Lecture 8: Implementation Techniques

Outline

1. Dowling-Gallier algorithm
2. Full sets
3. Search method for stable models
4. Implementing approximations
5. Branch&bound algorithm

1. DOWLING-GALLIER ALGORITHM

- W. Dowling and J. Gallier [1984] presented their method originally for testing the satisfiability of sets of Horn clauses.
- Due to close interconnection of Horn clauses with rules and constraints, we present the method for positive programs directly.
- The iteration sequence $\mathrm{T}_P \uparrow 0$, $\mathrm{T}_P \uparrow 1$, ... provides a basic method for computing $\mathrm{LM}(P) = \mathrm{lfp}(\mathrm{T}_P)$ for a positive program P.
- The least fixpoint, for which $\mathrm{T}_P \uparrow n = \mathrm{T}_P \uparrow n-1$ holds, is reached in at most $n = |\mathrm{Hb}(P)|+1$ applications of T_P.
- The time complexity is of $O(|\mathrm{Hb}(P)| \times ||P||)$ as for any $M \subseteq \mathrm{Hb}(P)$, the time required to compute $\mathrm{T}_P(M)$ depends linearly on $||P||$.

Data structures

- The iteration sequence of the operator T_P can be implemented more efficiently using certain precomputed data structures.
- The following data structures are used for a positive program P:
  1. An *array* occurs$[a]$ of sets of rules indexed by atoms $a \in \mathrm{Hb}(P)$ and precomputed as $\{r \mid r = h \leftarrow B \in P$ and $a \in B\}$ for each a.
  2. An *array* count$[r]$ of integers indexed by rules $r \in P$. The initial value for a rule $r = h \leftarrow B$ is the number $|B|$ of body atoms.
  3. A *set* of atoms $M \subseteq \mathrm{Hb}(P)$ initialized as $\mathrm{T}_P \uparrow 1 = \mathrm{T}_P(\emptyset)$.
- The execution of the function LeastModel(M) given on the next slide will gradually extend M to $\mathrm{LM}(P)$.

Algorithm for Computing $\mathrm{LM}(P)$

```
function LeastModel(M : atom set): atom set;
var Q: atom set; a: atom; r: rule;
  Q := M;
  while Q ≠ ∅ do
    a := pick(Q);
    Q := Q \ {a};
    for r in occurs[a] do
      count[r] := count[r] − 1;
      if count[r] = 0 and Head(r) ∉ M then do
        Q := Q ∪ {Head(r)}; M := M ∪ {Head(r)};
      done
    done
  done
  LeastModel := M;
```

Example. Consider a positive program P having the following rules:

$r_0 : c.\quad r_1 : d.\quad r_2 : e \leftarrow c, d.\quad r_3 : f \leftarrow c.\quad r_4 : g \leftarrow g, f.\quad r_5 : h \leftarrow d, f.$

The algorithm computes $\mathrm{LM}(P) = \{c,d,e,f,h\}$ as follows:

1. $M := \{c,d\}$	11. $a := d$	21. $r := r_4$
2. $Q := \{c,d\}$	12. $Q := \{f\}$	22. count$[r_4] := 1$
3. $a := c$	13. $r := r_2$	23. $r := r_5$
4. $Q := \{d\}$	14. count$[r_2] := 0$	24. count$[r_5] := 0$
5. $r := r_2$	15. $Q := \{f,e\}$	25. $Q := \{e,h\}$
6. count$[r_2] := 1$	16. $M := \{c,d,e,f\}$	26. $M := \{c,d,e,f,h\}$
7. $r := r_3$	17. $r := r_5$	27. $a := e$
8. count$[r_3] := 0$	18. count$[r_5] := 1$	28. $Q := \{h\}$
9. $Q := \{d,f\}$	19. $a := f$	29. $a := h$
10. $M := \{c,d,f\}$	20. $Q := \{e\}$	30. $Q := \emptyset$

Observations about the Algorithm

- The operation of the **while**-loop is governed by *invariants*
  1. $Q \subseteq M \subseteq \mathrm{LM}(P)$.
  2. $\forall r = a \leftarrow B \in P$: count$[r] = |B \setminus (M \setminus Q)|$.
  3. $\forall a \in \mathrm{Hb}(P)$: $(a \in M \iff \exists r = a \leftarrow B \in P$: count$[r] = 0)$.
- The progress of the **while**-loop is witnessed by $|\mathrm{LM}(P) \setminus M| + |Q|$.
- The cumulative work done by the **for**-loop is proportional to the number of positive body literals in the rules of P.
- Assuming constant costs for set operations and arithmetics, the algorithm runs in time linear with respect to $||P||$.
- The implementation of Q determines the way how computation proceeds: e.g., breadth-first (LIFO) or depth-first (FIFO).

2. FULL SETS

- The set of atoms $a \in \mathrm{Hb}(P)$ that appear in the negative body literals of a normal program P is denoted by $\mathrm{NBA}(P)$.
- The members of $\mathrm{NBA}(P)$ affect the reduct P^M play a major role when the stable models M of a normal program P are determined.
- The stable models $M \in \mathrm{SM}(P)$ can be characterized in terms of sets of *negative default literals* $\sim a$ based on atoms $a \in \mathrm{NBA}(P)$.
- The *least model* associated with a normal logic program P and a set of negative default literals F is $\mathrm{LM}(P_F)$ where

$$P_F = \{a \leftarrow B \mid a \leftarrow B, \sim C \in P \text{ and } \sim C \subseteq F\}.$$

Definition. A set F of negative default literals is P-*full* if and only if for all $a \in \mathrm{NBA}(P)$, the literal $\sim a \in F \iff a \notin \mathrm{LM}(P_F)$.

Properties of Full Sets

Theorem. Let P be a normal program and F a subset of $\sim\mathrm{NBA}(P)$.

1. If F is P-full, then $M = \mathrm{LM}(P_F) \in \mathrm{SM}(P)$.
2. If $M \in \mathrm{SM}(P)$, then $F = \{\sim a \mid a \in \mathrm{NBA}(P) \setminus M\}$ is P-full and $M = \mathrm{LM}(P_F)$.

Example. The set of atoms $\mathrm{NBA}(P) = \{a,b,d\}$ for a normal program

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

1. The set of literals $F_1 = \{\sim b, \sim d\}$ is P-full, since $P_{F_1} = \{a \leftarrow c.\ \ c.\ \}$ and $\mathrm{LM}(P_{F_1}) = \{a,c\}$. Thus $M = \{a,c\}$ is stable.
2. But e.g. $F_2 = \{\sim b\}$ is not P-full because $P_{F_2} = \{a \leftarrow c.\ \}$, $\mathrm{LM}(P_{F_2}) = \emptyset$, and, for instance, $\sim a \notin F_2$.

3. SEARCH METHOD FOR STABLE MODELS

- The goal is to compute—as efficiently as possible—one or several stable models for a normal program P given as input.
- The characterization of stable models based on full sets suggests that the search space essentially consists of subsets of $\mathrm{NBA}(P)$.
- Following the general *branch&bound* search strategy, we gradually build a set L of default literals that constrains stable models being computed and try to prune the search space.
  1. Assumptions about models are made one by one.
  2. At each point of the search space, stable models that satisfy all the assumptions introduced so far (the set L) are approximated.
  3. If a conflict is found, the search backtracks, and the search for other models takes place similarly, if a model is found.

Approximation Criteria

- Stable models being computed for a normal program P are specified in terms of a set of default literals L over $\mathrm{Hb}(P)$:
  1. If $a \in L$, then $a \in M$ for stable models M being computed.
  2. If $\sim a \in L$, then $a \notin M$ for stable models M being computed ($\sim a \in F$ holds for the respective full sets F).
- Such a relationship between a stable model $M \in \mathrm{SM}(P)$ and a set of default literals can be understood as a form of *compatibility*.

Example. Consider the normal logic program $P =$

$$\{a \leftarrow \sim b.\ \ b \leftarrow \sim a.\ \ c \leftarrow \sim d.\ \ d \leftarrow \sim c.\ \ e \leftarrow \sim f.\ \ f \leftarrow \sim e.\ \}.$$

Now, for instance, the set of default literals $L = \{a, \sim c\}$ is compatible with stable models $M_1 = \{a, d, e\}$ and $M_2 = \{a, d, f\}$ in $\mathrm{SM}(P)$.

Lower and Upper Bounds

- A *lower bound* $\mathrm{LB}(P,L) \supseteq L$ is a set of literals which is compatible with any stable model $M \in \mathrm{SM}(P)$ compatible with L.
- An *upper bound* $\mathrm{UB}(P,L) \subseteq \mathrm{Hb}(P)$ is a set of atoms that contains every $M \in \mathrm{SM}(P)$ compatible with L.
- An approximation $\mathrm{Expand}(P,L)$ is the *least* set of literals L' which contains L and is *closed* in the following senses:
  (i) If a default literal $l \in \mathrm{LB}(P,L')$, then $l \in L'$.
  (ii) If an atom $a \notin \mathrm{UB}(P,L')$, then $\sim a \in L'$.
- The approximation $\mathrm{Expand}(P,L)$ can be obtained by computing lower and upper bounds iteratively and applying (i) and (ii).

4. IMPLEMENTING APPROXIMATIONS

- For the sake of efficiency, it is important that the bounds $\mathrm{LB}(P,L)$ and $\mathrm{UB}(P,L)$ can be computed in linear time.
- The resulting approximation at any point of the search space L should be at least as accurate as $\mathrm{WFM}(P)$, i.e., $\mathrm{WFM}(P) \subseteq L$.
- Assumptions embodied in L should be taken fully into account.

Definition. For a normal logic program P and a set of default literals L over $\mathrm{Hb}(P)$, the set of *active rules* of P given L is

$$\mathrm{ActR}(P,L) = \{a \leftarrow B, \sim C \in P \mid L \cap (\sim B \cup C) = \emptyset\}.$$

Remark. The bodies of rules in $\mathrm{ActR}(P,L)$ are not falsified by L!

Lower Bound

Definition. The lower bound $\mathrm{LB}(P,L)$ is the least set of literals L' which contains L and is closed under the following principles:

P1: If $a \leftarrow B, \sim C \in \mathrm{ActR}(P,L')$ and $B \cup \sim C \subseteq L'$, then $a \in L'$.

P2: If $b \neq a$ for every rule $a \leftarrow B, \sim C$ in $\mathrm{ActR}(P,L')$, then $\sim b \in L'$.

P3: If $a \in L'$ is the head of *exactly one* rule $a \leftarrow B, \sim C$ in $\mathrm{ActR}(P,L')$, then the body $B \cup \sim C \subseteq L'$.

P4: If $\sim a \in L'$, a rule $a \leftarrow l_1, \ldots, l_n \in \mathrm{ActR}(P,L')$, and $\{l_1, \ldots, l_{i-i}, l_{i+1}, \ldots, l_n\} \subseteq L'$, then the complement $\overline{l_i} \in L'$.

P5: If for some atom $a \in \mathrm{Hb}(P)$ both $a \in L'$ and $\sim a \in L'$, then all literals over $\mathrm{Hb}(P)$ belong to L'.

Example

Let us compute $L_1 = \mathrm{LB}(P_1, \{\sim b\})$ and $L_2 = \mathrm{LB}(P_1, \{b\})$ for P_1:

$$r_1: a \leftarrow \sim b. \quad r_2: c \leftarrow a. \quad r_3: b \leftarrow a, \sim c, \sim d. \quad r_4: d \leftarrow c, \sim e.$$

L_1		$\mathrm{ActR}(P_1, L_1)$
$\sim b$		r_1, r_2, r_3, r_4
a	P1	r_1, r_2, r_3, r_4
c	P1	r_1, r_2, r_4
$\sim e$	P2	r_1, r_2, r_4
d	P1	r_1, r_2, r_4

L_2		$\mathrm{ActR}(P_1, L_2)$
b		r_2, r_3, r_4
$a, \sim c, \sim d$	P3	r_2, r_3
c	P1	r_2
all	P5	

$\Longrightarrow$ The approximation $\mathrm{Expand}(P_1, \{\sim b\}) = \{\sim b, a, c, \sim e, d\}$ determines a stable model $M = \{a, c, d\}$. The set $\mathrm{Expand}(P_1, \{b\})$ contains all literals. Thus there is no $M \in \mathrm{SM}(P)$ such that $b \in M$.

Upper Bound

Definition. The upper bound $\mathrm{UB}(P,L) = \mathrm{LM}(\mathrm{ActR}(P,L)^{\emptyset})$ where the reduct $\mathrm{ActR}(P,L)^{\emptyset}$ is $\mathrm{ActR}(P,L)$ with all negative literals removed.

Example. Consider the following normal program P_2:

$$a \leftarrow \sim b. \quad b \leftarrow \sim a. \quad c \leftarrow \sim a. \quad d \leftarrow \sim c. \quad e \leftarrow \sim d.$$

Verify the following upper bounds for P_2:

1. $\mathrm{UB}(P_2, \emptyset) = \{a, b, c, d, e\}$.
   $\Longrightarrow$ $\mathrm{Expand}(P_2, \emptyset) = \emptyset$ because also $\mathrm{LB}(P_2, \emptyset) = \emptyset$ holds.
2. $\mathrm{UB}(P_2, \{a\}) = \mathrm{LM}(\{a \leftarrow \sim b. \ d \leftarrow \sim c. \ e \leftarrow \sim e. \}^{\emptyset}) = \{a, d, e\}$.
3. $\mathrm{UB}(P_2, \{\sim a\}) = \{a, b, c, d, e\}$.

Further Examples

Example. Let us compute approximations $\mathrm{Expand}(P_2, \{a\})$ and $\mathrm{Expand}(P_2, \{\sim a\})$ for the preceding program P_2:

$$a \leftarrow \sim b. \quad b \leftarrow \sim a. \quad c \leftarrow \sim a. \quad d \leftarrow \sim c. \quad e \leftarrow \sim d.$$

1. $\mathrm{LB}(P_2, \{a\}) = \{a, \sim b, \sim c, d, \sim e\} = \mathrm{Expand}(P_2, \{a\})$.
2. $\mathrm{LB}(P_2, \{\sim a\}) = \{\sim a, b, c, \sim d, e\} = \mathrm{Expand}(P_2, \{\sim a\})$.

Example. Let us then analyze a normal program P_3 having two rules:

$$a \leftarrow \sim b. \quad b \leftarrow b.$$

Now $\mathrm{LB}(P_3, \emptyset) = \emptyset$ but $\mathrm{UB}(P_3, \emptyset) = \{a\}$ so that $\sim b \in \mathrm{Expand}(P_3, \emptyset)$.

$\Longrightarrow$ $\mathrm{Expand}(P_3, \emptyset) = \{a, \sim b\}$ because $\mathrm{LB}(P_3, \{\sim b\}) = \{a, \sim b\}$.

Implementing Bounds

- The lower and upper bounds can be implemented as linear time algorithms that resemble the Dowling-Gallier presented above.
- The accuracy of Expand is at least as good as that of the well-founded model. In fact, we have Expand$(P,\emptyset) = \mathrm{WFM}(P)$.
- However, assumptions about stable models to compute, a set of default literals $L \not\subseteq \mathrm{WFM}(P)$, make Expand$(P,L)$ more accurate.

Example. For the normal program P_4 consisting of

$$a \leftarrow \sim b. \quad b \leftarrow \sim a. \quad a \leftarrow b.$$

we obtain Expand$(P_4,\emptyset) = \emptyset$ and Expand$(P_4,\{b\}) = \{a,b,\sim a,\sim b\}$.

But the conflict is not detected for $P_5 = P_4 \cup \{b.\ \}$: $\mathrm{WFM}(P) = \{a,b\}$.

5. BRANCH&BOUND ALGORITHM

- The purpose of the function Smodels(P,L) is to *check* whether P has a stable model M which is compatible with L.
- The underlying algorithm is based on a number of primitives:
  1. The function Expand(P,L) returns a tightened approximation of stable models that are compatible with L.
  2. The function Conflict(P,L) checks whether the approximation L obtained so far is contradictory ($\{a,\sim a\} \subseteq L$ for some a).
  3. The function Covered(P,L) checks whether the approximation L obtained so far covers all atoms of $\mathrm{NBA}(P)$, i.e., for each $a \in \mathrm{NBA}(P)$, either $a \in L$ or $\sim a \in L$.
  4. The function Choose(P,L) implements the search heuristics, i.e., it picks the next *literal* for branching.

The `smodels` Algorithm

function Smodels(P,L): boolean;
var A: literal set; l: literal;
  $A :=$ Expand(P,L);
  if Conflict(P,A) **then return** $\bot$;
  if Covered(P,A) **then return** $\top$;
  $l :=$ Choose(P,A);
  if Smodels$(P,A \cup \{l\})$ **then**
    return $\top$;
  else return Smodels$(P,A \cup \{\bar{l}\})$;

Remarks. Recall the complements $\overline{a} = \sim a$ and $\overline{\sim a} = a$.

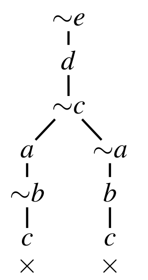
The sets of literals L and A (the new approximation) can be represented in space linear with respect to $|\mathrm{Hb}(P)| \leq ||P||$.

Example: Cautious Reasoning

Let us show $\langle P,e \rangle \in$ CAUTIOUS for the following normal program P:

$$a \leftarrow \sim b. \quad b \leftarrow \sim a. \quad c \leftarrow a, \sim b. \quad c \leftarrow b, \sim a. \quad d \leftarrow \sim c. \quad e \leftarrow \sim d.$$

The *non-existence* of counter-examples is showed using $L = \{\sim e\}$:



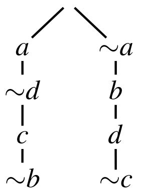
Remark. Only one choice was necessary although $|\mathrm{NBA}(P)| = 4$.

Example: Search for Stable Models

Let us search the stable models of the following normal program P:

$$a \leftarrow c, \sim b. \quad b \leftarrow \sim a. \quad c \leftarrow \sim d. \quad d \leftarrow \sim a.$$

In the beginning, the set of assumptions $L = \emptyset$ for this task:



$\Longrightarrow$ The sets $F_1 = \{\sim b, \sim d\}$ and $F_2 = \{\sim a, \sim c\}$ are P-full—giving rise to stable models $M_1 = \{a, c\}$ and $M_2 = \{b, d\}$ in $\mathrm{SM}(P)$.

Search Heuristics

- In the *look ahead* search strategy, refined approximations

  $$L_1 = \mathrm{Expand}(P, L \cup \{a\}) \text{ and } L_2 = \mathrm{Expand}(P, L \cup \{\sim a\})$$

  are computed for each $a \in \mathrm{NBA}(P)$ *not covered* by L.

- These can be used to recursively refine the approximation L:

  1. If L_1 contains $\sim a$ (a conflict), then $\sim a$ is added to L.
  2. If L_2 contains a (a conflict), then a is added to L.

  The search at L can be stopped if both a and $\sim a$ are added.

- If both L_1 and L_2 are consistent, they provide an estimate of the size of the remaining search space.

- The search heuristics of the `smodels` solver selects for branching a literal expected to create the smallest search space.

OBJECTIVES

- You understand the operation of the Dowling-Gallier-algorithm and are able to simulate it for a given normal program.
- You are familiar with the branch&bound algorithm that underlies the `smodels` system.
- Given a smallish normal program, you are able to determine its stable models and to reason about them (e.g., cautiously).
- You know at least some approximation principles and how to exploit them efficiently in the computation of stable models.

TIME TO PONDER

Recall the technique for removing a choice rule

$$\{a_1, \ldots, a_h\} \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m$$

by translating it into $2h+1$ rules

$$\begin{array}{lll} a_1 \leftarrow b, \sim \overline{a_1}. & \ldots & a_h \leftarrow b, \sim \overline{a_h}. \\ \overline{a_1} \leftarrow \sim a_1. & \ldots & \overline{a_h} \leftarrow \sim a_h. \\ b \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m. & & \end{array}$$

Given this interconnection, consider the applicability of the principles P1–P4 involved in the lower bound $\mathrm{LB}(P, L)$ to choice rules.