Distributed Nonmonotonic Multi-Context Systems

Minh Dao-Tran  Thomas Eiter
Michael Fink    Thomas Krennwallner

KBS Group, Institute of Information Systems, Vienna University of Technology

KR 2010 - May 13, 2010
Overview

Multi-Context Systems

Distributed Algorithm for Evaluating Nonmonotonic MCS

Loop Formulas for Multi-Context Systems

Experiments

Conclusions
Multi-Context Systems (MCS)

- MCSen introduced by [Giunchiglia and Serafini, 1994]:
  - represent inter-contextual information flow
  - express reasoning w.r.t. contextual information
  - allow decentralized, pointwise information exchange
  - monotonic, homogeneous logic

- Framework extended for integrating heterogeneous and nonmonotonic logics [Brewka and Eiter, 2007]
Syntax of Multi-Context Systems

- multi-context system
  - a collection $M = (C_1, \ldots, C_n)$ of contexts

- context $C_i = (L_i, kb_i, br_i)$
  - $L_i$: a logic
  - $kb_i$: a knowledge base of logic $L_i$
  - $br_i$: a set of bridge rules

Which belief sets are accepted by a knowledge base?
Syntax of Multi-Context Systems

- **multi-context system**
  - A collection $M = (C_1, \ldots, C_n)$ of contexts

- **context** $C_i = (L_i, kb_i, br_i)$
  - $L_i$: a logic
  - $kb_i$: a knowledge base of logic $L_i$
  - $br_i$: a set of bridge rules

- **logic** $L = (\text{KB}_L, \text{BS}_L, \text{ACC}_L)$
  - $\text{KB}_L$: set of well-formed knowledge bases
  - $\text{BS}_L$: is the set of possible belief sets
  - $\text{ACC}_L$: acceptability function $\text{KB}_L \mapsto 2^{\text{BS}_L}$
    Which belief sets are accepted by a knowledge base?
Semantics of Multi-Context Systems (2)

- **multi-context system**
  \[ M = (C_1, \ldots, C_n) \]

- **context**
  \[ C_i = (L_i, kb_i, br_i) \]

- **logic**
  \[ L = (KB_i, BS_i, ACC_i) \]
Syntax of Multi-Context Systems (bridge rules)

- **multi-context system**
  
  \[ M = (C_1, \ldots, C_n) \]

- **context**
  
  \[ C_i = (L_i, kb_i, br_i) \]

- **logic**
  
  \[ L_i = (KB_i, BS_i, ACC_i) \]

- **Bridge rule** \( r \in br_i \) of a context \( C_i \)
  
  \[ s \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), \]
  
  \[ not (c_{j+1} : p_{j+1}), \ldots, not (c_m : p_m) \]

  - \( (c_k : p_k) \) looks at belief \( p_k \) in context \( C_{c_k} \)
  - \( r \) is applicable \( \iff \) positive/negative beliefs are present/absent
  - we add the head \( s \) to \( kb_i \) if \( r \) is applicable
Semantics of Multi-Context Systems

- **multi-context system**
  \[ M = (C_1, \ldots, C_n) \]

- **context**
  \[ C_i = (L_i, kb_i, br_i) \]

- **logic**
  \[ L_i = (KB_i, BS_i, ACC_i) \]

- **knowledge base of a context** \( C_i \)
  \[ kb_i \in KB_i \]

- **set of bridge rules** \( br_i \) of a context \( C_i \) of form
  \[ s \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), not (c_{j+1} : p_{j+1}), \ldots, not (c_m : p_m) \]

- Contexts \( C_1, \ldots, C_n \) are knowledge bases with semantics in terms of accepted belief sets

- \( S = (S_1, \ldots, S_n) \) is a belief state of \( M \) with each \( S_i \in BS_i \)
Semantics of Multi-Context Systems

- **multi-context system**

\[ M = (C_1, \ldots, C_n) \]

- **context**

\[ C_i = (L_i, kb_i, br_i) \]

- **logic**

\[ L_i = (KB_i, BS_i, ACC_i) \]

- **Equilibrium semantics**

  - A belief state \( S = (S_1, \ldots, S_n) \) with \( S_i \in BS_i \)

  ... makes certain bridge rules applicable,

  ... add applicable bridge heads to \( kb_i \)

\[ \Rightarrow S \text{ is an equilibrium } \iff \]

each \( kb_i \) plus acceptable bridge heads from \( br_i \) accepts \( S_i \)

\[ S_i \in ACC_i(kb_i \cup \{head(r) \mid r \in app(br_i, S)\}) \]
Toward Distributed Equilibria building for MCS

Obstacles:

- abstraction of contexts
- information hiding and security aspects
- lack of system topology
- cycles between contexts

We need to capture:

- dependencies between contexts
- representation of partial knowledge
- combination/join of local results
Import Closure

Import neighborhood of $C_k$

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

Import neighborhood of $C_k$

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

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,

$$IC(k) = \{ k \} \cup \bigcup_{j \geq 0} IC^j(k),$$

where

$IC^0(k) = In(k)$, and
$IC^{j+1}(k) = \bigcup_{i \in IC^j(k)} In(i)$.
Let $M = (C_1, \ldots, C_n)$ be an MCS, and let $\epsilon \notin \bigcup_{i=1}^{n} BS_i$. 
Partial Belief States and Equilibria

Let $M = (C_1, \ldots, C_n)$ be an MCS, and let $\epsilon \notin \bigcup_{i=1}^{n} \text{BS}_i$

A partial belief state of $M$ is a sequence $S = (S_1, \ldots, S_n)$, where $S_i \in \text{BS}_i \cup \{\epsilon\}$, for $1 \leq i \leq n$
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 $S = (S_1, \ldots, S_n)$, where $S_i \in BS_i \cup \{\epsilon\}$, for $1 \leq i \leq n$

$S = (S_1, \ldots, S_n)$ is a partial equilibrium of $M$ w.r.t. a context $C_k$ iff for $1 \leq i \leq n$,

- if $i \in IC(k)$ then $S_i \in ACC_i(kb_i \cup \{head(r) \mid r \in app(br_i, S)\})$
- otherwise, $S_i = \epsilon$
Joining Partial Belief States

Join $S \Join T$ of belief sets $S$ and $T$: like join of tuples in a database.

\[ S = \begin{bmatrix} S_1 & \ldots & \epsilon & \ldots & \epsilon & \ldots & S_n \end{bmatrix} \]

\[ T = \begin{bmatrix} \epsilon & \ldots & \epsilon & \ldots & T_i & \ldots & T_n \end{bmatrix} \]

\[ S \Join T = \begin{bmatrix} S_1 & \ldots & \epsilon & \ldots & T_i & \ldots & S_n = T_n \end{bmatrix} \]

$S \Join T$ is undefined, if $\epsilon \neq S_j \neq T_j \neq \epsilon$ for some $j$.

\[ S \Join \mathcal{T} = \{ S \Join T \mid S \in S, T \in \mathcal{T} \} \]
Algorithm DMCS

Input: an MCS $M$ and a starting context $C_k$
Output: all partial equilibria of $M$ w.r.t. $C_k$
Algorithm DMCS

Input: an MCS $M$ and a starting context $C_k$
Output: all partial equilibria of $M$ w.r.t. $C_k$

Requirement: solver $\text{lsolve}(S)$ for each context $C_k$ is available which computes $\text{ACC}_k(kb_k \cup \text{app}_k(S))$
Algorithm DMCS

Input: an MCS $M$ and a starting context $C_k$
Output: all partial equilibria of $M$ w.r.t. $C_k$

Requirement: solver $\text{Isolve}(S)$ for each context $C_k$ is available which computes $\text{ACC}_k(kb_k \cup \text{app}_k(S))$

Input parameters for DMCS:
- $V$: set of “interesting” variables (to project the partial equilibria)
- $\text{hist}$: visited path
Algorithm DMCS

**Input:** an MCS $M$ and a starting context $C_k$

**Output:** all partial equilibria of $M$ w.r.t. $C_k$

Requirement: solver $\text{lsolve}(S)$ for each context $C_k$ is available which computes $\text{ACC}_k(kb_k \cup \text{app}_k(S))$

Input parameters for DMCS:

- $V$: set of “interesting” variables (to project the partial equilibria)
- $\text{hist}$: visited path

Strategy: DFS-traversal of $M$ starting with $C_k$, visiting all $C_i$ for $i \in IC(k)$

Instances of DMCS

- running at each context node,
- communicating with each other for exchanging sets of belief states
**Acyclic case**

Leaf context $C_k (br_k = \emptyset)$

\[
\text{lsolve}((\epsilon, \ldots, \epsilon)) = S
\]
Acyclic case

Leaf context $C_k$ ($br_k = \emptyset$)

$$lsolve((\epsilon, \ldots, \epsilon)) = S$$

Intermediate context $C_k$

$$((i : p), (j : q) \text{ appear in } br_k)$$

$$S_i \triangleleft S_j$$
Acyclic case

Leaf context $C_k$ ($br_k = \emptyset$)

Intermediate context $C_k$

$((i : p), (j : q) \text{ appear in } br_k)$

Isolve($((\epsilon, \ldots, \epsilon)) = S$)
**Acyclic case**

**Leaf context** $C_k (br_k = ∅)$

$\text{lsolve}((\epsilon, \ldots, \epsilon)) = S$

**Intermediate context** $C_k$

$((i : p), (j : q) \text{ appear in } br_k)$

$\text{lsolve}(S_i \bowtie S_j) = S_k$

$(V, \text{hist})$
Cycle breaking

\[ \text{hist} = \{ \ldots, k, \ldots \} \]

- \( V \)
- \( C_k \)
- \( C_i \)
- \( C_j \)
- \( C_\ell \)
Cycle breaking

$C_k$ detects a cycle in $hist$

$hist = \{ \ldots, k, \ldots \}$
Cycle breaking

$C_k$ detects a cycle in $\text{hist}$

- Guessing local belief sets
Cycle breaking

$C_k$ detects a cycle in $\text{hist}$

- Guessing local belief sets
- return them to invoking context
Cycle breaking

$C_k$ detects a cycle in $hist$

- Guessing local belief sets
- return them to invoking context

- on the way back, partial belief states w.r.t. bad guesses will be pruned by $\nabla$
Cycle breaking

$C_k$ detects a cycle in $hist$

- Guessing local belief sets
- return them to invoking context
- on the way back, partial belief states w.r.t. bad guesses will be pruned by $\nabla$
- eventually, $C_k$ will remove wrong guesses by calling $lsolve$ on each received partial belief state
Example

A run with $C_1.DMCS(V, \emptyset)$, where $V = \{a, b, c, f, g\}$. 

$kb_1 = \emptyset$
$br_1 = \{a \leftarrow (2:b), (3:c)\}$

$kb_2 = \emptyset$
$br_2 = \{b \leftarrow (4:g)\}$

$kb_3 = \{c \leftarrow d, d \leftarrow c\}$
$br_3 = \{c \lor e \leftarrow \text{not}(4:f)\}$

$kb_4 = \{f \lor g \leftarrow \}$
$br_4 = \emptyset$
Example

A run with $C_1 \cdot \text{DMCS}(V, \emptyset)$, where $V = \{a, b, c, f, g\}$.
Example

A run with $C_1 . \text{DMCS}(V, \emptyset)$, where $V = \{a, b, c, f, g\}$.

- $kb_1 = \emptyset$
- $br_1 = \{a \leftarrow (2 : b), (3 : c)\}$

- $kb_2 = \emptyset$
- $br_2 = \{b \leftarrow (4 : g)\}$

- $kb_3 = \{c \leftarrow d\}$
  - $d \leftarrow c$
- $br_3 = \{c \vee e \leftarrow \text{not}(4 : f)\}$

- $kb_4 = \{f \vee g \leftarrow \}$
- $br_4 = \emptyset$

- $C_2$
- $C_3$
  - $(\epsilon, \epsilon, \{\neg c, \neg d, e\}, \{\neg f, g\})$
  - $(\epsilon, \epsilon, \{c, d, \neg e\}, \{\neg f, g\})$
  - $(\epsilon, \epsilon, \{\neg c, \neg d, \neg e\}, \{f, \neg g\})$
  - $(\epsilon, \epsilon, \epsilon, \{\neg f, g\})$
  - $(\epsilon, \epsilon, \epsilon, \{f, \neg g\})$

- $C_4$
A run with $C_1\cdot DMCS(V, \emptyset)$, where $V = \{a, b, c, f, g\}$.

- $kb_1 = \emptyset$
- $br_1 = \{a \leftarrow (2 : b), (3 : c)\}$

- $kb_2 = \emptyset$
- $br_2 = \{b \leftarrow (4 : g)\}$

- $kb_3 = \{c \leftarrow d, d \leftarrow c\}$
- $br_3 = \{c \lor e \leftarrow \text{not} (4 : f)\}$

- $kb_4 = \{f \lor g \leftarrow \}$
- $br_4 = \emptyset$

- $(\epsilon, \{b\}, \epsilon, \{\neg f, g\})$
- $(\epsilon, \{\neg b\}, \epsilon, \{f, \neg g\})$
- $(\epsilon, \epsilon, \{\neg c, \neg d, e\}, \{\neg f, g\})$
- $(\epsilon, \epsilon, \{c, d, \neg e\}, \{\neg f, g\})$
- $(\epsilon, \epsilon, \{\neg c, \neg d, \neg e\}, \{f, \neg g\})$
- $(\epsilon, \epsilon, \epsilon, \{\neg f, g\})$
- $(\epsilon, \epsilon, \epsilon, \{f, \neg g\})$
Example

A run with $C_1 \cdot \text{DMCS}(V, \emptyset)$, where $V = \{a, b, c, f, g\}$.
Example

A run with $C_1.DMCS(V, \emptyset)$, where $V = \{a, b, c, f, g\}$.

$kb_1 = \emptyset$
$br_1 = \{ a \leftarrow (2 : b), (3 : c) \}$

$kb_2 = \emptyset$
$br_2 = \{ b \leftarrow (4 : g) \}$

$kb_3 = \{ c \leftarrow d \}
\quad d \leftarrow c$
$br_3 = \{ c \lor e \leftarrow not (4 : f) \}$

$kb_4 = \{ f \lor g \leftarrow \}$
$br_4 = \emptyset$

$kb_4 = \{ f \lor g \leftarrow \}$
$br_4 = \emptyset$
DMCS is using `lsolve()` to incorporate the bridge rules into the local knowledge base: this must be done for every intermediate result.

Some logics allow to combine $br_i$ and $kb_i$:
- contexts with answer set programs, or
- contexts with propositional formulas

Benefit: a single call to a SAT solver is sufficient to compute the local semantics of a context.

This is used to adapt DMCS and provide a prototype implementation.
Experiments

Diamond

<table>
<thead>
<tr>
<th></th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>#1944</td>
<td>#5240</td>
<td>#15088</td>
<td>#348</td>
<td>#12</td>
</tr>
<tr>
<td>P2</td>
<td>#1014</td>
<td>#56</td>
<td>#32</td>
<td>#14596</td>
<td>#5220</td>
</tr>
<tr>
<td># of equilibria</td>
<td>10</td>
<td>4</td>
<td>14</td>
<td>56</td>
<td>32</td>
</tr>
</tbody>
</table>

P1=(7,8,4,4)  P2=(10,12,6,6)  # equilibria

evaluation time / secs (logscale)
Experiments

P1=(7,8,4,4)  P2=(10,12,6,6)  # equilibria

evaluation time / secs (logscale)

Ring

R1  R2  R3  R4  R5
#176  #208  #80  #111  #2200  #120736  #98656  #19388  #2119  #19388
Experiments

Diamond

Parameter $P_i=(n,s,b,r)$  # equilibria

DMCSOPT

DMCS

#43.125
#1235
#35.25
#5772.8
#30.75
#80.5

#43.125  #1235  #35.25  #5772.8  #30.75  #80.5
Conclusions

- MCS is a general framework for integrating diverse formalisms
- First attempt for distributed MCS evaluation
- In certain settings, we can compile bridge rules away and use SAT solvers locally to generate partial equilibria (loop formulas for MCS)
- Initial experiments with a prototype implementation

Future work:
- Improve scalability
- Move away from "knowing-nothing" to "knowing-something"
- Approximation semantics
- Syntactic restrictions
- Specialized algorithms for some types of topologies
- How to deal with dynamic setting?
Conclusions

- MCS is a general framework for integrating diverse formalisms
- First attempt for distributed MCS evaluation
- In certain settings, we can compile bridge rules away and use SAT solvers locally to generate partial equilibria (loop formulas for MCS)
- Initial experiments with a prototype implementation

Future work:

- improve scalability
  - move away from “knowing-nothing” to “knowing-something”
  - approximation semantics
  - syntactic restrictions
  - specialized algorithms for some types of topologies
- how to deal with dynamic setting?
Related work

Frameworks/Platforms

- Framework for P2P inference systems [Hirayama and Yokoo, 2005]: consequence finding v.s. model building
- MWeb [Analyti et al., 2008]: scope and context for modular web rule bases on the Web

Distributed Reasoning

- Satisfiability checking for homogeneous, monotonic MCS [Roelofsen et al., 2004]: (co-inductive) fixpoint strategy, not truly distributed
- DisSAT [Hirayama and Yokoo, 2005]: finding single models (randomize)
- Distributed Description Logic [Serafini and Tamilin, 2005], [Serafini et al., 2005]
  - reasoning v.s. (distributed) model building
  - loose v.s. tight integration (signatures, meaning of symbols)


Katsutoshi Hirayama and Makoto Yokoo.  
The distributed breakout algorithms.  

Floris Roelofsen, Luciano Serafini, and Alessandro Cimatti.  

Luciano Serafini and Andrei Tamilin.  
Drago: Distributed reasoning architecture for the semantic web.  