule $r \in br_i$ of form (1):

- If $r \not\in app(br_i, S)$, then there either exists $(c_i : p_i)$ in $r$ such that $p_i \not\in S_i$, for $1 \leq i \leq j$, or there exists $(c : p)$ in $r$ such that $p_k \not\in S_{i_k}$ for $j + 1 \leq k \leq m$. Since the sets $S_i$ are pairwise disjoint, we get that $S$ does not satisfy the antecedent of $\kappa(\ell(r))$.

- If $r \in app(br_i, S)$, then for all $(c_i : p_i)$ for $1 \leq i \leq j$, $p_i \in S_{i_k}$, and for all $(c : p)$ for $j + 1 \leq k \leq m$, $p_k \not\in S_{i_k}$. As the sets $S_i$ are pairwise disjoint, we have that $S$ satisfies the antecedent of $\kappa(\ell(r))$. Furthermore, since $r$ is applicable in $S$, $head(r)$ was added to $R_i$ to determine $S_i$, thus $S_i \models head(r)$ and so we have that $S$ satisfies the consequent of $\kappa(\ell(r))$.

In both cases we can derive that $S \models \kappa(\ell(r))$ for all $r \in br_i$, and eventually $S \models \kappa(\ell(br_i))$.

Part (iii): Now take an arbitrary loop $L \subseteq A_i$ of $C_i$. We have to show that

$$S \models \left( \bigvee L \right) \supset \varepsilon(L, ER(L, kb_i) \cup SR(L, \ell(br_i)))$$

(3)

holds. If $L \cap S_i = \emptyset$, then (3) holds vacuously. Otherwise, $S \models \bigvee L$, so we have to show that $S$ satisfies the consequent of (3), i.e., for some $r \in ER(L, kb_i) \cup SR(L, \ell(br_i))$, we get

$$S \models \bigwedge B^+(r) \land \bigwedge \neg B^-(r) \land \bigwedge \neg (H(r) \land L).$$

(4)

Recall that $S_i$ is an answer set of $R_i$. We obtain that $T \models \bigwedge B^+(r)$ and $T \not\models H(r)$, thus $B^+(r) \subseteq T$ and $B^-(r) \cap T = \emptyset$.

We have that there is a rule $r \in R_i$ such that $t$ is the reduced rule $r$. Since $T \subseteq S_i$, we get $S_i \models B(r)$, as $S_i \models B^+(r)$ follows from $t \in R_i$. This means that $S_i \models H(r)$, hence $S_i \cap H(r) \neq \emptyset$. Since $T \not\models H(r)$, we also get that $T \cap H(r) = \emptyset$. Thus, $L \cap H(r) \neq \emptyset$.

We obtain two cases for $r \in R_i$:

- $r \in kb_i$: from $B^+(r) \subseteq T$ we get $B^+(r) \cap L = \emptyset$, and from $L \cap H(r) \neq \emptyset$ we can then conclude $r \in ER(L, kb_i)$.

We have to show that $S_i \models \varepsilon(L, r)$, thus

- $S_i \models \bigwedge B^+(r)$,

- $S_i \models \bigwedge \neg B^-(r)$, and

- $S_i \models \neg (H(r) \land L)$.

The first two items hold by $S_i \models B(r)$. The last item holds since $T \cap H(r) = \emptyset$, hence $(S_i \cap (A_i \setminus L)) \cap H(r) = S_i \cap (H(r) \setminus L) = \emptyset$. Since $r \in ER(L, kb_i)$, we set $r = r$ and obtain that (4) is true, hence also (3) holds.

- $r \in \{head(r') \mid r' \in app(br_i, S)\}$: there exists a rule $r' \in app(br_i, S)$ of form (1) such that $r = head(r')$ and for all $(c_i : p_i)$ in $r'$, $1 \leq i \leq j$, we have $p_i \not\in S_i$, and for all $(c : p)$ in $r'$, $j + 1 \leq k \leq m$, we have $p_k \not\in S_{i_k}$. From $L \cap H(r) \neq \emptyset$ we have that $L \cap H(head(r')) \neq \emptyset$ and thus $\ell(r') \in SR(L, \ell(br_i))$. We have to show for each $S_i$ that $S_i \models \varepsilon(L, r)$, then

- $S_i \models \bigwedge B^+(t(r))$,

- $S_i \models \bigwedge \neg B^-(t(r))$, and

- $S_i \models \neg (H(r') \land L)$.

Since the sets $\Sigma_i$ are pairwise disjoint, the first two hold as $S_i \models B(r)$ and $S$ is then a model for the body of $\ell(r')$. The last one holds since $T \cap H(head(r')) = \emptyset$, hence $(S_i \cap (A_i \setminus L)) \cap H(head(r')) = S_i \cap (H(head(r')) \setminus L) = \emptyset$. Since $\ell(r') \in SR(L, \ell(br_i))$, we set $\ell(r') = r$ and obtain that (4) is true, hence also (3) holds.

We have shown that (i)–(iii) holds, and as a result, we get that $S$ is a model of $\pi(M)$.

$(\Leftarrow)$ Let $S$ be a model of $\pi(M)$. We can create a belief state $S = (S_1, \ldots, S_n)$, where each $S_i = S_i \upharpoonright \downarrow V_i$, and show that $S$ is an equilibrium of $M$ (note that the sets $S_i$ are pairwise disjoint as the sets $\Sigma_i$ are pairwise disjoint). We have to show that each $S_i \in ACC_i(kb_i \cup H_i)$, where $H_i = \{head(r) \mid r \in app(br_i, S)\}$, i.e., each $S_i$ is an answer set of $kb_i \cup H_i$.

We show the following:

(i) $S_i$ is a model of $(kb_i \cup H_i)^{S_i} = kb_i^{S_i} \cup H_i$ and

(ii) $S_i$ is minimal.

Part (i): By $S \models \kappa(kb_i)$ we immediately get that $S_i \models \kappa(kb_i)$ (since the sets $\Sigma_i$ are pairwise disjoint), hence $S_i \models kb_i$ and also $S_i \models kb_i^{S_i}$.

Moreover, we have that $S \models \kappa(\ell(br_i))$, that is, for all $r_i \in \ell(br_i)$ we have $S \models \kappa(\ell(r_i))$. Let $r_i \in br_i$ of form (1) such that $r_i = \ell(\ell(r))$. We have two cases:

- $r_i \in kb_i$: from $B^+(r) \subseteq T$ we get $B^+(r) \cap L = \emptyset$, and from $L \cap H(r) \neq \emptyset$ we can then conclude $r_i \in ER(L, kb_i)$.

We have to show that $S_i \models \varepsilon(L, r)$, thus

- $S_i \models \bigwedge B^+(r)$,

- $S_i \models \bigwedge \neg B^-(r)$, and

- $S_i \models \neg (H(r) \land L)$.

The first two items hold by $S_i \models B(r)$. The last item holds since $T \cap H(r) = \emptyset$, hence $(S_i \cap (A_i \setminus L)) \cap H(r) = S_i \cap (H(r) \setminus L) = \emptyset$. Since $r_i \in ER(L, kb_i)$, we set $r_i = r$ and obtain that (4) is true, hence also (3) holds.

We have shown that (i)–(iii) holds, and as a result, we get that $S$ is a model of $\pi(M)$. 


• $\neg\bigwedge B^+(r_1)$ or $\neg\bigwedge B^-(r_2)$: there exists a $p_i \in B^+(r_1)$ such that $p_i \notin S$ or a $p_k \in B^-(r_2)$ such that $p_k \in S$. Since $p_i, p_k$ are uniquely determined (follows from our disjoint language assumption), we have that there exists $(c_i : p_i)$ in $r_1$, $1 \leq i \leq j$, or $(c_k : p_k)$ in $r_2$, $j + 1 \leq k \leq m$. Thus, from our construction of $S$, we obtain $p_i \notin S_{c_i}$ or $p_k \in S_{c_k}$, and so we have that $r \notin app(br_i, S)$. We conclude that $head(r) \notin H_i$.

• $\models \bigwedge B^+(r_2)$ and $\models \bigwedge B^-(r_2)$ and $\models \bigvee H(r_1)$: for all $p_i \in B^+(r_1)$, $p_k \in B^-(r_1)$, we have $p_i \in S$ and $p_k \notin S$. Moreover, there exists a $p_h \in H(r_1)$ such that $p_h \in S$. As the sets $S_i$ are pairwise disjoint, all $p_i, p_k, p_h$ are uniquely determined, i.e., for all of $p_i, p_k, p_h$ we have $(c_i : p_i)$ from $r_1$ $(1 \leq i \leq j)$, $(c_k : p_k)$ from $r_2$ $(j + 1 \leq k \leq m)$, and a $p_h \in H(s)$, where $s = head(r)$. By our construction of $S$, we obtain that all $p_i \in S_{c_i}$, and all $p_k \notin S_{c_k}$, thus $r \notin app(br_i, S)$, and so is $head(r) \in H_i$. Since there exists a $p_h \in S$, we obtain that $p_h \in S_i$, and so $S_i \models head(r)$.

To sum up, $S_i \models H_i$ and $S_i \models kb_i^S$, therefore (i) holds.

Part (ii): Assume that there is a model $T_i \subseteq S_i$ of $kb_i^S \cup H_i$. One can show that this leads to a contradiction and the claim follows.

Proof Sketch of Theorem 6

The proof is similar to the one for Theorem 5. The essential difference is that loops are defined over the whole MCS.

References


