IKI30820 – Logic Programming
Semantic Aspects of Prolog Programs
Slide 03

Ari Saptawijaya (slides) Adila A. Krisnadhi (LaTeXadaptation)

Fakultas Ilmu Komputer
Universitas Indonesia

2009/2010 • Semester Gasal

Outline

Semantics of Prolog program

- Declarative semantics
  - Defines whether a goal is true w.r.t. a given program, and if it is true, for what instantiation of variables it is true
- Procedural meaning
  - Defines a procedure for satisfying a list of goals in the context of a given program
  - The procedure outputs the truth or falsity of the goal list and the corresponding instantiations of variables
  - The procedure automatically **backtracks** to examine alternatives

Example

```
P :- Q, R.
```

where `P`, `Q`, and `R` are terms.

- Declarative meaning:
  - `P` is true if `Q` and `R` are true.
  - From `Q` and `R` follows `P`.
- Procedural meaning:
  - To solve problem `P`, *first* solve the subproblem `Q` and *then* the subproblem `R`.
  - To satisfy `P`, *first* satisfy `Q` and *then* `R`

Declarative vs. procedural semantics

- Declarative meaning does not depend on the order of the clauses and the order of the goals in clauses
- Procedural meaning does depend on the order of goals and clauses.
  - The order can affect the efficiency of the program.
  - An unsuitable order may even lead to infinite recursive calls

Outline

Declarative semantics

- The declarative semantics of logic programs is based on the standard model-theoretic semantics of first-order logic
- Prerequisites:
  - Herbrand Universe
  - Herbrand Base
  - Interpretation
  - Model
  - Minimal model

Instance and Variant

Definition (Instance)

An **instance** of a clause C is the clause C with each of its variables substituted by some terms

Definition (Variant)

A **variant** of a clause C is such an instance of the clause C where each variable is substituted by another variable

Example: consider the clause

```
hasachild(X) :- parent(X,Y).
```

- A variant of the clause:

  ```
  hasachild(A) :- parent(A,B).
  ```

- An instance of the clause:

  ```
  hasachild(sandro) :- parent(sandro,D).
  ```

Herbrand universe I

Definition (Herbrand universe)

Let P be a logic program.
The **Herbrand universe** of P, denoted by $U(P)$, is the set of all ground terms that can be formed from the constants and function symbols appearing in P.

Let P_1 be the following program. What is $U(P_1)$?

```
parent(tukul,budi).
parent(budi,doni).
parent(doni,harto).
parent(harto,tomi).

ancestor(X,Y) :- parent(X,Y).
ancestor(X,Z) :- parent(X,Y),ancestor(Y,Z).
```

Let P_1 be the following program. What is $U(P_1)$?

```
parent(tukul,budi).
parent(budi,doni).
parent(doni,harto).
parent(harto,tomi).

ancestor(X,Y) :- parent(X,Y).
ancestor(X,Z) :- parent(X,Y),ancestor(Y,Z).
```

$$U(P_1) \coloneqq \{\texttt{tukul}, \texttt{budi}, \texttt{doni}, \texttt{harto}, \texttt{tomi}\}$$

Let P_2 be the following program.

```
nat(0).
nat(s(X)) :- nat(X).
```

What is $U(P_2)$?

Let P_2 be the following program.

```
nat(0).
nat(s(X)) :- nat(X).
```

What is $U(P_2)$?

$$U(P_2) \coloneqq \{0, s(0), s(s(0)), s(s(s(0))), \ldots\}$$

Herbrand base

Definition (Herbrand base)

The **Herbrand Base**, denoted by $B(P)$, is the set of all ground goals that can be formed from the predicates in P and the terms in the Herbrand universe.

- What is $B(P_1)$?
- What is $B(P_2)$?

$$B(P_1) \coloneqq \{\texttt{parent}(\texttt{tukul},\texttt{tukul}), \texttt{parent}(\texttt{tukul},\texttt{budi}), \ldots,$$
$$\texttt{parent}(\texttt{tomi},\texttt{doni}), \texttt{parent}(\texttt{tomi},\texttt{tomi}),$$
$$\texttt{ancestor}(\texttt{tukul},\texttt{tukul}), \texttt{ancestor}(\texttt{tukul},\texttt{budi}), \ldots,$$
$$\texttt{ancestor}(\texttt{tomi},\texttt{doni}), \texttt{ancestor}(\texttt{tomi},\texttt{tomi})\}$$

Since there are two predicates with arity 2 and $|U(P_1)| = 4$, then $|B(P_1)| = 2 \cdot 2^4 = 32$.

$$B(P_2) \coloneqq \{\texttt{nat}(\texttt{0}), \texttt{nat}(\texttt{s}(\texttt{0})), \texttt{nat}(\texttt{s}(\texttt{s}(\texttt{0}))), \ldots\}$$

Interpretation

Definition (Interpretation)

An **interpretation** for a logic program is a subset of the Herbrand base

Important!

An interpretation assigns truth and falsity to the elements of the Herbrand base.

- A goal in the Herbrand base is *true* w.r.t. an interpretation if it is a member of it.
- Otherwise, *false*.

Note that an interpretation is simply a subset of the Herbrand base. It may or may not be a *model* w.r.t. a program (it may be correct or incorrect w.r.t. a program).

Model

Sometimes (esp. in the context of semantics), we denote a clause
`A :- B1, ..., Bn`
by $A \leftarrow B_1, \ldots, B_n$.

Definition (Model)

An interpretation I is a **model** for a logic program if for each ground instance of a clause in the program $A \leftarrow B_1, \ldots, B_n$, A is in I if $B_1, \ldots, B_n$ are in I.

Intuitively, models are interpretation that *respect* the *declarative* reading of the clauses of a program.

- `parent(tukul,budi)` must be in a model of P_1. What else are in a model of P_1?
- Is `nat(0)` in a model of P_2? What else are in a model of P_2?

$$M(P_1) \coloneqq \{\texttt{parent(tukul,budi)}, \texttt{parent(budi,doni)},$$
$$\texttt{parent(doni,harto)}, \texttt{parent(harto,tomi)},$$
$$\texttt{ancestor(tukul,budi)}, \texttt{ancestor(budi,doni)},$$
$$\texttt{ancestor(doni,harto)}, \texttt{ancestor(harto,tomi)},$$
$$\texttt{ancestor(tukul,doni)}, \texttt{ancestor(budi,harto)},$$
$$\texttt{ancestor(doni,tomi)},$$
$$\texttt{ancestor(tukul,harto)}, \texttt{ancestor(budi,tomi)},$$
$$\texttt{ancestor(tukul,tomi)}\}$$

$$M(P_2) \coloneqq \{\texttt{nat(0)}, \texttt{nat(s(0))}, \texttt{nat(s(s(0)))}, \ldots\}$$

Consequence operator

- Let P be a program, $ground(P)$ be set of all ground instances of clauses in P, and I be an interpretation. Then:

$$T_P(I) = \{A \mid A \leftarrow B_1, \ldots, B_n \in ground(P), \{B_1, \ldots, B_n\} \subseteq I\}$$

- An interpretation I is a **model** of P iff

$$T_P(I) \subseteq I$$

Example

Consider the following simple program “Happy”

```
happy :- graduated, married.
married :- has_fiancee.
married :- graduated.
graduated.
```

- Is `{graduated,married}` a model of “Happy”?
- Is `{graduated,married,happy}` a model of “Happy”?
- Is `{graduated,married,happy,has_fiancee}` a model of “Happy”?

Minimal model

- The model obtained as the intersection of all models is known as the **minimal model** and denoted by M(P).
- The minimal model is the **declarative meaning** of a logic program.

Example

- What is the declarative meaning of “Happy”?
- What is the declarative meaning of P_2?

Computing declarative meaning (iterating operator)

- Let I be an interpretation.
  - $T \uparrow 0(I) = I$
  - $T \uparrow (n+1)(I) = T(T \uparrow n(I))$
  - . . . until a **least fixpoint**, $T \uparrow \omega(I)$, is reached.
- Declarative meaning of a logic program is characterized by $T \uparrow \omega(\emptyset)$.
- Verify the declarative meaning of “Happy” using this iterating operator.

Conjunction and disjunction of goals

- A list of goals is true if all the goals are true for the **same** instantiation of variables.
  - The values of the variables result from the **most general** instantiation
- A comma between goals denotes the **conjunction** of goals
  - All have to be true
- A semicolon between goals denotes the **disjunction** of goals
  - Any one of the goals has to be true.

```
P :- Q;R
```

(P is true if Q is true or R is true) is equivalent to:

```
P :- Q.
P :- R.
```

Example

Rewrite the following program without using the semicolon notation

```
translate(Number,Word) :-
  Number=1, Word=satu;
  Number=2, Word=dua;
  Number=3, Word=tiga;
  Number=4, Word=empat.
```

Outline

Example

```
wild(bear).              % clause 1
wild(lion).              % clause 2
pet(cat).                % clause 3
brown(bear).             % clause 4
brown(lion).             % clause 5
black(cat).              % clause 6

dark(Z) :- black(Z). % clause 7: anything black is dark
dark(Z) :- brown(Z). % clause 8: anything brown is dark
```

Query: who is dark and wild?

```
?- dark(X), wild(X).
```

Procedural meaning I

To execute the list of goals: $G_1, G_2, \ldots, G_m$, the procedure **execute** does the following:

1. If the goal list is empty then terminate with ***success***.
2. If the goal list is not empty then continue with (the following) operation called '***SCANNING***'.
3. ***SCANNING***: scan through the clauses in the program from *top* to *bottom* until the first clause, C, is found such that the head of C *matches* the first goal G_1.
   1. If there is no such clause then terminate with ***failure***.
   2. If there is such a clause C of the form H :- $B_1, \ldots, B_n$, then
      - rename variables in C to obtain a variant C', such that C' and $G_1, \ldots, G_m$ have **no** common variables; let C' be H' :- $B'_1, \ldots, B'_n$;
      - match G_1 and H' and let the resulting instantiation be S;
      - replace the goal list with the new goal list: $B'_1, \ldots, B'_n, G_2, \ldots, G_m$ with variables in this new goal list is substituted by the values as specified in S;

Procedural meaning II

4. Execute (recursively) the new goal list:
   1. If the execution of this new goal list terminates with success, then the execution of the original goal list also terminates with success (and gives the instantiation of all variables appear in the original goal list)
   2. If the execution of the new goal list is **not** successful, then abandon this new goal list and go back to ***SCANNING*** through the program (**backtrack**):
      - Continue the scanning with the clause that immediately follows the clause that was last used (that led to failure) and try to find successful termination.

Monkey and Banana problem I

```
% move( State1, Move, State2): making Move in State1 results in
%      State2; a state is represented by a term:
%  state( HorizontalPos, VerticalPos, BoxPos, HasBanana)

move( state( middle, onbox, middle, hasnot),    % Before move
      grasp,                                    % Grasp banana
      state( middle, onbox, middle, has) ).     % After move

move( state( P, onfloor, P, H),
      climb,                                    % Climb box
      state( P, onbox, P, H) ).

move( state( P1, onfloor, P1, H),
      push( P1, P2),                  % Push box from P1 to P2
      state( P2, onfloor, P2, H) ).

move( state( P1, onfloor, B, H),
      walk( P1, P2),                  % Walk from P1 to P2
      state( P2, onfloor, B, H) ).
```

Monkey and Banana problem II

```
% canget( State): monkey can get banana in State

canget( state( _, _, _, has) ). % can 1: Monkey already has it

canget( State1) :-              % can 2: Do some work to get it
  move( State1, Move, State2),          % Do something
  canget( State2).                      % Get it now
```

The query:

```
?- canget(state(atdoor,onfloor,atwindow,hasnot)).
```

Build a **Prolog tree** to represent the computation w.r.t. the above query.

Four version of the predecessor program I

```
% The original version
pred1(X, Z) :- parent(X, Z).
pred1(X, Z) :- parent(X, Y), pred1(Y, Z).

% Variation a: swap clauses of the original version
pred2(X, Z) :- parent(X, Y), pred2(Y, Z).
pred2(X, Z) :- parent(X, Z).

% Variation b: swap goals in 2nd clause of the original version
pred3(X, Z) :- parent(X, Z).
pred3(X, Z) :- pred3(X, Y), parent(Y, Z).

% Variation c: swap goals and clauses of the original version
pred4(X, Z) :- pred4(X, Y), parent(Y, Z).
pred4(X, Z) :- parent(X, Z).
```

Questions (recall the family relations in worksheet 1): Is Tukul a predecessor of Fifi? Is Liz a predecessor of Jojon?

General heuristic in problem solving: it is usually best to try the simplest idea first!