Models

One person's syntax is another person's semantics.

One particular view of semantics is that is a regression of languages. Consider our translational semantics:

$$\text{source language} \longrightarrow \text{stack machine language} \longrightarrow \text{Prolog}$$

If we only consider the first half of this diagram then we are assigning semantics to the source language in terms of syntactic snippets of the stack machine language.

If we only consider the second part of the diagram then we are interpreting stack machine language syntax in terms of Prolog constructs (the thing we have done all along in this course).

Models

A fundamental question in semantics is: When is it reasonable to end this regression?

The answer: when the final language in this regression is a formal language that can be understood in mathematical terms.

There is a reason why we chose Prolog as our semantic modeling language, because Prolog itself has a semantics – Model Theory,

$$\text{Prolog} \rightarrow \text{Model Theory}$$

That is, every Prolog program can be interpreted in a "first order model" sometimes also called the *Herbrand Universe*.[1]

[1] en.wikipedia.org/wiki/Term_algebra

Models

Now, if look at what we have done for the major part of the course and then attach the semantics of Prolog we get this diagram,

$$\text{source language} \longrightarrow \text{Prolog} \longrightarrow \text{Model Theory}$$

Considering that interpretation is transitive this means by using Prolog as our defining language for our source languages we obtain formal (mathematical) models for our own languages!

Elements of Model Theory

- Model theory is the study of semantics for logic and provides the formal justification for using Prolog as a defining language and as a theorem prover.
- Similar to our models we constructed in this class, the models in Model Theory provide an interpretation for the syntactic units appearing in a logic program.

Elements of Model Theory

- Let's start with what we know, we know how to write logic programs and deduce knowledge from them.
- Let's consider the following logic program *P*:

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

  Here the functor `s` represents the successor function that takes an integer value and produces the integer value that follows that integer. That is `s(0)` represents 1 and `s(s(0))` represents 2, etc.
- Given this program we now can pose queries to extract knowledge:

  ```
  ?- odd(s(s(s(s(s(0)))))).
  true

  ?- odd(s(s(0))).
  false.

  ?-
  ```

Elements of Model Theory

- Given our program P and our query q:

  ```
  ?- odd(s(s(s(s(s(0)))))).
  true
  ```

  We say that q *derives* from P and we write

  $$P \vdash q$$

- Here the $\vdash$ operator represents inferencing in logic and in this particular case it represents inferencing via *resolution*.

Elements of Model Theory

- Now, when we wrote our program P:

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

  we had a particular model in mind. In particular, we probably thought about the natural numbers $\mathbb{N}$.

- Taking the natural numbers together with the *nodd* operation gives us our model M:

  $$(\mathbb{N}, \mathit{nodd})$$

  where $\mathit{nodd} : \mathbb{N} \to \{\mathit{true}, \mathit{false}\}$ is defined as the function[2]

  $$\mathit{nodd}(x) = \begin{cases} \mathit{true} & \text{if } x \bmod 2 = 0, \\ \mathit{false} & \text{if } x \bmod 2 = 1, \end{cases}$$

  for all $x \in \mathbb{N}$.

[2]Notice that with this definition 0 is an even number.

Elements of Model Theory

- It is now easy to show that each sentence $p \in P$ is *satisfied* by this model, *i.e.*, each sentence $p \in P$ is true in model M, where P is our odd program.
- We can formally show this for the first sentence,

  ```
  odd(s(0)).
  ```

  if we let $\texttt{s}(0) \mapsto 1$ and $\texttt{odd} \mapsto \mathit{nodd}$ then $\mathit{nodd}(1)$ is true in M.
- Similarly for the sentence `odd(s(s(X))) :- odd(X)`, this can be shown by induction over the odd natural numbers.

Elements of Model Theory

- If our model satisfies some sentence p then we write:

  $$M \models p$$

  and we say that *sentence p is true in model M*.
- If our model M satisfies every sentence p in P then we write

  $$M \models P$$

Elements of Model Theory

Assume that we have $M \models P$,

- We say that our logic is *sound* if anything that we can derive from our program P is also true in our model M. Formally,

$$P \vdash p \Rightarrow M \models p$$

- We say that our logic is *complete* if anything that is true in the model can be derived from the program P. Formally,

$$M \models p \Rightarrow P \vdash p$$

It has been shown that resolution is sound and complete,[3] that means, anything we can query from the program will be true in the model, and anything that is true in the model can be deduced from the program using queries.[4]

[3] *Foundations of Logic Programming*,Lloyd, J.W.,Springer-Verlag, 1987.

[4] Well, with some help - Prolog's implementation of resolution turns out not to be complete.

Elements of Model Theory

- In terms of programming language semantics, let P be a description of a programming language model, let M be the intended model, then because of soundness and completeness, any characteristic c about our programming language that can be deduced from P will also be true in the intended model,

  $$P \vdash c \Rightarrow M \models c$$

  and any characteristic c that is true in M can be proven,

  $$M \models c \Rightarrow P \vdash c$$

- That means, we are justified to use Prolog as a theorem prover to prove characteristics about our programming language models.

Elements of Model Theory

THE END