Slide 1

Lecture 3

- Logic programs as a subclass of default logic
- Stable models of logic programs
- Programs with variables
- Implementing logic programs with stable models

Slide 2

Interesting Subclasses of Default Logic

- Normal and semi-normal defaults: $\frac{\varphi : \gamma}{\gamma}$ / $\frac{\varphi : \psi \wedge \gamma}{\gamma}$
- Default logic and logic programs closely related.
- Extensions correspond to stable models.
- Logic program rules can be seen as a subclass of default rules.
- Logic program rules with disjunctions offer an expressive framework for knowledge representation.

Example. Strategic companies

$s(C_1) \vee s(C_2) \leftarrow \mathit{produced\_by}(P, C_1, C_2)$
$s(C_0) \leftarrow \mathit{controlled\_by}(C_0, C_1, C_2, C_3), s(C_1), s(C_2), s(C_3)$

Slide 3

- Disjunctive logic programs = clauses with consistency requirements + minimization:

  A logic program rule

  $$A_1 \vee \ldots \vee A_k \leftarrow B_1, \ldots, B_m, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_n$$

  corresponds to $\frac{: \neg C_1, \ldots, \neg C_n}{B_1 \wedge \cdots \wedge B_m \rightarrow A_1 \vee \ldots \vee A_k}$

  with each atom A minimized: $\frac{: \neg A}{\neg A}$
- For ground programs:
  (i) Each stable model of P is the set of atoms in some extension of the corresponding default theory D_P and
  (ii) the set of atoms in an extension of D_P is a stable model of P.
- An implementation of disjunctive logic programs with the stable model semantics:
  the `dlv` system `http://www.dbai.tuwien.ac.at/proj/dlv/`

Slide 4

Disjunctive vs. Normal Programs

- Deciding whether a disjunctive program has a stable model is $\mathbf{\Sigma_2^P}$-complete.
- Deciding whether a non-disjunctive program has a stable model is **NP**-complete.

  ☞ For handling problems in **NP** non-disjunctive programs are enough.
- Normal (non-disjunctive) programs enable efficient implementation methods:

  monotonic closures can be implemented efficiently.

Slide 5

Normal rules offer a natural framework for many applications:

Example. Rules with exceptions:

$fly(X) \leftarrow bird(X), not\ exc\_bird(X)$
$exc\_bird(X) \leftarrow pinguin(X)$
$exc\_bird(X) \leftarrow no\_wings(X)$
$exc\_bird(X) \leftarrow bird(X), footinconcrete(X)$
...

Example. Frame axioms:

$on(X, Y, T2) \leftarrow nextstate(T2, T1), on(X, Y, T1), not\ moving(X, T1)$
$moving(X, T) \leftarrow moveop(X, Y, T)$
...

Example. Configuration

$disk(X) \leftarrow pc(X)$
$disktype(X, ide) \leftarrow not\ disktype(X, scsi), disk(X)$
$disktype(X, scsi) \leftarrow not\ disktype(X, ide), disk(X)$
$diskcontroller(X, scsi) \leftarrow disktype(X, scsi)$

Slide 6

Stable Models of Normal Programs

Ground programs

- A set of ground atoms is a stable model of a program P iff S is the minimal model of the set of Horn clauses

$$P^S = \{B_1 \wedge \ldots \wedge B_m \rightarrow A \mid \ A \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n \in P, \{C_1, \ldots, C_n\} \cap S = \emptyset\}$$

- A minimal model of a set of Horn clauses $B_1 \wedge \ldots \wedge B_m \rightarrow A$ coincides with the closure of the corresponding rules $\frac{B_1, \ldots, B_m}{A}$ (which we often write as $A \leftarrow B_1, \ldots, B_m$)
- We denote the closure of rules R by $\mathrm{Cl}_R(W)$ (= the minimal model of the corresponding Horn clauses). Note: $\mathrm{Cl}_R(W)$ is a set of atoms (not closed under CL).

Slide 7

- Using this notation: A set of atoms S is a stable model of P iff $S = \mathrm{Cl}_{P^S}(\emptyset)$ where

$$P^S = \{A \leftarrow B_1, \ldots, B_m \mid \ A \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n \in P, \{C_1, \ldots, C_n\} \cap S = \emptyset\}$$

Example. Consider P: $p \leftarrow p$
$q \leftarrow not\ p$

- $S_1 = \{q\}$ is a stable model of P as P^{S_1}: $p \leftarrow p$
$q \leftarrow$
and $\mathrm{Cl}_{P^{S_1}}(\emptyset) = \{q\}$.
- $S_2 = \{p\}$ is not a stable model of P as P^{S_2}: $p \leftarrow p$ and $\mathrm{Cl}_{P^{S_2}}(\emptyset) = \{\}$.

Slide 8

Full set for Normal Programs

- We use the "partial" representation for a full set (a set of not-literals).
- $\mathrm{NA}(P)$: the set of atoms appearing in the scope of *not* in P. **Example.** $\mathrm{NA}(\{p \leftarrow q, not\ r, not\ p\}) = \{r, p\}$
- A set FS of not-literals is P-full iff for all $A \in \mathrm{NA}(P)$

$$\mathrm{not}(A) \in FS \quad \text{iff} \quad A \notin \mathrm{Cl}_{\mathrm{Mon}(P,FS)}(\emptyset)$$
$$A \in FS \quad \text{iff} \quad A \in \mathrm{Cl}_{\mathrm{Mon}(P,FS)}(\emptyset)$$

where $\mathrm{Mon}(P, FS) = \{\ A \leftarrow B_1, \ldots, B_m \mid A \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n \in P, \{not\ C_1, \ldots, not\ C_n\} \subseteq FS\}$

Slide 9

Full sets capture stable models in the following way:

1. For every P-full FS, $\mathrm{Cl}_{\mathrm{Mon}(P,FS)}(\emptyset)$ is a stable model of P.
2. For every stable model M,
   $FS = \{\mathit{not}\ A \mid A \in \mathrm{NA}(P) - M\} \cup \{A \mid A \in \mathrm{NA}(P) \cap M\}$
   is the unique corresponding full set s.t. $M = \mathrm{Cl}_{\mathrm{Mon}(P,F)}(\emptyset)$.

Example. P: $p \leftarrow \mathit{not}\ q, r$ $\quad\quad \mathrm{NA}(P) = \{q, p\}$.

$q \leftarrow \mathit{not}\ p, r$

$r \leftarrow \mathit{not}\ q$

- $FS = \{\mathit{not}\ q, p\}$ is P-full as $\mathrm{Mon}(P, FS)$: $p \leftarrow r$
  $r \leftarrow$
  $\mathrm{Cl}_{\mathrm{Mon}(P,FS)}(\emptyset) = \{p, r\}$.
- $FS' = \{\mathit{not}\ p, q\}$ is not P-full since $\mathrm{Mon}(P, FS')$: $q \leftarrow r$
  $\mathrm{Cl}_{\mathrm{Mon}(P,FS')}(\emptyset) = \emptyset$.

Slide 10

Programs with Variables

- Variables as assumed to universally quantified and the semantics is based on **Herbrand models**.

  Example. $r(X,Y) \leftarrow \mathit{edge}(X,Y)$

  $r(X,Y) \leftarrow \mathit{edge}(X,Z), r(Z,Y)$
- The *Herbrand universe* of a program is the set of ground terms constructed from the constants and functions in the program.
- The *Herbrand base* of a program is the set of atomic ground formulae built from the Herbrand universe and the predicate symbols of the program.
- A *Herbrand model* is a subset of the Herbrand base.

Slide 11

- Notice that the Herbrand base of a finite program is finite iff the program contains no function symbols.
- The ground instantiation P_H of P is the set of ground rules obtained from the rules in P by replacing variables in the rules by ground terms given in the Herbrand universe of P.
- Stable models are Herbrand models of the program, i.e., subsets of its Herbrand base, defined using the ground instantiation:

Definition. For a non-ground program P, the stable models of P are those of the ground instantiation P_H of P.

Slide 12

Example. Herbrand universe of P: $\{a, b, c\}$

Herbrand base: $\{d_1(a), d_1(b), d_1(c), d_2(a), d_2(b), \ldots, s(b,c,c), s(c,c,c)\}$.

P :	P_H :
$d_1(a) \leftarrow$	$d_1(a) \leftarrow$
$d_1(b) \leftarrow$	$d_1(b) \leftarrow$
$d_1(c) \leftarrow$	$d_1(c) \leftarrow$
$s(X,Y,Z) \leftarrow\ d_1(X), d_2(Y),$	$s(a,a,a) \leftarrow d_1(a), d_2(a), d_3(a), \mathit{not}\ d_3(a)$
$\quad d_3(Z), \mathit{not}\ d_3(X)$	$s(a,a,b) \leftarrow d_1(a), d_2(a), d_3(b), \mathit{not}\ d_3(a)$
$s(X,X,X) \leftarrow d_1(X)$	$\vdots$
	$s(c,c,c) \leftarrow d_1(c)$

A unique stable model: $\{d_1(a), d_1(b), d_1(c), s(a,a,a), s(b,b,b), s(c,c,c)\}$.

Slide 13

Implementations of Stable Model Semantics

- DeReS (Marek and Truszczynski)
  http://www.cs.engr.uky.edu/~lpnmr/DeReS.html
- dlv (Eiter, Leone et al.)
  http://www.dbai.tuwien.ac.at/proj/dlv/
- Smodels (Niemelä and Simons)
  http://www.tcs.hut.fi/Software/smodels/
- SLG (Chen and Warren)
- Mixed integer programming (Bell, Nerode, Ng, Subrahmanian)
- WFS + branch and bound = stable models (Subrahmanian, Nau, and Vago)

Slide 14

Implementing Stable Models

All current methods employ a **two phase approach**:

- **1. Grounding**: find a set P_g of ground instance of P such that P_g has exactly the same stable models as P.
- **2. Stable model search**: solve the decision problems using the ground program P_g.

Observations:

- Grounding needs to be done intelligently as the Herbrand instantiation of a program is typically huge!
- Stable model search is still **NP**-hard: highly efficient methods should be used.

  We employ the framework developed for default logic and further optimize it for logic programs.