Answer Set Programming with External Source Access

Reasoning Web Summer School 2017

DLVHEX

Thomas Eiter, Tobias Kaminski, Christoph Redl,
Peter Schüller, Antonius Weinzierl

{eiter,kaminski,redl,aweinz}@kr.tuwien.ac.at, peter.schuller@marmara.edu.tr

London, UK, July 11, 2017

Outline

Introduction

- Answer Set Programming (ASP): recent problem solving approach
- Term coined by DBLP:conf/iclp/Lifschitz99 [DBLP:conf/iclp/Lifschitz99,lifs-2002], proposed by others 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 Satisfiability (SAT) solving and Constraint Programming (CP)
- Books: [Baral, 2003], [Gebser *et al.*, 2012], compact survey: [Brewka *et al.*, 2011]

Fall 2016

ANSWER SET PROGRAMMING ARTICLES

Logic Programming – Prolog

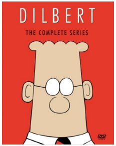
1960s/70s: Logic as a programming language (??)

- Breakthrough: Robinson's Resolution Principle (1965)

Kowalski (1979): **ALGORITHM = LOGIC + CONTROL**

- Knowledge for problem solving (LOGIC)
- "Processing" of the knowledge (CONTROL)

Prolog = "Programming in Logic"



Logic Programming – Prolog

1960s/70s: Logic as a programming language (??)

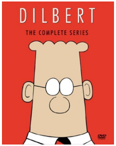
- Breakthrough: Robinson's Resolution Principle (1965)

Kowalski (1979): **ALGORITHM = LOGIC + CONTROL**

- Knowledge for problem solving (LOGIC)
- "Processing" of the knowledge (CONTROL)

Prolog = "Programming in Logic"

Example: Dilbert



$$man(dilbert).$$
$$person(X) \leftarrow man(X).$$
$$\text{query} \quad ?- \; person(X)$$

Logic Programming – Prolog

1960s/70s: Logic as a programming language (??)

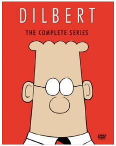
- Breakthrough: Robinson's Resolution Principle (1965)

Kowalski (1979): **ALGORITHM = LOGIC + CONTROL**

- Knowledge for problem solving (LOGIC)
- "Processing" of the knowledge (CONTROL)

Prolog = "Programming in Logic"

Example: Dilbert



$$man(dilbert).$$
$$person(X) \leftarrow man(X).$$

query $?-\ person(X)$

answer $X = dilbert$

The key: techniques to search for proofs

- Proofs provide answers, based on SLD resolution
- Understanding the resolution mechanism is important
- It may make a difference which logically equivalent form is used (e.g., termination).

The key: techniques to search for proofs

- Proofs provide answers, based on SLD resolution
- Understanding the resolution mechanism is important
- It may make a difference which logically equivalent form is used (e.g., termination).

Example: reverse lists

$$reverse([X|Y], Z) \leftarrow append(U, [X], Z), reverse(Y, U). \quad (1)$$

vs

$$reverse([X|Y], Z) \leftarrow reverse(Y, U), append(U, [X], Z). \quad (2)$$

query: $?- \; reverse([a|X], [b, c, d, b])$

The key: techniques to search for proofs

- Proofs provide answers, based on SLD resolution
- Understanding the resolution mechanism is important
- It may make a difference which logically equivalent form is used (e.g., termination).

Example: reverse lists

$$reverse([X|Y], Z) \leftarrow append(U, [X], Z), reverse(Y, U). \quad (1)$$

vs

$$reverse([X|Y], Z) \leftarrow reverse(Y, U), append(U, [X], Z). \quad (2)$$

query: $?-\ reverse([a|X], [b, c, d, b])$

- (1) yields answer "*no*", (2) does not terminate

The key: techniques to search for proofs

- Proofs provide answers, based on SLD resolution
- Understanding the resolution mechanism is important
- It may make a difference which logically equivalent form is used (e.g., termination).

Example: reverse lists

$$reverse([X|Y], Z) \leftarrow append(U, [X], Z), reverse(Y, U). \quad (1)$$

vs

$$reverse([X|Y], Z) \leftarrow reverse(Y, U), append(U, [X], Z). \quad (2)$$

query: $?-\ reverse([a|X], [b, c, d, b])$

- (1) yields answer "*no*", (2) does not terminate

Is this truly declarative programming?

Negation in Logic Programs

Why negation?

- Natural linguistic concept
- Facilitates convenient, declarative descriptions (definitions)

  E.g., ”Men who are not husbands are singles.”

Negation in Logic Programs

Why negation?

- Natural linguistic concept
- Facilitates convenient, declarative descriptions (definitions)

  E.g., ”Men who are not husbands are singles.”

Prolog: “not $\langle X \rangle$” means “Negation as Failure (to prove $\langle X \rangle$)”

Different from negation in classical logic!

Negation in Logic Programs

Why negation?

- Natural linguistic concept
- Facilitates convenient, declarative descriptions (definitions)

  E.g., ”Men who are not husbands are singles.”

Prolog: “$\mathrm{not}\ \langle X \rangle$” means “Negation as Failure (to prove $\langle X \rangle$)”

Different from negation in classical logic!

Example: Dilbert cont’d

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

% fail = ”false” in Prolog

Negation in Logic Programs

Why negation?

- Natural linguistic concept
- Facilitates convenient, declarative descriptions (definitions)

  E.g., ”Men who are not husbands are singles.”

Prolog: “$\mathrm{not}\ \langle X \rangle$” means “Negation as Failure (to prove $\langle X \rangle$)”

Different from negation in classical logic!

Example: Dilbert cont’d

$$man(dilbert).$$
$$single(X) \leftarrow man(X), \mathrm{not}\, husband(X).$$
$$husband(X) \leftarrow fail.$$ % fail = ”false” in Prolog

query $?-\ \ single(X)$

Negation in Logic Programs

Why negation?

- Natural linguistic concept
- Facilitates convenient, declarative descriptions (definitions)

  E.g., ”Men who are not husbands are singles.”

Prolog: “not $\langle X\rangle$” means “Negation as Failure (to prove $\langle X\rangle$)”

Different from negation in classical logic!

Example: Dilbert cont’d

$$man(dilbert).$$
$$single(X) \leftarrow man(X), \mathrm{not}\, husband(X).$$
$$husband(X) \leftarrow fail.$$ % fail = ”false” in Prolog

query $?-\ single(X)$
answer $X = dilbert$

Cyclic Negation

(cont'd)

Modifying the last rule of the Dilbert program, we obtain:

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

query $?-\ \ single(X)$

answer in Prolog ????

Cyclic Negation

(cont'd)

Modifying the last rule of the Dilbert program, we obtain:

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

query $?- \ single(X)$

answer in Prolog ????

Problem: not a single intuitive model!

Cyclic Negation

(cont'd)

Modifying the last rule of the Dilbert program, we obtain:

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

query $?- \;\; single(X)$

answer in Prolog ????

Problem: not a single intuitive model!

Two intuitive models:

$$\begin{aligned} M_1 &= \{man(dilbert), single(dilbert)\}, \\ M_2 &= \{man(dilbert), husband(dilbert)\} \;\; . \end{aligned}$$

Which one to choose?

Cyclic Negation

(cont'd)

Modifying the last rule of the Dilbert program, we obtain:

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

query $?-\ single(X)$

answer in Prolog ????

Problem: not a single intuitive model!

Two intuitive models:

$$
\begin{aligned}
M_1 &= \{man(dilbert), single(dilbert)\},\\
M_2 &= \{man(dilbert), husband(dilbert)\} \quad .
\end{aligned}
$$

Which one to choose? Answer set semantics: both!

LP Desiderata

Relieve the programmer from several concerns:

- the order of program rules does not matter;
- the order of subgoals in a rule does not matter;
- termination is not subject to such order.

LP Desiderata

Relieve the programmer from several concerns:

- the order of program rules does not matter;
- the order of subgoals in a rule does not matter;
- termination is not subject to such order.

“Pure” declarative programming

- Prolog does not satisfy these desiderata
- Satisfied by the *answer set semantics* of logic programs

Outline

Answer Set Programs: Syntax

Starting point: *relational signature* $S = (\mathcal{C}, \mathcal{P}, \mathcal{X})$ of pairwise disjoint sets

- $\mathcal{C}$ of *constants*,
- $\mathcal{P}$ of *predicate symbols* p/n (arity $n \geq 0$), and
- $\mathcal{X}$ of *variables*

Basic building blocks:

- *terms* are elements of $\mathcal{C} \cup \mathcal{X}$
- *atoms* are formulas $p(t_1, \ldots, t_n)$, where $p/n \in \mathcal{P}$
- *literals* are formulas a or $\mathrm{not}\, a$, where a is an atom

Answer Set Programs: Syntax

Starting point: *relational signature* $S = (\mathcal{C}, \mathcal{P}, \mathcal{X})$ of pairwise disjoint sets

- $\mathcal{C}$ of *constants*,
- $\mathcal{P}$ of *predicate symbols* p/n (arity $n \geq 0$), and
- $\mathcal{X}$ of *variables*

Basic building blocks:

- *terms* are elements of $\mathcal{C} \cup \mathcal{X}$
- *atoms* are formulas $p(t_1, \ldots, t_n)$, where $p/n \in \mathcal{P}$
- *literals* are formulas a or $\mathrm{not}\, a$, where a is an atom

Example

Typically, S is not stated explicitly if it is clear from the context; variables start with upper case letter

- terms X, bob, 123
- atoms $day()$, written as day, $firstname(bob)$, $reachable(a, Y)$
- literals $firstname(bob)$, day, $\mathrm{not}\, day$

Answer Set Programs: Syntax (cont'd)

Programs consist of rules written in "A if B" form

Rules and Programs

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

$$A_1 \vee \ldots \vee A_m \leftarrow L_1 \ldots, L_n, \quad m, n \geq 0$$

where all A_i are atoms and all L_j are literals.

- $head(r) = \{A_1, \ldots, A_m\}$ is the *head* (conclusion)
- $body(r) = \{L_1, \ldots, L_n\}$ is the *body* (premise)

Rules r with $body(r) = \emptyset$ are *facts*, and with $head(r) = \emptyset$ are *constraints*

Answer Set Programs: Syntax (cont'd)

Programs consist of rules written in "A if B" form

Rules and Programs

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

$$A_1 \vee \ldots \vee A_m \leftarrow L_1 \ldots, L_n, \quad m, n \geq 0$$

where all A_i are atoms and all L_j are literals.

- $head(r) = \{A_1, \ldots, A_m\}$ is the *head* (conclusion)
- $body(r) = \{L_1, \ldots, L_n\}$ is the *body* (premise)

Rules r with $body(r) = \emptyset$ are *facts*, and with $head(r) = \emptyset$ are *constraints*

Example

$$day \vee night.$$
$$\leftarrow sunshine, raining.$$
$$sunshine \leftarrow day, \text{not}\, raining.$$

Safety and Recursion

Technical Requirement (by Solvers)

Each variable in a rule r must occur in $body(r)$ unnegated (*safety*).

Example

$r_1 : \ p(X) \leftarrow q(X,Y), at, \text{not}\, r(X).$ safe ✓

$r_2 : \ p(X) \leftarrow \text{not}\, t(Z).$ unsafe ×

Safety and Recursion

Technical Requirement (by Solvers)

Each variable in a rule r must occur in $body(r)$ unnegated (*safety*).

Example

$$r_1 : \; p(X) \leftarrow q(X,Y), at, \text{not}\, r(X). \qquad \text{safe} \quad \checkmark$$

$$r_2 : \; p(X) \leftarrow \text{not}\, t(Z). \qquad \text{unsafe} \quad \times$$

Example: Reachability/Unreachability

$$r_1 : \quad reachable(X,Y) \leftarrow connection(X,Y).$$

$$r_2 : \quad reachable(X,Z) \leftarrow reachable(X,Y), reachable(Y,Z).$$

$$r_3 : \quad not\_reachable(X,Y) \leftarrow location(X), location(Y), \text{not}\, reachable(X,Y).$$

- Rules r_1 and r_2 express reachability (recursion)
- Rule r_3 expresses *unreachability* on top – not expressible in first-order logic!

Outline

Semantics

- Consider *ground* (i.e. *variable-free*) rules and programs
- This is lifted to arbitrary programs by variable elimination (*grounding*)

Herbrand Universe, Herbrand Base, Interpretations

Given a relational signature $S = (\mathcal{C}, \mathcal{P}, \mathcal{X})$,

- the *Herbrand universe* HU are all ground terms (i.e. $\mathcal{C}$),
- the *Herbrand base* HB is the set of all ground atoms wrt. S,
- a (Herbrand) *interpretation* is any set $I \subseteq HB$.

Intuitively, $a \in I$ means a is true in I, and false otherwise.

Semantics

- Consider *ground* (i.e. *variable-free*) rules and programs
- This is lifted to arbitrary programs by variable elimination (*grounding*)

Herbrand Universe, Herbrand Base, Interpretations

Given a relational signature $S = (\mathcal{C}, \mathcal{P}, \mathcal{X})$,

- the *Herbrand universe* HU are all ground terms (i.e. $\mathcal{C}$),
- the *Herbrand base* HB is the set of all ground atoms wrt. S,
- a (Herbrand) *interpretation* is any set $I \subseteq HB$.

Intuitively, $a \in I$ means a is true in I, and false otherwise.

Example

$P = \{ friend(X,Y) \leftarrow friend(Y,X);\ happy(X) \leftarrow friend(bob,X);\ friend(joy,bob) \}$

- $HU = \{ joy, bob \}$
- $HB = \{ friend(bob,bob), friend(bob,joy),$
  $friend(joy,bob), friend(joy,joy), happy(bob), happy(joy) \}$
- $I = \{ friend(joy,bob), friend(bob,joy), happy(joy) \}$

Semantics (cont'd)

Satisfaction of formulas, programs etc α in interpretation I, denoted $I \models \alpha$, is defined bottom up

Satisfaction, Model

An interpretation I *satisfies* (is a *model* of)

- a ground atom a, if $a \in I$;
- a literal $\text{not}\, a$, if $I \not\models a$;
- a conj. $L_1, \ldots, L_n$ of ground literals, $I \models L_i$ for $i = 1, \ldots, n$;
- a disj. $A_1 \vee \ldots \vee A_m$ of ground atoms if $I \models A_k$ for some $1 \leq k \leq m$;
- a ground rule r, if $I \models body(r)$ implies that $I \models head(r)$;
- a ground program P, if $I \models r$ for each rule $r \in P$.

Semantics (cont'd)

Satisfaction of formulas, programs etc α in interpretation I, denoted $I \models \alpha$, is defined bottom up

Satisfaction, Model

An interpretation I *satisfies* (is a *model* of)

- a ground atom a, if $a \in I$;
- a literal $\mathrm{not}\, a$, if $I \not\models a$;
- a conj. $L_1, \ldots, L_n$ of ground literals, $I \models L_i$ for $i = 1, \ldots, n$;
- a disj. $A_1 \vee \ldots \vee A_m$ of ground atoms if $I \models A_k$ for some $1 \leq k \leq m$;
- a ground rule r, if $I \models body(r)$ implies that $I \models head(r)$;
- a ground program P, if $I \models r$ for each rule $r \in P$.

Example (cont'd)

$I = \{friend(joy, bob), friend(bob, joy), happy(joy)\}$

- $I \models happy(joy)$; $\quad I \not\models happy(bob)$
- $I \models friend(bob, joy) \leftarrow friend(joy, bob)$
- $I \models happy(joy) \vee happy(bob) \leftarrow friend(bob, joy), \mathrm{not}\, friend(joy, bob)$

Semantics (cont'd)

Example

$$P = \{\, b. \quad a \leftarrow b. \quad c \leftarrow d. \,\}$$

- $I_1 = \{b, a\}$ is a model of P
- $I_2 = \{b, a, c\}$ is a model of P as well

why should c being true in I_2 be accepted?

Semantics (cont'd)

Example

$$P = \{\, b. \quad a \leftarrow b. \quad c \leftarrow d. \,\}$$

- $I_1 = \{b, a\}$ is a model of P
- $I_2 = \{b, a, c\}$ is a model of P as well

why should c being true in I_2 be accepted?

CWA Rationale

- Respect reit-78's [reit-78] *Closed World Assumption (CWA)*: If c is not derivable, assume it is false
- Semantically, prefer *minimal models*: a model I of P is *minimal*, if no model $J \subsetneq I$ of P exists.

Semantics (cont'd)

Example

$$P = \{\, b. \quad a \leftarrow b. \quad c \leftarrow d. \,\}$$

- $I_1 = \{b, a\}$ is a model of P
- $I_2 = \{b, a, c\}$ is a model of P as well

why should c being true in I_2 be accepted?

CWA Rationale

- Respect reit-78's [reit-78] *Closed World Assumption (CWA)*: If c is not derivable, assume it is false
- Semantically, prefer *minimal models*: a model I of P is *minimal*, if no model $J \subseteq I$ of P exists.

Example: CWA on mutual recursion

$$P = \{\, a \leftarrow b. \quad b \leftarrow a. \,\},$$

- $I = HB = \{a, b\}$ is a model (if P has no constraints)
- the minimal model is $I = \emptyset$

Answer Sets

Guiding Idea

- rules must be obeyed (= model)
- model must be generated by firing rules
- incorporate CWA (minimality)

Answer Sets

Guiding Idea

- rules must be obeyed (= model)
- model must be generated by firing rules
- incorporate CWA (minimality)

FLP-Reduct

The *FLP-reduct* P^I of a ground program P wrt. an interpretation I is obtained as follows: delete from P all rules r with false bodies:

$$P^I = \{r \in grnd(P) \mid I \models body(r)\}.$$

Answer sets of a program P are then defined as follows:

Answer Set

An interpretation I is an *answer set* of P, if I is a minimal model of P^I.

Answer Sets (cont'd)

Example: Restaurant

program P:

$$r_1: \quad restaurant(osteria).$$

$$r_2: \quad indoor(osteria) \leftarrow restaurant(osteria), \text{not}\, outdoor(osteria).$$

Answer Sets (cont'd)

Example: Restaurant

program P:

$$r_1 : \quad restaurant(osteria).$$

$$r_2 : \quad indoor(osteria) \leftarrow restaurant(osteria), \text{not}\, outdoor(osteria).$$

- $I_1 = \{restaurant(osteria), indoor(osteria)\}$: answer set ✓

  reduct $P^I = \{r_1, r_2\} = P$

Answer Sets (cont'd)

Example: Restaurant

program P:

$$r_1: \quad restaurant(osteria).$$

$$r_2: \quad indoor(osteria) \leftarrow restaurant(osteria), \text{not}\, outdoor(osteria).$$

- $I_1 = \{restaurant(osteria), indoor(osteria)\}$: answer set ✓

  reduct $P^I = \{r_1, r_2\} = P$

- $I_2 = \{restaurant(osteria), outdoor(osteria)\}$: no answer set ×

  reduct $P^I = \{r_1\}$

Answer Sets (cont'd)

Example: Restaurant with Decision Making

$$
\begin{array}{ll}
r_1 & restaurant(osteria). \\
r_2 & indoor(osteria) \vee outdoor(osteria) \leftarrow restaurant(osteria). \\
r_3 & eat(osteria) \leftarrow indoor(osteria), raining. \\
r_4 & eat(osteria) \leftarrow outdoor(osteria), \mathrm{not}\, raining.
\end{array}
$$

Answer Sets (cont'd)

Example: Restaurant with Decision Making

$$
\begin{array}{ll}
r_1 & restaurant(osteria). \\
r_2 & indoor(osteria) \vee outdoor(osteria) \leftarrow restaurant(osteria). \\
r_3 & eat(osteria) \leftarrow indoor(osteria), raining. \\
r_4 & eat(osteria) \leftarrow outdoor(osteria), \text{not}\, raining.
\end{array}
$$

answer sets:

- $I_1 = \{restaurant(osteria), indoor(osteria)\}$ ✓
  reduct $P^{I_1} = \{r_1, r_2\}$
- $I_2 = \{restaurant(osteria), outdoor(osteria), eat(osteria)\}$ ✓
  reduct $P^{I_2} = \{r_1, r_2, r_4\}$
- $I_3 = \{restaurant(osteria), indoor(osteria), raining\}$ ×
  reduct $P^{I_3} = \{r_1, r_2, r_3\}$
- all other I: ×

Non-Ground Programs

General Case: Variable Elimination (*Grounding*)

(ground) substitution: mapping $\sigma : \mathcal{X} \cup \mathcal{C} \rightarrow \mathcal{C}$ s.t. $\sigma(c) = c$ for any $c \in \mathcal{C}$

The *grounding* of (i) a rule r is $grnd(r) = \{r\sigma \mid \sigma$ is a substitution$\}$;
(ii) a program P is $grnd(P) = \bigcup_{r \in P} grnd(r)$.

The answer-sets of a non-ground program P are those of $grnd(P)$

Non-Ground Programs

General Case: Variable Elimination (*Grounding*)

(ground) substitution: mapping $\sigma : \mathcal{X} \cup \mathcal{C} \rightarrow \mathcal{C}$ s.t. $\sigma(c) = c$ for any $c \in \mathcal{C}$

The *grounding* of (i) a rule r is $grnd(r) = \{r\sigma \mid \sigma \text{ is a substitution}\}$;
(ii) a program P is $grnd(P) = \bigcup_{r \in P} grnd(r)$.

The answer-sets of a non-ground program P are those of $grnd(P)$

Example

- P
$$reach(X,Y) \leftarrow conn(X,Y).$$
$$reach(X,Z) \leftarrow reach(X,Y), reach(Y,Z).$$

$grnd(P) = \emptyset$ as P has no constants (in theory, let then $\mathcal{C} = \{c\}$)

Non-Ground Programs

General Case: Variable Elimination (*Grounding*)

(ground) substitution: mapping $\sigma : \mathcal{X} \cup \mathcal{C} \rightarrow \mathcal{C}$ s.t. $\sigma(c) = c$ for any $c \in \mathcal{C}$

The *grounding* of (i) a rule r is $grnd(r) = \{r\sigma \mid \sigma \text{ is a substitution}\}$;
(ii) a program P is $grnd(P) = \bigcup_{r \in P} grnd(r)$.

The answer-sets of a non-ground program P are those of $grnd(P)$

Example

- P

$$reach(X,Y) \leftarrow conn(X,Y).$$
$$reach(X,Z) \leftarrow reach(X,Y), reach(Y,Z).$$

$grnd(P) = \emptyset$ as P has no constants (in theory, let then $\mathcal{C} = \{c\}$)

- $P' = P \cup \{\, conn(a,b). \quad conn(b,c).\, \}$

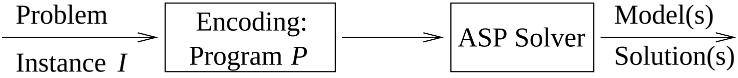
$$\begin{array}{ll}
reach(a,b) \leftarrow conn(a,b). & reach(a,b) \leftarrow reach(a,b), reach(a,b). \\
reach(b,a) \leftarrow conn(b,a). & reach(b,a) \leftarrow reach(b,a), reach(b,a). \\
reach(b,c) \leftarrow conn(b,c). & reach(b,c) \leftarrow reach(b,c), reach(b,c). \\
reach(c,b) \leftarrow conn(c,b). & reach(c,b) \leftarrow reach(c,b), reach(c,b). \\
reach(c,a) \leftarrow conn(c,a). & reach(c,a) \leftarrow reach(c,a), reach(c,a). \\
reach(a,c) \leftarrow conn(a,c). & reach(a,c) \leftarrow reach(a,c), reach(a,c).
\end{array}$$

answer set $I = \{conn(a,b), conn(b,a), reach(a,b), reach(b,c), reach(a,c)\}$

ASP Paradigm

General idea: answer sets are solutions!

Reduce solving a problem instance I to computing answer sets of an LP



- Method:
  1. *encode* 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/all models (for multiple/all solutions)
- Often: decompose I into *problem specification* and *data*
- Use a *guess and check* approach

Outline

Lack of Answer Sets: Incoherence

Programs with not might lack answer sets.

Example

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

NO answer set is possible ("derive p if it is not derivable")

Is this bad??

Lack of Answer Sets: Incoherence

Programs with not might lack answer sets.

Example

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

NO answer set is possible ("derive p if it is not derivable")

Is this bad??

Russell's Barber Paradox:

$$\begin{aligned} man(bertrand). \\ barber(bertrand). \\ shaves(X,Y) \leftarrow barber(X), man(Y),\ not\ shaves(Y,Y). \end{aligned}$$

Lack of Answer Sets: Incoherence

Programs with not might lack answer sets.

Example

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

NO answer set is possible ("derive p if it is not derivable")

Is this bad??

Russell's Barber Paradox:

$$man(bertrand).$$
$$barber(bertrand).$$
$$shaves(X,Y) \leftarrow barber(X), man(Y),\ not\ shaves(Y,Y).$$

- Adding $p \leftarrow q_1, \ldots, q_m, \text{not } r_1, \ldots, \text{not } r_n, \text{not } p.$

  to P, where p is fresh, "kills" all answer sets of P that (i) contain $q_1, \ldots, q_m$, and (ii) do not contain $r_1, \ldots, r_n$.
- This is equivalent to the constraint $\leftarrow q_1, \ldots, q_m, \text{not } r_1, \ldots, \text{not } r_n.$

Incomparability and Minimality

- Answer sets are minimal models of P^I.
- What about P itself?

Incomparability and Minimality

- Answer sets are minimal models of P^I.
- What about P itself?

Proposition (Incomparability)

If I is an answer set I of a program P, then $I \models P$ and no answer set $I' \subset I$ of P exists (i.e., with $I' \subseteq I$ s.t. $I' \neq I$).

Example

- $P = \{a \leftarrow \text{not } b\}$, answer set $I = \{a\}$
- $P = \{a \leftarrow \text{not } b; \quad b \leftarrow \text{not } a; \}$, answer sets $I_1 = \{a\}$, $I_2 = \{b\}$

Incomparability and Minimality

- Answer sets are minimal models of P^I.
- What about P itself?

Proposition (Incomparability)

If I is an answer set I of a program P, then $I \models P$ and no answer set $I' \subset I$ of P exists (i.e., with $I' \subseteq I$ s.t. $I' \neq I$).

Example

- $P = \{a \leftarrow \text{not } b\}$, answer set $I = \{a\}$
- $P = \{a \leftarrow \text{not } b; \quad b \leftarrow \text{not } a;\}$, answer sets $I_1 = \{a\}$, $I_2 = \{b\}$

In fact, answer sets satisfy a stronger property in the spirit of CWA:

Proposition (Minimality)

Every answer set I of a program P is a minimal model of P.

Non-Monotonicity

Answer sets violate the monotonicity of classical logic

Proposition (Non-monotonicity)

Given some programs P, P' and an atom a, that $I \models a$ for every answer set of P does not imply that $I \models a$ for every answer set of $P \cup P'$.

Non-Monotonicity

Answer sets violate the monotonicity of classical logic

Proposition (Non-monotonicity)

Given some programs P, P' and an atom a, that $I \models a$ for every answer set of P does not imply that $I \models a$ for every answer set of $P \cup P'$.

Example: Plain Restaurant

- restaurant program P:

  $$restaurant(osteria).$$

  $$indoor(osteria) \leftarrow restaurant(osteria), \text{not}\, outdoor(osteria).$$

  answer set
  $I = \{restaurant(osteria), indoor(osteria)\} \models indoor(osteria)$

- $P \cup \{outdoor(osteria)\}$ has the answer set
  $I = \{restaurant(osteria), outdoor(osteria)\} \not\models indoor(osteria)$

Can be exploited to declare default behaviour!

Supportedness

Presence of atoms in answer sets must be supported by rules

Example

- rule $r: \ a \leftarrow b, \mathrm{not}\, c,$ model $I = \{a, b\}$
- a is supported by the “firing” rule r

Supportedness

Presence of atoms in answer sets must be supported by rules

Example

- rule $r: \; a \leftarrow b, \text{not}\, c$, model $I = \{a, b\}$
- a is supported by the "firing" rule r

Proposition (Supportedness)

Any answer set I of a program P is a supported model, i.e., for each $a \in I$ some rule $r \in grnd(P)$ exists s.t. $I \models body(r)$ and $I \cap head(r) = \{a\}$.

Example (cont'd)

- For $P = \{b; \quad a \leftarrow b, \text{not}\, c\}$, $I = \{a, b\}$ is an answer set
- For $P = \{a \leftarrow b, \text{not}\, c\}$, $I = \{a, b\}$ is no answer set (b lacks support)

Supportedness

Presence of atoms in answer sets must be supported by rules

Example

- rule $r: \; a \leftarrow b, \mathrm{not}\, c$, model $I = \{a, b\}$
- a is supported by the "firing" rule r

Proposition (Supportedness)

Any answer set I of a program P is a supported model, i.e., for each $a \in I$ some rule $r \in grnd(P)$ exists s.t. $I \models body(r)$ and $I \cap head(r) = \{a\}$.

Example (cont'd)

- For $P = \{b; \quad a \leftarrow b, \mathrm{not}\, c\}$, $I = \{a, b\}$ is an answer set
- For $P = \{a \leftarrow b, \mathrm{not}\, c\}$, $I = \{a, b\}$ is no answer set (b lacks support)

But: stable $\neq$ minimal + supported!

Example

$$P = \{a \leftarrow a; \; a \leftarrow \mathrm{not}\; a\}$$

Computational Complexity

An answer set program P is *normal*, if each rule $r \in P$ is *normal*, defined as $|head(r)| \leq 1$.

Theorem

Deciding whether a normal program P has some answer set is

- NP*-complete in the ground (propositional) case;*
- NEXPTIME*-complete in the non-ground case.*

Theorem

Deciding whether an answer set program P has some answer set is

- Σ_2^p*-complete in the propositional case* ($\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$);
- $\mathrm{NEXPTIME}^{\mathrm{NP}}$*-complete in the non-ground case.*

Note: the relational (i.e., function-free) non-ground case as considered here is also called *datalog case*

More on complexity: [Dantsin *et al.*, 2001]

Outline

Extensions of ASP

Language extensions like aggregates, complex formula syntax are within same semantic / computational framework

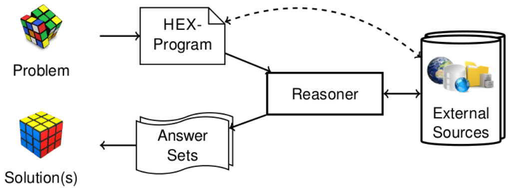
Need

- interoperability with other logics, e.g. Description Logics
- interfacing with programming languages, e.g. $C{+}{+}$, Python
- access to general *external* sources of information, e.g. WordNet

Approaches

- embedded ASP: akin to embedded SQL
- bilateral interaction: e.g. JASP
- ASP + concrete theories: constraint ASP, ASP + ontologies
- *ASP + abstract theories*: clingo, HEX/ DLVHEX

External Information Access



Examples

- import external RDF triples into the program

$$triple(S, P, O) \leftarrow \&rdf[\text{"http://}\langle Nick\rangle\text{.livejournal.com/data/foaf"}](S, P, O).$$

- access external graph

$$reachable(X) \leftarrow \&reachable[conn, a](X).$$

- perform auxiliary / data structure computations

$$fullname(Z) \leftarrow \&concat[X, Y](Z), firstname(X), lastname(Y).$$

External Information Access (cont'd)

Issues

External Information Access (cont'd)

Issues

- **Formal Model of External Atoms**
  - predicate input
  - allow arbitrary external code
    $\Rightarrow$ "impedance mismatch"

External Information Access (cont'd)

Issues

- **Formal Model of External Atoms**
  - predicate input
  - allow arbitrary external code
    $\Rightarrow$ "impedance mismatch"
- **Semantics**
  - e.g. cyclic reference (web graphs!)
  - non-monotonic external sources
    $\Rightarrow$ no simple fixpoint computation

External Information Access (cont'd)

Issues

- **Formal Model of External Atoms**
  - predicate input
  - allow arbitrary external code
    $\Rightarrow$ "impedance mismatch"
- **Semantics**
  - e.g. cyclic reference (web graphs!)
  - non-monotonic external sources
    $\Rightarrow$ no simple fixpoint computation
- **Value Invention**
  - new ground terms might appear

Outline

Syntax

New element: $\mathcal{G}$ external predicate names $\&g$ that have $in(\&g)$ many "input" arguments and $out(\&g)$ many "output" arguments

External Atom

An *external atom* over a rel. signature $S = (\mathcal{C}, \mathcal{P}, \mathcal{X}, \mathcal{G})$ is of the form

$$\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m)$$

where

- $Y_1, \ldots, Y_n$ are terms and predicate names from $\mathcal{C} \cup \mathcal{X} \cup \mathcal{P}$ (*input list*)
- $X_1, \ldots, X_m$ are terms from $\mathcal{C} \cup \mathcal{X}$ (*output list*)
- $\&g \in \mathcal{G}$ is an external predicate name with $in(\&g) = n$, $out(\&g) = m$

Syntax

New element: $\mathcal{G}$ external predicate names $\&g$ that have $in(\&g)$ many "input" arguments and $out(\&g)$ many "output" arguments

External Atom

An *external atom* over a rel. signature $S = (\mathcal{C}, \mathcal{P}, \mathcal{X}, \mathcal{G})$ is of the form

$$\&g[Y_1, \dots, Y_n](X_1, \dots, X_m)$$

where

- $Y_1, \dots, Y_n$ are terms and predicate names from $\mathcal{C} \cup \mathcal{X} \cup \mathcal{P}$ (*input list*)
- $X_1, \dots, X_m$ are terms from $\mathcal{C} \cup \mathcal{X}$ (*output list*)
- $\&g \in \mathcal{G}$ is an external predicate name with $in(\&g) = n$, $out(\&g) = m$

Examples

- $\&rdf[U](S, P, O)$: intuitively, from a given concrete "input" URL U (a constant), retrieve (one by one) all "output" triples (S, P, O)
- $\&reachable[connection, a](X)$: intuitively, all nodes X reachable from node a in a graph represented by atoms of form $connection(u, v)$.

External Atoms

Examples (cont'd)

- $\&concat[X, Y](Z)$: intuitively, concatenate two strings
  - $\&concat[bob, dylan](bobdylan)$ is true
  - $\&concat[bob, dylan](Z)$ is true for $Z = bobdylan$
  - $\&concat[bob, Y](bobdylan)$ is true for $Y = dylan$

External Atoms

Examples (cont'd)

- $\&concat[X,Y](Z)$: intuitively, concatenate two strings
  - $\&concat[bob,dylan](bobdylan)$ is true
  - $\&concat[bob,dylan](Z)$ is true for $Z = bobdylan$
  - $\&concat[bob,Y](bobdylan)$ is true for $Y = dylan$

External atoms can be of any nature (non-logical) nature

Example

$$\&weatherreport[dateLocationPredicate](WeatherConditions)$$

query a web-based weather report

- input $dateLocationPredicate$ is a binary predicate with tuples (d,l) of dates d and locations l (facts $dateLocationPredicate(d,l)$)
- output $WeatherConditions$ are (one by one) all weather conditions that occur at some input date & location

$\&weatherreport[goto](W)$ where $goto = \{(1,paris),(1,london),(2,paris),(2,london)\}$ returns all weather conditions on dates 1/2 for London/Paris

HEX Programs

HEX rule and program

A HEX *program* is a set P of *(HEX) rules* r of the form

$$A_1 \vee \ldots \vee A_m \leftarrow L_1 \ldots, L_n, \quad m, n \geq 0,$$

where all A_i are atoms, and all L_j are either literals or HEX-literals, i.e. either

- an ordinary literal,
- an external atom,
- or a default-negated external atom.

That is, like ordinary ASP rules/programs but external atoms can occur in rule bodies

Examples

- $reachable(X) \leftarrow \&reachable[connection, a](X).$
- $fullname(Z) \leftarrow \&concat[X, Y](Z), firstname(X), lastname(Y).$
- $\leftarrow \&weatherreport[goto](W), badweather(W).$

HEX Programs (cont'd)

Example: City Trip

Plan to visit Paris and London, under the condition the weather isn't bad

Program Π_{goto}:

$$
\begin{array}{lr}
r_1 & badweather(rain). \quad badweather(snow). \\
r_2 & goto(1, paris) \vee goto(1, london). \\
r_3 & goto(2, paris) \vee goto(2, london). \\
r_4 & \leftarrow \&weatherreport[goto](W), badweather(W).
\end{array}
$$

- state what bad weather means (r_1)
- decide on what day to go to which city (r_2, r_3)
- exclude trips where the (external) weather report indicates bad weather during the trip (r_4)

Outline

Semantics

Analogous to ordinary ASP:

- the *Herbrand base* HB for HEX program P
- the *grounding* of a rule r, $grnd(r)$, and of P, $grnd(P) = \bigcup_{r \in P} grnd(r)$.
- *interpretations* are subsets $I \subseteq HB$ with no external atoms

To define satisfaction, key issue is the semantics of external atoms.

Semantics

Analogous to ordinary ASP:

- the *Herbrand base* HB for HEX program P
- the *grounding* of a rule r, $grnd(r)$, and of P, $grnd(P) = \bigcup_{r \in P} grnd(r)$.
- *interpretations* are subsets $I \subseteq HB$ with no external atoms

To define satisfaction, key issue is the semantics of external atoms.

Oracle Function

Every $\&g \in \mathcal{G}$, has an associated decidable *oracle function*

$$f_{\&g} : 2^{HB_P} \times (\mathcal{C} \cup \mathcal{P})^n \times \mathcal{C}^m \to \{\mathbf{T}, \mathbf{F}\}, \quad n = in(\&g), m = out(\&g)$$

that maps each $(I, \vec{y}, \vec{x})$, where $I \subseteq HB$ is an interpretation, $\vec{y} = y_1, \ldots, y_n$ on $\mathcal{C} \cup \mathcal{P}$ is "input", and $\vec{x} = x_1, \ldots, x_m$ on $\mathcal{C}$ is "output", to $\mathbf{T}$ or $\mathbf{F}$.

Pragmatic assumptions:

- for any $I, \vec{y}$, only finitely many $\vec{x}$ yield $f_{\&g}(I, \vec{y}, \vec{x}) = \mathbf{T}$
- output $\vec{x}$ is independent of the extensions of the predicates that do not occur in the input $\vec{y}$

Oracle Functions

Example: String Concatenation

for the external predicate $\&concat$, the associated function is

$$f_{\&concat}(I, X, Y, Z) = \begin{cases} \mathbf{T}, & \text{if } XY = Z; \\ \mathbf{F}, & \text{otherwise} \end{cases}$$

(where XY is concatenation of X and Y)

Oracle Functions

Example: String Concatenation

for the external predicate *&concat*, the associated function is

$$f_{\&concat}(I, X, Y, Z) = \begin{cases} \mathbf{T}, & \text{if } XY = Z; \\ \mathbf{F}, & \text{otherwise} \end{cases}$$

(where XY is concatenation of X and Y)

Example: City Trip (cont'd)

- weather forecast Paris: sun on day 1 and day 2
- weather forecast London: rain on day 1 and day 2

the corresponding oracle function is ($wr = weatherreport$)

$$f_{\&wr}(I, goto, W) = \begin{cases} \mathbf{T}, \text{ if } \{goto(1, london), goto(2, london)\} \subseteq I \text{ and } W = rain, \\ \mathbf{T}, \text{ if } \{goto(1, london), goto(2, paris)\} \subseteq I \text{ and } W \in \{sun,\ rain\}, \\ \mathbf{T}, \text{ if } \{goto(1, paris), goto(2, london)\} \subseteq I \text{ and } W \in \{sun,\ rain\}, \\ \mathbf{T}, \text{ if } \{goto(1, paris), goto(2, paris)\} \subseteq I \text{ and } W = sun, \\ \mathbf{F}, \text{ otherwise.} \end{cases}$$

Satisfaction and Models

Satisfaction of External Atom

An interpretation $I \subseteq HB$ *satisfies* (is a *model* of) a ground external atom $a = \&g[\vec{y}](\vec{x})$, denoted $I \models a$, if $f_{\&g}(I, \vec{y}, \vec{x}) = \mathbf{T}$.

Satisfaction and Models

Satisfaction of External Atom

An interpretation $I \subseteq HB$ *satisfies* (is a *model* of) a ground external atom $a = \&g[\vec{y}](\vec{x})$, denoted $I \models a$, if $f_{\&g}(I, \vec{y}, \vec{x}) = \mathbf{T}$.

Example: String Concatenation

I plays no role for concatenation:

- $I \models \&concat[bob, dylan](bobdylan)$ holds for every interpretation I
- $I \not\models \&concat[bob, dylan](bobbydylan)$ for every interpretation I

Satisfaction and Models

Satisfaction of External Atom

An interpretation $I \subseteq HB$ *satisfies* (is a *model* of) a ground external atom $a = \&g[\vec{y}](\vec{x})$, denoted $I \models a$, if $f_{\&g}(I, \vec{y}, \vec{x}) = \mathbf{T}$.

Example: String Concatenation

I plays no role for concatenation:

- $I \models \&concat[bob, dylan](bobdylan)$ holds for every interpretation I
- $I \not\models \&concat[bob, dylan](bobbydylan)$ for every interpretation I

Example: City Trip (cont'd)

For weather forecast as above:

- $I \models \&weatherreport[goto](sun)$ holds if $I \models goto(1, paris)$, or if $I \models goto(2, paris)$.
- $I \models \&weatherreport[goto](rain)$ if $I \models goto(1, london)$ or if $I \models goto(2, london)$,

Answer Sets for HEX Programs

Answer sets naturally extend to HEX-programs

Answer Set of a HEX Program

An interpretation $I \subseteq HB$ is an *answer set* of a HEX program P, if I is a minimal model of the *FLP-reduct*

$$P^I = \{r \in grnd(P) \mid I \models body(r)\}.$$

$AS(P)$ = the set of all answer sets of P

Answer Sets for HEX Programs

Answer sets naturally extend to HEX-programs

Answer Set of a HEX Program

An interpretation $I \subseteq HB$ is an *answer set* of a HEX program P, if I is a minimal model of the *FLP-reduct*

$$P^I = \{r \in grnd(P) \mid I \models body(r)\}.$$

$AS(P)$ = the set of all answer sets of P

Remarks:

- For ordinary P (no external atoms), the answer sets are as usual
- For aggregates modeled as external atoms (e.g. $\&count[goto](N)$), the answer sets coincide with FLP-answer sets [Faber *et al.*, 2011]
- Alternative (more restrictive) notions of answer sets exist [Shen *et al.*, 2014]

Answer Sets for HEX Programs

Example: City Trip (cont'd)

$$
\begin{array}{ll}
\Pi_{goto} & badweather(rain). \quad badweather(snow). \\
& goto(1, paris) \vee goto(1, london). \\
& goto(2, paris) \vee goto(2, london). \\
& \leftarrow \&weatherreport[goto](W), badweather(W).
\end{array}
$$

- For the above weather report, Π_{goto} has one answer set: $\{goto(1, paris), goto(2, paris), badweather(snow), badweather(rain)\}$

Answer Sets for HEX Programs

Example: City Trip (cont'd)

$$\Pi_{goto} \qquad badweather(rain). \qquad badweather(snow).$$
$$goto(1, paris) \vee goto(1, london).$$
$$goto(2, paris) \vee goto(2, london).$$
$$\leftarrow \&weatherreport[goto](W), badweather(W).$$

- For the above weather report, Π_{goto} has one answer set: $\{goto(1, paris), goto(2, paris), badweather(snow), badweather(rain)\}$
- For a different weather report saying it's always sunny, 3 more answer sets exist:
  - $\{goto(1, paris), goto(2, london), badweather(snow), badweather(rain)\}$
  - $\{goto(1, london), goto(2, paris), badweather(snow), badweather(rain)\}$
  - $\{goto(1, london), goto(2, london), badweather(snow), badweather(rain)\}$

Answer Sets for HEX Programs

Example: City Trip (cont'd)

$$\Pi_{goto} \qquad badweather(rain). \quad badweather(snow).$$
$$goto(1, paris) \vee goto(1, london).$$
$$goto(2, paris) \vee goto(2, london).$$
$$\leftarrow \&weatherreport[goto](W), badweather(W).$$

- For the above weather report, Π_{goto} has one answer set: $\{goto(1, paris), goto(2, paris), badweather(snow), badweather(rain)\}$
- For a different weather report saying it's always sunny, 3 more answer sets exist:
  - $\{goto(1, paris), goto(2, london), badweather(snow), badweather(rain)\}$
  - $\{goto(1, london), goto(2, paris), badweather(snow), badweather(rain)\}$
  - $\{goto(1, london), goto(2, london), badweather(snow), badweather(rain)\}$
- Finally if the weather report for both cities is *snow* for days 1 and 2, no answer set exists.

Outline

Basic Properties

The basic properties of answer sets extend to HEX-programs:

- answer sets are incomparable
- answer sets are minimal models
- answer sets are supported models
- non-monotonicity

Basic Properties

The basic properties of answer sets extend to HEX-programs:

- answer sets are incomparable
- answer sets are minimal models
- answer sets are supported models
- non-monotonicity

The computational complexity depends on external atoms: deciding answer set existence is

- Σ_2^p-complete for ground programs, if evaluating external atoms, i.e. deciding whether $f_{\&g}(I, \vec{y}, \vec{x}) = \mathbf{T}$ holds, is feasible in polynomial time with an NP oracle;
- Σ_2^p-hard already for Horn ground programs (no disjunction, no negation) and polynomial-time external atoms.
- Thus, *minimality checking of answer set candidates for* HEX*-programs is a challenging problem*

Outline

Basic Methodology

Modeling techniques from ordinary ASP carry over to HEX-programs.

Basic Methodology

Modeling techniques from ordinary ASP carry over to HEX-programs.

Guess and check paradigm

1. Generate a superset of the desired solutions.
   $\Rightarrow$ Use disjunctive rules or default negation to span a search space.

Basic Methodology

Modeling techniques from ordinary ASP carry over to HEX-programs.

Guess and check paradigm

1. Generate a superset of the desired solutions.
   $\Rightarrow$ Use disjunctive rules or default negation to span a search space.
2. Use constraints to eliminate spurious solutions.

Basic Methodology

Modeling techniques from ordinary ASP carry over to HEX-programs.

Guess and check paradigm

1. Generate a superset of the desired solutions.
   $\Rightarrow$ Use disjunctive rules or default negation to span a search space.
2. Use constraints to eliminate spurious solutions.

Example: 3-Colorability of a Graph

Consider a graph $G = (V, E)$
given by facts $node(v)$ for all $v \in V$ and $edge(u, v)$ for all $(u, v) \in E$.

Basic Methodology

Modeling techniques from ordinary ASP carry over to HEX-programs.

Guess and check paradigm

1. Generate a superset of the desired solutions.
   $\Rightarrow$ Use disjunctive rules or default negation to span a search space.
2. Use constraints to eliminate spurious solutions.

Example: 3-Colorability of a Graph

Consider a graph $G = (V, E)$
given by facts $node(v)$ for all $v \in V$ and $edge(u, v)$ for all $(u, v) \in E$.

$$
\begin{aligned}
r(X) \vee g(X) \vee b(X) &\leftarrow node(X) \\
&\leftarrow r(X), r(Y), edge(X, Y) \\
&\leftarrow g(X), g(Y), edge(X, Y) \\
&\leftarrow b(X), b(Y), edge(X, Y)
\end{aligned}
$$

Basic Methodology (cont'd)

Saturation technique

1. Check whether all possible guesses satisfy a certain property Pr.

Basic Methodology (cont'd)

Saturation technique

1. Check whether all possible guesses satisfy a certain property Pr.
2. To test a property Pr we
   - design a program P and an answer set candidate I_{sat} such that I_{sat} is the single answer set of P if the property Pr holds, and
   - P has other answer sets (excluding I_{sat}) otherwise.

Basic Methodology (cont'd)

Saturation technique

1. Check whether all possible guesses satisfy a certain property Pr.
2. To test a property Pr we
   - design a program P and an answer set candidate I_{sat} such that I_{sat} is the single answer set of P if the property Pr holds, and
   - P has other answer sets (excluding I_{sat}) otherwise.

Example: Non-3-Colorability of a Graph

$$
\begin{aligned}
b(X) \vee r(X) \vee g(X) &\leftarrow node(X)\\
non\_col &\leftarrow r(X), r(Y), edge(X,Y)\\
non\_col &\leftarrow g(X), g(Y), edge(X,Y)\\
non\_col &\leftarrow b(X), b(Y), edge(X,Y)\\
r(X) &\leftarrow non\_col, node(X)\\
g(X) &\leftarrow non\_col, node(X)\\
b(X) &\leftarrow non\_col, node(X)
\end{aligned}
$$

Basic Methodology (cont'd)

Extension with External Atoms

- The existing techniques can be combined with external atoms.

Basic Methodology (cont'd)

Extension with External Atoms

- The existing techniques can be combined with external atoms.
- Example: Checks can be outsourced to external sources.

Basic Methodology (cont'd)

Extension with External Atoms

- The existing techniques can be combined with external atoms.
- Example: Checks can be outsourced to external sources.

Example: 3-Colorability of a Graph

Consider a graph $G = (V, E)$
given by facts $node(v)$ for all $v \in V$ and $edge(u, v)$ for all $(u, v) \in E$.

Basic Methodology (cont'd)

Extension with External Atoms

- The existing techniques can be combined with external atoms.
- Example: Checks can be outsourced to external sources.

Example: 3-Colorability of a Graph

Consider a graph $G = (V, E)$
given by facts $node(v)$ for all $v \in V$ and $edge(u, v)$ for all $(u, v) \in E$.

$$
\begin{aligned}
r(X) \vee g(X) \vee b(X) &\leftarrow node(X) \\
&\leftarrow \text{not } \&check[edge, r, g, b]()
\end{aligned}
$$

Outline

Methodology for Using External Atoms

Main Usages of External Atoms

- **Computation Outsourcing:**
  Send the definition of a subproblem to an external source and retrieve its result.

Methodology for Using External Atoms

Main Usages of External Atoms

- **Computation Outsourcing:**
  Send the definition of a subproblem to an external source and retrieve its result.
- **Information Outsourcing:**
  External sources import information while reasoning itself is done in the logic program.

Methodology for Using External Atoms

Main Usages of External Atoms

- **Computation Outsourcing:**
  Send the definition of a subproblem to an external source and retrieve its result.
- **Information Outsourcing:**
  External sources import information while reasoning itself is done in the logic program.

Note:

- Both types of outsourcing may be used together in a program.

Methodology for Using External Atoms

Main Usages of External Atoms

- **Computation Outsourcing:**
  Send the definition of a subproblem to an external source and retrieve its result.
- **Information Outsourcing:**
  External sources import information while reasoning itself is done in the logic program.

Note:

- Both types of outsourcing may be used together in a program.
- External sources may combine both use cases.

Methodology for Using External Atoms

Main Usages of External Atoms

- **Computation Outsourcing:**
  Send the definition of a subproblem to an external source and retrieve its result.
- **Information Outsourcing:**
  External sources import information while reasoning itself is done in the logic program.

Note:

- Both types of outsourcing may be used together in a program.
- External sources may combine both use cases.
- Important: Both usages are based on the same language features!

Computation Outsourcing

On-demand Constrains

- Constraints of form
  $$\leftarrow \&forbidden[p_1, \ldots, p_n]()$$
  eliminate certain extensions of predicates $p_1, \ldots, p_n$.

Computation Outsourcing

On-demand Constrains

- Constraints of form
$$\leftarrow \&forbidden[p_1, \ldots, p_n]()$$
eliminate certain extensions of predicates $p_1, \ldots, p_n$.

- Advantage:
Explicit grounding of ASP constraints representing the forbidden combinations is avoided
(cf. constraint ASP [Ostrowski and Schaub, 2012]).

Computation Outsourcing

On-demand Constrains

- Constraints of form
$$\leftarrow \&forbidden[p_1, \ldots, p_n]()$$
eliminate certain extensions of predicates $p_1, \ldots, p_n$.

- Advantage:
Explicit grounding of ASP constraints representing the forbidden combinations is avoided
(cf. constraint ASP [Ostrowski and Schaub, 2012]).

- The external evaluation may notify the reasoner about reasons for conflicts to restrict the search space (see later).

Computation Outsourcing

On-demand Constrains

- Constraints of form
$$\leftarrow \&\mathit{forbidden}[p_1, \dots, p_n]()$$
eliminate certain extensions of predicates $p_1, \dots, p_n$.

- Advantage:
Explicit grounding of ASP constraints representing the forbidden combinations is avoided
(cf. constraint ASP [Ostrowski and Schaub, 2012]).

- The external evaluation may notify the reasoner about reasons for conflicts to restrict the search space (see later).

- Example:
Efficient planning in robotics where external atoms verify the feasibility of a 3D motion [Erdem *et al.*, 2016b].

Computation Outsourcing (cont'd)

Accessing Procedural Computations

- Accessing algorithms which cannot (easily or efficiently) be expressed by rules.

Computation Outsourcing (cont'd)

Accessing Procedural Computations

- Accessing algorithms which cannot (easily or efficiently) be expressed by rules.

- Example:
  AngryHEX is an AI agent for the game *AngryBirds* that needs to perform physics simulations [Calimeri *et al.*, 2013b].

Computation Outsourcing (cont'd)

Accessing Procedural Computations

- Accessing algorithms which cannot (easily or efficiently) be expressed by rules.

- Example:
  AngryHEX is an AI agent for the game *AngryBirds* that needs to perform physics simulations [Calimeri *et al.*, 2013b].

Complexity Lifting

- Computations with a complexity higher than the complexity of ordinary ASP programs.

Computation Outsourcing (cont'd)

Accessing Procedural Computations

- Accessing algorithms which cannot (easily or efficiently) be expressed by rules.
- Example:
  AngryHEX is an AI agent for the game *AngryBirds* that needs to perform physics simulations [Calimeri *et al.*, 2013b].

Complexity Lifting

- Computations with a complexity higher than the complexity of ordinary ASP programs.
- External sources can also be other ASP or HEX programs, which allows for encoding other formalisms of higher complexity in HEX programs, e.g., *abstract argumentation frameworks* [Dung, 1995].

Information Outsourcing

Data Sources

- RDF triplet stores:
  $p(X, Y) \leftarrow url(U), \&rdf[U](X, Y, Z)$
- Geographic data
- Description logic ontologies
- Multi-context systems
- Relational databases

Information Outsourcing

Data Sources

- RDF triplet stores:
  $p(X, Y) \leftarrow url(U), \&rdf[U](X, Y, Z)$
- Geographic data
- Description logic ontologies
- Multi-context systems
- Relational databases

Note:
Some external sources may realize a combination of data and computation outsourcing (e.g. complex queries over ontologies).

Outline

Modeling an Application

How to realize an application on top of HEX-programs?

Modeling an Application

How to realize an application on top of HEX-programs?

Typical Procedure

1. Identify and realize the required external atoms.

Modeling an Application

How to realize an application on top of HEX-programs?

Typical Procedure

1. Identify and realize the required external atoms.
2. Write the HEX-program which uses these external atoms.

Modeling an Application

How to realize an application on top of HEX-programs?

Typical Procedure

1. Identify and realize the required external atoms.
2. Write the HEX-program which uses these external atoms.

These steps might be repeated or interleaved.

Modeling an Application

How to realize an application on top of HEX-programs?

Typical Procedure

1. Identify and realize the required external atoms.
2. Write the HEX-program which uses these external atoms.

These steps might be repeated or interleaved.

External atoms might be reused for multiple applications.

Outline

Applications of HEX-Programs

Some examples:

- Queries of Web resources (RDF triplet stores, social graphs, etc)
- Multi-context Systems (interconnection of knowledge-bases)
- DL-programs (integration of ASP with ontologies)
- Constraint ASP (programs with constraint atoms)
- Physics simulation (e.g. AngryBirds agent)
- Route planning (possibly semantically enriched)
- Robotics applications (planning)
- ACTHEX (programs with action atoms)

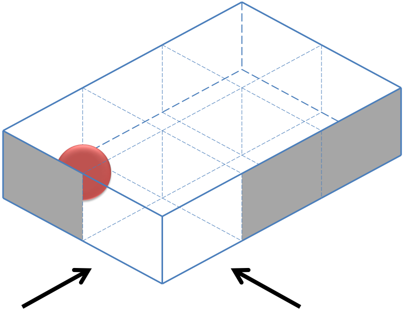
Applications of HEX-Programs

Some example

- Queoi
- Multi-conte
- DL-progra
- Constraint
- Physics si
- Route plar
- Robotics a
- ACTHEX (p

raphs, etc)

ɔases)



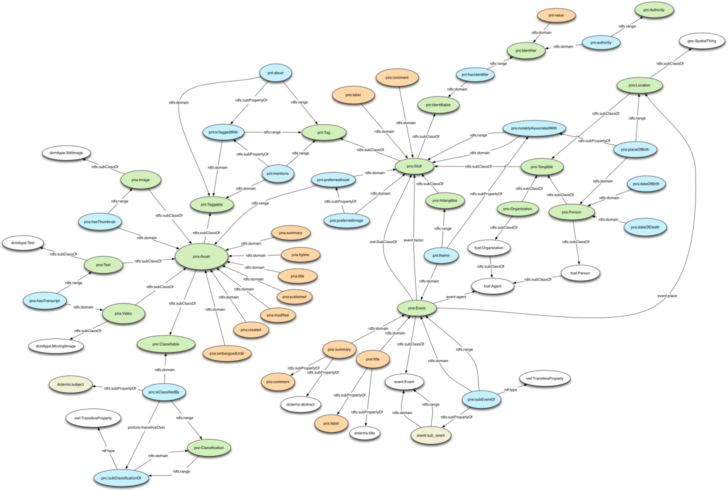
Applications of HEX-Programs

Some examples:

- Queries of Web resources (RDF triplet stores, social graphs, etc)
- Multi-context Systems (interconnection of knowledge-bases)
- DL-programs (integration of ASP with ontologies)
- Constraint ASP (programs with constraint atoms)
- Physics simulation (e.g. AngryBirds agent)
- Route planning (possibly semantically enriched)
- Robotics applications (planning)
- ACTHEX (programs with action atoms)

Applications of HEX-Programs

Son



Applications of HEX-Programs

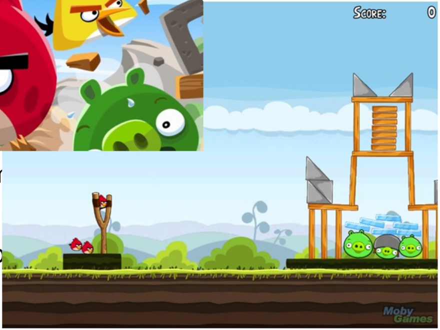
Some examples:

- Queries of Web resources (RDF triplet stores, social graphs, etc)
- Multi-context Systems (interconnection of knowledge-bases)
- DL-programs (integration of ASP with ontologies)
- Constraint ASP (programs with constraint atoms)
- Physics simulation (e.g. AngryBirds agent)
- Route planning (possibly semantically enriched)
- Robotics applications (planning)
- ACTHEX (programs with action atoms)

Applications of HEX-Programs

Some e

- Qu


social graphs, etc)


- Mu
- DL
- Co
- Ph
- Route planning (possibly sem
- Robotics applications (planni
- ACTHEX (programs with actio

Applications of HEX-Programs

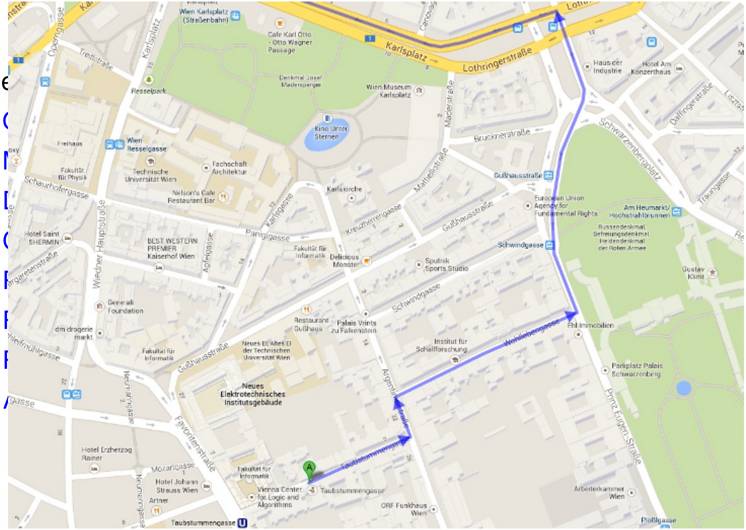
Some examples:

- Queries of Web resources (RDF triplet stores, social graphs, etc)
- Multi-context Systems (interconnection of knowledge-bases)
- DL-programs (integration of ASP with ontologies)
- Constraint ASP (programs with constraint atoms)
- Physics simulation (e.g. AngryBirds agent)
- Route planning (possibly semantically enriched)
- Robotics applications (planning)
- ACTHEX (programs with action atoms)

Applications of HEX-Programs

Some

- C
- N
- D
- C
- F
- F
- F
- A



Applications of HEX-Programs

Some examples:

- Queries of Web resources (RDF triplet stores, social graphs, etc)
- Multi-context Systems (interconnection of knowledge-bases)
- DL-programs (integration of ASP with ontologies)
- Constraint ASP (programs with constraint atoms)
- Physics simulation (e.g. AngryBirds agent)
- Route planning (possibly semantically enriched)
- Robotics applications (planning)
- ACTHEX (programs with action atoms)

Example: Semantic Web Application

Example: Friend-of-a-Friend

Use the FOAF (Friend-of-a-friend) RDF schema to return all pairs of nicknames that know each other, as stored in a FOAF RDF datasource:

$$explore(\text{"http://}\langle Nick\rangle\text{.livejournal.com/data/foaf"})$$

$$triple(S,P,O) \leftarrow \&rdf[What](S,P,O),\ explore(What)$$

$$\begin{aligned} knows(Nick_1, Nick_2) \leftarrow\ & triple(Id_1, \text{"http://xmlns.com/foaf/0.1/knows"}, Id_2), \\ & triple(Id_1, \text{"http://xmlns.com/foaf/0.1/nick"}, Nick_1),\ Nick_1 < Nick_2, \\ & triple(Id_2, \text{"http://xmlns.com/foaf/0.1/nick"}, Nick_2). \end{aligned}$$

$$knows(A,C) \leftarrow knows(A,B),\ knows(B,C)$$

Example: Semantic Web Application (cont'd)

Example: Recursive FOAF querying with limited depth

$explore($"http://$\langle Nick \rangle$.livejournal.com/data/foaf"$)$

$explore\_to(What, 3) \leftarrow explore(What)$

$triple\_at(S, P, O, D) \leftarrow \&rdf[Uri](S, P, O),\ explore\_to(Uri, D),\ D > 1$

$explore\_to(U, D_2) \leftarrow D_2 = D_1 - 1,$

$\quad triple\_at(Id,$ "http://www.w3.org/2000/01/rdf-schema#seeAlso"$, U, D_1),$

$\quad triple\_at(Id,$ "http://xmlns.com/foaf/0.1/nick"$, Nick, D_1)$

$found(Nick) \leftarrow triple\_at(S,$ "http://xmlns.com/foaf/0.1/nick"$, Nick, D).$

Example: Physics Simulation

Example: AngryHEX

Fundamental strategy:
Maximize the estimated damage to obstacles and pigs.

$$
\begin{aligned}
shootable(O, Type, Tr) \leftarrow\ & \&shootable[O, Tr, V, Sx, Sy, Sw, Sh, B, bb](O), \\
& birdType(B),\ velocity(V),\ objectType(O, Type), \\
& slingshot(Sx, Sy, Sw, Sh),\ trajectory(Tr) \\
tgt(O, Tr) \vee ntgt(O, Tr) \leftarrow\ & shootable(O, Type, Tr) \\
\leftarrow\ & target(X, \_),\ target(Y, \_), X \neq Y. \\
\leftarrow\ & target(\_, T_1),\ target(\_, T_2),\ T_1 \neq T_2 \\
target\_ex \leftarrow\ & target(\_, \_) \\
\leftarrow\ & \text{not}\ target\_ex. \\
directDmg(O, P, E) \leftarrow\ & target(O, Tr),\ objectType(O, T),\ birdType(Bird), \\
& dmgProbability(Bird, T, P), \\
& energyLoss(Bird, T, E)
\end{aligned}
$$

Example: Physics Simulation

Example: AngryHEX (cont'd)

$$
\begin{aligned}
exDirectDmg(O) &\leftarrow directDmg(O, \_, \_) \\
nexDirectDmg(O) &\leftarrow \text{not } exDirectDmg(O),\ objectType(O, \_) \\
goodObject(O) &\leftarrow objectType(O, pig) \\
goodObject(O) &\leftarrow objectType(O, tnt) \\
&\leftsquigarrow nexDirectDmg(O), goodObject(O) \quad [1@4, O, nexDirectDmg] \\
&\leftsquigarrow nexDirectDmg(O). \quad [1@1, O, nexDirectDmg]
\end{aligned}
$$

The DLVHEX-System

DLVHEX

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

- Based on GRINGO and CLASP from the Potassco suite.
- Supported platforms: Linux-based, OS X, Windows.
  Pre-compiled binaries available.
- External sources are implemented as plugins using a plugin API (available for C++ or Python).
- Support for the ASP-Core-2 standard.
- Online demo:
  `http://www.kr.tuwien.ac.at/research/systems/dlvhex/demo.php`.
- User manual available (see system website).

System Architecture

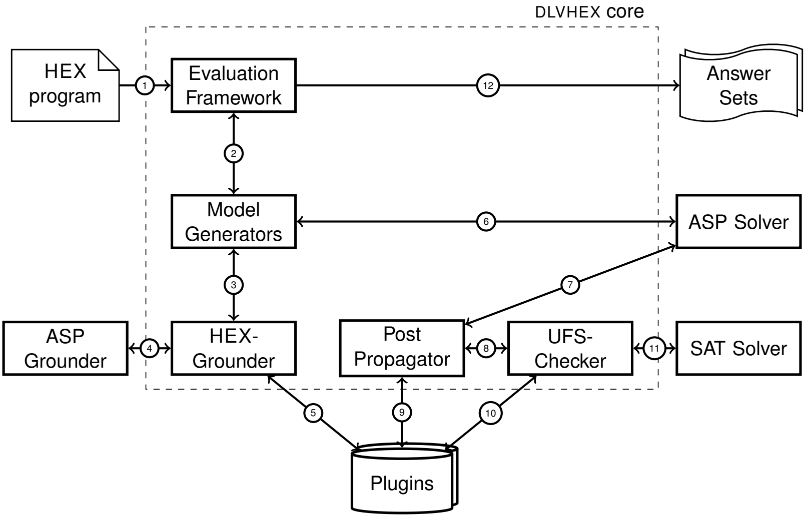


Figure: Architecture of DLVHEX

Outline

Python Programming Interface

More convenient interface

Previously only C++ support, but Python preferred by many developers:

- No overhead due to makefiles, compilation, linking, etc.
- High-level features.
- Negligible overhead compared to plugins implemented in C++.

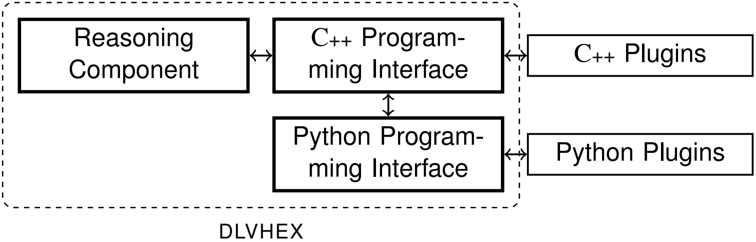


Figure: Architecture of the Python Programming Interface

Python Programming Interface (cont'd)

Example

Program

$$\Pi = \left\{ \begin{array}{l} r_1: start(s). \\ r_2: reach(X) \leftarrow start(X). \quad r_3: reach(Y) \leftarrow reach(X), \&edge[X](Y). \end{array} \right\}$$

compute the nodes reachable from a start node s in a graph.

Implementation of $\&edge[X](Y)$:

```
def edge(x):
  graph=((1,2),(1,3),(2,3))          # simplified implementation
  for edge in graph:                 # search for out-edges of node x
    if edge[0]==x.intValue():
      dlvhex.output((edge[1],))      # output edge target

def register():
  prop = dlvhex.ExtSourceProperties() # inform dlvhex about
  prop.addFiniteOutputDomain(0)       # finiteness of the graph
  dlvhex.addAtom("edge", (dlvhex.CONSTANT, ), 1, prop)
```

Outline

From Black-box to Grey-box

Overcoming the Evaluation Bottleneck

- By default, external sources are seen as black boxes.
- Behavior under an interpretation does not allow for drawing conclusions about other interpretations.
- Algorithmic improvements require meta-information about external sources.

From Black-box to Grey-box

Overcoming the Evaluation Bottleneck

- By default, external sources are seen as black boxes.
- Behavior under an interpretation does not allow for drawing conclusions about other interpretations.
- Algorithmic improvements require meta-information about external sources.

Idea

- Developers of external sources and/or implementer of HEX-program might have useful additional information.
- Provide a (predefined) list of possible properties of external sources.
- Let the developer and/or user specify which properties are satisfied.
- Algorithms exploit them for various purposes, most importantly:
  - efficiency improvements and
  - language flexibility (reducing syntactic restrictions).

From Black-box to Grey-box

Overcoming the Evaluation Bottleneck

- By default, external sources are seen as black boxes.
- Behavior under an interpretation does not allow for drawing conclusions about other interpretations.
- Algorithmic improvements require meta-information about external sources.

Idea

- Developers of external sources and/or implementer of HEX-program might have useful additional information.
- Provide a (predefined) list of possible properties of external sources.
- Let the developer and/or user specify which properties are satisfied.
- Algorithms exploit them for various purposes, most importantly:
  - efficiency improvements and
  - language flexibility (reducing syntactic restrictions).

Important:
User specifies them but does **not** need to know how they are exploited!

Specifying Properties

Available properties (examples)

- **Functionality**: $\&add[X, Y](Z)\langle functional\rangle$
  Adds integers X and Y and is true for their sum Z.
  It provides exactly one output for a given input.

Specifying Properties

Available properties (examples)

- **Functionality**: $\&add[X, Y](Z)\langle functional \rangle$
  Adds integers X and Y and is true for their sum Z.
  It provides exactly one output for a given input.
- **Well-ordering**: $\&decrement[X](Z)\langle wellordering\ 0\ 0 \rangle$
  Decrements a given integer.
  The 0-th output is no greater than the 0-th input (wrt. some ordering).

Specifying Properties

Available properties (examples)

- **Functionality**: $\&add[X, Y](Z)\langle functional\rangle$
  Adds integers X and Y and is true for their sum Z.
  It provides exactly one output for a given input.
- **Well-ordering**: $\&decrement[X](Z)\langle wellordering\ 0\ 0\rangle$
  Decrements a given integer.
  The 0-th output is no greater than the 0-th input (wrt. some ordering).
- **Three-valued semantics**:
  The external source can be evaluated under partial interpretations.

Specifying Properties

Available properties (examples)

- **Functionality**: $\&add[X,Y](Z)\langle functional\rangle$
  Adds integers X and Y and is true for their sum Z.
  It provides exactly one output for a given input.
- **Well-ordering**: $\&decrement[X](Z)\langle wellordering\ 0\ 0\rangle$
  Decrements a given integer.
  The 0-th output is no greater than the 0-th input (wrt. some ordering).
- **Three-valued semantics**:
  The external source can be evaluated under partial interpretations.
- …

Specifying Properties

Available properties (examples)

- **Functionality**: $\&add[X, Y](Z)\langle functional\rangle$
  Adds integers X and Y and is true for their sum Z.
  It provides exactly one output for a given input.
- **Well-ordering**: $\&decrement[X](Z)\langle wellordering\ 0\ 0\rangle$
  Decrements a given integer.
  The 0-th output is no greater than the 0-th input (wrt. some ordering).
- **Three-valued semantics**:
  The external source can be evaluated under partial interpretations.
- ...

How to specify them?

- During development of external source using the plugin API.

Specifying Properties

Available properties (examples)

- **Functionality**: $\&add[X, Y](Z)\langle functional\rangle$
  Adds integers X and Y and is true for their sum Z.
  It provides exactly one output for a given input.
- **Well-ordering**: $\&decrement[X](Z)\langle wellordering\ 0\ 0\rangle$
  Decrements a given integer.
  The 0-th output is no greater than the 0-th input (wrt. some ordering).
- **Three-valued semantics**:
  The external source can be evaluated under partial interpretations.
- ...

How to specify them?

- During development of external source using the plugin API.
- As part of the HEX-program using property tags $\langle \cdots \rangle$.

Specifying Properties

Available properties (examples)

- **Functionality**: $\&add[X, Y](Z)\langle functional\rangle$
  Adds integers X and Y and is true for their sum Z.
  It provides exactly one output for a given input.
- **Well-ordering**: $\&decrement[X](Z)\langle wellordering\ 0\ 0\rangle$
  Decrements a given integer.
  The 0-th output is no greater than the 0-th input (wrt. some ordering).
- **Three-valued semantics**:
  The external source can be evaluated under partial interpretations.
- …

How to specify them?

- During development of external source using the plugin API.
- As part of the HEX-program using property tags $\langle \cdots \rangle$.
  Example:
  $\&greaterThan[p, 10]()$ is true if $\sum_{p(c)\in I} c > 10$.
  It is monotonic for positive integers.

Exploiting Properties for Efficiency Improvement

Conflict-driven Solving

- ASP program is internally represented by nogoods (sets of literals which cannot be simultaneously true).
- Additional nogoods learned from conflicting interpretations.
- HEX-solver further learns nogoods from external sources which describe parts of their behavior to avoid future wrong guesses.

Exploiting Properties for Efficiency Improvement

Conflict-driven Solving

- ASP program is internally represented by nogoods (sets of literals which cannot be simultaneously true).
- Additional nogoods learned from conflicting interpretations.
- HEX-solver further learns nogoods from external sources which describe parts of their behavior to avoid future wrong guesses.

Example

- We evaluate $\&\mathit{diff}[p,q](X)$ under $I = \{p(a), q(b)\}$.
- It is true for $X = a$ (and false otherwise), i.e., $I \models \&\mathit{diff}[p,q](a)$.
- $\Rightarrow$ Learn nogood $N = \{p(a), \neg q(a), \neg p(b), q(b), \neg \&\mathit{diff}[p,q](a)\}$.

Exploiting Properties for Efficiency Improvement

Conflict-driven Solving

- ASP program is internally represented by nogoods (sets of literals which cannot be simultaneously true).
- Additional nogoods learned from conflicting interpretations.
- HEX-solver further learns nogoods from external sources which describe parts of their behavior to avoid future wrong guesses.

Example

- We evaluate $\&\mathit{diff}[p,q](X)$ under $I = \{p(a), q(b)\}$.
- It is true for $X = a$ (and false otherwise), i.e., $I \models \&\mathit{diff}[p,q](a)$.
- $\Rightarrow$ Learn nogood $N = \{p(a), \neg q(a), \neg p(b), q(b), \neg \&\mathit{diff}[p,q](a)\}$.

Exploiting Properties

- Known properties used to shrink nogoods to their essential part.
- Example: $\&\mathit{diff}[p,q](X)$ is monotonic in p:
  Shrink above nogood N to $N' = \{p(a), \neg q(a), q(b), \neg \&\mathit{diff}[p,q](a)\}$.
  (If $p(b)$ turns to true, $\&\mathit{diff}[p,q](a)$ is still true $\Rightarrow \neg p(b)$ not needed.)

Exploiting Properties for Language Flexibility

Grounding and Safety

- External atoms may introduce new constants: value invention.
- $\Rightarrow$ Can lead to programs which cannot be finitely grounded.

Exploiting Properties for Language Flexibility

Grounding and Safety

- External atoms may introduce new constants: value invention.
- $\Rightarrow$ Can lead to programs which cannot be finitely grounded.

Example

$$\Pi = \left\{ \begin{array}{l} r_1: start(s). \\ r_2: reach(X) \leftarrow start(X). \quad r_3: reach(Y) \leftarrow reach(X), \&edge[X](Y). \end{array} \right\}$$

Exploiting Properties for Language Flexibility

Grounding and Safety

- External atoms may introduce new constants: value invention.
- $\Rightarrow$ Can lead to programs which cannot be finitely grounded.

Example

$$\Pi = \left\{ \begin{array}{l} r_1: start(s). \\ r_2: reach(X) \leftarrow start(X). \quad r_3: reach(Y) \leftarrow reach(X), \&edge[X](Y). \end{array} \right\}$$

Solution: Syntactic Restrictions (Safety)

- Traditionally: strong safety; essentially no recursive value invention!

Exploiting Properties for Language Flexibility

Grounding and Safety

- External atoms may introduce new constants: value invention.
- $\Rightarrow$ Can lead to programs which cannot be finitely grounded.

Example

$$\Pi = \left\{ \begin{array}{l} r_1 : start(s). \\ r_2 : reach(X) \leftarrow start(X). \quad r_3 : reach(Y) \leftarrow reach(X), \&edge[X](Y). \end{array} \right\}$$

Solution: Syntactic Restrictions (Safety)

- Traditionally: strong safety; essentially no recursive value invention!
- But: overly restrictive.

Exploiting Properties for Language Flexibility

Grounding and Safety

- External atoms may introduce new constants: value invention.
- $\Rightarrow$ Can lead to programs which cannot be finitely grounded.

Example

$$\Pi = \left\{ \begin{array}{l} r_1 : \mathit{start}(s). \\ r_2 : \mathit{reach}(X) \leftarrow \mathit{start}(X). \quad r_3 : \mathit{reach}(Y) \leftarrow \mathit{reach}(X), \&\mathit{edge}[X](Y). \end{array} \right\}$$

Solution: Syntactic Restrictions (Safety)

- Traditionally: strong safety; essentially no recursive value invention!
- But: overly restrictive.

Exploiting Properties

- Properties may allow for identifying finite groundability even in presence of recursive value invention (in some cases).
- Example:
  Known finiteness of the graph above allows for establishing safety.

Outline

Use Case: Semantic Trip Planning in Vienna

Requirements

- Find shortest trip visiting predefined locations
- Long trip ⇒ add lunch location using an ontology
- Choose restaurant depending on weather report

DEMO

Trip Planning

- Transport data might be:
  - Extremely large
  - Remote/not accessible
- Access external transport information (information outsourcing)
- Use dedicated algorithm to compute shortest connection (computation outsourcing)

External atom:
`&route[File,Loc1,Loc2](Stp1,Stp2,Costs,Line)`

$\Rightarrow$ Obtain shortest trip by using weak constraints

DEMO

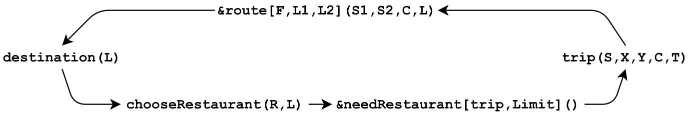
Adding Lunch Location

- Adjustment of the trip based on its length
- Add on-demand constraint (no output needed)
- Boolean output depends monotonically on the input
  - Specify according property

External atom:

`&needRestaurant[trip,Limit]()`

Introduces cyclic dependency, not strongly safe:



DEMO

Partial Evaluation

- `&needRestaurant[trip,Limit]()` usually evaluated only after extension of `trip` is decided
  - Truth value not fixed before
- Often truth value can be decided early during search
- Partial assignments: atoms can be *true*, *false* or *unassigned*
- Use both methods isTrue() and isFalse()
  - Everything else is unassigned
- Use both methods output() and outputUnknown() to declare outputs
  - All other outputs are implicitly false
- Requirement: assignment monotonicity

Example

Learned nogood: $\{\neg t(0,1), t(1,1), t(2,1), t(3,1), \&nR[t,3]()\}$

DEMO

DL-Lite Plugin

- We use the DL-Lite Plugin for semantically enriched route planning (inspired by [Eiter *et al.*, 2016c])
- Interfaces to OWL ontologies using DL reasoner
- Provides external atoms for concept and role queries:
  - `&cDL[File,rp,rm,cp,cm,C](X)`
  - `&rDL[File,rp,rm,cp,cm,R](X,Y)`
- Bidirectional interaction by adding elements to concepts and roles, resp. to their complements

Link:
http://www.kr.tuwien.ac.at/research/systems/dlvhex/dlliteplugin.html

Restaurant Ontology

$$
\begin{aligned}
BeerGarden &\sqsubseteq Restaurant \\
BeerGarden &\sqsubseteq \neg IndoorRestaurant \\
IndoorRestaurant &\sqsubseteq Restaurant \\
IndoorRestaurant &\sqsubseteq \neg BeerGarden \\
IndoorRestaurant &\sqsubseteq \neg WurstStand \\
Restaurant &\sqsubseteq \exists closeTo.Location \\
WurstStand &\sqsubseteq Restaurant \\
WurstStand &\sqsubseteq \neg IndoorRestaurant
\end{aligned}
$$

$$
\begin{aligned}
&Location(Karlsplatz) \\
&Location(Museumsquartier) \\
&Location(Praterstern) \\
&BeerGarden(bg1) \\
&closeTo(bg1, Praterstern) \\
&IndoorRestaurant(ir1) \\
&closeTo(ir1, Museumsquartier) \\
&WurstStand(ws1) \\
&closeTo(ws1, Karlsplatz)
\end{aligned}
$$

DEMO

Weather Data

- Goal: retrieve weather data from http://openweathermap.org/
- Importing dynamic data from remote location
- General plugin for retrieving JSON data from API
  - Data represented by nested key-value pairs:
    `{"weather":[{"id":803,"main":"Clouds",`
    `"description":"clouds", "icon":"04d"}], ...}`
- Input type `dlvhex.TUPLE` for arbitrary number of constants
  - Provide sequence of keys

External atom:

`&getJSON[Url,Keys.TUPLE](Value)`

DEMO

Summary of the Case Study

- Encoding uses four different external atoms in combination
  - `&route`-Plugin for information and computation outsourcing
  - `&needRestaurant`-Plugin for external check
  - DL-Lite-Plugin for interfacing an external DL-reasoner
  - `&getJson`-Plugin for accessing remote information on the web
- Complete implementation and more examples at:
  https://github.com/hexhex/manual/tree/master/RW2017/

Outline

HEX$^{\exists}$ Programs

- By value invention external atoms can generate witnesses
- Used to model query answering from existential rules

Example

Not possible in standard ASP:

$$\exists X\colon \mathit{office}(Y,X) \leftarrow \mathit{employee}(Y).$$

Encoding with external atom:

$$\mathit{office}(Y,X) \leftarrow \mathit{employee}(Y), \&\mathit{exists}[r_1, Y](X).$$

HEX Programs with Function Symbols

- External atoms can simulate composition and decomposition of function terms
- Allows external data type checking and argument generation

Example

Not possible in standard ASP:

$$
\begin{aligned}
q(f(X)) &\leftarrow p(X). \\
r(Y) &\leftarrow q(f(Y)).
\end{aligned}
$$

Encoding with external atom:

$$
\begin{aligned}
q(A) &\leftarrow p(X), \&comp[f, X](A). \\
r(Y) &\leftarrow q(B), \&decomp[B](f, Y).
\end{aligned}
$$

ACTHEX

- Extension of HEX for execution of declaratively scheduled actions
- Action atoms in rule heads operate on an external environment
- Environment can influence truth values of external atoms
  - Enables stateful behaviour

Example

$$
\begin{aligned}
\#robot[clean, kitchen]\{c, 2\} &\leftarrow night \\
\#robot[clean, bedroom]\{c, 2\} &\leftarrow day \\
\#robot[goto, charger]\{b, 1\} &\leftarrow \&sensor[bat](low) \\
night \vee day &\leftarrow
\end{aligned}
$$

Constraint HEX Programs

- Grounding issues when encoding constraints in ASP
- Contain ordinary, external and constraint atoms
  - Comparisons of arithmetic expressions
- Allow to combine diverse background theories

Example

$$food(P) \leftarrow \&sql[\text{“Select price from Food”}](P)$$
$$drink(P) \leftarrow \&sql[\text{“Select price from Drink”}](P)$$
$$inMenu(F,D) \vee outMenu(F,D) \leftarrow food(F), drink(D)$$
$$F + D < P \leftarrow inMenu(F,D), max\_price(P)$$

Encoding of constraint with external atom:

$$con(F,+,D,<,P) \vee con(F,+,D,\geq,P) \leftarrow inMenu(F,D), max\_price(P)$$
$$\leftarrow \text{not } \&check[con]()$$

Nested HEX [Eiter *et al.*, 2013]

- External atoms for evaluating subprograms and inspecting their answer sets:
  $\&callhex$, $\&callhexfile$, $\&answersets$, $\&predicates$, $\&arguments$
- A new instance of DLVHEX is called and results stored in an answer cache assigning unique handles

Example

$$
\begin{aligned}
p_1(x,y) &\leftarrow \\
p_2(a) &\leftarrow \\
p_2(b) &\leftarrow \\
handle(PH) &\leftarrow \&callhexfile[\texttt{"sub.hex"}, p_1, p_2](PH) \\
ash(PH, AH) &\leftarrow \&callhex[\texttt{"a v b :-"}](PH), \&answersets[PH](AH)
\end{aligned}
$$

Outline

Related Work

- Many approaches, different degrees of integration
- DLV^{DB} offers access to relational databases via *ODBC* interface
- ONTODLV for information retrieval from OWL ontologies, extends ASP with classes, inheritance, relations and axioms
- DLV-EX programs early generic integration approach
  - Introduction of new terms by value invention
  - Only terms as inputs to external sources
  - Nonmonotonic aggregates not expressible
- CLINGO supports custom functions implemented in *Lua* or *Python*
  - Import extensions of user-defined predicates during grounding
  - Customisable built-in atoms
  - No cyclic dependencies

Related Work (cont'd)

- CLINGO 5 provides generic interfaces for theory solving in ASP
  - Semantics differs from HEX unfounded support of theory atoms allowed $\Rightarrow$ consider $p \leftarrow \&id[p]()$
  - Theory atoms interrelated via external theory (orthogonal to HEX)
  - No value invention based on answer set
  - Well-suited for system developers, rich infrastructure
- Extensions of ASP with specific external sources:
  - Constraint ASP solvers, e.g. CLINGCON, lc2casp, ezcsp, EZSMT
  - Extensions of ASP with SMT, e.g. dingo (difference logic), ASPMT

Outline

Summary

- HEX is a powerful formalism, wide range of applications
- Extends ASP with external sources via API-style interface
- Bi-directional interaction and value invention possible
- Methodology from ASP generalises to HEX
- Implemented in the DLVHEX system
  - Plugins in Python and C++
  - Exploiting external source properties

DLVHEX

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

Outline

Further Resources

- All executable examples from this course:
  `https://github.com/hexhex/manual/tree/master/RW2017/`
- Slides of tutorial "ASP for the Semantic Web" and many executable ASP/HEX-examples:
  `http://asptut.gibbi.com/`
- An online demo of the DLVHEX system:
  `http://www.kr.tuwien.ac.at/research/systems/dlvhex/demo.php`
- Pre-built binaries of DLVHEX for Linux, OS X and Windows:
  `http://www.kr.tuwien.ac.at/research/systems/dlvhex/downloadb.html`
- The source code of DLVHEX and corresponding plugins, best place for bug reports:
  `https://github.com/hexhex/`
- Python-based HEX implementation for a fragment of the HEX language and a subset of features
  `https://github.com/hexhex/hexlite`

References I

Chitta Baral.
Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge Univ. Press, 2003.

Markus Bögl, Thomas Eiter, Michael Fink, and Peter Schüller.
The MCS-IE system for explaining inconsistency in multi-context systems.
In *In Proceedings of the Twelfth European Conference on Logics in Artificial Intelligence (JELIA 2010)*, pages 356–359, 2010.

Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski.
Answer set programming at a glance.
Commun. ACM, 54(12):92–103, 2011.

Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Thomas Krennwallner Roland Kaminski, Nicola Leone, Francesco Ricca, and Torsten Schaub.
ASP-Core-2 Input Language Format, 2013.

Francesco Calimeri, Michael Fink, Stefano Germano, Giovambattista Ianni, Christoph Redl, and Anton Wimmer.
AngryHEX: an artificial player for angry birds based on declarative knowledge bases.
In *National Workshop and Prize on Popularize Artificial Intelligence*, pages 29–35, 2013.

Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov.
Complexity and Expressive Power of Logic Programming.
ACM Computing Surveys, 33(3):374–425, 2001.

Phan Minh Dung.
On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games.
Artificial Intelligence, 77(2):321–357, 1995.

References II

Thomas Eiter, Thomas Krennwallner, and Christoph Redl.
HEX-Programs with Nested Program Calls.
In Hans Tompits, Salvador Abreu, Johannes Oetsch, Jörg Pührer, Dietmar Seipel, Masanobu Umeda, and Armin Wolf, editors, *19th International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2011)*, volume 7773 of *LNAI*, pages 1–10. Springer, October 2013.

Thomas Eiter, Michael Fink, Thomas Krennwallner, Christoph Redl, and Peter Schüller.
Efficient HEX-program evaluation based on unfounded sets.
Journal of Artificial Intelligence Research, 49:269–321, February 2014.

Thomas Eiter, Michael Fink, Thomas Krennwallner, and Christoph Redl.
Domain expansion for asp-programs with external sources.
Artif. Intell., 233:84–121, 2016.

Thomas Eiter, Tobiask Kaminski, Christoph Redl, and Antonius Weinzierl.
Exploiting partial assignments for efficient evaluation of answer set programs with external source access.
In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI 2016), July 9–15, 2016, New York City, New York, USA*, July 2016.

Thomas Eiter, Thomas Krennwallner, Matthias Prandtstetter, Christian Rudloff, Patrik Schneider, and Markus Straub.
Semantically enriched multi-modal routing.
Int. J. Intelligent Transportation Systems Research, 14(1):20–35, 2016.

Esra Erdem, Volkan Patoglu, and Peter Schüller.
A Systematic Analysis of Levels of Integration between High-Level Task Planning and Low-Level Feasibility Checks.
AI Communications, IOS Press, 2016.

References III

Esra Erdem, Volkan Patoglu, and Peter Schüller.
A Systematic Analysis of Levels of Integration between Low-Level Reasoning and Task Planning.
AI Communications, 29(2):319–349, 2016.

Wolfgang Faber, Nicola Leone, and Gerald Pfeifer.
Semantics and complexity of recursive aggregates in answer set programming.
Artificial Intelligence, 175(1):278–298, 2011.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub.
Answer Set Solving in Practice.
Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

Giovambattista Ianni, Francesco Calimeri, Stefano Germano, Andreas Humenberger, Christoph Redl, Daria Stepanova, Andrea Tucci, and Anton Wimmer.
Angry-HEX: an artificial player for angry birds based on declarative knowledge bases.
IEEE Transactions on Computational Intelligence and AI in Games, 2016.

Victor W. Marek and Mirosław Truszczyński.
Stable Models and an Alternative Logic Programming Paradigm.
In *The Logic Programming Paradigm – A 25-Year Perspective*, pages 375–398. Springer, 1999.

Ilkka Niemelä.
Logic programming with stable model semantics as constraint programming paradigm.
Annals of Mathematics and Artificial Intelligenc, 25(3–4):241–273, 1999.

Max Ostrowski and Torsten Schaub.
ASP modulo CSP: the clingcon system.
Theory and Practice of Logic Programming (TPLP), 12(4-5):485–503, 2012.

References IV

Yi-Dong Shen, Kewen Wang, Thomas Eiter, Michael Fink, Christoph Redl, Thomas Krennwallner, and Jun Deng.
FLP answer set semantics without circular justifications for general logic programs.
Artificial Intelligence, 213:1–41, 2014.

Hande Zirtiloglu and Pinar Yolum.
Ranking semantic information for e-government: complaints management.
In Alistair Duke, Martin Hepp, Kalina Bontcheva, and Marc B. Vilain, editors, *Proceedings of the First International Workshop on Ontology-supported Business Intelligence, OBI 2008, Karlsruhe, Germany, October 27, 2008*, volume 308 of *ACM International Conference Proceeding Series*, page 5. ACM, 2008.