Datalogi V – Programming Languages

Andrzej Filinski Robert Glück Neil D. Jones

Spring 2007

Datalogisk Institut
Københavns Universitet

The remainder of this course will be dedicated to formulating a variety of programming language concepts using operational semantics. Before proceeding into details, we outline its general characteristics:

- A language definition is given by a finite set of inference rules. Any one computation is described by a finite "proof tree" that is built from the inference rules.
- Unlike denotational semantics, there is not a hard separation between syntax and semantics.
- The mathematical objects dealt with are all *discrete* and *finitary*: only finitely representable values can be used. Thus a value may not be an infinite set, or a function with an infinite domain.
- Some but not all operational semantic definitions are executable.
- Operational semantic definitions are better-suited for describing finite computations than for describing infinite processes.[4]
- The meaning of a recursive construction such as a `while` loop or a recursively defined function is obtained by "syntactic unfolding" rather than by fixpoints on domains.
- Operational semantics is *first order*: higher-order functions are not invoked to explain the meanings of programs or their fragments. Finite-domain functions are allowed, for example stores and environments mapping variables to values.
- Operational semantics are *structural*, being described in terms of the syntax of the program whose meaning is being defined. This is close to *compositionality* from denotational semantics: the effect of a compound language construction is found using the effects of its syntactic subphrases.[5]

As a first step towards structural operational semantics, we first present an important underlying concept: that of an "inference system."

5.4 Inference systems

We will use inference systems to define operational semantics and type systems in programming languages. Inference systems also have many other applications, e.g., in mathematical logic for defining logical systems, reasoning about provability (e.g., Gödel's completeness and incompleteness theorems).

An inference system is a finte set of *inference rules* used to define one or more *semantic relations.*

[4]Denotational semantics can handle both, if sufficiently sophisticated domains are used.

[5]A difference: denotational semantics uses "fixpoint" operators such as *fix* seen for the `while` statement in Figure 5.1. This necessarily involves use of a higher-order function, Operational semantics would instead use first-order functions, and express the effect of iterative `while` execution by "unwinding" iterations or recursions.

An inference rule has some finite number of judgments $P_1, P_2, \ldots$ as *premises*; and a single judgment C as *conclusion*. If a rule has no premises, it is called an *axiom*. An inference rule may have *side conditions*: conditions involving values appearing in the premises and/or conclusion. These conditions must be satisfied in order for the rule to be usable in a proof tree. An inference rule may also be named for reference.

We follow the common practice of writing a rule with the premises above and the conclusion below a solid line. The most general format is:

$$\text{Rule name: } \frac{P_1 \quad P_2 \quad \ldots P_k}{C} \quad \text{(Side conditions)} \qquad \begin{array}{l}\text{Premises above the line (0 or more)}\\[1em] \text{Conclusion below the line}\end{array}$$

A rule expresses that the judgment below the line follows logically (or by definition) from the judgments above the line. An example of a rule is:

$$\frac{\mathit{Even}(x)}{\mathit{Even}(x+2)}$$

This rule expresses that if x is an even number, then $x+2$ is also an even number. An axiom is usually written without premises, and sometimes without the solid line. Such a rule states something unconditional: the judgment below the line holds *without preconditions*. Example: 0 is an even number:

$$\frac{}{\mathit{Even}(0)}$$

Mathematically, an n-ary *relation* (also called a *predicate*) is a set of *n-tuples*, i.e., a set $R \subseteq V_1 \times \ldots V_n$, where V_i is the set of all possible values for v_i in $R(v_1, v_2, \ldots, v_n)$. For any $(v_1, v_2, \ldots, v_n) \in V_1 \times \ldots V_n$ we say "$R(v_1, v_2, \ldots, v_n)$ is true," or "$R(v_1, v_2, \ldots, v_n)$ holds,", just in case $(v_1, v_2, \ldots, v_n) \in R$. A unary (same as 1-ary) relation $R(x)$ on set V is a subset of V.

For example, if $\mathit{Even} \subseteq \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ is some unary relation over the integers, then we can write either $\mathit{Even}(x)$ or $x \in \mathit{Even}$: they mean the same thing. Further, if *Even* satisfies the two inference rules above, we must have:

$$\mathit{Even} = \{0, 2, 4, 6, \ldots\}$$

Axioms and inference rules are used inductively: an axiom states that certain tuples are in a relation, with no prior assumptions. The inductive step: an inference rule may state that certain tuples are in a relation, *provided* some other relations hold. These other relations in a rule are its premises.

5.4.1 Some examples of inference systems defining unary relations

Even natural numbers. *Even* is a unary relation on $\boldsymbol{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$. (Unary = 1-ary.) *Even* is the subset $\{0, 2, 4, \ldots\}$ of $\boldsymbol{Z}$. It can be defined by two rules, one an axiom and one with one premise:

$$\text{A: } \frac{}{\mathit{Even}(0)} \qquad \text{B: } \frac{\mathit{Even}(x)}{\mathit{Even}(x+2)}$$

$\mathit{Even}(0)$ holds because of rule A, which is an axiom. Once we know this, we can further apply rule B to deduce that $\mathit{Even}(2)$ holds.

Well-balanced corner bracket strings: This is a unary relation on strings: WB $\subseteq A^*$, where $A = \{\texttt{<},\texttt{>}\}$ is an alphabet (finite set of symbols). Informally, WB is "the set of all well-balanced bracket strings." WB can be defined by three rules, one an axiom, one with one premise, and with two premises:

$$\text{Base: } \frac{}{\text{WB}(\texttt{<>})} \qquad \text{Nest: } \frac{\text{WB}(s)}{\text{WB}(\texttt{<}s\texttt{>})} \qquad \text{Sequence: } \frac{\text{WB}(s) \quad \text{WB}(s')}{\text{WB}(ss')}$$

WB(<<><>><>) can be seen to hold (be true) by the following:

1. WB(<>) by the Base rule
2. WB(<><>) by 1, 1 and the Sequence rule
3. WB(<<><>>) by 2 and the Nest rule
4. WB(<<><>><>) by 3, 1 and the Sequence rule

Another way to present this reasoning is via the following *proof tree*:

$$\text{Sequence: } \frac{\text{Nest: } \dfrac{\text{Sequence: } \dfrac{\text{Base: } \dfrac{}{\text{WB}(\texttt{<>})} \quad \text{Base: } \dfrac{}{\text{WB}(\texttt{<>})}}{\text{WB}(\texttt{<><>})}}{\text{WB}(\texttt{<<><>>})} \qquad \text{Base: } \dfrac{}{\text{WB}(\texttt{<>})}}{\text{WB}(\texttt{<<><>><>})}$$

A more compact presentation without the rule names:

$$\frac{\dfrac{\dfrac{\dfrac{}{\text{WB}(\texttt{<>})} \quad \dfrac{}{\text{WB}(\texttt{<>})}}{\text{WB}(\texttt{<><>})}}{\text{WB}(\texttt{<<><>>})} \qquad \dfrac{}{\text{WB}(\texttt{<>})}}{\text{WB}(\texttt{<<><>><>})}$$

Syntax of arithmetic expressions in + and $*$. These rules define unary relations on strings over the alphabet $A = \{\texttt{A}, \texttt{B}, \ldots, \texttt{Z}, \texttt{(}, \texttt{)}, \texttt{+}, \texttt{*}\}$. This involves three judgments: *Expression*(E), *Term*(T), *Primary*(P), where E, T, P range over strings.

$$\frac{}{\mathit{Primary}(\texttt{A})} \quad \frac{}{\mathit{Primary}(\texttt{B})} \quad \cdots \quad \frac{}{\mathit{Primary}(\texttt{Z})} \quad \frac{\mathit{Expression}(E)}{\mathit{Primary}(\ \texttt{(}\,E\,\texttt{)}\ )}$$

$$\frac{\mathit{Primary}(P)}{\mathit{Term}(P)} \qquad \frac{\mathit{Term}(T) \quad \mathit{Primary}(P)}{\mathit{Term}(T\texttt{*}P)}$$

$$\frac{\mathit{Term}(T)}{\mathit{Expression}(T)} \qquad \frac{\mathit{Expression}(E) \quad \mathit{Term}(T)}{\mathit{Expression}(E\texttt{+}T)}$$

The example can be expressed in a more familiar form, as a *context-free grammar*:

$$\begin{array}{lcl} P & ::= & \texttt{A} \mid \texttt{B} \mid \ldots \mid \texttt{Z} \mid \texttt{(}E\texttt{)} \\ T & ::= & P \mid T \texttt{*} P \\ E & ::= & T \mid E \texttt{+} T \end{array}$$

A proof tree establishing *Expression*(A+B*C) resembles the expression's parse tree, turned upside down:

Proof tree:

$$\dfrac{\dfrac{\dfrac{\mathit{Primary}(\texttt{A})}{\mathit{Term}(\texttt{A})}}{\mathit{Expression}(\texttt{A})} \qquad \dfrac{\dfrac{\mathit{Primary}(\texttt{B})}{\mathit{Term}(\texttt{B})} \quad \dfrac{}{\mathit{Primary}(\texttt{C})}}{\mathit{Term}(\texttt{B*C})}}{\mathit{Expression}(\texttt{A+B*C})}$$

5.4.2 More about inference systems

A relation may, for convenience, readability etc., be written in other ways than $R(v_1, \ldots, v_n)$. In ordinary mathematics the binary (2-ary) relation $<$ is usually written in *infix* notation, for example $x < y$ rather than $< (x, y)$.

A syntax for a relation $R(v_1, \ldots, v_n)$ is called a **judgement form**. For example a binary judgement about arithmetic expression evaluation might be written:

$$\mathit{Expression} \Longrightarrow \mathit{value}$$

Further, a judgment about programs might naturally be ternary (3-ary)

$$\mathit{program} : \mathit{input} \Rightarrow \mathit{output}$$

Metavariables. Variables ranging over given sets such as commands, expressions, variables or stores, are very common in inference rules. For example, in the rules above we have used metavariables x, s, E, T, P ranging over *natural numbers, strings, expressions, terms*, and *primaries*, respectively. Examples of sets of values in our application area include program fragments, the values that program fragments denote, and auxiliary constructions such as stores.

A *rule instance* is obtained by instantiating metavariables to particular elements of their respective sets. Note that *all* occurrences of a metavariable in a rule must be instantiated to the *same* value to obtain a valid rule instance. Metavariables in *different* rules, however, can be instantiated independently of each other. For example, both

$$\frac{\mathit{Even}(2)}{\mathit{Even}(4)} \qquad \text{and} \qquad \frac{\mathit{Even}(3)}{\mathit{Even}(5)}$$

are rule instances of the inference rule

$$\frac{\mathit{Even}(x)}{\mathit{Even}(x+2)}$$

Side conditions in a rule constrain which instantiations of the metavariables are allowed in a rule instance. For example, the rule

$$\frac{}{\mathit{Primary}(P)} \ (P \in \{\texttt{A}, \texttt{B}, \ldots, \texttt{Z}\})$$

has the side condition $P \in \{\texttt{A}, \texttt{B}, \ldots, \texttt{Z}\}$, limiting the possible values of P.

5.4.3 Derivations, or proof trees

A *derivation* or *proof tree* is a nonempty finite tree, where the conclusion of a rule instance is the root, and the derivations of the premises are its subtrees. It is a finite demonstration that a judgment is derived from a number of premises by a rule instance, and where each of the premises is in turn derived in the same fashion.

Derivations must be finite: an inference rule has only a finite number of premises, so by tracing a path from a judgment to a premise and then to one of *its* premises and so on we must eventually end up with an instance of an axiom. In other words, every path from the root must eventually reach a leaf.

The judgment at the root of a derivation is called a *derivable judgment.* Such judgments can be thought of as *facts*; that is, as true statements.

An example proof tree might have form:

$$\dfrac{\dfrac{P_1 \quad P_2}{C_3} \qquad P_4 \qquad \dfrac{P_5 \quad \dfrac{P_6 \quad P_7}{C_8} \quad P_9}{C_{10}}}{C_{11}}$$

This proves C_{11}. A full proof tree must be *locally correct* at every node (internal, root, or leaf). A tree is locally correct at the root node of a subtree if the root is a consequence of the nodes just above it by one of the inference rules, and all side conditions are satisfied. For instance, the truth of C_8 must follow from the truth of P_6 and P_7; and C_{10} must follow from P_5, C_8, P_9. Since the local correctness condition applies to *all* nodes, this implies that every leaf ($P_{1,2,4,5,6,7,9}$ above) must be an instance of an axiom, i.e., true without need for further justifications. The tree above thus involves 7 axiom uses for the 7 leaf nodes, and non-axiom rule uses for the 4 non-leaf nodes.

5.4.4 Connections with logic

Inference systems originated in *Mathematical Logic*, for the purpose of making a precise formulation of mathematical reasoning, for example proofs in geometry from Euclid's axioms. A concrete "formal system" is often presented by beginning with definitions of some syntactic categories and then by presenting inference systems for reasoning about them. Formal proof procedures work by symbol manipulation, and are often presented in the form of inference systems.

A judgment such as $\Gamma \vdash F$ usually has an intuitive reading, for instance "propositional formula F is true, provided the assumptions listed in Γ hold." An example of an inference rule is the *modus ponens* rule:

$$\text{If } \Gamma \vdash F \Rightarrow G \text{ and } \Gamma \vdash F, \text{ then } \Gamma \vdash G$$

Expressed in tree form this becomes:

$$\text{Modus ponens: } \frac{\Gamma \vdash F \Rightarrow G \qquad \Gamma \vdash F}{\Gamma \vdash G}$$

5.5 Big-step semantics: expression evaluation by inference rules

We now show how expressions in a programming language can be evaluated, relating the *syntactic* world of expressions to their *semantics*, i.e. the mathematical values that they denote.

Suppose `e` is an expression, such as `x+y`, which contains occurrences of the variables `x` and `y`. Then the value of `e` can only be determined under some *value assumptions* about the values of `x` and `y`. Such assumptions can be represented by a finite function $store = [\texttt{x} \mapsto v, \ldots]$ that for instance maps `x` to its value, so $store(\texttt{x}) = v$. Function *store* is usually called a "store" in an imperative programming language, or an "environment" in a functional programming language.

The assertion that "if $\texttt{x} = 5$ and $\texttt{y} = 6$, then $\texttt{x+y} = 11$" can be written as follows: