11.

Answer Set Programming

Outline

- Different Approaches to Deal with Defaults
- Answer Set Semantics
- Answer Set Programming

Comparing Classical Approaches to Defaults

Approach	Advantage	Disadvantage
Closed World	Simplicity for propositional case	Minimizes all predicates
Circumscription	Minimizes selected predicates	Forces to talk about abnormality
Default Logic	Add only selected negative literals	Cannot reason about defaults
Auto epistemic Logic	Represents defaults & consistent beliefs	Practicality

Why Answer Set Semantics

- Logic programming community has searched for a simple, clean and computationally elegant ways to handle default negation
- There are two approaches that have been most popular
  - Well founded semantics
  - Answer set semantics

Semantics of Logic Programs

- Well Founded Semantics
  - 3-valued models: true, false, undefined
  - Every KB has a unique well founded model
  - Well founded model (for datalog[1]) poly-computable
- Answer Set Semantics
  - Classical 2-valued models
  - KB may have 0 or more stable models
  - Stable model (for datalog) NP-complete
    - We will consider this in more detail

1. Datalog: function free horn logic programs

Answer Set Semantics

- Basic definitions
- Interpretations
- Positive programs
- General programs

Rules and Constraints

$$b_1 \text{ or } \ldots \text{ or } b_m \leftarrow a_1, \ldots, a_n, \text{ not } a_{n+1}, \ldots, \text{ not } a_{n+k}$$

- a_i, b_j: atoms in a first order language.
- *not a*: a *negation-as-failure atom* (naf-atom).

Reading 1

If $a_1, \ldots, a_n$ are true and none of $a_{n+1}, \ldots, a_{n+k}$ can be proven to be true then at least one of $b_1, \ldots, b_m$ must be true.

Reading 2

If $a_1, \ldots, a_n$ are believed to be true and there is no reason to believe that any of $a_{n+1}, \ldots, a_{n+k}$ is true then at least one of $b_1, \ldots, b_m$ must be true.

Rules and Constraints

$$b_1 \text{ or } \ldots \text{ or } b_m \leftarrow a_1, \ldots, a_n, \text{ not } a_{n+1}, \ldots, \text{ not } a_{n+k}$$

- a_i, b_j: atoms in a first order language.
- *not a*: a *negation-as-failure atom* (naf-atom).

Reading 1

If $a_1, \ldots, a_n$ are true and none of $a_{n+1}, \ldots, a_{n+k}$ can be proven to be true then at least one of $b_1, \ldots, b_m$ must be true.

Reading 2

If $a_1, \ldots, a_n$ are believed to be true and there is no reason to believe that any of $a_{n+1}, \ldots, a_{n+k}$ is true then at least one of $b_1, \ldots, b_m$ must be true.

Rules and Constraints

$$b_1 \text{ or } \ldots \text{ or } b_m \leftarrow a_1, \ldots, a_n, \text{ not } a_{n+1}, \ldots, \text{ not } a_{n+k}$$

- a_i, b_j: atoms in a first order language.
- *not a*: a *negation-as-failure atom* (naf-atom).

Reading 1

If $a_1, \ldots, a_n$ are true and none of $a_{n+1}, \ldots, a_{n+k}$ can be proven to be true then at least one of $b_1, \ldots, b_m$ must be true.

Reading 2

If $a_1, \ldots, a_n$ are believed to be true and there is no reason to believe that any of $a_{n+1}, \ldots, a_{n+k}$ is true then at least one of $b_1, \ldots, b_m$ must be true.

Notations

$$r: \quad \underbrace{b_1 \text{ or } \ldots \text{ or } b_m}_{head(r)} \leftarrow \underbrace{a_1, \ldots, a_n, \text{ not } a_{n+1}, \ldots, \text{ not } a_{n+k}}_{body(r)}.$$

- $head(r) = \{b_1, \ldots, b_m\}$
- $pos(r) = \{a_1, \ldots, a_n\}$
- $neg(r) = \{a_{n+1}, \ldots, a_{n+k}\}$

Special cases

- $n = k = 0$: r encodes a fact;
- $k = 0$: r is a positive rule; and
- $m = 0$: r encodes a constraint.

Program

- Program: a set of rules.
- Herbrand universe: the set of ground terms constructed from function symbols and constants occurring in the program. (U_π)
- Herbrand base: the set of ground atoms constructed from predicate symbols and ground terms from the Herbrand universe. (B_π)
- Rule with variables: shorthand for the collection of its ground instances.
- Program with variables: collection of ground instances of its rules.

  $\Rightarrow$ "programs" $\equiv$ "ground programs"

Program

- Program: a set of rules.
- Herbrand universe: the set of ground terms constructed from function symbols and constants occurring in the program. (U_π)
- Herbrand base: the set of ground atoms constructed from predicate symbols and ground terms from the Herbrand universe. (B_π)
- Rule with variables: shorthand for the collection of its ground instances.
- Program with variables: collection of ground instances of its rules.

  $\Rightarrow$ "programs" $\equiv$ "ground programs"

Logic Program Examples

Example (Tweety the penguin)

$$\pi_1 = \begin{cases} fly(X) \leftarrow bird(X),\ not\ ab_f(X) \\ bird(X) \leftarrow penguin(X) \\ ab_f(X) \leftarrow penguin(X) \\ penguin(t) \leftarrow \end{cases}$$

$U_{\pi_1} = \{t\}$

$B_{\pi_1} = \{fly(t), bird(t), penguin(t), ab_f(t)\}$

$$ground(\pi_1) = \begin{cases} fly(t) \leftarrow bird(t),\ not\ ab_f(t) \\ bird(t) \leftarrow penguin(t) \\ ab_f(t) \leftarrow penguin(t) \\ penguin(t) \leftarrow \end{cases}$$

Logic Program Examples

Example (Tweety the penguin)

$$\pi_1 = \begin{cases} fly(X) \leftarrow bird(X),\ not\ ab_f(X) \\ bird(X) \leftarrow penguin(X) \\ ab_f(X) \leftarrow penguin(X) \\ penguin(t) \leftarrow \end{cases}$$

$U_{\pi_1} = \{t\}$

$B_{\pi_1} = \{fly(t), bird(t), penguin(t), ab_f(t)\}$

$$ground(\pi_1) = \begin{cases} fly(t) \leftarrow bird(t),\ not\ ab_f(t) \\ bird(t) \leftarrow penguin(t) \\ ab_f(t) \leftarrow penguin(t) \\ penguin(t) \leftarrow \end{cases}$$

Example

Example

$$\pi_2 = \begin{cases} a \leftarrow \ not\ b \\ b \leftarrow \ not\ a \end{cases}$$

$U_{\pi_1} = \emptyset$
$B_{\pi_1} = \{a, b\}$
$ground(\pi_2) = \pi_2$

Example (Disjunctive Program)

$$\pi_3 = \begin{cases} a \ or\ b \leftarrow \ not\ c \\ b \leftarrow \ not\ a \end{cases}$$

$ground(\pi_3) = \pi_3$

Interpretation and Satisfaction

Herbrand interpretation: $I \subseteq B_\pi$

- $I \models a$ if $a \in I$ (I satisfies a);
- $I \models head(r)$ if $head(r) \cap I \neq \emptyset$;
- $I \models body(r)$ if $pos(r) \subseteq I$ and $neg(r) \cap I = \emptyset$;
- I satisfies a rule $r \in \pi$ if $I \models head(r)$ or $I \not\models body(r)$;
- I satisfies π if I satisfies all rules in π (I is a model of π).

Example

$\pi_2 = \{r_1 : a \leftarrow not\ b\ ;\ r_2 : b \leftarrow not\ a\}$ and $X = \{a\}$.
X satisfies r_1 because $head(r_1) \cap X \neq \emptyset$.
X satisfies r_2 because $X \not\models body(r_2)$.

Interpretation and Satisfaction

Herbrand interpretation: $I \subseteq B_\pi$

- $I \models a$ if $a \in I$ (I satisfies a);
- $I \models head(r)$ if $head(r) \cap I \neq \emptyset$;
- $I \models body(r)$ if $pos(r) \subseteq I$ and $neg(r) \cap I = \emptyset$;
- I satisfies a rule $r \in \pi$ if $I \models head(r)$ or $I \not\models body(r)$;
- I satisfies π if I satisfies all rules in π (I is a model of π).

Example

$\pi_2 = \{r_1 : a \leftarrow not\ b\ ;\ r_2 : b \leftarrow not\ a\}$ and $X = \{a\}$.
X satisfies r_1 because $head(r_1) \cap X \neq \emptyset$.
X satisfies r_2 because $X \not\models body(r_2)$.

Answer Set: Positive Programs

For a positive program π, $X \subseteq B_\pi$ is an answer set of π if X is a minimal model (w.r.t. $\subseteq$) of π.

Example

- $\pi_4 = \{a \leftarrow ;\ b \leftarrow a ;\ c \leftarrow d\}$.
  $X = \{a, b, c\}$ is a model of π_4 but not an answer set of π_4.
  $X = \{a, b\}$ is an answer set of π_4.
- $\pi_5 = \{a \leftarrow ;\ \leftarrow a\}$ has no answer set.
- $\pi_6 = \{a \ or \ b \leftarrow\}$ has two answer sets $\{a\}$ and $\{b\}$.

Property of programs without *or* (non-disjunctive)

- Unique if exists.
- Least fixpoint of $T_\pi(X) = \{head(r) \mid X \models body(r)\}$.

Answer Set: Positive Programs

For a positive program π, $X \subseteq B_\pi$ is an answer set of π if X is a minimal model (w.r.t. $\subseteq$) of π.

Example

- $\pi_4 = \{a \leftarrow ;\ b \leftarrow a ;\ c \leftarrow d\}$.
  $X = \{a, b, c\}$ is a model of π_4 but not an answer set of π_4.
  $X = \{a, b\}$ is an answer set of π_4.
- $\pi_5 = \{a \leftarrow ;\ \leftarrow a\}$ has no answer set.
- $\pi_6 = \{a \ or \ b \leftarrow\}$ has two answer sets $\{a\}$ and $\{b\}$.

Property of programs without *or* (non-disjunctive)

- Unique if exists.
- Least fixpoint of $T_\pi(X) = \{head(r) \mid X \models body(r)\}$.

Answer Set: Positive Programs

For a positive program π, $X \subseteq B_\pi$ is an answer set of π if X is a minimal model (w.r.t. $\subseteq$) of π.

Example

- $\pi_4 = \{a \leftarrow ;\ b \leftarrow a ;\ c \leftarrow d\}$.
  $X = \{a, b, c\}$ is a model of π_4 but not an answer set of π_4.
  $X = \{a, b\}$ is an answer set of π_4.
- $\pi_5 = \{a \leftarrow ;\ \leftarrow a\}$ has no answer set.
- $\pi_6 = \{a \ or \ b \leftarrow\}$ has two answer sets $\{a\}$ and $\{b\}$.

Property of programs without *or* (non-disjunctive)

- Unique if exists.
- Least fixpoint of $T_\pi(X) = \{head(r) \mid X \models body(r)\}$.

Answer Set: General Programs

For a program π and $X \subseteq B_\pi$, π^X is obtained from π by

1. Deleting from π any rule r such that $neg(r) \cap X \neq \emptyset$.
2. Removing all of the naf-atoms from the remaining rules.

Answer Set

X is an answer set of π if X is the answer set of π^X.

Example

$\pi_2 = \{a \leftarrow not\ b\ ; b \leftarrow not\ a\}$

$X_1 = \{a\}$ $\quad \pi_2^{X_1} = \{a \leftarrow\}$ $\quad X_1 = M_{\pi_{X_1}}$ $\quad X_1$: answer set of π_2.

$X_2 = \{b\}$ $\quad \pi_2^{X_2} = \{b \leftarrow\}$ $\quad X_2 = M_{\pi_{X_2}}$ $\quad X_2$: answer set of π_2.

$X_3 = \{a, b\}$ $\quad \pi_2^{X_3} = \emptyset$ $\quad X_3 \neq M_{\pi_{X_3}}$ $\quad X_3$: not answer set of π_2.

Answer Set: General Programs

For a program π and $X \subseteq B_\pi$, π^X is obtained from π by

1. Deleting from π any rule r such that $neg(r) \cap X \neq \emptyset$.
2. Removing all of the naf-atoms from the remaining rules.

Answer Set

X is an answer set of π if X is the answer set of π^X.

Example

$\pi_2 = \{a \leftarrow not\ b\ ; b \leftarrow not\ a\}$

$X_1 = \{a\}$ $\quad \pi_2^{X_1} = \{a \leftarrow\}$ $\quad X_1 = M_{\pi_{X_1}}$ $\quad X_1$: answer set of π_2.

$X_2 = \{b\}$ $\quad \pi_2^{X_2} = \{b \leftarrow\}$ $\quad X_2 = M_{\pi_{X_2}}$ $\quad X_2$: answer set of π_2.

$X_3 = \{a, b\}$ $\quad \pi_2^{X_3} = \emptyset$ $\quad X_3 \neq M_{\pi_{X_3}}$ $\quad X_3$: not answer set of π_2.

Answer Set: General Programs

For a program π and $X \subseteq B_\pi$, π^X is obtained from π by

1. Deleting from π any rule r such that $neg(r) \cap X \neq \emptyset$.
2. Removing all of the naf-atoms from the remaining rules.

Answer Set

X is an answer set of π if X is the answer set of π^X.

Example

$\pi_2 = \{a \leftarrow not\ b\ ; b \leftarrow not\ a\}$

$X_1 = \{a\}$ $\quad \pi_2^{X_1} = \{a \leftarrow\}$ $\quad X_1 = M_{\pi_{X_1}}$ $\quad X_1$: answer set of π_2.

$X_2 = \{b\}$ $\quad \pi_2^{X_2} = \{b \leftarrow\}$ $\quad X_2 = M_{\pi_{X_2}}$ $\quad X_2$: answer set of π_2.

$X_3 = \{a, b\}$ $\quad \pi_2^{X_3} = \emptyset$ $\quad X_3 \neq M_{\pi_{X_3}}$ $\quad X_3$: not answer set of π_2.

Answer Set: General Programs

For a program π and $X \subseteq B_\pi$, π^X is obtained from π by

1. Deleting from π any rule r such that $neg(r) \cap X \neq \emptyset$.
2. Removing all of the naf-atoms from the remaining rules.

Answer Set

X is an answer set of π if X is the answer set of π^X.

Example

$\pi_2 = \{a \leftarrow \ not\ b\ ; b \leftarrow \ not\ a\}$

$X_1 = \{a\}$ $\quad \pi_2^{X_1} = \{a \leftarrow\}$ $\quad X_1 = M_{\pi^{X_1}}$ $\quad X_1$: answer set of π_2.

$X_2 = \{b\}$ $\quad \pi_2^{X_2} = \{b \leftarrow\}$ $\quad X_2 = M_{\pi^{X_2}}$ $\quad X_2$: answer set of π_2.

$X_3 = \{a, b\}$ $\quad \pi_2^{X_3} = \emptyset$ $\quad X_3 \neq M_{\pi^{X_3}}$ $\quad X_3$: not answer set of π_2.

Some Complexity Results

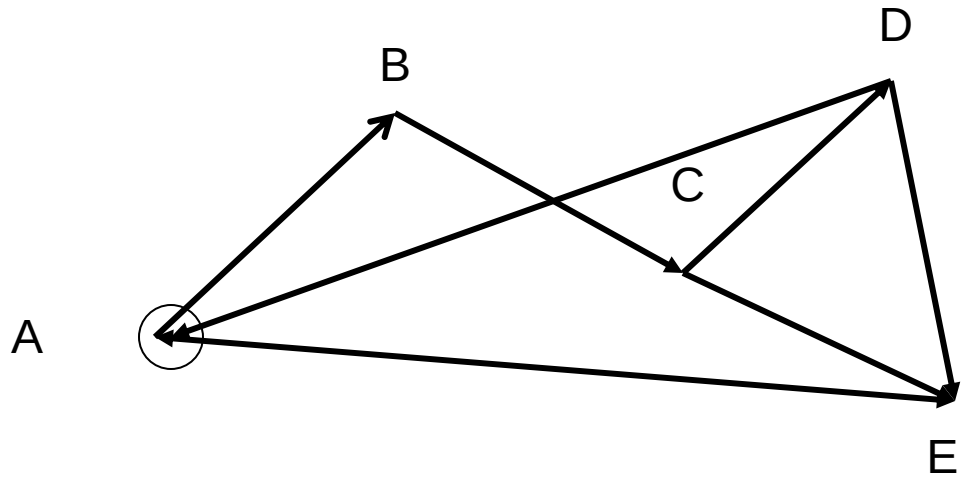
- Answer set checking: Decide whether X is an answer set
  - *Polynomial with no disjunction, Co-NP with disjunction*
- Answer set existence: Decide whether there is an answer set
  - *NP*
- Brave/Credulous reasoning: Decide whether there is an answer set containing a specific atom
  - *NP*
- Cautious/Skeptical reasoning: Decide whether a specific atom is in all the answer sets
  - *Co-NP*

Answer Set Programming

- Solving computational problems by reducing them to computing answers sets of logic programs is called answer set programming
- Useful for problems such as
  - Space shuttle and computer configuration
  - Solving puzzles and games (e.g., Sudoku)
- We will illustrate the process using an example

Computing Hamiltonian Paths using ASP

- Given a directed graph *G* and an initial vertex v_0, find a path from v_0 to v_0 which visits each vertex exactly once



A, B, C, D, E, A is a Hamiltonian path

Answer Set Programming Formulation

Represent the graph as ground facts

vertex(A)
vertex(B)
vertex(C)
……..
edge(A,B)
edge(B,C)
……
init(A)

Represent the Hamiltonian path by statements of form

in(v_0,v_1),…,*in* (v_k, v_0)

Defining Hamiltonian Path (1)

- P visits each vertex at most once

1. ← vertex(V1), vertex(V2), vertex (V),
   in (V1,V),
   in (V2, V)
   V1!=V2

2. ← vertex(V1), vertex(V2), vertex (V),
   in (V,V1),
   in (V, V2)
   V1!=V2

Defining Hamiltonian Path (2)

- P visits every vertex of the graph

3. reached(V2) ← vertex(V1), vertex(V2),
   init(V1),
   in(V1, V2)

4. reached (V2) ← vertex (V1), vertex(V2),
   reached (V1),
   in(V1, V2)

Defining Hamiltonian Path (3)

- Every vertex is reached

5. ← vertex(V), naf reached (V).

Defining Hamiltonian Path (4)

- Generate the collection of candidate paths

6. in (V1, V2) ∨ ¬in (V1, V2) ← edge (V1, V2)

Answer Set Solvers

- DLV – A disjunctive datalog system
  - http://www.dbai.tuwien.ac.at/proj/dlv/
- SMODELS (the front end is lparse)
  - See http://www.tcs.hut.fi/Software/smodels
- CLASP – A conflict driven nogood learning answer set solver
  - See http://www.cs.uni-potsdam.de/clasp/

Also see Answer Set Planning Competition

http://www.cs.kuleuven.be/~dtai/events/ASP-competition/index.shtml

Suggested Readings

- Knowledge Representation, Reasoning, and the Design of Intelligent Agents, by Michael Gelfond and Yulia Kahl, Chapters 2 and 6