COMP 317: Semantics of Programming Languages

Lecture 3 Summary

This lecture introduces the syntax and semantics of **SImpL**, a **S**imple **Imp**erative **L**anguage. Imperative languages are characterised by:

- **variables** that store values;
- **commands** such as assignments; and
- **flow of control** such as conditionals and loops.

In the interests of SImpLicity, the language SImpL has only one data type, integers, so that all variables store integer values; the only commands are assignments; and the only flow-of-control constructs are sequential composition (one command after another), conditionals, and while-loops. This keeps the language as simple as possible so that we can concentrate on the fundamental issues of syntax and semantics.

In this lecture we look at the syntax and semantics of SImpL expressions:

<Digit>	::=	`0` | `1` | ... | `9`
<Numeral>	::=	*<Digit>* | *<Numeral>* *<Digit>*
<Variable>	::=	`'a` | `'b` | ... | `'aa` | `'ab` | ...
<Expression>	::=	*<Numeral>* | *<Variable>* | *<Expression>* `+` *<Expression>* | *<Expression>* `*` *<Expression>* | `-`*<Expression>*

So Expressions include strings like `2 * 'x + 1`.

We described how expressions are evaluated by inductively defining the integer [[*E*]](*S*) as a function of Expression *E* and State *S*. States are tables that list the integer values stored by the SImpL Variables. For example, a typical state might be

'a	'b	...	'x	'y	...	'aa	'ab	...
3	0	...	7	25	...	-5	27	...

Such tables can be viewed as functions that take a variable and return an integer. Thus, table look-up is function application: For example, if *S* is the state pictured above, then *S*(`'x`) represents the value stored by `'x` in *S*, that is, *S*(`'x`) = 7.

The inductive definition of [[*E*]](*S*) follows the BNF definition of *<Expression>*s:

- For Numeral *N* and State *S*, [[*N*]](*S*) = [[*N*]], where [[*N*]] is the denotation of *N*
- For Variable *V* and State *S*, [[*V*]](*S*) = *S*(*V*)
- For Expressions *E1* and *E2* and State *S*, [[*E1*+*E2*]](*S*) = [[*E1*]](*S*)+[[*E2*]](*S*)
- For Expressions *E1* and *E2* and State *S*, [[*E1***E2*]](*S*) = [[*E1*]](*S*)*[[*E2*]](*S*)
- For Expression *E* and State *S*, [[-*E*]](*S*) = - [[*E*]](*S*).

Key Points

The inductive definition of [[*E*]](*S*) gives a means of calculating the value of [[*E*]](*S*) for any particular Expression *E* and State *S*.

You should be able to:

- calculate particular values like [[`2*'x+'y`]](S), where S is the state pictured above;
- give an inductive definition of the boolean value [[*B*]](*S*) as a function of Boolean Expression *B* and State *S* (see the Problem Sheet for Week 3).

Grant Malcolm