Slide 1

## Lecture 4

- Implementing ground logic programs
- Handling programs with variables
- Architecture of the *Smodels* System

Slide 2

## Implementing Ground Rules

- Our approach is based on the decision method for default logic.
- Polynomial space complexity
- Similarities to Davis-Putnam procedure
- Efficient search space pruning (using tight upper and lower bounds)



Slide 3

**function smodels**$(P, FS, \varphi)$
$FS' :=$ **expand**$(P, FS)$;
**if conflict**$(P, FS')$ returns true **then** return false
**else if** all negative literals in $P$ are covered by $FS'$ **then**
    return **test**$(P, FS', \varphi)$
**else**
    $\chi :=$ **choose**$(P, FS')$;
    **if smodels**$(P, FS' \cup \{\chi\}, \varphi)$ returns true **then** return true
    **else** return **smodels**$(P, FS' \cup \{\mathrm{compl}(\chi)\}, \varphi)$
    **end if**
**end if**

Slide 4

**expand** and **conflict**:

- Based on approximating stable models agreeing with $FS$ using lower and upper bounds.
- Efficient implementation techniques for approximations:
  (i) Ideas from Fitting semantics.
  (ii) Backward propagation under certain conditions.
  (iii) Dowling-Gallier type data structures for the deductive closures (linear time complexity).

☞ Approximation method generalizes the well-founded (WF) semantics.

WF model computed typically in linear time (quadratic in the worst case).

Slide 5

**Implementations of the bounds**

Tight bounds are obtained using a notion of **potentially applicable** rules $P/FS$ in a program $P$ under a set of assumptions $FS$ containing the rules with **bodies not false** in $FS$:

$$P/FS = \{\ A \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n \mid \{not\ B_1, \ldots, not\ B_m, C_1, \ldots, C_n\} \cap FS = \emptyset\}$$

**Example.**

| $P$ : | $P/\{not\ s, t\}$ : |
|---|---|
| $p \leftarrow q$ | $p \leftarrow q$ |
| $q \leftarrow p$ | $q \leftarrow p$ |
| $p \leftarrow s, not\ r$ | $t \leftarrow not\ r$ |
| $s \leftarrow not\ t, not\ p$ | |
| $r \leftarrow not\ t, not\ q$ | |
| $t \leftarrow not\ r$ | |

Slide 6

- $\mathbf{LB}(P, FS)$: the smallest set $FS'$ including $FS$ and closed under the rules
  1. If for $H \leftarrow L_1, \ldots, L_n \in P$, $\{L_1, \ldots, L_n\} \subseteq FS'$, then $H \in FS'$
  2. If an atom $H$ is not a head of any rule in $P/FS'$, then $\mathrm{not}(H) \in FS'$.
  3. If $H \in FS'$ is the head of only one rule $H \leftarrow L_1, \ldots, L_n$ in $P/FS'$, then $\{L_1, \ldots, L_n\} \subseteq FS'$.
  4. If $\mathrm{not}(H) \in FS'$ and $H$ is the head of a rule $H \leftarrow L_1, \ldots, L_n$ in $P/FS'$ with $\{L_1, L_{i-i}, L_{i+1}, \ldots, L_n\} \subseteq FS'$, then $\mathrm{compl}(L_i) \in FS'$.
- $\mathbf{UB}(P, FS) = \mathrm{Cl}_{P'}(\emptyset)$ where $P'$ is obtained by removing all negative literals from $P/FS$.



Slide 7

**Example.**

| $P$ : | **expand**$(P, \{p\})$ : | | **expand**$(P, \{not\ s\})$ : | |
|---|---|---|---|---|
| $p \leftarrow q$ | $p$ | | $not\ s$ | |
| $q \leftarrow p$ | $q$ | $LB1$ | $not\ p$ | $UB$ |
| $p \leftarrow s, not\ r$ | $not\ s$ | $LB2$ | $not\ q$ | $UB$ |
| $s \leftarrow not\ t, not\ p$ | $not\ r$ | $LB2$ | $t$ | $LB4$ |
| $r \leftarrow not\ t, not\ q$ | $t$ | $LB1$ | $not\ r$ | $LB3$ |
| $t \leftarrow not\ r$ | $not\ p$ | $UB$ | | |
| | $not\ q$ | $UB$ | | |

Slide 8

**Observations:**

- Rules 1–2: Fitting semantics
  Rules 3–4: backward propagation
- Upper and lower bound implementable in linear time (Dowling-Gallier type data structures)
- If there are no initial assumptions ($FS = \emptyset$), then **expand** computes the WF model as the first approximation.

**Slide 9**

**Search heuristics (choose)**:

➤ Typical heuristics:
- Start from the lowest strongly connected component in the dependency graph of the program.
- Maximum Occurrences in rules of Minimum size (Mom's) heuristics.

Advantages: fast to implement
Disadvantages: static, syntax dependent

➤ Our approach is based on lookahead
- Prefers choice points with a small remaining search space. by employing the approximation technique.
- Immediate conflicts are detected.
- Stability of the performance is enhanced.

**Slide 10**

**Implementing choose**

➤ Lookahead: test all choices before committing
For each atom $A$ appearing negatively in $P$ compute
$FS_1 =$ **expand**$(P, FS \cup \{A\})$ and
$FS_2 =$ **expand**$(P, FS \cup \{\mathrm{not}(A)\})$

➤ Immediate conflicts are detected:
If $FS_1$ is conflicting, add $\mathrm{not}(A)$ to $FS$ and
if $FS_2$ is conflicting, add $A$ to $FS$.

➤ Lookahead can be exploited for the search heuristics as $FS_1$ and $FS_2$ give an estimate $\mathrm{E}(A)$ of the remaining search if $A$ is chosen:

$$\mathrm{E}(A) = 2^{n_1} + 2^{n_2}$$

where $n_1$ ($n_2$) is the number of atoms in $\mathrm{NA}(P)$ not covered by $FS_1$ ($FS_2$).

➤ Heuristics: choose an atom $A$ with the smallest $\mathrm{E}(A)$.



**Slide 11**

**test**:

➤ Trivial **test** function (returning true) is often sufficient.

➤ Finding a stable model:
**smodels**$(P, \emptyset, \varphi)$ returns true iff there is a stable model of $P$.

➤ Credulous reasoning, i.e., deciding whether there is a stable model agreeing with a set of literals $S$:
**smodels**$(P, S, \varphi)$ returns true iff there is a stable model agreeing with $S$.

➤ Sceptical reasoning, i.e., deciding whether a literal $L$ is in every stable model of $P$:
**smodels**$(P, \{\mathrm{compl}(L)\}, \varphi)$ returns false iff $L$ is in every stable model of $P$.

**Slide 12**

**Example.** (a) All stable models of $P$: let **test** return false and run **smodels**$(P, \{\}, \varphi)$.
(b) Decide whether $s$ is in every stable model of $P$: let **test** return true and run **smodels**$(P, \{not\ s\}, \varphi)$.

| | | | | |
|---|---|---|---|---|
| $P:$ | | **smodels**$(P, \{\}, \varphi):$ | | **smodels**$(P,$ |
| $p \leftarrow not\ q, not\ u$ | | $1.\ not\ u$ | | $\{not\ s\}, \varphi):$ |
| $q \leftarrow not\ p, not\ u$ | | $\swarrow\searrow$ | | $1.\ not\ s$ |
| $r \leftarrow p$ | | $2.\ not\ t$ | $3.\ t$ | $2.\ not\ u$ |
| $s \leftarrow not\ q$ | | $4.\ s$ | $11.\ not\ s$ | $3.\ t$ |
| $s \leftarrow not\ r, q$ | $\swarrow\searrow$ | | $12.\ q$ | $4.\ q$ |
| $t \leftarrow not\ s$ | $5.\ not\ r$ | $6.\ r$ | $13.\ not\ p$ | $5.\ not\ p$ |
| $s \leftarrow not\ t$ | $7.\ not\ p$ | $9.\ p$ | $14.\ not\ r$ | $6.\ not\ r$ |
| | $8.\ q$ | $10.\ not\ q$ | $15.\ not\ q$ | $7.\ s$ |
| | | | $\times$ | $\times$ |

Slide 13

## Related Work

**smodels**: (i) Overall linear space complexity.
(ii) Efficient pruning of the search space.

- Mixed integer programming (Bell, Nerode, Ng, Subrahmanian)
  Exponential space requirements.
- WFS + branch and bound = stable models (Subrahmanian, Nau, and Vago)
  Search space: all atoms in the program
  Exponential space requirements
- SLG (Chen and Warren):
  Uses WFS once and then assume-and-reduce

Slide 14

## Handling Rules with Variables

- Handled using a compilation technique: a program with variables is transformed to a ground program preserving stable models.
- Top-down goal-oriented approach:
  For a query, use SLG resolution to get a ground residual program.
  Efficient implementation available: *XSB*
  http://www.cs.sunysb.edu/~sbprolog/
  For termination: bounded-term size property.
  Not sound w.r.t. stable models semantics.
- Bottom-up approach:
  Used in *dlv* and *Smodels* .
  Sound w.r.t. stable models semantics.
  For termination: no function symbols.
  Programs range-restricted (to avoid floundering problems).



Slide 15

## Rules with Variables in Smodels

- Function symbols not allowed.
- A rule with variables is compiled to a set of ground rules.
- For efficiency: domain-restricted programs
  (i) Each variable in a rule must appear in a positive domain predicate in the rule.
  (ii) Domain predicates: non-recursive definitions (views).
  **Example.** $e2(X,Y) \leftarrow e(X,Z), e(Z,Y), \mathit{not}\ e(Z,Z)$
- For domain-restricted programs, compilation can be implemented locally and efficiently.
- Easy to incorporate built-in functions and predicates, e.g. arithmetic (no floundering problems).

Slide 16

**Example.** Queens.

$q(X,Y) \leftarrow d(X), d(Y), \mathit{not}\ \mathit{negq}(X,Y)$

$\mathit{negq}(X,Y) \leftarrow d(X), d(Y), \mathit{not}\ q(X,Y)$

$\leftarrow d(X), d(Y), d(X'), q(X,Y), q(X',Y), X' \neq X$

$\leftarrow d(X), d(Y), d(Y'), q(X,Y), q(X,Y'), Y' \neq Y$

$\leftarrow d(X), d(Y), d(X'), d(Y'), q(X,Y), q(X',Y'), X \neq X', Y \neq Y',$
$\quad abs(X - X') = abs(Y - Y')$

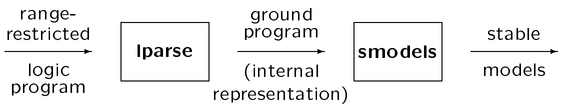
$\leftarrow d(X), \mathit{not}\ \mathit{hasq}(X)$

$\mathit{hasq}(X) \leftarrow d(X), d(Y), q(X,Y)$

$d(1..n) \leftarrow$

Slide 17

## Overall Architecture of the Smodels System




- **lparse**: parses the input and compiles rules with variables compiled to ground rules such that stable models are preserved.
- **smodels**: implements the stable model semantics for ground programs (and as a by-product also the well-founded semantics).

Slide 18

## Functionality

- Domain-restricted function-free programs with built-ins.
- Complete method: can find all/a given number of stable models.
- Supports query-evaluation by *focused model search*.
- Low space complexity.
- Coping with large programs (100k+ ground rules).
- Stable behavior (e.g., ordering and syntactic transformations do not cause major changes in performance).



Slide 19

## Availability of Smodels

- C++ implementation publicly available
  `http://www.tcs.hut.fi/smodels/`
  Source code, documentation, test cases, papers, . . .
  Developed under Linux
  Easily portable to any Unix system having a C++ compiler (+ make and tar).
- Two ways of using the *Smodels* system:
  (i) Stand-alone
  (ii) Embedded through API (as a C++ library)