IMP Concrete Semantics

CS 260

1 IMP Abstract Syntax

$$n \in \mathbb{Z} \qquad x \in \mathit{Variable}$$

$$\begin{aligned} p \in \mathit{Program} &::= \textbf{decl } \vec{x} \textbf{ in } \overrightarrow{s} \\ s \in \mathit{Stmt} &::= x := e \mid \textbf{if } e \overrightarrow{s_1} \textbf{ else } \overrightarrow{s_2} \mid \textbf{while } e \vec{s} \\ e \in \mathit{Exp} &::= \overline{n} \mid x \mid e_1 \oplus e_2 \\ \oplus \in \mathit{BinaryOp} &::= + \mid - \mid \times \mid \div \mid < \mid \leq \mid = \mid \neq \end{aligned}$$

Notation. By abuse of notation we use the vector notation $\overrightarrow{\cdot}$ to indicate an ordered sequence of unspecified size n, indexed from $0 \leq i < n$. We use the overline notation $\overline{\cdot}$ to indicate an unordered set. The length of a vector (respectively, set) is denoted by $|\overrightarrow{\cdot}|$ (respectively, $|\overline{\cdot}|$).

Syntax Summary. A *program* consists of a declaration giving the list of variables used in the program followed by a sequence of statements. A *statement* is an assignment, a conditional, or a while loop. An *expression* is a set of integers, a variable, or a binary operation. We use sets of integers to allow for nondeterministic execution without needing to specify I/O for the language.

2 IMP Concrete Semantics

We describe the semantic domains that constitute a state of the transition system (Section 2.1), state transition rules (Section 2.2), and the helper functions used by the transition rules (Section 2.3).

2.1 Concrete Semantic Domains

$$\begin{aligned} \varsigma \in \mathit{State} &= \mathit{Stmt}^? \times \mathit{Locals} \times \mathit{Kont}^\star \\ \rho \in \mathit{Locals} &= \mathit{Variable} \to \mathbb{Z} \\ \kappa \in \mathit{Kont} &= \textbf{stmtK } \mathit{Stmt} \uplus \textbf{whileK } \mathit{Exp} \times \mathit{Stmt}^\star \end{aligned}$$

Notation. We borrow notation from formal languages: $\cdot^?$ means 0 or 1 instances; $\cdot^\star$ means an ordered sequence of 0 or more instances; $\cdot^+$ means an ordered sequence of 1 or more instances. The $\uplus$ operator means disjoint union.

Domains Summary. A state consists of an optional statement to be processed, a map from the progam's variables to their values, and a continuation stack. The continuation stack is a sequence of **stmtK** continuations (holding statements to be processed) and **whileK** continuations (holding the guard and body of a currently executing while loop, so we can start the next iteration).

2.2 Concrete Transition Rules

Table 1: The concrete transition relation. Each rule describes how to take one concrete state $(s^?, \rho, \vec{\kappa})$ to the next concrete state $(s^?_{new}, \rho_{new}, \overrightarrow{\kappa_{new}})$, where $\vec{\kappa} = \kappa \cdot \overrightarrow{\kappa_1}$. The $s^?$ notation means a statement may or may not exist; we use $\bullet$ to indicate that there is no statement. The $\cdot$ operator used for the continuation stack indicates appending sequences; thus κ is the top of the continuation stack in the source state and $\overrightarrow{\kappa_1}$ is the rest of that continuation stack.

no.	$s^?$	**premises**	$s^?_{new}$	ρ_{new}	$\overrightarrow{\kappa_{new}}$
1	$x := e$	$[\![e]\!] = n$	$\bullet$	$\rho[x \mapsto n]$	$\vec{\kappa}$
2	**if** e $\overrightarrow{s_1}$ **else** $\overrightarrow{s_2}$	$[\![e]\!] \neq 0$	$\bullet$	ρ	$\texttt{toSK}(\overrightarrow{s_1}) \cdot \vec{\kappa}$
3	**if** e $\overrightarrow{s_1}$ **else** $\overrightarrow{s_2}$	$[\![e]\!] = 0$	$\bullet$	ρ	$\texttt{toSK}(\overrightarrow{s_2}) \cdot \vec{\kappa}$
4	**while** e $\vec{s}$	$[\![e]\!] \neq 0$	$\bullet$	ρ	$\texttt{toSK}(\vec{s}) \cdot$ **whileK**$(e, \vec{s}) \cdot \vec{\kappa}$
5	**while** e $\vec{s}$	$[\![e]\!] = 0$	$\bullet$	ρ	$\vec{\kappa}$
6	$\bullet$	$\kappa =$ **stmtK**(s_1)	s_1	ρ	$\overrightarrow{\kappa_1}$
7	$\bullet$	$\kappa =$ **whileK**$(e, \vec{s})$, $[\![e]\!] \neq 0$	$\bullet$	ρ	$\texttt{toSK}(\vec{s}) \cdot \vec{\kappa}$
8	$\bullet$	$\kappa =$ **whileK**$(e, \vec{s})$, $[\![e]\!] = 0$	$\bullet$	ρ	$\overrightarrow{\kappa_1}$

Notation. We use $[\![e]\!]$ as shorthand for $\eta(e, \rho)$ when ρ is obvious from context. We use $s^?$ to indicate 0 or 1 statements; $\bullet$ means there is no statement. For any map X, the notation $X[a \mapsto b]$ means a new map that is exactly the same as X except that a maps to b.

2.3 Concrete Helper Functions

We describe the helper functions used by the transition rules. The functions are listed in alphabetical order.

2.3.1 $\eta(e, \rho)$ **a.k.a.** $[\![e]\!]$

This function describes how to evaluate expressions to values. Note that sets of integers are evaluated by nondeterministically selecting an element from that set. Variables are looked up in the locals map; binary operators recursively evaluate the operands and then apply the appropriate operation to the result (the operators are described below).

$\eta : \mathit{Exp} \times \mathit{Locals} \to \mathbb{Z}$

$\eta(e, \rho) =$

$$\begin{cases} n & \text{if } e = \overline{n},\ n \in \overline{n} \\ \rho(x) & \text{if } e = x \\ [\![e_1]\!] \oplus [\![e_2]\!] & \text{if } e = e_1 \oplus e_2 \end{cases}$$

Arithmetic Operators. $\{+, -, \times, \div\} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ are the standard (unbounded width) integer arithmetic operators.

Relational Operators. $\{<, \leq, =, \neq\} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ are the standard relational operators on integers, except that instead of **true** or **false** they return 1 or 0 (respectively).

2.3.2 `initstate`

This function takes the program and generates the initial state.

$\texttt{initstate} \in \mathit{Program} \to \mathit{State}$

$\texttt{initstate}(p) = (\bullet, \rho, \vec{\kappa})$ where

$p =$ **decl** $\vec{x}$ **in** $\overrightarrow{s}$

$\rho = [\ x_i \mapsto 0 \ \mid \ x_i \in \vec{x}\ ]$

$\vec{\kappa} = \texttt{toSK}(\vec{s})$

2.3.3 `toSK`

This function maps a sequence of statements to a sequence of **stmtK** continuations containing those statements.

$\texttt{toSK} \in \mathit{Stmt}^\star \to \mathit{Kont}^\star$

$\texttt{toSK}(\vec{s}) = \vec{\kappa}$ where $\kappa_i \in \vec{\kappa} =$ **stmtK**(s_i) for $0 \leq i < |\vec{s}|$