FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Towards a Policy Language for Managing Inconsistency in Multi-Context Systems

Thomas Eiter Michael Fink Giovambattista Ianni Peter Schüller

KBS Group – Institut für Informationssysteme, Technische Universität Wien

Dipartimento di Matematica, Università della Calabria

Log-IC Workshop – May 16, 2011

supported by the Vienna Science and Technology Fund (WWTF) under grant ICT08-020

Outline

- Syntax and Semantics of MCSs
- Motivating Example for Managing Inconsistency
- IMPL Policy Language Overview
- Input for IMPL: Inconsistency Analysis
- Example IMPL Policies and their effects
- Syntax and Semantics of IMPL
- Future Work
  - Methodologies for applying IMPL in Practice
  - Realizing IMPL

Syntax and Semantics of MCSs

- A MCS $M = (C_1, \ldots, C_n)$ is a collection of contexts.
- A context $C = (L, kb, br)$ consists of
  - $L = (\mathbf{KB}_L, \mathbf{BS}_L, \mathbf{ACC}_L)$ a “logic”
    - $\mathbf{KB}_L$ well-formed knowledge bases
    - $\mathbf{BS}_L$ possible belief sets
    - $\mathbf{ACC}_L : \mathbf{KB}_L \rightarrow 2^{\mathbf{BS}_L}$ semantics (acceptability) function
  - kb_i context knowledge base
  - br_i context bridge rules
- A bridge rule

$$(onto : Marker(Id)) \leftarrow (lab : test(Id, blood, m1)).$$

1. “looks” at beliefs of source context(s), and
2. if the rule is applicable, it adds the head fact to its context KB.

Equilibrium Semantics

- Belief State $S = (S_1, \ldots, S_n)$ where $S_i \in \mathbf{BS}_{L_i}$ is a belief set at C_i.
- S makes certain bridge rules applicable
  $\Rightarrow$ we add their heads (H_i) to the respective knowledge bases (kb_i)
  $\Rightarrow$ equilibrium condition: $S_i \in \mathbf{ACC}(kb_i \cup H_i)$ for all C_i.
- Inconsistency is absence of an equilibrium.

Example MCS (inconsistent)

$$
\begin{aligned}
kb_{db} &= \{person(sue, 03/02/1985), allergy(sue, ab1)\},\\
kb_{lab} &= \{customer(sue, 02/03/1985), test(sue, xray, pneum),\\
&\quad test(sue, blood, m1), test(Id, X, Y) \rightarrow \exists D : customer(Id, D),\\
&\quad customer(Id, X) \land customer(Id, Y) \rightarrow X = Y\},\\
kb_{onto} &= \{Pneumonia \sqcap Marker \sqsubseteq AtypPneumonia\},\\
kb_{dss} &= \{give(Id, ab1) \lor give(Id, ab2) \leftarrow need(Id, ab).\\
&\quad give(Id, ab1) \leftarrow need(Id, ab1).\\
&\quad \neg give(Id, ab1) \leftarrow not\, allow(Id, ab1), need(Id, ab1).\}.
\end{aligned}
$$

$$
\begin{aligned}
r_1 &= (lab : customer(Id, Birthday)) &&\leftarrow (db : person(Id, Birthday)).\\
r_2 &= (onto : Pneumonia(Id)) &&\leftarrow (lab : test(Id, xray, pneum)).\\
r_3 &= (onto : Marker(Id)) &&\leftarrow (lab : test(Id, blood, m1)).\\
r_4 &= (dss : need(Id, ab)) &&\leftarrow (onto : Pneumonia(Id)).\\
r_5 &= (dss : need(Id, ab1)) &&\leftarrow (onto : AtypPneumonia(Id)).\\
r_6 &= (dss : allow(Id, ab1)) &&\leftarrow \textbf{not}\ (db : allergy(Id, ab1)).
\end{aligned}
$$

$(kb_{db},$

Example MCS (inconsistent)

$$
\begin{aligned}
kb_{db} &= \{person(sue, 03/02/1985), allergy(sue, ab1)\},\\
kb_{lab} &= \{customer(sue, 02/03/1985), test(sue, xray, pneum),\\
&\quad test(sue, blood, m1), test(Id, X, Y) \rightarrow \exists D : customer(Id, D),\\
&\quad customer(Id, X) \wedge customer(Id, Y) \rightarrow X = Y\},\\
kb_{onto} &= \{Pneumonia \sqcap Marker \sqsubseteq AtypPneumonia\},\\
kb_{dss} &= \{give(Id, ab1) \vee give(Id, ab2) \leftarrow need(Id, ab).\\
&\quad give(Id, ab1) \leftarrow need(Id, ab1).\\
&\quad \neg give(Id, ab1) \leftarrow not\, allow(Id, ab1), need(Id, ab1).\}.
\end{aligned}
$$

$$
\begin{aligned}
r_1 &= (lab : customer(Id, Birthday)) &&\leftarrow (db : person(Id, Birthday)).\\
r_2 &= (onto : Pneumonia(Id)) &&\leftarrow (lab : test(Id, xray, pneum)).\\
r_3 &= (onto : Marker(Id)) &&\leftarrow (lab : test(Id, blood, m1)).\\
r_4 &= (dss : need(Id, ab)) &&\leftarrow (onto : Pneumonia(Id)).\\
r_5 &= (dss : need(Id, ab1)) &&\leftarrow (onto : AtypPneumonia(Id)).\\
r_6 &= (dss : allow(Id, ab1)) &&\leftarrow \mathbf{not}\ (db : allergy(Id, ab1)).
\end{aligned}
$$

$(kb_{db}, \{customer(sue, 02/03/1985), customer(sue, 03/02/1985)\ \lightning,$
$test(sue, xray, pneum), test(sue, blood, m1)\},$

Example MCS (inconsistent)

$$
\begin{aligned}
kb_{db} &= \{person(sue, 03/02/1985), allergy(sue, ab1)\},\\
kb_{lab} &= \{customer(sue, 02/03/1985), test(sue, xray, pneum),\\
&\quad test(sue, blood, m1), test(Id, X, Y) \rightarrow \exists D : customer(Id, D),\\
&\quad customer(Id, X) \wedge customer(Id, Y) \rightarrow X = Y\},\\
kb_{onto} &= \{Pneumonia \sqcap Marker \sqsubseteq AtypPneumonia\},\\
kb_{dss} &= \{give(Id, ab1) \vee give(Id, ab2) \leftarrow need(Id, ab).\\
&\quad give(Id, ab1) \leftarrow need(Id, ab1).\\
&\quad \neg give(Id, ab1) \leftarrow not\, allow(Id, ab1), need(Id, ab1).\}.
\end{aligned}
$$

$$
\begin{aligned}
r_1 &= (lab : customer(Id, Birthday)) &&\leftarrow (db : person(Id, Birthday)).\\
r_2 &= (onto : Pneumonia(Id)) &&\leftarrow (lab : test(Id, xray, pneum)).\\
r_3 &= (onto : Marker(Id)) &&\leftarrow (lab : test(Id, blood, m1)).\\
r_4 &= (dss : need(Id, ab)) &&\leftarrow (onto : Pneumonia(Id)).\\
r_5 &= (dss : need(Id, ab1)) &&\leftarrow (onto : AtypPneumonia(Id)).\\
r_6 &= (dss : allow(Id, ab1)) &&\leftarrow \textbf{not}\ (db : allergy(Id, ab1)).
\end{aligned}
$$

$$(kb_{db}, \{customer(sue, 02/03/1985), test(sue, xray, pneum), test(sue, blood, m1)\},$$

Example MCS (inconsistent)

$$
\begin{aligned}
kb_{db} &= \{person(sue, 03/02/1985), allergy(sue, ab1)\},\\
kb_{lab} &= \{customer(sue, 02/03/1985), test(sue, xray, pneum),\\
&\quad test(sue, blood, m1), test(Id, X, Y) \rightarrow \exists D : customer(Id, D),\\
&\quad customer(Id, X) \land customer(Id, Y) \rightarrow X = Y\},\\
kb_{onto} &= \{Pneumonia \sqcap Marker \sqsubseteq AtypPneumonia\},\\
kb_{dss} &= \{give(Id, ab1) \lor give(Id, ab2) \leftarrow need(Id, ab).\\
&\quad give(Id, ab1) \leftarrow need(Id, ab1).\\
&\quad \neg give(Id, ab1) \leftarrow not\, allow(Id, ab1), need(Id, ab1).\}.
\end{aligned}
$$

$$
\begin{aligned}
r_1 &= (lab : customer(Id, Birthday)) &&\leftarrow (db : person(Id, Birthday)).\\
r_2 &= (onto : Pneumonia(Id)) &&\leftarrow (lab : test(Id, xray, pneum)).\\
r_3 &= (onto : Marker(Id)) &&\leftarrow (lab : test(Id, blood, m1)).\\
r_4 &= (dss : need(Id, ab)) &&\leftarrow (onto : Pneumonia(Id)).\\
r_5 &= (dss : need(Id, ab1)) &&\leftarrow (onto : AtypPneumonia(Id)).\\
r_6 &= (dss : allow(Id, ab1)) &&\leftarrow \mathbf{not}\ (db : allergy(Id, ab1)).
\end{aligned}
$$

$$
\begin{aligned}
&(kb_{db}, \{customer(sue, 02/03/1985), test(sue, xray, pneum), test(sue, blood, m1)\},\\
&\{Pneumonia(sue), Marker(sue), AtypPneumonia(sue)\},
\end{aligned}
$$

Example MCS (inconsistent)

$$kb_{db} = \{person(sue, 03/02/1985), allergy(sue, ab1)\},$$

$$kb_{lab} = \{customer(sue, 02/03/1985), test(sue, xray, pneum),$$
$$test(sue, blood, m1), test(Id, X, Y) \rightarrow \exists D : customer(Id, D),$$
$$customer(Id, X) \wedge customer(Id, Y) \rightarrow X = Y\},$$

$$kb_{onto} = \{Pneumonia \sqcap Marker \sqsubseteq AtypPneumonia\},$$

$$kb_{dss} = \{give(Id, ab1) \vee give(Id, ab2) \leftarrow need(Id, ab).$$
$$give(Id, ab1) \leftarrow need(Id, ab1).$$
$$\neg give(Id, ab1) \leftarrow not\, allow(Id, ab1), need(Id, ab1).\}.$$

$$r_1 = (lab : customer(Id, Birthday)) \leftarrow (db : person(Id, Birthday)).$$
$$r_2 = (onto : Pneumonia(Id)) \leftarrow (lab : test(Id, xray, pneum)).$$
$$r_3 = (onto : Marker(Id)) \leftarrow (lab : test(Id, blood, m1)).$$
$$r_4 = (dss : need(Id, ab)) \leftarrow (onto : Pneumonia(Id)).$$
$$r_5 = (dss : need(Id, ab1)) \leftarrow (onto : AtypPneumonia(Id)).$$
$$r_6 = (dss : allow(Id, ab1)) \leftarrow \textbf{not}\ (db : allergy(Id, ab1)).$$

$$(kb_{db}, \{customer(sue, 02/03/1985), test(sue, xray, pneum), test(sue, blood, m1)\},$$
$$\{Pneumonia(sue), Marker(sue), AtypPneumonia(sue)\},$$
$$\{need(sue, ab), need(sue, ab1), give(sue, ab1), \neg give(sue, ab1)\ \lightning\}$$

How to deal with Inconsistency?

- ~~Automatic repair?~~ (dangerous)
- ~~Manual repair?~~ (inefficient)

How to deal with Inconsistency?

- ~~Automatic repair?~~ (dangerous)
- ~~Manual repair?~~ (inefficient)

⇒ We propose the Inconsistency Management Policy Language "IMPL"

Overview of IMPL:

- inspired by ASP
- special input facts:
  - description of system as input facts
  - analysis of inconsistency as input facts
- special action atoms (derived in rules):
  - system modifications
  - user interaction for manual system modification

How to deal with Inconsistency?

- ~~Automatic repair?~~ (dangerous)
- ~~Manual repair?~~ (inefficient)

⇒ We propose the Inconsistency Management Policy Language “IMPL”

Overview of IMPL:

- inspired by ASP
- special input facts:
  - description of system as input facts
  - analysis of inconsistency as input facts
- special action atoms (derived in rules):
  - system modifications
  - user interaction for manual system modification
- for managing inconsistency with a policy
- semiautomatic system modifications

Inconsistency Analysis

Explaining inconsistency in MCSs:

- Diagnosis $(D_1, D_2) \Rightarrow$ consistency restored by
  removing bridge rules D_1 and adding bridge rules D_2
- Explanation $(E_1, E_2) \Rightarrow$ inconsistency caused by
  applicability of bridge rules E_1 and inapplicability of bridge rules E_2

(formal Definition in [Eiter et al. 2010] — KR2010).

Explaining Inconsistency

$$
\begin{aligned}
kb_{db} &= \{person(sue, 03/02/1985), allergy(sue, ab1)\},\\
kb_{lab} &= \{customer(sue, 02/03/1985), test(sue, xray, pneum),\\
&\quad test(sue, blood, m1), test(Id, X, Y) \rightarrow \exists D : customer(Id, D),\\
&\quad customer(Id, X) \wedge customer(Id, Y) \rightarrow X = Y\},\\
kb_{onto} &= \{Pneumonia \sqcap Marker \sqsubseteq AtypPneumonia\},\\
kb_{dss} &= \{give(Id, ab1) \vee give(Id, ab2) \leftarrow need(Id, ab).\\
&\quad give(Id, ab1) \leftarrow need(Id, ab1).\\
&\quad \neg give(Id, ab1) \leftarrow not\ allow(Id, ab1), need(Id, ab1).\}.
\end{aligned}
$$

$$
\begin{aligned}
r_1 &= (lab : customer(Id, Birthday)) \leftarrow (db : person(Id, Birthday)).\\
r_2 &= (onto : Pneumonia(Id)) \leftarrow (lab : test(Id, xray, pneum)).\\
r_3 &= (onto : Marker(Id)) \leftarrow (lab : test(Id, blood, m1)).\\
r_4 &= (dss : need(Id, ab)) \leftarrow (onto : Pneumonia(Id)).\\
r_5 &= (dss : need(Id, ab1)) \leftarrow (onto : AtypPneumonia(Id)).\\
r_6 &= (dss : allow(Id, ab1)) \leftarrow \mathbf{not}\ (db : allergy(Id, ab1).
\end{aligned}
$$

Explanations: $(\{r'_1\}, \emptyset)$, $(\{r'_2, r'_3, r'_5\}, \{r'_6\})$.

Explaining Inconsistency

$$
\begin{aligned}
kb_{db} &= \{person(sue, 03/02/1985), allergy(sue, ab1)\},\\
kb_{lab} &= \{customer(sue, 02/03/1985), test(sue, xray, pneum),\\
&\quad test(sue, blood, m1), test(Id, X, Y) \rightarrow \exists D : customer(Id, D),\\
&\quad customer(Id, X) \wedge customer(Id, Y) \rightarrow X = Y\},\\
kb_{onto} &= \{Pneumonia \sqcap Marker \sqsubseteq AtypPneumonia\},\\
kb_{dss} &= \{give(Id, ab1) \vee give(Id, ab2) \leftarrow need(Id, ab).\\
&\quad give(Id, ab1) \leftarrow need(Id, ab1).\\
&\quad \neg give(Id, ab1) \leftarrow not\, allow(Id, ab1), need(Id, ab1).\}.
\end{aligned}
$$

$$
\begin{aligned}
r_1 &= (lab : customer(Id, Birthday)) \leftarrow (db : person(Id, Birthday)).\\
r_2 &= (onto : Pneumonia(Id)) \leftarrow (lab : test(Id, xray, pneum)).\\
r_3 &= (onto : Marker(Id)) \leftarrow (lab : test(Id, blood, m1)).\\
r_4 &= (dss : need(Id, ab)) \leftarrow (onto : Pneumonia(Id)).\\
r_5 &= (dss : need(Id, ab1)) \leftarrow (onto : AtypPneumonia(Id)).\\
r_6 &= (dss : allow(Id, ab1)) \leftarrow \textbf{not}\ (db : allergy(Id, ab1)).
\end{aligned}
$$

Explanations: $(\{r'_1\}, \emptyset)$, $(\{r'_2, r'_3, r'_5\}, \{r'_6\})$.

Diagnoses: $(\{r'_1, r'_2\}, \emptyset)$, $(\{r'_1, r'_3\}, \emptyset)$, $(\{r'_1, r'_5\}, \emptyset)$, $(\{r'_1\}, \{r'_6\})$.

Example IMPL Policy (1)

% domain predicate for explanations
$expl(E) \leftarrow explNeed(E,R).$
$expl(E) \leftarrow explForbid(E,R).$
% find out whether one explanation only concerns bridge rules at C_{lab}
$incNotAtLab(E) \leftarrow explNeed(E,R), ruleHead(R,C,F), C \neq c_{lab}.$
$incNotAtLab(E) \leftarrow explForbid(E,R), ruleHead(R,C,F), C \neq c_{lab}.$
$incAtLab \leftarrow expl(E), not\ incNotAtLab(E).$
% guess a unique diagnosis to apply
$in(D) \leftarrow not\ out(D), diag(D), incAtLab.$
$out(D) \leftarrow not\ in(D), diag(D), incAtLab.$
$useOne \leftarrow in(D).$
$\bot \leftarrow in(A), in(B), A \neq B.$
$\bot \leftarrow not\ useOne, incAtLab.$
% apply diagnosis projected to C_{lab} if one was selected
$@applyModAtContext(D, c_{lab}) \leftarrow useDiag(D).$

Effect: $\Rightarrow$ input to C_{lab} is ignored automatically
$\Rightarrow$ other inconsistencies are not repaired

Example IMPL Policy (2)

Additionally create an inconsistency alert, if the automatic repair was done:

% inconsistency alert
$ruleHead(r_{alert}, c_{lab}, alert).$
$@addRule(r_{alert}) \leftarrow incAtLab.$

Effect: $\Rightarrow C_{lab}$ accepts an additional belief $alert$ in its belief set

Example IMPL Policy (2)

Additionally create an inconsistency alert, if the automatic repair was done:

% inconsistency alert
$ruleHead(r_{alert}, c_{lab}, alert).$
$@addRule(r_{alert}) \leftarrow incAtLab.$

Effect: $\Rightarrow$ C_{lab} accepts an additional belief *alert* in its belief set

Alternate Policy (fully manual):

% let the user choose from all diagnoses if there is a diagnosis
$member(md, X) \leftarrow diag(X).$
$@guiSelectMod(md).$

Effect: $\Rightarrow$ for any inconsistency, the user is asked to select a diagnosis

IMPL **Syntax (1)**

IMPL *policy* is a set of rules of the form

$$h \leftarrow a_1, \ldots, a_j, not\ a_{j+1}, \ldots, not\ a_k. \quad (1)$$

where h is an ordinary or an action atom, and a_i are literals.

Reserved predicates for representing

- the MCS: $ruleHead(BridgeRule, Context, Formula)$, $ruleBody^{+/-}(BridgeRule, Context, Belief)$.
- MCS modifications: $modAdd(Modification, BridgeRule)$, $modDel(Modification, BridgeRule)$.
- Inconsistency Analysis: $diag(Modification)$, $explNeed(Modification, BridgeRule)$, $explForbid(Modification, BridgeRule)$.
- sets of modifications: $member(SetName, Modification)$.

$\Rightarrow$ a policy can reason on the system structure

IMPL **Syntax (2)**

Limited Value Invention (modifying bridge rules requires new constants):

- $\#id(NewConstant, ExistingConstant, Integer)$

Actions for dealing with Inconsistency:

- Single Bridge Rules: $@delRule(Rule)$, $@addRule(Rule)$, $@addRuleCondition^{+/-}(Rule, Context, Belief)$, $@delRuleCondition^{+/-}(Rule, Context, Belief)$, $@makeRuleUnconditional(Rule)$.
- Diagnoses/Modifications: $@applyMod(Mod)$, $@applyModAtContext(Mod, Context)$.
- User Interaction: $@guiSelectMod(ModSet)$, $@guiEditMod(Mod)$ $@guiSelectModAtContext(ModSet, Context)$, $@guiEditModAtContext(ModSet, Context)$,

Core Fragment: $@delRule$, $@addRule$, $@guiSelectMod$, and $@guiEditMod$.

Three-step-semantics:

1. Action Determination: evaluate policy program (ASP semantics)
   ⇒ which actions are in an answer set?
2. Effect Determination: evaluate actions wrt. policy answer set
   ⇒ nondeterministic functions
   ⇒ input: policy answer set
   ⇒ output: set of added bridge rules
   ⇒ output: set of removed bridge rules
3. Effect Materialization:
   ⇒ overall set of added bridge rules
   ⇒ overall set of removed bridge rules
   ⇒ modify MCS

Methodologies, Realization

Usage Methodologies:

- Reason, apply modification, stop.
- Reason, apply modification,
  check consistency, restart if still inconsistent.

Future Work: formal properties/guarantees arising from methodologies

Methodologies, Realization

Usage Methodologies:

- Reason, apply modification, stop.
- Reason, apply modification,
  check consistency, restart if still inconsistent.

Future Work: formal properties/guarantees arising from methodologies

Realization:

- possible in acthex [Basol et al. 2010] – ICLP2010
- realization as rewriting:
  IMPL policy $\longrightarrow$ acthex program

Future Work: realize IMPL on top of acthex

Methodologies, Realization

Usage Methodologies:

- Reason, apply modification, stop.
- Reason, apply modification,
  check consistency, restart if still inconsistent.

Future Work: formal properties/guarantees arising from methodologies

Realization:

- possible in acthex [Basol et al. 2010] – ICLP2010
- realization as rewriting:
  IMPL policy $\longrightarrow$ acthex program

Future Work: realize IMPL on top of acthex

Future Work: applying IMPL in a real application scenario

References

- Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. Next Generat. Comput. 9(3–4), 365–386 (1991)
- Thomas Eiter, Michael Fink, Peter Schüller, and Antonius Weinzierl. Finding explanations of inconsistency in nonmonotonic multi-context systems. In *KR*, 2010.
- Basol, S., Erdem, O., Fink, M., Ianni, G.: HEX programs with action atoms. In: ICLP. pp. 24–33 (2010)

Extended Fragment and $\#id$

Instead of

$$@makeRuleUnconditional(R) \leftarrow foo(R).$$

We can write

% associate new constant with R to get identifier for rule derived from R
$aux(Rid, R) \leftarrow foo(R), \#id(Rid, R, 1).$
% copy existing rule heads (don't copy body literals)
$ruleHead(Rid, C, S) \leftarrow ruleHead(R, C, S), aux(Rid, R).$
% trigger actions
$@delRule(R) \leftarrow aux(Rid, R).$
$@addRule(Rid) \leftarrow aux(Rid, R).$