Lecture 11: Relationship with Propositional Logic

Outline

- Expressive power
- Clark's completion
- Loop formulas
- Characterization of stable models
- Tight programs

1. EXPRESSIVE POWER

- In the sequel, we concentrate on the class of normal programs although many results can be generalized for `smodels` programs.
- It can be formally proved that normal programs under stable model semantics are strictly more expressive than propositional theories.
- The proof is based on the existence of translations of specific kinds between normal programs and propositional theories.
- In this respect, the basic criteria imposed on a translation Tr are:

  1. *Faithfulness*: $T \equiv_{\mathrm{v}} \mathrm{Tr}(T)$.
  2. *Modularity*: $\mathrm{Tr}(T_1 \cup T_2) \equiv_{\mathrm{v}} \mathrm{Tr}(T_1) \cup \mathrm{Tr}(T_2)$.

  Here we assume that $\mathrm{Hb}_{\mathrm{v}}(T) = \mathrm{Hb}(T) \subseteq \mathrm{Hb}(\mathrm{Tr}(T))$, i.e., Tr may introduce new atoms which remain invisible in $\mathrm{Tr}(T)$.

Modular Representation for Clauses

- There is a *faithful* and *modular* translation Tr_{N} from sets of clauses into normal programs (involving constraints).

Definition. An individual clause $A \vee \neg B$ is translated into

$$\mathrm{Tr}_{\mathrm{N}}(A \vee \neg B) = \{a \leftarrow \sim\overline{a}.\ \ \overline{a} \leftarrow \sim a.\ \mid a \in A \cup B\} \cup \{\bot \leftarrow \sim A, \sim\overline{B}\}$$

and $\mathrm{Tr}_{\mathrm{N}}(S) = \bigcup\{\mathrm{Tr}_{\mathrm{N}}(A \vee \neg B) \mid A \vee \neg B \in S\}$ for a set of clauses S.

Theorem. For any sets of clauses S, S_1, and S_2,

$$S \equiv_{\mathrm{v}} \mathrm{Tr}_{\mathrm{N}}(S) \text{ and } \mathrm{Tr}_{\mathrm{N}}(S_1 \cup S_2) \equiv_{\mathrm{v}} \mathrm{Tr}_{\mathrm{N}}(S_1) \cup \mathrm{Tr}_{\mathrm{N}}(S_2).$$

Proof sketch. There is a bijection $f : \mathrm{CM}(S) \to \mathrm{SM}(\mathrm{Tr}_{\mathrm{N}}(S))$ defined by $f(M) = M \cup \{\overline{a} \mid a \in \mathrm{Hb}(S) \setminus M\}$ so that $f^{-1}(M) = M \cap \mathrm{Hb}(S)$. The modularity of Tr_{N} follows from $\mathrm{Tr}_{\mathrm{N}}(S_1 \cup S_2) = \mathrm{Tr}_{\mathrm{N}}(S_1) \cup \mathrm{Tr}_{\mathrm{N}}(S_2)$. □

An Impossibility Result

- Normal programs cannot be modularly represented with clauses.

Theorem. There is no faithful and modular translation Tr_{C} from normal programs into sets of clauses.

Proof. Assume the contrary, i.e., for all normal programs P, P_1, and P_2, $P \equiv_{\mathrm{v}} \mathrm{Tr}_{\mathrm{C}}(P)$ and $\mathrm{Tr}_{\mathrm{C}}(P_1 \cup P_2) \equiv_{\mathrm{v}} \mathrm{Tr}_{\mathrm{C}}(P_1) \cup \mathrm{Tr}_{\mathrm{C}}(P_2)$.

Consider normal programs $P_1 = \{a \leftarrow \sim a, \sim b.\ \}$ and $P_2 = \{b.\ \}$:

1. Now $\mathrm{SM}(P_1) = \emptyset$ implies that $\mathrm{CM}(\mathrm{Tr}_{\mathrm{C}}(P_1)) = \emptyset$.
2. Thus $\mathrm{CM}(\mathrm{Tr}_{\mathrm{C}}(P_1) \cup \mathrm{Tr}_{\mathrm{C}}(P_2)) = \emptyset$ and also $\mathrm{CM}(\mathrm{Tr}_{\mathrm{C}}(P_1 \cup P_2)) = \emptyset$.
3. It follows that $\mathrm{SM}(P_1 \cup P_2) = \emptyset$, because $P_1 \cup P_2 \equiv_{\mathrm{v}} \mathrm{Tr}_{\mathrm{C}}(P_1 \cup P_2)$.

A contradiction, since $\mathrm{SM}(P_1 \cup P_2) = \{\{b\}\}$. □

2. CLARK'S COMPLETION

- The preceding analysis shows that any *faithful* translation from normal programs into clauses is inherently *non-modular*.
- Thus there is no chance of obtaining a transformation that would work on a rule-by-rule basis (in analogy to Tr_{N} for clauses).
- Clark's *completion procedure* provides a non-modular translation of a normal program P into a propositional theory $\mathrm{Comp}(P)$.
- Although the translation $\mathrm{Comp}(\cdot)$ is not always faithful, it can be characterized in terms of *supported models* of normal programs.

Definition. Given a normal program P and an atom $a \in \mathrm{Hb}(P)$, let $\mathrm{Def}_P(a)$ denote the *definition* of a in P, i.e., the set of rules $a \leftarrow B, \sim C \in P$ having a as their head.

Translating Definitions of Atoms

Definition. For a *finite* normal program P, the theory $\mathrm{Comp}(P)$ includes an equivalence $\mathrm{a} \leftrightarrow ((B_1 \wedge \neg C_1) \vee \ldots \vee (B_n \wedge \neg C_n))$ for each atom $a \in \mathrm{Hb}(P)$ and its definition

$$\mathrm{Def}_P(a) = \{a \leftarrow B_1, \sim C_1. \quad \ldots. \quad a \leftarrow B_n, \sim C_n. \quad \}.$$

A number of observations about $\mathrm{Comp}(P)$ follow:

1. Clark's completion is inherently non-modular because, e.g., $\mathrm{Comp}(\{a \leftarrow b.\ a \leftarrow \sim b.\}) \not\equiv \mathrm{Comp}(\{a \leftarrow b.\}) \cup \mathrm{Comp}(\{a \leftarrow \sim b.\})$.
2. The respective transformation is not faithful in general because $\mathrm{SM}(P) = \{\emptyset\}$ and $\mathrm{CM}(\mathrm{Comp}(P)) = \{\emptyset, \{a\}\}$ for $P = \{a \leftarrow a.\ \}$.
3. The derivation of a CNF for $\mathrm{Comp}(P)$ is exponential in the worst case unless new atoms are introduced as "names" for rule bodies.

Supported Models

Definition. For a normal program P, an interpretation $M \subseteq \mathrm{Hb}(P)$ is a *supported model* of P if and only if $M = \mathrm{T}_{P^M}(M)$.

Proposition. If $M \subseteq \mathrm{Hb}(P)$ is a supported model of a normal program P and $a \in M$, then there is a *supporting rule* $a \leftarrow B, \sim C \in P$ such that a is the head of the rule and $M \models B \cup \sim C$.

Example. The normal program $P = \{\mathrm{a} \leftarrow \mathrm{b}.\ \ \mathrm{b} \leftarrow \mathrm{a}\}$ has two supported models $M_1 = \emptyset$ and $M_2 = \{\mathrm{a}, \mathrm{b}\}$ based on $P^{M_1} = P = P^{M_2}$.

However, only M_1 is stable, as

1. $\mathrm{LM}(P^{M_1}) = \mathrm{LM}(P) = \emptyset = M_1$ and
2. $\mathrm{LM}(P^{M_2}) = \mathrm{LM}(P) = \emptyset \neq M_2$.

Properties of Stable and Supported Models

Theorem. For a normal program P, it holds in general that

$$\mathrm{SM}(P) \subseteq \mathrm{SuppM}(P) = \mathrm{CM}(\mathrm{Comp}(P)).$$

Proposition. If a normal program P contains only *atomic* rules of the form $a \leftarrow \sim C$, then $\mathrm{SM}(P) = \mathrm{SuppM}(P) = \mathrm{CM}(\mathrm{Comp}(P))$.

$\Longrightarrow$ The completion $\mathrm{Comp}(\cdot)$ is faithful for *atomic* normal programs.

Example. Consider a normal program $P = \{a \leftarrow \sim b.\ \ b \leftarrow \sim a.\ \}$ and its completion $\mathrm{Comp}(P) = \{a \leftrightarrow \neg b,\ b \leftrightarrow \neg a\}$.

A perfect match of models results:

$$\mathrm{SM}(P) = \{\{a\}, \{b\}\} = \mathrm{CM}(\mathrm{Comp}(P)).$$

3. LOOP FORMULAS

- Since $\mathrm{Comp}(P)$ is faithful for certain programs, the question is whether it can be revised to be faithful for all normal programs.
- As suggested by preceding examples, the answer to this question goes back to positively interdependent atoms in programs.

Definition. Given a normal program P, a *loop* L is a set of atoms $\{a_1, \ldots, a_n\} \subseteq \mathrm{Hb}(P)$ such that $a_1 \leq_1 \ldots \leq_1 a_n$ and $a_n \leq_1 a_1$ in $\mathrm{DG}^+(\mathrm{P})$.

On the basis of this definition, we observe that

1. atoms in a loop L are mutually dependent in terms of $\leq$, and
2. a loop L does not have to be maximal, i.e., an SCC of $\mathrm{DG}^+(\mathrm{P})$.

Supporting Rules

- A supported model M of P has a set of *supporting rules*

$$\mathrm{SuppR}(P,M) = \{a \leftarrow B, \sim C \in P \mid M \models B \cup \sim C\}.$$

- A loop L for P must be similarly supported under stable models but the support for L must be external to L.

Definition. Given a loop L of a normal program P, the set $\mathrm{ExtSupp}(L,P)$ includes $B \land \neg C$ for each $a \in L$ and each *externally supporting rule* $a \leftarrow B, \sim C \in P$ such that $B \cap L = \emptyset$.

Definition. The disjunctive *loop formula* $\mathrm{LoopF}(L,P)$ associated with a loop L of a normal program P is

$$\bigvee L \rightarrow \bigvee \mathrm{ExtSupp}(L,P)$$

and $\mathrm{LoopF}(P) = \{\mathrm{LoopF}(L,P) \mid L \neq \emptyset \text{ is a loop of } P\}$.

Example

Consider the following normal logic program P:

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

1. Since $a \leq_1 b$ and $b \leq_1 a$ are the only positive dependencies in $\mathrm{DG}^+(\mathrm{P})$, there is only one nonempty loop $L = \{a, b\}$ for P.
2. The set $\mathrm{ExtSupp}(L,P) = \{\neg c, \neg d\}$.
3. The respective loop formula $\mathrm{LoopF}(L,P)$ is

$$a \lor b \rightarrow \neg c \lor \neg d.$$

Remark. If the last two rules of P were dropped, $\mathrm{LoopF}(L,P)$ would be revised to $a \lor b \rightarrow \bot$, which indicates that $\mathrm{LoopF}(P)$ is non-modular.

4. CHARACTERIZATION OF STABLE MODELS

Theorem. Let P be a *finite* normal logic program P and $M \subseteq \mathrm{Hb}(P)$ an interpretation. Then $M \in \mathrm{SM}(P)$ if and only if

$$M \models \mathrm{Comp}(P) \cup \mathrm{LoopF}(P).$$

Example. For the program P from the preceding example, we have

$$\mathrm{Comp}(P) \cup \mathrm{LoopF}(P) =$$
$$\{a \leftrightarrow b \lor \neg c,\ b \leftrightarrow a \lor \neg d,\ c \leftrightarrow \neg d,\ d \leftrightarrow \neg c,\ a \lor b \rightarrow \neg c \lor \neg d\}$$

which has two classical models $M_1 = \{a, b, c\}$ and $M_2 = \{a, b, d\}$.

Then $\mathrm{SM}(P) = \{M_1, M_2\}$ by the theorem above.

Summary of Properties

- The translation $\mathrm{Tr}_{\mathrm{CL}}(P) = \mathrm{Comp}(P) \cup \mathrm{LoopF}(P)$ is *faithful*.
- It is clearly *non-modular* because both $\mathrm{Comp}(P)$ and $\mathrm{LoopF}(P)$ may depend on several rules of P.
- Unfortunately, the translation is also *exponential* in the worst case.
- The last two reflect the difference between expressive powers of normal programs and propositional logic in a very concrete way.

Example. Consider, for instance, the number of loops for a program

$$P_n = \{a_i \leftarrow a_j. \mid 1 \leq i, j \leq n\}.$$

Any subset of $\mathrm{Hb}(P_n) = \{a_1, \ldots, a_n\}$ is a loop!

Computing Stable Models with SAT Solvers

- Despite the space complexity, the translation $\mathrm{Tr}_{\mathrm{CL}}(P)$ can be exploited in the computation of stable models *incrementally*.
- This can be highly effective, e.g., if only one stable model is computed, or the existence of stable models is checked.
- A number of primitives are needed for an implementation:

Completion(P):	Form the completion of P in clausal form.
Satisfy(C):	Compute one model (as a set of literals) for C.
Consistent(M):	Check the consistency of M.
Stable(M,P):	Check the stability of M with respect to P.
MaxLoop(M,P):	Find a maximal unsupported loop $L \subseteq M$.
MakeLoopF(L,P):	Form the loop formula for L in clausal form.

The assat Algorithm

```
function AsSAT(P): boolean;
var C: clause set; M: literal set; L: atom set;
  C := Completion(P);
  M := Satisfy(C);
  while Consistent(M) do
    if Stable(M,P) then return ⊤;
    L := MaxLoop(M,P);
    C := C ∪ MakeLoopF(L,P);
    M := Satisfy(C);
  done
  return ⊥;
```

Remark. If the stability test fails, we have $\mathrm{LM}(P^N) \subset N$ for $N = M \cap \mathrm{Hb}(P)$ which implies the existence of a loop $L \subseteq N \setminus \mathrm{LM}(P^N)$.

5. TIGHT PROGRAMS

- There are subclasses of normal programs P for which $\mathrm{Comp}(P)$ provides a sufficient translation and no loop formulas are needed.

Definition. A normal logic program P is *tight on an interpretation* $M \subseteq \mathrm{Hb}(P)$ if and only if there is a mapping $\lambda : M \to \mathbb{N}$ such that $\lambda(a) > \lambda(B) = \max\{\lambda(b) \mid b \in B\}$ for every $a \leftarrow B \in P^M$ with $B \subseteq M$.

Definition. A normal program P is *tight* if and only if it is tight on every $M \in \mathrm{CM}(\mathrm{Comp}(P)) = \mathrm{SuppM}(P)$.

Theorem. If a finite normal logic program P is *tight*, then

$$\mathrm{SM}(P) = \mathrm{CM}(\mathrm{Comp}(P)) = \mathrm{SuppM}(P).$$

Proof. Since $\mathrm{SM}(P) \subseteq \mathrm{SuppM}(P)$ in general, it remains to prove $\mathrm{SuppM}(P) \subseteq \mathrm{SM}(P)$ when P is tight. Consider any $M \in \mathrm{SuppM}(P)$.

Proof Continued

Now $M = \mathrm{T}_{P^M}(M)$ which implies that $\mathrm{LM}(P^M) \subseteq \mathrm{T}_{P^M}(M) = M$. We prove that $a \in M$ implies $a \in \mathrm{LM}(P^M)$ by complete induction on $\lambda(a)$.

1. For the base case, consider any atom $a \in M$ having the minimum value n for $\lambda(a)$. There must be a supporting rule $a \leftarrow B, \sim C \in P$ such that $M \models B \cup \sim C$, i.e., $a \leftarrow B \in P^M$ and $B \subseteq M$. Because P is tight on M, $\lambda(B) < \lambda(a)$ which implies $B = \emptyset$ because $\lambda(a)$ is the minimum. Thus a appears as a fact in P^M so that $a \in \mathrm{LM}(P^M)$.
2. Then consider any atom $a \in M$ for which $\lambda(a) > n$. As above, there is a supporting rule such that $a \leftarrow B \in P^M$, $B \subseteq M$, and $n \leq \lambda(B) < \lambda(a)$ as P is tight on M. It follows by the inductive hypothesis that $B \subseteq \mathrm{LM}(P^M)$. Thus also $a \in \mathrm{LM}(P^M)$.

To conclude, we have shown that $M = \mathrm{LM}(P^M)$, i.e., $M \in \mathrm{SM}(P)$. □

Example

Consider the following program P_n and $\mathrm{Gnd}(P_n)$:

$$\begin{array}{l}
\mathsf{Edge}(0,1). \quad \ldots \ \mathsf{Edge}(n-1,n). \quad \mathsf{Edge}(n,0). \\
\mathsf{In}(x,y) \leftarrow \sim\mathsf{Out}(x,y), \mathsf{Edge}(x,y). \quad \mathsf{Out}(x,y) \leftarrow \sim\mathsf{In}(x,y), \mathsf{Edge}(x,y). \\
\mathsf{F} \leftarrow \mathsf{In}(0,1), \ldots, \mathsf{In}(n-1,n), \mathsf{In}(n,0), \sim\mathsf{F}. \\
\mathsf{F} \leftarrow \mathsf{Out}(x,y), \mathsf{Out}(z,v), \sim\mathsf{F}, \mathsf{Edge}(x,y), \mathsf{Edge}(z,v), x \neq z. \\
\mathsf{Reach}(x,y) \leftarrow \mathsf{In}(x,y), \mathsf{Edge}(x,y). \quad \mathsf{Node}(x) \leftarrow \mathsf{Edge}(x,y). \\
\mathsf{Reach}(x,y) \leftarrow \mathsf{Reach}(x,z), \mathsf{In}(z,y), \mathsf{Node}(x), \mathsf{Edge}(z,y).
\end{array}$$

When $n = 2$, for instance, one of the $n+1 = 3$ supported models is

$$\begin{array}{l}
M = \{\mathsf{Edge}(0,1), \mathsf{Edge}(1,2), \mathsf{Edge}(2,0), \mathsf{Out}(0,1), \mathsf{In}(1,2), \mathsf{In}(2,0), \\
\quad \mathsf{Node}(0), \mathsf{Node}(1), \mathsf{Node}(2), \mathsf{Reach}(1,2), \mathsf{Reach}(2,0), \mathsf{Reach}(1,0)\ \}.
\end{array}$$

The program $\mathrm{Gnd}(P_n)$ is tight on M—indicating that M is stable.

OBJECTIVES

- You understand the difference of normal logic programs and propositional logic in terms of expressive power.
- You are able to define desirable properties for translations: *faithfulness*, *modularity*, and *polynomiality* (even *linearity*).
- You know the two major sources of *non-modularity* in ASP:
  1. The definition of an atom $\mathrm{Def}_P(a)$ may involve several rules.
  2. The definitions of mutually dependent atoms which belong to the same SCC S of $\mathrm{DG}^+(\mathrm{P})$ should go together.
- You are aware of SAT solvers as potential search engines for ASP.

TIME TO PONDER

The translation

$$\mathrm{Tr}_{\mathrm{CL}}(P) = \mathrm{Comp}(P) \cup \mathrm{LoopF}(P)$$

from normal logic programs to propositional logic is faithful but exponential in the worst case.

- Do you see any possibilities for polynomial transformation?
- Does the case of `smodels` programs present any further difficulties in view of a faithful translation?