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 $\text{Tr}$ are:
  1. Faithfulness: $T \equiv \text{Tr}(T)$.
  2. Modularity: $\text{Tr}(T_1 \cup T_2) \equiv \text{Tr}(T_1) \cup \text{Tr}(T_2)$.

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

Modular Representation for Clauses

- There is a faithful and modular translation $\text{Tr}_N$ from sets of clauses into normal programs (involving constraints).

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

$$\text{Tr}_N(A \lor \neg B) = \{a \leftarrow \neg \bar{a}, \bar{a} \leftarrow \neg a \mid a \in A \cup B\} \cup \{ \bot \leftarrow \neg A, \neg B \}$$

and $\text{Tr}_N(S) = \bigcup \{ \text{Tr}_N(A \lor \neg B) \mid A \lor \neg B \in S \}$ for a set of clauses $S$.

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

$$S \equiv_\forall \text{Tr}_N(S) \text{ and } \text{Tr}_N(S_1 \cup S_2) \equiv_\forall \text{Tr}_N(S_1) \cup \text{Tr}_N(S_2).$$

Proof sketch. There is a bijection $f : \text{CM}(S) \rightarrow \text{SM(Tr}_N(S))$ defined by $f(M) = M \cup \{ \bar{a} \mid a \in \text{Hb}(S) \setminus M \}$ so that $f^{-1}(M) = M \cap \text{Hb}(S)$. The modularity of $\text{Tr}_N$ follows from $\text{Tr}_N(S_1 \cup S_2) = \text{Tr}_N(S_1) \cup \text{Tr}_N(S_2)$. \qed

An Impossibility Result

- Normal programs cannot be modularly represented with clauses.

Theorem. There is no faithful and modular translation $\text{Tr}_C$ from normal programs into sets of clauses.

Proof. Assume the contrary, i.e., for all normal programs $P, P_1,$ and $P_2, P \equiv_\forall \text{Tr}_C(P)$ and $\text{Tr}_C(P_1 \cup P_2) \equiv_\forall \text{Tr}_C(P_1) \cup \text{Tr}_C(P_2)$.

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

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

A contradiction, since $\text{SM}(P_1 \cup P_2) = \{ \{b\} \}$. \qed
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 Comp(P).
- Although the translation Comp(·) 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 ∈ Hb(P), let Def_P(a) denote the definition of a in P, i.e., the set of rules a ← B, ¬C ∈ P having a as their head.

Translating Definitions of Atoms

Definition. For a finite normal program P, the theory Comp(P) includes an equivalence a ← (B_1 ∧ ¬C_1) ∨ ... ∨ (B_n ∧ ¬C_n) for each atom a ∈ Hb(P) and its definition

\[ \text{Def}_P(a) = \{ a \leftarrow B_1, \neg C_1 \}. \]

A number of observations about Comp(P) follow:

1. Clark’s completion is inherently non-modular because, e.g.,

\[ \text{Comp} \{ a \leftarrow b, \ a \leftarrow \neg b \} \neq \text{Comp} \{ a \leftarrow b \} \cup \text{Comp} \{ a \leftarrow \neg b \}. \]

2. The respective transformation is not faithful in general because

\[ \text{SM}(P) = \{0\} \text{ and } \text{CM}(\text{Comp}(P)) = \{0, \{a\}\} \text{ for } P = \{a \leftarrow a\}. \]

3. The derivation of a CNF for 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 ⊆ Hb(P) is a supported model of P if and only if \( M = T_{\text{post}}(M) \).

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

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

However, only \( M_1 \) is stable, as

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

Properties of Stable and Supported Models

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

\[ \text{SM}(P) \subseteq \text{Supp}(P) = \text{CM}(\text{Comp}(P)). \]

Proposition. If a normal program P contains only atomic rules of the form \( a \leftarrow \neg C \), then \( \text{SM}(P) = \text{Supp}(P) = \text{CM}(\text{Comp}(P)) \).

The completion Comp(·) is faithful for atomic normal programs.

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

A perfect match of models results:

\[ \text{SM}(P) = \{ \{a\}, \{b\} \} = \text{CM}(\text{Comp}(P)). \]
3. LOOP FORMULAS

- Since 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 \( \{a_1,\ldots,a_n\} \subseteq \text{Hb}(P) \) such that \( a_1 \leq \cdots \leq a_n \) and \( a_n \leq a_1 \) in \( \text{DG}^+(P) \).

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 \( \text{DG}^+(P) \).

Example

Consider the following normal logic program P:

\[
\begin{align*}
   a & \leftarrow b. \\
   b & \leftarrow a. \\
   c & \leftarrow \neg d. \\
   d & \leftarrow \neg c. \\
   a & \leftarrow \neg c. \\
   b & \leftarrow \neg d.
\end{align*}
\]

1. Since \( a \leq b \) and \( b \leq a \) are the only positive dependencies in \( \text{DG}^+(P) \), there is only one nonempty loop \( L = \{a, b\} \) for P.

2. The set \( \text{ExtSupp}(L, P) = \{\neg c, \neg d\} \).

3. The respective loop formula \( \text{LoopF}(L, P) \) is

\[
a \lor b \rightarrow \neg c \lor \neg d.
\]

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

Supporting Rules

- A supported model \( M \) of P has a set of supporting rules

\[
\text{SuppR}(P, M) = \{a \leftarrow B, \neg C \in P \mid M \models B \cup \neg C\}.
\]

- 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 \( \text{ExtSupp}(L, P) \) includes \( B \land \neg C \) for each \( a \in L \) and each externally supporting rule \( a \leftarrow B, \neg C \in P \) such that \( B \cap L = \emptyset \).

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

\[
\forall L \rightarrow \forall \text{ExtSupp}(L, P)
\]

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

4. CHARACTERIZATION OF STABLE MODELS

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

\[
M \models \text{Comp}(P) \cup \text{LoopF}(P).
\]

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

\[
\text{Comp}(P) \cup \text{LoopF}(P) = \{a \leftrightarrow b \lor \neg c, \ b \leftrightarrow a \lor \neg d, \ c \leftrightarrow \neg d, \ d \leftrightarrow \neg c, \ a \lor b \leftrightarrow \neg c \lor \neg d\}
\]

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

Then \( \text{SM}(P) = \{M_1, M_2\} \) by the theorem above.
Summary of Properties

- The translation $T_{CL}(P) = \text{Comp}(P) \cup \text{LoopF}(P)$ is **faithful**.
- It is clearly **non-modular** because both $\text{Comp}(P)$ and $\text{LoopF}(P)$ 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 $\text{Hb}(P_n) = \{a_1, \ldots, a_n\}$ is a loop!

---

The assat Algorithm

**function** AsSAT($P$): boolean;  
**var** $C$: clause set; $M$: literal set; $L$: atom set;  
$C := \text{Completion}(P)$;  
$M := \text{Satisfy}(C)$;  
while Consistent($M$) do  
  if Stable($M, P$) then return $\top$;  
  $L := \text{MaxLoop}(M, P)$;  
  $C := C \cup \text{MakeLoopF}(L, P)$;  
  $M := \text{Satisfy}(C)$;  
return $\bot$;

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

---

Computing Stable Models with SAT Solvers

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

---

5. TIGHT PROGRAMS

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

**Definition.** A normal logic program $P$ is **tight on an interpretation** $M \subseteq \text{Hb}(P)$ if and only if there is a mapping $\lambda : M \rightarrow \mathbb{N}$ 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 $M \in \text{CM}(\text{Comp}(P)) = \text{SuppM}(P)$.

**Theorem.** If a finite normal logic program $P$ is **tight**, then  
$\text{SM}(P) = \text{CM}(\text{Comp}(P)) = \text{SuppM}(P)$.

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

Now \( M = T_{pa}(M) \) which implies that \( LM(P^M) \subseteq T_{pa}(M) = M \). We prove that \( a \in M \) implies \( a \in LM(P^M) \) 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 \( a \in LM(P^M) \).

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 \( B \subseteq LM(P^M) \). Thus also \( a \in LM(P^M) \).

To conclude, we have shown that \( M = LM(P^M) \), i.e., \( M \in SM(P) \). \( \square \)

Example

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

\[
\begin{align*}
\text{Edge}(0,1). & \quad \ldots \quad \text{Edge}(n-1,n). & \quad \text{Edge}(n,0). \\
\text{In}(x,y) & \leftarrow \sim \text{Out}(x,y), \text{Edge}(x,y). & \quad \text{Out}(x,y) & \leftarrow \sim \text{In}(x,y), \text{Edge}(x,y). \\
F & \leftarrow \text{In}(0,1), \ldots, \text{In}(n-1,n), \text{In}(n,0), \sim F. & \\
F & \leftarrow \text{Out}(x,y), \text{Out}(z,v), \sim F, \text{Edge}(x,y), \text{Edge}(z,v), x \neq z. \\
\text{Reach}(x,y) & \leftarrow \text{In}(x,y), \text{Edge}(x,y). & \quad \text{Node}(x) & \leftarrow \text{Edge}(x,y). \\
\text{Reach}(x,y) & \leftarrow \text{Reach}(x,z), \text{In}(z,y), \text{Node}(x), \text{Edge}(z,y). \\
\end{align*}
\]

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

\[
M = \{ \text{Edge}(0,1), \text{Edge}(1,2), \text{Edge}(2,0), \text{Out}(0,1), \text{In}(1,2), \text{In}(2,0), \\
\text{Node}(0), \text{Node}(1), \text{Node}(2), \text{Reach}(1,2), \text{Reach}(2,0), \text{Reach}(1,0) \}.
\]

The program \( \text{Gnd}(P_n) \) 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 \( \text{Def}_P(a) \) may involve several rules.
  2. The definitions of mutually dependent atoms which belong to the same SCC \( S \) of DG\( ^+(P) \) should go together.
- You are aware of SAT solvers as potential search engines for ASP.

Time to Ponder

The translation \( \text{Tr}_\text{CL}(P) = \text{Comp}(P) \cup \text{LoopF}(P) \) 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 \text{smo}dels programs present any further difficulties in view of a faithful translation?