Combining Nonmonotonic Knowledge Bases with External Sources

Thomas Eiter

Institute of Information Systems, TU Vienna
eiter@kr.tuwien.ac.at

with Gerhard Brewka (U Leipzig); Giovambattista Ianni (U Calabria); Minh Dao-Tran, Michael Fink, Thomas Krennwallner (TU Vienna)

FroCoS, Trento, September 16-18, 2009

- Austrian Science Fund (FWF) grant P20840, P20841
- Vienna Science and Technology Fund (WWTF) grant ICT08-020.
- ICT Ontorule (FP7 231875)

Recent Developments

- Traditional KR: monolithic, closed reasoning systems
- The World Wide Web
- Wealth of data / knowledge sources
- Distributed, open systems

Urgent Need

- access to external sources
- cope with heterogenity
- incompleteness
- recurrent data access
- dynamics

Issues:

- Semantics
- Algorithms, Implementations

Aim of this Talk

- Present some formalisms that combine possibly nonmonotonic knowledge bases with external sources
- Nonmonotonic formalisms have long tradition in Knowledge Representation and Reasoning
- **Focus:** recent work of KBS Group @ TU Vienna and colleagues
- **Observations:**
  - Principled issues
  - Research problems (theory, implementation)
  - On target for combining systems
  - . . . to new frontiers

Historic Background

Work @ KBS/DBAI groups of TU Vienna in the 1990's:

- Nonmonotonic Reasoning, Logic Progamming (DLV)

Historic Background

Work @ KBS/DBAI groups of TU Vienna in the 1990's:

- Nonmonotonic Reasoning, Logic Progamming (DLV)
- Logic programs with *generalized quantifiers* [E_ *et al.*, 1997]

  Incorporate Lindström type quantifiers ("majority", ...) into LPs

  $$friendly\_guy(X) \leftarrow most[likes](X)$$

  Similar to use of GQs in databases / finite model theory (avg, min, ...)

Historic Background

Work @ KBS/DBAI groups of TU Vienna in the 1990's:

- Nonmonotonic Reasoning, Logic Progamming (DLV)
- Logic programs with *generalized quantifiers* [E_ *et al.*, 1997]

  Incorporate Lindström type quantifiers ("majority", ...) into LPs

  $$friendly\_guy(X) \leftarrow most[likes](X)$$

  Similar to use of GQs in databases / finite model theory (avg, min, ...)

- IMPACT agent platform [Subrahmanian *et al.*, 2000]

  $$\mathbf{Do}\, notify(P, M) \leftarrow \mathbf{P}\, inform(P, M),\ in(P, db{:}getClients()),\ \mathrm{not}\ urgent(M)$$

- **But:** embryonic; limitations, drawbacks

Historic Background

Work @ KBS/DBAI groups of TU Vienna in the 1990's:

- Nonmonotonic Reasoning, Logic Progamming (DLV)
- Logic programs with *generalized quantifiers* [E_ *et al.*, 1997]

  Incorporate Lindström type quantifiers ("majority", ...) into LPs

  $$friendly\_guy(X) \leftarrow most[likes](X)$$

  Similar to use of GQs in databases / finite model theory (avg, min, ...)
- IMPACT agent platform [Subrahmanian *et al.*, 2000]

  $$\mathbf{Do}\, notify(P,M) \leftarrow \mathbf{P}\, inform(P,M),\ in(P, db{:}getClients()),\ \text{not}\ urgent(M)$$
- **But:** embryonic; limitations, drawbacks

Around 2000: Emergence of Answer Set Programming

Answer Set Programming (ASP)

- Answer Set Programming (ASP) is a recent declarative problem solving approach.
- The term was coined by Lifschitz [1999,2002].
- Proposed by other people at about the same time, e.g. [Marek and Truszczyński, 1999], [Niemelä, 1999].
- It has roots in KR, logic programming, and nonmonotonic reasoning.
- At an abstract level, relates to SAT solving and CSP.
- Early book: [Baral, 2003]
- To date, ASP languages and systems are a major tool for building non-monotonic knowledge bases.

Roadmap

Logic Programs with Negation

- "War of Semantics" in Logic Programming (1980/90ies)

  Meaning of programs with negation "not" like the following:

$$
\begin{aligned}
&& man(joe). \\
single(X) &\leftarrow man(X), \mathrm{not}\, husband(X). \\
husband(X) &\leftarrow man(X), \mathrm{not}\, single(X).
\end{aligned}
$$

  Intuitive models: $M_1 = \{man(joe), single(joe)\}$, $M_2 = \{man(joe), husband(joe)\}$.

  Prolog: ???

Logic Programs with Negation

- "War of Semantics" in Logic Programming (1980/90ies)

  Meaning of programs with negation "not" like the following:

$$
\begin{aligned}
man(joe)&.\\
single(X) \leftarrow man(X), \mathrm{not}\, husband(X)&.\\
husband(X) \leftarrow man(X), \mathrm{not}\, single(X)&.
\end{aligned}
$$

  Intuitive models: $M_1 = \{man(joe), single(joe)\}$, $M_2 = \{man(joe), husband(joe)\}$.
  Prolog: ???

- Great Schism: Single model vs. multiple model semantics

Logic Programs with Negation

- "War of Semantics" in Logic Programming (1980/90ies)

  Meaning of programs with negation "not" like the following:

$$\begin{aligned} && man(joe). \\ single(X) &\leftarrow man(X), \text{not}\, husband(X). \\ husband(X) &\leftarrow man(X), \text{not}\, single(X). \end{aligned}$$

  Intuitive models: $M_1 = \{man(joe), single(joe)\}$, $M_2 = \{man(joe), husband(joe)\}$.

  Prolog: ???

- Great Schism: Single model vs. multiple model semantics
  - *Well-Founded Semantics* [Van Gelder *et al.*, 1991]: partial model, where $man(joe)$ is true, $single(joe)$, $husband(joe)$ are unknown

Logic Programs with Negation

- "War of Semantics" in Logic Programming (1980/90ies)

  Meaning of programs with negation "not" like the following:

$$
\begin{aligned}
& man(joe). \\
single(X) &\leftarrow man(X), \text{not}\, husband(X). \\
husband(X) &\leftarrow man(X), \text{not}\, single(X).
\end{aligned}
$$

  Intuitive models: $M_1 = \{man(joe), single(joe)\}$, $M_2 = \{man(joe), husband(joe)\}$.

  Prolog: ???

- Great Schism: Single model vs. multiple model semantics
  - *Well-Founded Semantics* [Van Gelder *et al.*, 1991]: partial model, where $man(joe)$ is true, $single(joe)$, $husband(joe)$ are unknown
  - *Answer Set (Stable Model) Semantics* by Gelfond and Lifschitz [1988,1991]: Alternative models M_1, M_2.

Logic Programs with Negation

- "War of Semantics" in Logic Programming (1980/90ies)

  Meaning of programs with negation "not" like the following:

$$\begin{aligned} & man(joe). \\ single(X) &\leftarrow man(X), \text{not}\, husband(X). \\ husband(X) &\leftarrow man(X), \text{not}\, single(X). \end{aligned}$$

  Intuitive models: $M_1 = \{man(joe), single(joe)\}$, $M_2 = \{man(joe), husband(joe)\}$.

  Prolog: ???

- Great Schism: Single model vs. multiple model semantics
  - *Well-Founded Semantics* [Van Gelder *et al.*, 1991]: partial model, where $man(joe)$ is true, $single(joe)$, $husband(joe)$ are unknown
  - *Answer Set (Stable Model) Semantics* by Gelfond and Lifschitz [1988,1991]: Alternative models M_1, M_2.

- Shift in LP: compute *Answer Sets (=models)*, not proofs!

ASP Paradigm

General idea: answer sets provide solutions!

$$\xrightarrow[\text{Instance } I]{\text{Problem}} \boxed{\begin{array}{c}\text{Encoding:}\\ \text{Program } P\end{array}} \longrightarrow \boxed{\text{ASP Solver}} \xrightarrow[\text{Solution(s)}]{\text{Model(s)}}$$

1. *Encode* problem instance I as a (non-monotonic) logic program P, such that solutions of I are represented by models of P
2. *Compute* some model M of P, using an ASP solver
3. *Extract* a solution for I from M.

Variant: Compute multiple models (for multiple / all solutions)

Often: Decompose I into *problem specification* and *data*

Note: Related to SAT Solving/CSP, but ASP offers special features (variables, supports transitivity)

Answer Set Solvers

DLV	`http://www.dbai.tuwien.ac.at/proj/dlv/` *
Smodels	`http://www.tcs.hut.fi/Software/smodels/` **
GnT	`http://www.tcs.hut.fi/Software/gnt/`
Cmodels	`http://www.cs.utexas.edu/users/tag/cmodels/`
ASSAT	`http://assat.cs.ust.hk/`
NoMore(++)	`http://www.cs.uni-potsdam.de/~linke/nomore/`
Platypus	`http://www.cs.uni-potsdam.de/platypus/`
clasp	`http://www.cs.uni-potsdam.de/clasp/`
XASP	`http://xsb.sourceforge.net`, distributed with XSB v2.6
aspps	`http://www.cs.engr.uky.edu/ai/aspps/`
ccalc	`http://www.cs.utexas.edu/users/tag/cc/`

* + extensions (DLVHEX, DVL^{DB}, DLT, ...)   ** + Smodels_cc

- Several provide a number of extensions to the language described here.
- ASP Solver competition: see LPNMR conference (2009 edition this week!);
- Benchmark platform: `http://asparagus.cs.uni-potsdam.de/`
- **Note:** clasp wins the *crafted instances* categories a) SAT+UNSAT and b) UNSAT instances of the SAT Competition 2009.

Answer Set Programs

Disjunctive Logic Program

A *(disjunctive) logic program* P is a (finite) set of rules of the form

$$a_1 \vee \cdots \vee a_l \leftarrow \ b_1, \ldots, b_m, \text{not} \ \ c_1, \ldots, \text{not} \ \ c_n$$

where all a_i, b_j, c_k are literals of the form p or $\neg p$, where p is a first-order atom over a (classical) first-order vocabulary.

- Standard ASP has no function symbols
- "$\neg$" is called strong negation (also written as "–")
- In *normal programs*, the rule head is a single literal ($l = 1$)

(Extended) Herbrand Base

HB_P is the set of all ground (variable-free) literals p and $\neg p$ with predicates and ground terms constructible from P.

Answer Sets

- Answer Sets are based on *3-valued Herbrand Interpretations (=consistent sets of ground literals $M \subseteq HB_P$)*, with incomplete information
- For programs without "$\neg$," they are also called "stable models" and viewed 2-valued, with complete information about the world.

Satisfaction

An interpretation $M \subseteq HB_P$ satisfies

- a ground rule $a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$, if $\{b_1, \ldots, b_m\} \subseteq M$ and $M \cap \{c_1, \ldots, c_n\} = \emptyset$ implies $M \cap \{a_1, \ldots, a_k\} \neq \emptyset$.
- a ground program P, if M satisfies each $r \in P$.
- a rule r, if M satisfies each $r' \in grnd(r)$, where $grnd(r)$ is the set of of all ground instances of r.
- a program P, if M satisfies $grnd(P) = \bigcup_{r \in P} grnd(r)$.

- For not*-free ("positive") programs*, an intuitive semantics are *minimal models*:

Minimal Model

An interpretation $M \subseteq HB_P$ is minimal model of P, if (i) M satisfies P and (ii) no $N \subset M$ satisfies P.

- For not-*free ("positive") programs*, an intuitive semantics are *minimal models*:

Minimal Model

An interpretation $M \subseteq HB_P$ is minimal model of P, if (i) M satisfies P and (ii) no $N \subset M$ satisfies P.

- **Key idea for arbitrary programs:** elimination of not

Gelfond-Lifschitz (GL) reduct P^M

Given program P, remove from $grnd(P)$

1. every rule $a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$ where some c_i is in M, and
2. all literals $\text{not } c_j$ from the remaining rules.

Use M as an *assumption* on how negation finally evaluates.

Answer Set

M is an *answer set* of P iff M is a minimal model of P^M.

Answer Set

M is an *answer set* of P iff M is a minimal model of P^M.

- M satisfies all rules of P

Answer Set

M is an *answer set* of P iff M is a minimal model of P^M.

- M satisfies all rules of P
- Moreover, P can "reproduce" M with an *assumption* on how negation finally evaluates *(stability)*

Answer Set

M is an *answer set* of P iff M is a minimal model of P^M.

- M satisfies all rules of P
- Moreover, P can "reproduce" M with an *assumption* on how negation finally evaluates *(stability)*
- Note: for normal P, "a minimal" = "the least"

Answer Set

M is an *answer set* of P iff M is a minimal model of P^M.

- M satisfies all rules of P
- Moreover, P can "reproduce" M with an *assumption* on how negation finally evaluates *(stability)*
- Note: for normal P, "a minimal" = "the least"
- For positive P, $P^M = P$, so the answer sets coincide with the minimal models

Answer Set

M is an *answer set* of P iff M is a minimal model of P^M.

- M satisfies all rules of P
- Moreover, P can "reproduce" M with an *assumption* on how negation finally evaluates *(stability)*
- Note: for normal P, "a minimal" = "the least"
- For positive P, $P^M = P$, so the answer sets coincide with the minimal models
- Many equivalent definitions of answer sets / stable models exist [Lifschitz, 2008]

  E.g., Answer sets can be reconstructed in the logic of Here and There (*Equilibrium Logic* [Pearce, 2006])

Example

$$
\begin{aligned}
P = \{\ & person(joey);\\
& male(X) \vee female(X) \leftarrow person(X);\\
& bachelor(X) \leftarrow male(X), \mathrm{not}\, married(X)\ \}
\end{aligned}
$$

Example

$$
\begin{aligned}
grnd(P) = \{\ & person(joey);\\
& male(joey) \vee female(joey) \leftarrow person(joey);\\
& bachelor(joey) \leftarrow male(joey), \mathrm{not}\, married(joey)\ \}
\end{aligned}
$$

- Grounding of P

Example

$$
\begin{aligned}
grnd(P) = \{\ & person(joey);\\
& male(joey) \vee female(joey) \leftarrow person(joey);\\
& bachelor(joey) \leftarrow male(joey), \mathrm{not}\, married(joey)\ \}
\end{aligned}
$$

- $M_1 = \{person(joey), male(joey), bachelor(joey)\}$ is "stable"

Example

$$P^{M_1} = \{\, person(joey);$$
$$male(joey) \vee female(joey) \leftarrow person(joey);$$
$$bachelor(joey) \leftarrow male(joey), \cancel{\text{not}\, married(joey)} \,\}$$

- $M_1 = \{person(joey), male(joey), bachelor(joey)\}$ is "stable"

  M_1 is a minimal model of P^{M_1}

Example

$$\begin{aligned} grnd(P) = \{\ & person(joey); \\ & male(joey) \vee female(joey) \leftarrow person(joey); \\ & bachelor(joey) \leftarrow male(joey), \text{not}\, married(joey)\ \} \end{aligned}$$

- $M_1 = \{person(joey), male(joey), bachelor(joey)\}$ is “stable”
- $M_2 = \{person(joey), male(joey), married(joey)\}$ is not stable

Example

$$
\begin{aligned}
P^{M_2} = \{\, & person(joey); \\
& male(joey) \vee female(joey) \leftarrow person(joey); \\
& \cancel{bachelor(joey) \leftarrow male(joey), \text{not}\, married(joey)}\ \}
\end{aligned}
$$

- $M_1 = \{person(joey), male(joey), bachelor(joey)\}$ is “stable”

- $M_2 = \{person(joey), male(joey), married(joey)\}$ is not stable

  M_2 is not a minimal model of P^{M_2}

Example

$$
\begin{aligned}
P = \{\ & \mathit{person}(\mathit{joey});\\
& \mathit{male}(X) \vee \mathit{female}(X) \leftarrow \mathit{person}(X);\\
& \mathit{bachelor}(X) \leftarrow \mathit{male}(X), \mathrm{not}\, \mathit{married}(X)\ \}
\end{aligned}
$$

- $M_1 = \{\mathit{person}(\mathit{joey}), \mathit{male}(\mathit{joey}), \mathit{bachelor}(\mathit{joey})\}$ is “stable”
- $M_2 = \{\mathit{person}(\mathit{joey}), \mathit{male}(\mathit{joey}), \mathit{married}(\mathit{joey})\}$ is not stable
- Further answer set: $M_3 = \{\mathit{person}(\mathit{joey}), \mathit{female}(\mathit{joey})\}$

Constraints

- Consider the program

$$P = \{\ p \leftarrow \text{not}\ p.\ \}$$

- This program has NO answer sets.

Constraints

- Consider the program

$$P = \{\ p \leftarrow \text{not } p.\ \}$$

- This program has NO answer sets.
- Let P be a program and p be a new atom.
- Then, adding

$$p \leftarrow \text{not } p, a_1, \ldots, a_n \text{ not } b_1, \ldots, \text{not } b_m.$$

to P "kills" each answer set M of P containing all a_i and no b_j.

Constraints

- Consider the program

$$P = \{\, p \leftarrow \text{not } p. \,\}$$

- This program has NO answer sets.
- Let P be a program and p be a new atom.
- Then, adding

$$p \leftarrow \text{not } p, a_1, \ldots, a_n \text{ not } b_1, \ldots, \text{not } b_m.$$

to P “kills” each answer set M of P containing all a_i and no b_j.

Constraint

$$\leftarrow a_1, \ldots, a_n \text{ not } b_1, \ldots, \text{not } b_m.$$

ASP Applications

See http://www.kr.tuwien.ac.at/projects/WASP/report.html

- information integration
- constraint satisfaction, configuration
- planning, routing
- diagnosis
- security analysis
- Semantic Web
- computer-aided verification
- biology / biomedicine
- knowledge management
- ...

ASP Showcase: http://www.kr.tuwien.ac.at/projects/WASP/showcase.html

ASP with External Sources

Issues

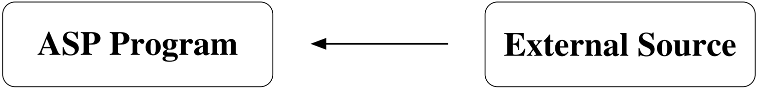
- Interface / integrate with external sources.
- Despite possible heterogenous semantics, keep ASP spirit for the semantics.

ASP with External Sources

Issues

- Interface / integrate with external sources.
- Despite possible heterogenous semantics, keep ASP spirit for the semantics.

Scenarios



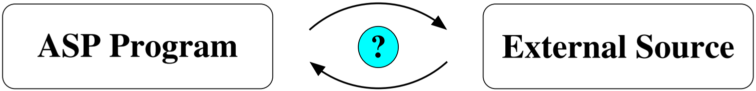
- Import of information: add facts

ASP with External Sources

Issues

- Interface / integrate with external sources.
- Despite possible heterogenous semantics, keep ASP spirit for the semantics.

Scenarios



- Import of information: add facts
- Bidirectional information flow:
  - For ASP, a nontrivial aspect in general
  - Specifically, in case of recursion (minimality, stability)

Formalisms and Systems

- A variety of formalisms and systems has been proposed, e.g.,
  - GQLPs [E_ *et al.*, 1997], MLPs [Dao Tran *et al.*, 2009], DLP Functions [Janhunen *et al.*, 2007]
  - DLVEX [Calimeri *et al.*, 2007], HEX programs [E_ *et al.*, 2005], DLV^{DB} [Terracina *et al.*, 2008]
  - Nonmonotonic Multi-Context Systems [Brewka and E_, 2007]
- Related: Macros [Baral *et al.*, 2006], Templates [Ianni *et al.*, 2003], MWeb [Analyti *et al.*, 2008] etc.
- The proposals are different, yet not unrelated. Superficially,
  - MLPs can be viewed as special setting for HEX programs
  - MCSs are a kind of generalization of HEX programs
- **But:** relation not by intent; underlying philosophy/assumptions vary
- Systematic view helps

Two Major Aspects

world view	

Collection $KB = KB_1, \ldots, KB_n$ of knowledge bases / sources KB_i

- **environment (world) view**

Two Major Aspects

world view	
local model	

Collection $KB = KB_1, \ldots, KB_n$ of knowledge bases / sources KB_i

- **environment (world) view**
  - individual: purely *local models* M_i for each KB_i; semantics of KB emerges implicitly

Two Major Aspects

world view	
local model	GQLPs, HEX

Collection $KB = KB_1, \ldots, KB_n$ of knowledge bases / sources KB_i

- **environment (world) view**
  - individual: purely *local models* M_i for each KB_i; semantics of KB emerges implicitly

Two Major Aspects

world view	
local model	GQLPs, HEX
globale state	

Collection $KB = KB_1, \ldots, KB_n$ of knowledge bases / sources KB_i

- **environment (world) view**
  - individual: purely *local models* M_i for each KB_i; semantics of KB emerges implicitly
  - societal: *global state* $S = (S_1, \ldots, S_n)$ of local models S_i of KB_i; S is *explicitly* accessible $\Rightarrow$ global state preference

Two Major Aspects

world view	
local model	GQLPs, HEX
globale state	MCS, MLPs

Collection $KB = KB_1, \ldots, KB_n$ of knowledge bases / sources KB_i

- **environment (world) view**
  - individual: purely *local models* M_i for each KB_i; semantics of KB emerges implicitly
  - societal: *global state* $S = (S_1, \ldots, S_n)$ of local models S_i of KB_i; S is *explicitly* accessible $\Rightarrow$ global state preference

Two Major Aspects

world view	
local model	GQLPs, HEX
globale state	MCS, MLPs

Collection $KB = KB_1, \ldots, KB_n$ of knowledge bases / sources KB_i

- **environment (world) view**
  - individual: purely *local models* M_i for each KB_i; semantics of KB emerges implicitly
  - societal: *global state* $S = (S_1, \ldots, S_n)$ of local models S_i of KB_i; S is *explicitly* accessible $\Rightarrow$ global state preference

Loosely speaking, *Nash equilibria* vs *Pareto-optimality*.

reduct world view	
local model globale state	

Collection $KB = KB_1, \ldots, KB_n$ of knowledge bases

- **program reduct**:

reduct / world view	GL-style
local model / globale state	

Collection $KB = KB_1, \ldots, KB_n$ of knowledge bases

- **program reduct**:
  - *GL-style reduct* P^I

reduct / world view	GL-style
local model	GQLPs
globale state	MCS

Collection $KB = KB_1, \ldots, KB_n$ of knowledge bases

- **program reduct**:
  - *GL-style reduct* P^I

reduct / world view	GL-style	FLP
local model	GQLPs	
globale state	MCS	

Collection $KB = KB_1, \ldots, KB_n$ of knowledge bases

- **program reduct**:
  - *GL-style reduct* P^I
  - *FLP reduct* fP^I [Faber *et al.*, 2004]

$$fP^I = \{Head \leftarrow Body \in grnd(P) \mid I \text{ satisfies } Body\}.$$

- for ordinary ASP programs, GL and FLP reduct are equivalent
- For ASP extensions, FLP retains minimality of models, but not GL.

reduct / world view	GL-style	FLP
local model	GQLPs	HEX
globale state	MCS	MLPs

Collection $KB = KB_1, \ldots, KB_n$ of knowledge bases

- **program reduct**:
  - *GL-style reduct* P^I
  - *FLP reduct* fP^I [Faber *et al.*, 2004]

$$fP^I = \{Head \leftarrow Body \in grnd(P) \mid I \text{ satisfies } Body\}.$$

- for ordinary ASP programs, GL and FLP reduct are equivalent
- For ASP extensions, FLP retains minimality of models, but not GL.

reduct / world view	GL-style	FLP
local model	GQLPs	HEX
globale state	MCS	MLPs

Collection $KB = KB_1, \ldots, KB_n$ of knowledge bases

- **program reduct**:
  - *GL-style reduct* P^I
  - *FLP reduct* fP^I [Faber *et al.*, 2004]

$$fP^I = \{Head \leftarrow Body \in grnd(P) \mid I \text{ satisfies } Body\}.$$

- for ordinary ASP programs, GL and FLP reduct are equivalent
- For ASP extensions, FLP retains minimality of models, but not GL.

Other formalisms fit (e.g., dl-programs (ASP+DL): GL-style/local model)

HEX Programs

- Designed to meet needs of heterogenous data access on the Web
- Generalizes earlier *description logic programs* which provide ASP programs with query access to an OWL logic ontology.
- Allow to access sources of whatever type (no restriction; abstract modeling)
- **Features:**
  - Higher-Order atoms: variables for predicate names (syntactic sugar)
  - External atoms: access to external sources (increases expressivity)
- **Type:** FLP reduct / local model

An Example

$$
\begin{aligned}
invites(john, X) \vee skip(X) \leftarrow & X \neq john, \\
& \&DL\_Query[my\_ontology, relativeOf](john, X). \\
someInvited \leftarrow & invites(john, X). \\
\leftarrow & \text{not } someInvited. \\
\leftarrow & \&degs[invites](Min, Max), Max > 2.
\end{aligned}
$$

Example

Input: Data about *John*'s relatives (from an ontology)

Output: Possible picks for persons John might want to invite, according to some constraints (some evaluated externally)

An Example

$$\begin{aligned}
invites(john, X) \lor skip(X) \leftarrow & X \neq john, \\
& \&DL_Query[my_ontology, relativeOf](john, X). \\
someInvited \leftarrow & invites(john, X). \\
\leftarrow & \text{not } someInvited. \\
\leftarrow & \&degs[invites](Min, Max), Max > 2.
\end{aligned}$$

Example

Input: Data about *John*'s relatives (from an ontology)

Output: Possible picks for persons John might want to invite, according to some constraints (some evaluated externally)

An Example

$$\begin{aligned}
\mathit{invites}(\mathit{john}, X) \vee \mathit{skip}(X) \leftarrow & X \neq \mathit{john}, \\
& \mathit{\&DL_Query}[\mathit{my_ontology}, \mathit{relativeOf}](\mathit{john}, X). \\
\mathit{someInvited} \leftarrow & \mathit{invites}(\mathit{john}, X). \\
\leftarrow & \text{not } \mathit{someInvited}. \\
\leftarrow & \mathit{\&degs}[\mathit{invites}](\mathit{Min}, \mathit{Max}), \mathit{Max} > 2.
\end{aligned}$$

Example

Input: Data about *John*'s relatives (from an ontology)

Output: Possible picks for persons John might want to invite, according to some constraints (some evaluated externally)

An Example

$$
\begin{aligned}
invites(john, X) \vee skip(X) \leftarrow & X \neq john, \\
& \&DL\_Query[my\_ontology, relativeOf](john, X). \\
someInvited \leftarrow & invites(john, X). \\
\leftarrow & \text{not } someInvited. \\
\leftarrow & \&degs[invites](Min, Max), Max > 2.
\end{aligned}
$$

Example

Input: Data about *John*'s relatives (from an ontology)

Output: Possible picks for persons John might want to invite, according to some constraints (some evaluated externally)

$$\&DL\_Query[my\_ontology, relativeOf](john, X) \tag{1}$$

$$\&degs[invites](Min, Max) \tag{2}$$

External Atom

In general, an *external atom* a is of the form

$$\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m) \ , \tag{3}$$

where $Y_1, \ldots, Y_n$ and $X_1, \ldots, X_m$ are two lists of terms (called *input* and *output* lists, respectively), and $\&g$ is an external predicate name.

- External atoms may occur only in rule bodies; disregard $\neg$.
- Each $\&g$ is associated with an evaluation function $f_{\&g}$

Example

$\&DL\_Query$ corresponds to $f_{\&DL\_Query}$.

- Informally,

$$\&DL\_Query[my\_ontology, relativeOf](john, c)$$

  is true if $relativeOf(john, c)$ is provable in $my\_ontology$.

- This is formally captured via $f_{\&DL\_Query}$:

  For a given interpretation I,

$$I \models \&DL\_Query[my\_ontology, relativeOf](john, c)$$

  iff

$$f_{\&DL\_Query}(I, my\_ontology, relativeOf, john, c) = 1$$

Semantics of HEX programs P

- Higher order atoms $T_0(T_1, \ldots, T_n)$ are grounded to $t_0(t_1, \ldots, t_n)$.
- Herbrand base HB_P: all ground (ordinary, external) atoms.

Interpretations

An *interpretation* is any subset $I \subseteq HB_P$ containing only ordinary atoms.

Satisfaction and Answer Sets

As for ordinary ASP programs, where

- I satisfies any ground higher-order atom $a \in HB_P$ iff $a \in I$.
- I satisfies any ground $a = \&g[y_1, \ldots, y_n](x_1, \ldots, x_m)$ iff $f_{\&g}(I, y_1, \ldots, y_n, x_1, \ldots, x_m) = 1$, where $f_{\&g}$ is a *fixed* $(n+m+1)$-ary function with range $\{0, 1\}$ for $\&g$ ($I \subseteq HB_P$, x_i, y_j ground terms).

For answer sets, use FLP reduct instead of GL reduct:

- Interpretation I is an answer set of P, iff I is a minimal model of fP^I.

Choice of FLP Reduct

Proposition

Every answer set of a HEX-program P is a minimal model of P.

Choice of FLP Reduct

Proposition

Every answer set of a HEX-program P is a minimal model of P.

This fails for the GL-reduct P^I in place of fP^I.

Example

$$p(a) \leftarrow \text{not } \&neg[p](a)$$

Suppose $f_{\&neg}(I, p)$ computes the complement of p (negation)

- Under GL-reduct, both $\emptyset$ and $\{p\}$ are answer sets
- Under FLP-reduct, only $\emptyset$ is an answer set

Choice of FLP Reduct

Proposition

Every answer set of a HEX-program P is a minimal model of P.

This fails for the GL-reduct P^I in place of fP^I.

Example

$$p(a) \leftarrow \text{not } \&neg[p](a)$$

Suppose $f_{\&neg}(I, p)$ computes the complement of p (negation)

- Under GL-reduct, both $\emptyset$ and $\{p\}$ are answer sets
- Under FLP-reduct, only $\emptyset$ is an answer set

However, GL and FLP reduct are equivalent for monotonic external atoms.

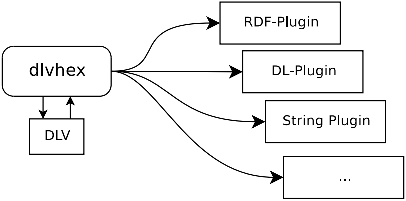
Theorem

Suppose in P all external atoms α are monotonic, i.e., for each $\alpha' \in grnd(\alpha)$, $I \subseteq J \subseteq HB_P \wedge I \models \alpha'$ implies $J \models \alpha'$. Then $ans_{GL}(P) = ans_{FLP}(P)$.

Implementation

- Algorithms: reduction to ordinary ASP, generalization of techniques
- System prototype: dlvhex

  `http://www.kr.tuwien.ac.at/research/systems/dlvhex/`



- Flexible, modular architecture
- External atoms are realized by plugins (loaded at run-time)
- Pool of plugins available
- New plugins can be defined by the user

Applications

- Fuzzy ASP [Nieuwenborgh *et al.*, 2007a], [Heymans and Toma, 2008]
- Planning with Sensing [Nieuwenborgh *et al.*, 2007b]
- Biomedical ontologies [Hoehndorf *et al.*, 2007]
- Haplotype inference
- Web querying (SPARQL) [Polleres, 2007]
- Data integration
- Trust management [Schindlauer, 2006]
- Process management in building construction [Rybenko, 2009]

Modular Nonmonotonic Logic Programs (MLPs)

- Goal: Structured programming
- In ASP, different directions:
  - Programming in the large: compositional operators

    E.g., DLP-functions [Janhunen *et al.*, 2007]
  - Programming in the small: abstraction and scoping

    E.g., Generalized Quantifiers [E_ *et al.*, 1997], Macros [Baral *et al.*, 2006], Templates [Ianni *et al.*, 2003]
- **Our aim**: Provide module ("procedure") concept as in ordinary programming
  - realize libraries, code reuse
- MLPs: look like special HEX programs, but are different
- **Type:** FLP reduct / global state

Program Modules

Conventional programming:

- Definition:

  proc p(var x, y: int): int
  begin
  ...
  end p;

- Use: $x := p(y, z)$;

Program Modules

Conventional programming:

- Definition:

  proc p(var x, y: int): int
  begin
  ...
  end p;

- Use: $x := p(y, z);$

Nonmonotonic LP:

- Definition:

  Module $m = (P[q_1, q_2], R)$, where
  - P is a module name
  - q_1, q_2 are predicate names
  - R is a set of rules

- Use: $p(X) \leftarrow P[r, s].even$

Modular Logic Program

A *modular (nonmonotonic) logic program* (MLP) $\mathbf{P} = (m_1, \ldots, m_n)$, $n \geq 1$, consists of modules $m_i = (P_i[\vec{q}_i], R_i)$ where at least one m_i has void $\vec{q}_i$.

Rule bodies may contain *module atoms* $P[p_1, \ldots, p_k].o(t_1, \ldots, t_l)$, where $p_1, \ldots, p_k$ are predicate names and $o(t_1, \ldots, t_l)$ is an ordinary atom.

Semantics (Essentials)

- For module $m_i = (P_i[\vec{q}_i]; R_i)$, each interpretation S of $\vec{q}_i$ yields an instance of m_i, named $P_i[S]$.
- An interpretation $\mathbf{M} = (M_i/S \mid P_i[S])$ of $\mathbf{P}$ consists of ordinary interpretations M_i/S for all instances $P_i[S]$ of all modules m_i in $\mathbf{P}$.
  (*global state*)
- In $P_i[S]$,
  - ordinary $o(\vec{t})$ evaluates to $o(\vec{t}) \in M_i/S$;
  - $P_j[\vec{p}_j].o(\vec{t})$ evaluates to $o(\vec{t}) \in M_j/S'$ where S' *takes the value of* $\vec{p}$ *in* M_i/S (*call by value*).
- For answer sets, extend notion of minimal model and FLP reduct to $\mathbf{P}$ (componentwise, i.e., for all $P_i[S]$).

Semantics (Essentials)

- For module $m_i = (P_i[\vec{q}_i]; R_i)$, each interpretation S of $\vec{q}_i$ yields an instance of m_i, named $P_i[S]$.
- An interpretation $\mathbf{M} = (M_i/S \mid P_i[S])$ of $\mathbf{P}$ consists of ordinary interpretations M_i/S for all instances $P_i[S]$ of all modules m_i in $\mathbf{P}$.
  (*global state*)
- In $P_i[S]$,
  - ordinary $o(\vec{t})$ evaluates to $o(\vec{t}) \in M_i/S$;
  - $P_j[\vec{p}_j].o(\vec{t})$ evaluates to $o(\vec{t}) \in M_j/S'$ where S' *takes the value of* $\vec{p}$ *in* M_i/S (*call by value*).
- For answer sets, extend notion of minimal model and FLP reduct to $\mathbf{P}$ (componentwise, i.e., for all $P_i[S]$).

Natural Question

Can't each module be simply cast to a HEX program
(module atom = external atom)?

- Difference: Global minimization (essential for loops, recursion)

- Difference: Global minimization (essential for loops, recursion)

Example

$$P_1: \quad a \leftarrow P_2[\,].b \qquad P_2: \quad b \leftarrow P_1[\,].a$$

- Answer set: $\mathbf{M}_1 = (\emptyset, \emptyset)$
- Non-minimal model: $\mathbf{M}_2 = (\{a\}, \{b\})$
- As HEX programs, P_1 and P_2 have also $\mathbf{M}_2$ as answer set.

- Difference: Global minimization (essential for loops, recursion)

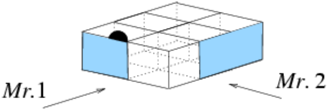
Example

$$P_1 : \quad a \leftarrow P_2[\,].b \qquad P_2 : \quad b \leftarrow P_1[\,].a$$

- Answer set: $\mathbf{M}_1 = (\emptyset, \emptyset)$
- Non-minimal model: $\mathbf{M}_2 = (\{a\}, \{b\})$
- As HEX programs, P_1 and P_2 have also $\mathbf{M}_2$ as answer set.

- Note: MLPs exclude infinite recursion.
- Still the semantics is very expressive (2-NEXP$^{\mathrm{NP}}$ vs. NEXP$^{\mathrm{NP}}$).
- Preliminary GQLPs had no recursion, used GL reduct and local models
- Refined MLP semantics takes *relevant* module calls into account.

Multi-Context Systems



- In AI, McCarthy [1987] first investigated contexts.
- Intuitively, a multi-context system describes the information available in several contexts (to people / agents/ databases etc)
- The Trento School (Giunchiglia, Serafini et al.):

  Information flow via *bridge rules* between contexts

  - Heterogeneous MCS [Giunchiglia and Serafini, 1994]
  - Nonmonotonic bridge rules [Roelofsen and Serafini, 2005]
  - Extension to Contextual Default Logic [Brewka *et al.*, 2007]
- Nonmonotonic Multi-Context Systems [Brewka and E_, 2007]:
  - abstract “logics” (description / modal / default logics, ASP, . . .)

Nonmonotonic Multi-Context Systems (MCSs)

Multi-Context System

Formally, a Multi-Context System

$$M = (C_1, \ldots, C_n)$$

consists of contexts

$$C_i = (L_i, kb_i, br_i),\ i \in \{1, \ldots, n\},$$

where

- each L_i is a "logic,"
- each kb_i is a knowledge base in L_i, and
- each br_i is a set of L_i-bridge rules over M's logics.

Logic

A *logic* L is a tuple $L = (\mathbf{KB}_L, \mathbf{BS}_L, \mathbf{ACC}_L)$, where

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

Thus, logic L caters for multiple extensions of a knowledge base.

Logic

A *logic* L is a tuple $L = (\mathbf{KB}_L, \mathbf{BS}_L, \mathbf{ACC}_L)$, where

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

Thus, logic L caters for multiple extensions of a knowledge base.

Bridge Rules

A L_i-bridge rule over logics $L_1, \ldots, L_n$, $1 \leq i \leq n$, is of the form

$$s \leftarrow (r_1 : p_1), \ldots, (r_j : p_j), \textbf{not } (r_{j+1} : p_{j+1}), \ldots, \textbf{not } (r_m : p_m)$$

where $kb \cup \{s\} \in \mathbf{KB}_i$ for each $kb \in \mathbf{KB}_i$, each $r_k \in \{1, \ldots, n\}$, and each p_k is in some belief set of L_{r_k}.

Note: Such rules are akin to rules of normal logic programs!

Example

Suppose a MCS $M = (C_1, C_2)$ has contexts that express the individual views of a paper by the two authors.

- C_1:
  - L_1 = Classical Logic
  - $kb_1 = \{\, unhappy \supset revision \,\}$
  - $br_1 = \{\, unhappy \leftarrow (2 : work) \,\}$
- C_2:
  - L_2 = Reiter's Default Logic
  - $kb_2 = \{\, good : accepted/accepted \,\}$
  - $br_2 = \{\, work \leftarrow (1 : revision),$
    $good \leftarrow$ **not** $(1 : unhappy) \,\}$

Equilibrium Semantics

Belief State

A *belief state* is a sequence $S = (S_1, \ldots, S_n)$ of belief sets S_i in L_i

Applicable Bridge Rules

For $M = (C_1, \ldots, C_n)$ and belief state $S = (S_1, \ldots, S_n)$, the bridge rule

$$s \leftarrow (r_1 : p_1), \ldots, (r_j : p_j), \textbf{not } (r_{j+1} : p_{j+1}), \ldots, \textbf{not } (r_m : p_m)$$

is *applicable in* S iff (1) $p_i \in S_{r_i}$, for $1 \leq i \leq j$, and (2) $p_k \notin S_{r_k}$, for $j < k \leq m$.

Equilibrium

A belief state $S = (S_1, \ldots, S_n)$ of M is an equilibrium iff for all $i = 1, \ldots, n$,

$$S_i \in \mathbf{ACC}_i(kb_i \cup \{head(r) \mid r \in br_i \text{ is applicable in } S\}) \,.$$

Note: Interpretable as Nash-equilibrium of an n-player game

Example (ctd)

Reconsider $M = (C_1, C_2)$:

- $kb_1 = \{\, unhappy \supset revision \,\}$ (Classical Logic)
- $br_1 = \{\, unhappy \leftarrow (2 : work) \,\}$

- $kb_2 = \{\, good : accepted / accepted \,\}$ (Default Logic)
- $br_2 = \{\, work \leftarrow (1 : revision),$
  $good \leftarrow \textbf{not}\ (1 : unhappy) \,\}$

M has two equilibria:

- $E_1 = (Th(\{unhappy, revision\}), Th(\{work\}))$ and
- $E_2 = (Th(\{unhappy \supset revision\}), Th(\{good, accepted\}))$

Groundedness

- Problem: Equilibria admit self-justifying beliefs (loops)

Example (ctd)

Intuitively, E_1 is ungrounded, since *unhappy* has a cyclic justification:

- Accept *unhappy* in C_1, since *work* is accepted in C_2, since *revision* is accepted in C_1, since *unhappy* is accepted in C_1.

Groundedness

- Problem: Equilibria admit self-justifying beliefs (loops)

Example (ctd)

Intuitively, E_1 is ungrounded, since *unhappy* has a cyclic justification:

- Accept *unhappy* in C_1, since *work* is accepted in C_2, since *revision* is accepted in C_1, since *unhappy* is accepted in C_1.

- "Groundedness" may be achieved if the logics L_i have *monotonic cores* ML_i (kb_i has a single, monotonically growing belief set).
- $M = (C_1, \ldots, C_n)$ has a unique minimal equilibrium wrt. the ML_i.
- Reduce M, given a belief state S, to $M^S = (C_1^S, \ldots, C_n^S)$ in the ML_i's.
- For bridge rules, a GL-style reduct br_i^S is used.

MCS vs. HEX programs

- MCSs take a global state view, HEX programs a local model view
- Modeling $M = (C_1, \ldots, C_n)$
  - as a collection $(P_1, \ldots, P_n)$ of HEX programs is not feasible.
  - in a *single* HEX program P_M is feasible (under conditions).

MCS vs. HEX programs

- MCSs take a global state view, HEX programs a local model view
- Modeling $M = (C_1, \ldots, C_n)$
  - as a collection $(P_1, \ldots, P_n)$ of HEX programs is not feasible.
  - in a *single* HEX program P_M is feasible (under conditions).
- **Idea**: Model formulas $(r_l : p_l)$ in bridge rules by external atoms $\&con\_r_l[\,](a_{p_l})$ being true iff p_l is in belief set S_{r_l} (a_{p_l} is a name for p_l).

MCS vs. HEX programs

- MCSs take a global state view, HEX programs a local model view
- Modeling $M = (C_1, \ldots, C_n)$
  - as a collection $(P_1, \ldots, P_n)$ of HEX programs is not feasible.
  - in a *single* HEX program P_M is feasible (under conditions).
- **Idea**: Model formulas $(r_l : p_l)$ in bridge rules by external atoms $\&con\_r_l[\,](a_{p_l})$ being true iff p_l is in belief set S_{r_l} (a_{p_l} is a name for p_l).
- Subtle problem: nondeterminism in context C_i

For the same kb_i, C_i might have multiple possible belief sets;

How to ensure that different atoms $\&con\_r_l[\,](\cdot)$ model access to the *same* belief set?

MCS vs. HEX programs

- MCSs take a global state view, HEX programs a local model view
- Modeling $M = (C_1, \ldots, C_n)$
  - as a collection $(P_1, \ldots, P_n)$ of HEX programs is not feasible.
  - in a *single* HEX program P_M is feasible (under conditions).
- **Idea**: Model formulas $(r_l : p_l)$ in bridge rules by external atoms $\&con\_r_l[\,](a_{p_l})$ being true iff p_l is in belief set S_{r_l} (a_{p_l} is a name for p_l).
- Subtle problem: nondeterminism in context C_i

> For the same kb_i, C_i might have multiple possible belief sets;
> How to ensure that different atoms $\&con\_r_l[\,](\cdot)$ model access to the *same* belief set?

- Possible, if each belief set S_i is uniquely identified by a (small) subset (*kernel*, exists in many logics)

Ongoing Work at KBS

- *Modular* HEX *programs*:
  - Formalisms and reasoning techniques
  - Algorithms (local and distributed)
  - Reasoning framework (e.g., host for distributed SPARQL)
- *Inconsistency Management for Knowledge Integration Systems*:
  - A general formalism and basic methods for inconsistency management in MCSs.
  - Algorithms for their practical realization.
  - Applications; e.g., Argumentation Context Systems (ACSs) [Brewka and E_, 2009]
    - integrate individual Dung-style argumentation frameworks $\mathcal{A}_1, \ldots, \mathcal{A}_n$
    - *mediator* M_i configures $\mathcal{A}_i$ with input from $\mathcal{A}_j$'s and manages arising inconsistency.

Theory, proofs of concepts, prototypes

Conclusion

Summary

- Need for knowledge bases with access to external sources
- Several ASP extensions address this, featuring non-monotonicty
- Different types and settings (environment view, reduct)
- An interesting area of research

Issues

- Formalisms and semantics: incompleteness, approximation
- Algorithms and methods: heterogenity, distribution, optimization (e.g., source access)
- Implementation: reasoning platforms
- Applications

References I

Anastasia Analyti, Grigoris Antoniou, and Carlos Viegas Damásio.

A principled framework for modular web rule bases and its semantics.

In *Proc. 11th Int'l Conf. Principles of Knowledge Representation and Reasoning (KR2008)*, pages 390–400. AAAI Press, September 2008.

Chitta Baral, Juraj Dzifcak, and Hiro Takahashi.

Macros, Macro calls and Use of Ensembles in Modular Answer Set Programming.

In *Proceedings of the 22th International Conference on Logic Programming (ICLP 2006)*, number 4079 in LNCS, pages 376–390. Springer, 2006.

Chitta Baral.

Knowledge Representation, Reasoning and Declarative Problem Solving.

Cambridge University Press, 2003.

References II

Gerd Brewka and Thomas Eiter.
Equilibria in Heterogeneous Nonmonotonic Multi-Context Systems.
In *AAAI-2007*, pages 385–390. AAAI Press, 2007.

Gerd Brewka and Thomas Eiter.
Argumentation context systems: A framework for abstract group argumentation.
In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2009)*, volume 5753 of *LNCS*, pages 44–57. Springer, 2009.

G. Brewka, F. Roelofsen, and L. Serafini.
Contextual default reasoning.
In *International Joint Conference on Artificial Intelligence (IJCAI 07)*, 2007.

References III

Francesco Calimeri, Susanna Cozza, and Giovambattista Ianni.
External sources of knowledge and value invention in logic programming.
Annals of Mathematics and Artificial Intelligence, 50(3-4):333–361, 2007.

Minh Dao Tran, Thomas Eiter, Michael Fink, and Thomas Krennwallner.
Modular nonmonotonic logic programming revisited.
In P.M. Hill and D.S. Warren, editors, *Proceedings 25th International Conference on Logic Programming (ICLP 2009)*, number 5649 in LNCS, pages 145–159. Springer, 2009.

Phan Minh Dung.
On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games.
Artif. Intell., 77(2):321–358, 1995.

Thomas Eiter, Georg Gottlob, and Helmuth Veith.
Modular Logic Programming and Generalized Quantifiers.
In *LPNMR-1997*, volume 1265 of *LNCS*, pages 290–309. Springer, 1997.

Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits.
A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming.
In *IJCAI-05*, pages 90–96. Professional Book Center, 2005.

Wolfgang Faber, Nicola Leone, and Gerald Pfeifer.
Recursive aggregates in disjunctive logic programs: Semantics and complexity.
In *JELIA 2004*, volume 3229 of *LNCS*, pages 200–212. Springer, September 2004.

Michael Gelfond and Vladimir Lifschitz.
The Stable Model Semantics for Logic Programming.
In *ICLP-1988*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.

Michael Gelfond and Vladimir Lifschitz.
Classical negation in logic programs and deductive databases.
New Generation Computing, 9:365–385, 1991.

F. Giunchiglia and L. Serafini.
Multilanguage hierarchical logics, or: How we can do without modal logics.
Artificial Intelligence, 65(1):29–70, 1994.

Stijn Heymans and Ioan Toma.
Ranking services using fuzzy hex-programs.
In Diego Calvanese and Georg Lausen, editors, *RR 2008*, volume 5341 of *LNCS*, pages 181–196. Springer, 2008.

References VI

Robert Hoehndorf, Frank Loebe, Janet Kelso, and Heinrich Herre.
Representing default knowledge in biomedical ontologies: Application to the integration of anatomy and phenotype ontologies.
BMC Bioinformatics, 8(1):377, 2007.

Giovambattista Ianni, Guiseppe Ielpa, Adriana Pietramala, and Maria Carmela Santoro.
Answer Set Programming with Templates.
In *Proceedings of the 2nd International Answer Set Programming Workshop (ASP'03)*, CEUR Workshop Proceedings. CEUR WS, 2003.

Tomi Janhunen, Emilia Oikarinen, Hans Tompits, and Stefan Woltran.
Modularity Aspects of Disjunctive Stable Models.
In *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 4483 of *LNCS*, pages 175–187. Springer, May 2007.

Vladimir Lifschitz.
Answer set planning.
In *ICLP*, pages 23–37, 1999.

Vladimir Lifschitz.
Answer Set Programming and Plan Generation.
Artificial Intelligence, 138:39–54, 2002.

Vladimir Lifschitz.
Twelve definitions of a stable model.
In *ICLP 2008*, pages 37–51, 2008.

Victor W. Marek and Mirosław Truszczyński.
Stable Models and an Alternative Logic Programming Paradigm.
In K. Apt, V. W. Marek, M. Truszczyński, and D. S. Warren, editors, *The Logic Programming Paradigm – A 25-Year Perspective*, pages 375–398. Springer, 1999.

J. McCarthy.
Generality in artificial intelligence.
Commun. ACM, 30(12):1029–1035, 1987.

Ilkka Niemelä.
Logic Programming with Stable Model Semantics as Constraint Programming Paradigm.
Annals of Mathematics and Artificial Intelligence, 25(3–4):241–273, 1999.

Davy Van Nieuwenborgh, Martine De Cock, and Dirk Vermeir.
Computing Fuzzy Answer Sets Using dlvhex.
In *ICLP 2007*, volume 4670 of *LNCS*, pages 449–450. Springer, 2007.

Davy Van Nieuwenborgh, Thomas Eiter, and Dirk Vermeir.
Conditional Planning with External Functions.
In *LPNMR 2007*, volume 4483 of *LNCS*, pages 214–227. Springer, 2007.

References IX

David Pearce.
Equilibrium logic.
Annals of Mathematics and Artificial Intelligence, 47(1-2):3–41, 2006.

Axel Polleres.
From SPARQL to rules (and back).
In *Proceedings of the 16th International Conference on World Wide Web (WWW)*, pages 787–796. ACM, 2007.

F. Roelofsen and L. Serafini.
Minimal and absent information in contexts.
In *Proc. IJCAI-05*, 2005.

Ksenia Rybenko.
Collaborative process management in construction by means of rules and ontologies, June 2009.

References X

Roman Schindlauer.
Answer-Set Programming for the Semantic Web.
PhD thesis, Vienna University of Technology, Austria, December 2006.

V.S. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Ozcan, and R. Ross.
Heterogeneous Agent Systems: Theory and Implementation.
MIT Press, 2000.

Giorgio Terracina, Nicola Leone, Vincenzino Lio, and Claudio Panetta.
Experimenting with recursive queries in database and logic programming systems.
TPLP, 8(2):129–165, 2008.

Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf.
The Well-Founded Semantics for General Logic Programs.
Journal of the ACM, 38(3):620–650, 1991.

Example MLP: Checking Even

$\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$, $m_2 = (P_2[q_2], R_2)$, $m_3 = (P_3[q_3], R_3)$.

$$R_1 = \{q(a).\quad q(b).\quad ok \leftarrow P_2[q].even.\}$$

$$R_2 = \left\{ \begin{array}{rcl} q_2'(X) \vee q_2'(Y) & \leftarrow & q_2(X), q_2(Y), \\ & & X \neq Y. \\ skip_2 & \leftarrow & q_2(X), \text{not } q_2'(X). \\ even & \leftarrow & \text{not } skip_2. \\ even & \leftarrow & skip_2, P_3[q_2'].odd. \end{array} \right\}$$

$$R_3 = \left\{ \begin{array}{rcl} q_3'(X) \vee q_3'(Y) & \leftarrow & q_3(X), q_3(Y), \\ & & X \neq Y. \\ skip_3 & \leftarrow & q_3(X), \text{not } q_3'(X). \\ odd & \leftarrow & skip_3, P_2[q_3'].even. \end{array} \right\}$$

$main()$:

$n := |q|$
if $even(n)$ **then return** ok

$even(n)$:

$n' := n - 1$
if $n' < 0$ **then return** true
if $n' = 0$ **then return** false
if $odd(n')$ **then return** true
else return false

$odd(n)$:

$n' := n - 1$
if $even(n')$ **then return** true
else return false

Example MLP: Checking Even

$\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$, $m_2 = (P_2[q_2], R_2)$, $m_3 = (P_3[q_3], R_3)$.

$R_1 = \{q(a). \quad q(b). \quad ok \leftarrow P_2[q].even.\}$

$$R_2 = \left\{ \begin{array}{rcl} q_2'(X) \vee q_2'(Y) & \leftarrow & q_2(X), q_2(Y), \\ & & X \neq Y. \\ skip_2 & \leftarrow & q_2(X), \text{not } q_2'(X). \\ even & \leftarrow & \text{not } skip_2. \\ even & \leftarrow & skip_2, P_3[q_2'].odd. \end{array} \right\}$$

$$R_3 = \left\{ \begin{array}{rcl} q_3'(X) \vee q_3'(Y) & \leftarrow & q_3(X), q_3(Y), \\ & & X \neq Y. \\ skip_3 & \leftarrow & q_3(X), \text{not } q_3'(X). \\ odd & \leftarrow & skip_3, P_2[q_3'].even. \end{array} \right\}$$

$main()$:

$n := |q|$
if $even(n)$ **then return** ok

$even(n)$:

$n' := n - 1$
if $n' < 0$ **then return** true
if $n' = 0$ **then return** false
if $odd(n')$ **then return** true
else return false

$odd(n)$:

$n' := n - 1$
if $even(n')$ **then return** true
else return false

Example MLP: Checking Even

$\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$, $m_2 = (P_2[q_2], R_2)$, $m_3 = (P_3[q_3], R_3)$.

$R_1 = \{q(a). \quad q(b). \quad ok \leftarrow P_2[q].even.\}$

$$R_2 = \left\{ \begin{array}{lcl} q'_2(X) \vee q'_2(Y) & \leftarrow & q_2(X), q_2(Y), \\ & & X \neq Y. \\ skip_2 & \leftarrow & q_2(X), \text{not } q'_2(X). \\ even & \leftarrow & \text{not } skip_2. \\ even & \leftarrow & skip_2, P_3[q'_2].odd. \end{array} \right\}$$

$$R_3 = \left\{ \begin{array}{lcl} q'_3(X) \vee q'_3(Y) & \leftarrow & q_3(X), q_3(Y), \\ & & X \neq Y. \\ skip_3 & \leftarrow & q_3(X), \text{not } q'_3(X). \\ odd & \leftarrow & skip_3, P_2[q'_3].even. \end{array} \right\}$$

$main()$:

$n := |q|$
if $even(n)$ **then return** ok

$even(n)$:

$n' := n - 1$
if $n' < 0$ **then return** true
if $n' = 0$ **then return** false
if $odd(n')$ **then return** true
else return false

$odd(n)$:

$n' := n - 1$
if $even(n')$ **then return** true
else return false

Example MLP: Checking Even

$\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$, $m_2 = (P_2[q_2], R_2)$, $m_3 = (P_3[q_3], R_3)$.

$R_1 = \{q(a). \quad q(b). \quad ok \leftarrow P_2[q].even.\}$

$$R_2 = \left\{ \begin{array}{rcl} q_2'(X) \vee q_2'(Y) & \leftarrow & q_2(X), q_2(Y), \\ & & X \neq Y. \\ skip_2 & \leftarrow & q_2(X), \text{not } q_2'(X). \\ even & \leftarrow & \text{not } skip_2. \\ even & \leftarrow & skip_2, P_3[q_2'].odd. \end{array} \right\}$$

$$R_3 = \left\{ \begin{array}{rcl} q_3'(X) \vee q_3'(Y) & \leftarrow & q_3(X), q_3(Y), \\ & & X \neq Y. \\ skip_3 & \leftarrow & q_3(X), \text{not } q_3'(X). \\ odd & \leftarrow & skip_3, P_2[q_3'].even. \end{array} \right\}$$

$main()$:

$n := |q|$
if $even(n)$ **then return** ok

$even(n)$:

$n' := n - 1$
if $n' < 0$ **then return** true
if $n' = 0$ **then return** false
if $odd(n')$ **then return** true
else return false

$odd(n)$:

$n' := n - 1$
if $even(n')$ **then return** true
else return false

Example MLP: Checking Even

$\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$, $m_2 = (P_2[q_2], R_2)$, $m_3 = (P_3[q_3], R_3)$.

$$R_1 = \{q(a). \quad q(b). \quad ok \leftarrow P_2[q].even.\}$$

$$R_2 = \left\{ \begin{array}{lcl} q_2'(X) \vee q_2'(Y) & \leftarrow & q_2(X), q_2(Y), \\ & & X \neq Y. \\ skip_2 & \leftarrow & q_2(X), \text{not } q_2'(X). \\ even & \leftarrow & \text{not } skip_2. \\ even & \leftarrow & skip_2, P_3[q_2'].odd. \end{array} \right\}$$

$$R_3 = \left\{ \begin{array}{lcl} q_3'(X) \vee q_3'(Y) & \leftarrow & q_3(X), q_3(Y), \\ & & X \neq Y. \\ skip_3 & \leftarrow & q_3(X), \text{not } q_3'(X). \\ odd & \leftarrow & skip_3, P_2[q_3'].even. \end{array} \right\}$$

$main()$:

$n := |q|$
if $even(n)$ **then return** ok

$even(n)$:

$n' := n - 1$
if $n' < 0$ **then return** true
if $n' = 0$ **then return** false
if $odd(n')$ **then return** true
else return false

$odd(n)$:

$n' := n - 1$
if $even(n')$ **then return** true
else return false

Example MLP: Checking Even

$\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$, $m_2 = (P_2[q_2], R_2)$, $m_3 = (P_3[q_3], R_3)$.

$R_1 = \{q(a). \quad q(b). \quad ok \leftarrow P_2[q].even.\}$

$$R_2 = \left\{ \begin{array}{rcl} q'_2(X) \vee q'_2(Y) & \leftarrow & q_2(X), q_2(Y), \\ & & X \neq Y. \\ skip_2 & \leftarrow & q_2(X), \text{not } q'_2(X). \\ even & \leftarrow & \text{not } skip_2. \\ even & \leftarrow & skip_2, P_3[q'_2].odd. \end{array} \right\}$$

$$R_3 = \left\{ \begin{array}{rcl} q'_3(X) \vee q'_3(Y) & \leftarrow & q_3(X), q_3(Y), \\ & & X \neq Y. \\ skip_3 & \leftarrow & q_3(X), \text{not } q'_3(X). \\ odd & \leftarrow & skip_3, P_2[q'_3].even. \end{array} \right\}$$

$main()$:

$n := |q|$
if $even(n)$ **then return** ok

$even(n)$:

$n' := n - 1$
if $n' < 0$ **then return** true
if $n' = 0$ **then return** false
if $odd(n')$ **then return** true
else return false

$odd(n)$:

$n' := n - 1$
if $even(n')$ **then return** true
else return false

Example MLP: Checking Even

$\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$, $m_2 = (P_2[q_2], R_2)$, $m_3 = (P_3[q_3], R_3)$.

$$R_1 = \{q(a). \quad q(b). \quad ok \leftarrow P_2[q].even.\}$$

$$R_2 = \left\{ \begin{array}{rcl} q_2'(X) \vee q_2'(Y) & \leftarrow & q_2(X), q_2(Y), \\ & & X \neq Y. \\ skip_2 & \leftarrow & q_2(X), \text{not } q_2'(X). \\ even & \leftarrow & \text{not } skip_2. \\ even & \leftarrow & skip_2, P_3[q_2'].odd. \end{array} \right\}$$

$$R_3 = \left\{ \begin{array}{rcl} q_3'(X) \vee q_3'(Y) & \leftarrow & q_3(X), q_3(Y), \\ & & X \neq Y. \\ skip_3 & \leftarrow & q_3(X), \text{not } q_3'(X). \\ odd & \leftarrow & skip_3, P_2[q_3'].even. \end{array} \right\}$$

$main()$:

$n := |q|$
if $even(n)$ **then return** ok

$even(n)$:

$n' := n - 1$
if $n' < 0$ **then return** true
if $n' = 0$ **then return** false
if $odd(n')$ **then return** true
else return false

$odd(n)$:

$n' := n - 1$
if $even(n')$ **then return** true
else return false

Checking Even (ctd)

$\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$, $m_2 = (P_2[q_2], R_2)$, $m_3 = (P_3[q_3], R_3)$.

$$R_1 = \{q(a). \quad q(b). \quad ok \leftarrow P_2[q].even.\}$$

$$R_2 = \left\{ \begin{array}{rcl} q_2'(X) \vee q_2'(Y) & \leftarrow & q_2(X), q_2(Y), \\ & & X \neq Y. \\ skip_2 & \leftarrow & q_2(X), \text{not } q_2'(X). \\ even & \leftarrow & \text{not } skip_2. \\ even & \leftarrow & skip_2, P_3[q_2'].odd. \end{array} \right\}$$

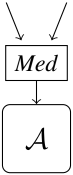
$$R_3 = \left\{ \begin{array}{rcl} q_3'(X) \vee q_3'(Y) & \leftarrow & q_3(X), q_3(Y), \\ & & X \neq Y. \\ skip_3 & \leftarrow & q_3(X), \text{not } q_3'(X). \\ odd & \leftarrow & skip_3, P_2[q_3'].even. \end{array} \right\}$$

$\mathbf{M}$
$M_1/\emptyset : \{ok, q(a), q(b)\}$
$M_2/\{q_2(a), q_2(b)\} : \left\{ \begin{array}{l} even, skip_2, \\ q_2(a), q_2(b), q_2'(b) \end{array} \right\}$
$M_2/\emptyset : \{even\}$
$\vdots$
$M_3/\{q_3(b)\} : \{odd, skip_3, q_3(b)\}$
$\vdots$

Argumentation Context Systems (ACSs)

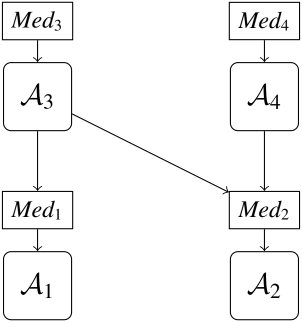
- Nonmonotonic MCS neglect two important aspects:
  1. What if information provided by different contexts is conflicting?
  2. What if a context does not only add information?
- ACSs provide an answer to these questions.
- Focus on a particular type of local reasoners:
  Dung-style argumentation frameworks [Dung, 1995]
- Goals are achieved by introducing mediators.

Argumentation Modules



- An argumentation module $\mathcal{M}$ is equipped with a mediator Med which can “listen" to other modules and “talk" to the argumentation framework $\mathcal{A}$ of $\mathcal{M}$.
- Med sets an *argumentation context* for $\mathcal{A}$ (semantics, reasoning mode, etc) expressed in a description language, depending on local and imported information, using bridge rules
- inconsistencies in the setting are treated using a parametric *inconsisteny handling method*

Example ACS



An argumentation context system.