COMP 317: Semantics of Programming Languages

Lecture 5 Summary

This lecture completes the inductive definition of [[*P*]](*S*). The remaining case is when *P* is a while-loop. A loop of the form

```
while (B) { P }
```

is executed by evaluating the Boolean Expression `B`; if this returns false, then we exit the loop:

> for all States *S* and Programs *P* and Boolean Expressions *B*,
> [[`while (`*B*`) {`*P*`}`]](*S*) = *S*   if not [[*B*]](*S*)

If evaluating `B` returns true, then we execute `P`; if our starting state is *S*, this gives us the state [[`P`]](*S*); and then we repeat the whole process:

> for all States *S* and Programs *P* and Boolean Expressions *B*,
> [[`while (`*B*`) {`*P*`}`]](*S*) = [[`while (`*B*`) {`*P*`}`]]([[*P*]](*S*)) if [[*B*]](*S*)

Thus, the semantics of while-loops is given by

> for all States *S* and Programs *P* and Boolean Expressions *B*,
> [[`while (`*B*`) {`*P*`}`]](*S*) = *S*   if not [[*B*]](*S*)
> [[`while (`*B*`) {`*P*`}`]](*S*) = [[`while (`*B*`) {`*P*`}`]]([[*P*]](*S*)) if [[*B*]](*S*)

And this completes the inductive definition of [[*P*]].

When we define a function, we have to say what the output is for any given input. Defining a function inductively does that: whatever the actual program *P* is, and whatever state *S* is, our definition tells us what the resulting state [[*P*]](*S*) is. For example, if *P* is the program

```
'x := 0;
while ('x < 2) {
   'x := 'x + 1;
}
```

and *S* is the *initial* state (where all variables have the value 0), then [[*P*]](*S*) is the state *initial*[`'x <-` 2].

But we also want to be able to *calculate* outputs for any given inputs, just as we can for the example given above. However, if we take *P* to be `while` `(true) { skip }`, then for any state *S*, our calculation goes on and on and on:

```
[[while (true) { skip }]](S) =
[[while (true) { skip }]]([[skip]](S)) =
[[while (true) { skip }]](S) =
[[while (true) { skip }]]([[skip]](S)) =
...
```

We can't find the state that results from executing this program because there is no such state. We say the function [[*P*]] is *partial*: for some programs *P* and some states *S*, there is no state [[*P*]](*S*) - just as, for the function f(x) = 1/x, there is no number f(0). When there is no output for a given input, we say the function is *undefined* at that input: for example, f is undefined at 0, and the function [[`while (true) { skip }`]] is undefined at all states. The function [[`while (! 'x == 2) { 'x := 'x + 1; }`]] is undefined for all states *S* for which *S*(`'x`) > 2 (the program goes into an infinite loop), but is defined for all other states (and has output *S*[`'x <-` 2]).

Key Points

- The function [[*P*]] is partial. The function is defined at a state *S* precisely when the program *P* terminates when run in the state *S*.
- We now have a complete semantics for SImpL; the denotation functions say how to evaluate expressions and Boolean expressions, and how to execute programs.

Grant Malcolm