01/25/19 Recitation Notes

17-355/17-665/17-819: Program Analysis (Spring 2019)
Jenna Wise
jlwise@andrew.cmu.edu

1 Reminders

- Homework 2 is due next **Thursday, January 31, 2019 at 11:59pm**. Homework instructions and the LaTeX helper files can be found on the course website
- If you want feedback on your in-class exercises, please let me know
- Office hours are weekly on Tuesday from 5-6pm in Jonathan's office and Thursday from 1-3pm in Jenna's office or by appointment

2 Lecture Review for Homework

2.1 Judgments & Inference Rules

2.1.1 Judgments

Judgments are statements written in a "metalanguage" — a language in which the symbols and rules for manipulating another language (in our case the WHILE or WHILE3ADDR languages) are formulated. For example, to formulate big-step operational semantics for WHILE (see Section 2.2.1 for more information about big-step operational semantics) we write and give meaning to this statement, which manipulates program statements in the WHILE language:

$$\langle S, E\rangle \Downarrow E' \; for \; S \in \texttt{Stmt} \; and \; E, E' \in Var \to \mathbb{Z}$$

2.1.2 Inference Rules

An inference rule is made up of a set of judgments above the line, known as *premises*, and a judgment below the line, known as the *conclusion*. The meaning of an inference rule is that the conclusion holds if all of the premises hold:

$$\frac{premise_1 \quad premise_2 \quad \ldots \quad premise_n}{conclusion}$$

An inference rule with no premises is an *axiom*, which is always true. We use sets of inference rules to express various program semantics, as seen in Section 2.2.

2.2 Program Semantics

A program's semantics define its meaning. There are three main classes of formal semantics denotational, operational, and axiomatic. So far, we've covered operational semantics and its two classes: big-step operational semantics and small-step operational semantics.

Operational semantics mimics, at a high level, the operation of a computer executing the program.

Both big-step and small-step operational semantics depend on a program state $E \in \mathit{Var} \to \mathbb{Z}$, which maps program variables to their corresponding integer values.

2.2.1 Big-step Operational Semantics

Big-step operational semantics specifies the entire execution of a program at runtime. Inference rules define the big-step operational semantics for the WHILE language and these three main judgments:

$$\begin{array}{ll} \langle a, E\rangle \Downarrow_a n & \mathit{for}\ a \in \texttt{Aexp}\ \mathit{and\ integer\ literal}\ n \\ \langle P, E\rangle \Downarrow_b b & \mathit{for}\ P \in \texttt{Bexp}\ \mathit{and}\ b \in \{\texttt{true}, \texttt{false}\} \\ \langle S, E\rangle \Downarrow E' & \mathit{for}\ S \in \texttt{Stmt} \end{array}$$

Each judgment above indicates complete execution of an arithmetic expression, boolean expression, or program statement (respectively) starting from program state E, which results in an integer literal, the true or false boolean expression, or a potentially different program state E' (respectively).

Big-step operational semantics for the WHILE language

$\langle \boldsymbol{a}, \boldsymbol{E}\rangle \Downarrow_{\boldsymbol{a}} \boldsymbol{n}$

$$\frac{}{\langle n, E\rangle \Downarrow_a n}\ \textit{big-int} \qquad \frac{}{\langle x, E\rangle \Downarrow_a E(x)}\ \textit{big-var} \qquad \frac{\langle a_1, E\rangle \Downarrow_a n_1 \quad \langle a_2, E\rangle \Downarrow_a n_2}{\langle a_1\ op_a\ a_2, E\rangle \Downarrow_a n_1\ op_a\ n_2}\ \textit{big-aop}$$

$\langle \boldsymbol{P}, \boldsymbol{E}\rangle \Downarrow_{\boldsymbol{b}} \boldsymbol{b}$ **where** $\boldsymbol{b} \in \{\texttt{true}, \texttt{false}\}$

$$\frac{}{\langle b, E\rangle \Downarrow_b b}\ \textit{big-b} \qquad \frac{\langle P, E\rangle \Downarrow_b b}{\langle \text{not } P, E\rangle \Downarrow_b \neg\, b}\ \textit{big-not} \qquad \frac{\langle P_1, E\rangle \Downarrow_b b_1 \quad \langle P_2, E\rangle \Downarrow_b b_2}{\langle P_1\ op_b\ P_2, E\rangle \Downarrow_b b_1\ op_b\ b_2}\ \textit{big-bop}$$

$$\frac{\langle a_1, E\rangle \Downarrow_a n_1 \quad \langle a_2, E\rangle \Downarrow_a n_2}{\langle a_1\ op_r\ a_2, E\rangle \Downarrow_b n_1\ op_r\ n_2}\ \textit{big-rop}$$

$\langle \boldsymbol{S}, \boldsymbol{E}\rangle \Downarrow \boldsymbol{E'}$

$$\frac{}{\langle \texttt{skip}, E\rangle \Downarrow E}\ \textit{big-skip} \qquad \frac{\langle a, E\rangle \Downarrow_a n}{\langle x := a, E\rangle \Downarrow E[x \mapsto n]}\ \textit{big-assign} \qquad \frac{\langle S_1, E\rangle \Downarrow E' \quad \langle S_2, E'\rangle \Downarrow E''}{\langle S_1; S_2, E\rangle \Downarrow E''}\ \textit{big-seq}$$

$$\frac{\langle P, E\rangle \Downarrow_b \texttt{true} \quad \langle S_1, E\rangle \Downarrow E'}{\langle \texttt{if } P \texttt{ then } S_1 \texttt{ else } S_2, E\rangle \Downarrow E'}\ \textit{big-iftrue} \qquad \frac{\langle P, E\rangle \Downarrow_b \texttt{false} \quad \langle S_2, E\rangle \Downarrow E'}{\langle \texttt{if } P \texttt{ then } S_1 \texttt{ else } S_2, E\rangle \Downarrow E'}\ \textit{big-iffalse}$$

$$\frac{\langle P, E\rangle \Downarrow_b \texttt{true} \quad \langle S, E\rangle \Downarrow E' \quad \langle \texttt{while } P \texttt{ do } S, E'\rangle \Downarrow E''}{\langle \texttt{while } P \texttt{ do } S, E\rangle \Downarrow E''}\ \textit{big-whiletrue}$$

$$\frac{\langle P, E\rangle \Downarrow_b \texttt{false}}{\langle \texttt{while } P \texttt{ do } S, E\rangle \Downarrow E}\ \textit{big-whilefalse}$$

2.2.2 Small-step Operational Semantics

Small-step operational semantics specifies the execution of a program at runtime one step at a time until a final result or program configuration ($\langle \texttt{skip}, E\rangle$) is reached (if it can be reached). Inference rules define the small-step operational semantics for the WHILE language and these three main judgments:

$$\begin{array}{ll} \langle a, E\rangle \rightarrow_a a' & for\ a, a' \in \texttt{Aexp} \\ \langle P, E\rangle \rightarrow_b P' & for\ P, P' \in \texttt{Bexp} \\ \langle S, E\rangle \rightarrow \langle S', E'\rangle & for\ S, S' \in \texttt{Stmt} \end{array}$$

Each judgment above indicates one step of execution of an arithmetic expression, boolean expression, or program statement (respectively) starting from program state E, which results in a potentially different arithmetic expression, boolean expression, or program configuration (respectively). The above judgments with $\rightarrow_a$, $\rightarrow_b$, and $\rightarrow$ replaced by $\rightarrow_a^*$, $\rightarrow_b^*$, and $\rightarrow^*$ respectively indicate zero or more steps of execution of an arithmetic expression, boolean expression, or program statement (respectively) starting from program state E, which results in a potentially different arithmetic expression, boolean expression, or program configuration (respectively).

Small-step operational semantics for the WHILE language

$\boldsymbol{\langle a, E\rangle \rightarrow_a a'}$

$$\frac{}{\langle n, E\rangle \rightarrow_a n}\ \textit{small-int} \qquad \frac{}{\langle x, E\rangle \rightarrow_a E(x)}\ \textit{small-var}$$

$$\frac{\langle a_1, E\rangle \rightarrow_a a_1'}{\langle a_1\ op_a\ a_2, E\rangle \rightarrow_a a_1'\ op_a\ a_2}\ \textit{small-aop-congruence-a1}$$

$$\frac{\langle a_2, E\rangle \rightarrow_a a_2'}{\langle n\ op_a\ a_2, E\rangle \rightarrow_a n\ op_a\ a_2'}\ \textit{small-aop-congruence-a2} \qquad \frac{}{\langle n_1\ op_a\ n_2, E\rangle \rightarrow_a n_1\ op_a\ n_2}\ \textit{small-aop}$$

$\boldsymbol{\langle P, E\rangle \rightarrow_b P'}$

$$\frac{b \in \{\texttt{true}, \texttt{false}\}}{\langle b, E\rangle \to_b b}\ \textit{small-b} \qquad \frac{\langle P, E\rangle \to_b P'}{\langle \text{not } P, E\rangle \to_b \text{not } P'}\ \textit{small-not-congruence}$$

$$\frac{b \in \{\texttt{true}, \texttt{false}\}}{\langle \text{not } b, E\rangle \to_b \neg\, b}\ \textit{small-not} \qquad \frac{\langle P_1, E\rangle \to_b P_1'}{\langle P_1\ op_b\ P_2, E\rangle \to_b P_1'\ op_b\ P_2}\ \textit{small-bop-congruence-p1}$$

$$\frac{b \in \{\texttt{true}, \texttt{false}\} \quad \langle P_2, E\rangle \to_b P_2'}{\langle b\ op_b\ P_2, E\rangle \to_b b\ op_b\ P_2'}\ \textit{small-bop-congruence-p2}$$

$$\frac{b_1, b_2 \in \{\texttt{true}, \texttt{false}\}}{\langle b_1\ op_b\ b_2, E\rangle \to_b b_1\ op_b\ b_2}\ \textit{small-bop} \qquad \frac{\langle a_1, E\rangle \to_a a_1'}{\langle a_1\ op_r\ a_2, E\rangle \to_b a_1'\ op_r\ a_2}\ \textit{small-rop-congruence-a1}$$

$$\frac{\langle a_2, E\rangle \to_a a_2'}{\langle n\ op_r\ a_2, E\rangle \to_b n\ op_r\ a_2'}\ \textit{small-rop-congruence-a2} \qquad \frac{}{\langle n_1\ op_r\ n_2, E\rangle \to_b n_1\ op_r\ n_2}\ \textit{small-rop}$$

$\langle$skip, $E\rangle$ final

$$\frac{}{\langle \texttt{skip}, E\rangle\ \ \texttt{final}}\ \textit{small-skip}$$

$\langle S, E\rangle \to \langle S', E'\rangle$

$$\frac{\langle a, E\rangle \to_a a'}{\langle x := a, E\rangle \to \langle x := a', E\rangle}\ \textit{small-assign-congruence} \qquad \frac{}{\langle x := n, E\rangle \to \langle \texttt{skip}, E[x \mapsto n]\rangle}\ \textit{small-assign}$$

$$\frac{\langle S_1, E\rangle \to \langle S_1', E'\rangle}{\langle S_1; S_2, E\rangle \to \langle S_1'; S_2, E'\rangle}\ \textit{small-seq-congruence} \qquad \frac{}{\langle \texttt{skip}; S_2, E\rangle \to \langle S_2, E\rangle}\ \textit{small-seq}$$

$$\frac{\langle P, E\rangle \to_b\ P'}{\langle \texttt{if}\ P\ \texttt{then}\ S_1\ \texttt{else}\ S_2, E\rangle \to \langle \texttt{if}\ P'\ \texttt{then}\ S_1\ \texttt{else}\ S_2, E\rangle}\ \textit{small-if-congruence}$$

$$\frac{}{\langle \texttt{if true then}\ S_1\ \texttt{else}\ S_2, E\rangle \to \langle S_1, E\rangle}\ \textit{small-iftrue}$$

$$\frac{}{\langle \texttt{if false then}\ S_1\ \texttt{else}\ S_2, E\rangle \to \langle S_2, E\rangle}\ \textit{small-iffalse}$$

$$\frac{}{\langle \texttt{while}\ P\ \texttt{do}\ S, E\rangle \to \langle \texttt{if}\ P\ \texttt{then}\ (S; \texttt{while}\ P\ \texttt{do}\ S)\ \texttt{else skip}, E\rangle}\ \textit{small-while}$$

2.2.3 Derivations

We can prove that concrete program expressions will evaluate to particular values and concrete program statements will evaluate to particular program states (in the case of big-step operational semantics) or configurations (in the case of small-step operational semantics). This is done by chaining together rules of inference into *derivations*, for example:

$$\dfrac{\dfrac{\dfrac{}{\langle 4, E_1\rangle \Downarrow 4}\ \textit{big-int} \quad \dfrac{}{\langle 2, E_1\rangle \Downarrow 2}\ \textit{big-int}}{\langle 4 * 2, E_1\rangle \Downarrow 8}\ \textit{big-aop} \quad \dfrac{}{\langle 6, E_1\rangle \Downarrow 6}\ \textit{big-int}}{\langle (4 * 2) - 6, E_1\rangle \Downarrow 2}\ \textit{big-aop}$$

Therefore, we have proven that $(4 * 2) - 6$ evaluates to 2 starting in the program state E_1.

2.2.4 Proof Techniques - Structural Induction

A precise language specification lets us precisely prove properties of our language or programs written in it (including analyses that we write). Structural induction is the main proof technique that we use to do this.

Structural Induction
is a special case of well-founded induction where a well-founded relation, $\prec \subseteq A \times A$, is defined on the recursive structure of a program or derivation. Recall that well-founded induction says that to prove $\forall x \in A.P(x)$ it is enough to prove $\forall x \in A.[\forall y \prec x \Rightarrow P(y)] \Rightarrow P(x)$; the base case arises when there is no $y \prec x$, and so the part of the formula within the brackets [] is vacuously true.

You can induct on the structure of an abstract syntax construct (ie. arithmetic expressions) or you can induct on the structure of a derivation (which you will do in `hw2`).

Induction on the Structure of Derivations
Derivation trees are defined inductively and are built of sub-derivations, so we can induct on their structure. To prove that property P holds for a program statement, we will prove that P holds for all possible derivations of that statement. Such a proof consists of the following steps:

Base Case(s): Show that P holds for each atomic derivation rule with no premises (of the form $\overline{S}$).
Inductive Case(s): For each derivation rule of the form

$$\frac{H_1 \ldots H_n}{S}$$

By the induction hypothesis, P holds for H_i, where $i = 1 \ldots n$. We then have to prove that the property is preserved by the derivation using the given rule of inference.

Inversion: A key technique for induction on derivations is *inversion.* Because the number of forms of rules of inference is finite, we can tell which inference rules might have been used last in the derivation. For example, if D is the derivation that proves $\langle \text{while } P \text{ do } S, E\rangle \Downarrow E'$, then (by inversion) the last rule used in D was either the `big-whiletrue` rule or the `big-whilefalse` rule.

Example Proof: In `hw2` you will be partially proving by induction on the structure of derivations that if a statement terminates, the big- and small-step semantics for WHILE will obtain equivalent results, expressed formally as:

$$\forall S \in Stmt. \quad \forall E, E' \in Var \to \mathbb{Z}. \quad \langle S, E\rangle \to^* \langle \texttt{skip}, E'\rangle \quad \Leftrightarrow \quad \langle S, E\rangle \Downarrow E' \quad (1)$$

You will also be able to assume that this property holds for arithmetic and boolean expressions in WHILE, expressed formally as:

$$\begin{array}{llll} \forall a \in \texttt{AExp}. & \forall n \in \mathbb{Z}. & \langle a, E\rangle \to_a^* n & \Leftrightarrow \quad \langle a, E\rangle \Downarrow n \quad (2) \\ \forall P \in \texttt{Bexp}. & \forall b \in \{\texttt{true}, \texttt{false}\}. & \langle P, E\rangle \to_b^* b & \Leftrightarrow \quad \langle P, E\rangle \Downarrow b \quad (3) \end{array}$$

The $\langle S, E\rangle \to^* \langle \texttt{skip}, E'\rangle \Rightarrow \langle S, E\rangle \Downarrow E'$ direction of the proof of equation (1) requires Lemma 1 to be proven. The structure of the proof of this lemma and the proof for one inductive case (this proof is by structural induction on the structure of a derivation) for this lemma will be shown below:

Lemma 1.
$\forall S, S' \in Stmt.\ \forall E, E', E'' \in Var \to \mathbb{Z}.\ \langle S, E\rangle \to \langle S', E'\rangle$ *and* $\langle S', E'\rangle \Downarrow E'' \Rightarrow \langle S, E\rangle \Downarrow E''$

Proof. By structural induction on the derivation of $\langle S, E\rangle \to \langle S', E'\rangle$

*** We get to assume both the arbitrary $\langle S, E\rangle \to \langle S', E'\rangle$ and $\langle S', E'\rangle \Downarrow E''$ judgments have derivations which prove them for this proof, so we can perform structural induction on either derivation tree; I chose the derivation of $\langle S, E\rangle \to \langle S', E'\rangle$ (choosing the right one to find a proof requires trial and error).

[BASE CASE(S)]

$$\frac{}{\langle x := n, E\rangle \to \langle \texttt{skip}, E[x \mapsto n]\rangle}\ \textit{small-assign} \qquad \frac{}{\langle \texttt{skip}; S_2, E\rangle \to \langle S_2, E\rangle}\ \textit{small-seq}$$

$$\frac{}{\langle \texttt{if true then } S_1 \texttt{ else } S_2, E\rangle \to \langle S_1, E\rangle}\ \textit{small-iftrue}$$

$$\frac{}{\langle \texttt{if false then } S_1 \texttt{ else } S_2, E\rangle \to \langle S_2, E\rangle}\ \textit{small-iffalse}$$

$$\frac{}{\langle \texttt{while } P \texttt{ do } S, E\rangle \to \langle \texttt{if } P \texttt{ then } (S; \texttt{while } P \texttt{ do } S) \texttt{ else skip}, E\rangle}\ \textit{small-while}$$

*** For each base case (inference rule) listed above, we would have to prove that given the conclusion of the inference rule and $\langle S', E'\rangle \Downarrow E''$ adjusted for the rule, then we can arrive at $\langle S, E\rangle \Downarrow E''$ adjusted for the rule.

[INDUCTIVE CASE:]

$$\left[\frac{\langle P, E\rangle \to_b P'}{\langle \texttt{if } P \texttt{ then } S_1 \texttt{ else } S_2, E\rangle \to \langle \texttt{if } P' \texttt{ then } S_1 \texttt{ else } S_2, E\rangle}\ \textit{small-if-congruence}\right]$$

We are given that $\langle \texttt{if } P' \texttt{ then } S_1 \texttt{ else } S_2, E\rangle \Downarrow E''$.

By inversion on $\langle \texttt{if } P' \texttt{ then } S_1 \texttt{ else } S_2, E\rangle \Downarrow E''$, we get

$$\langle P', E\rangle \Downarrow_b \texttt{true} \textbf{ and } \langle S_1, E\rangle \Downarrow E'' \textbf{ or}$$
$$\langle P', E\rangle \Downarrow_b \texttt{false} \textbf{ and } \langle S_2, E\rangle \Downarrow E''$$

If $\langle P', E\rangle \Downarrow_b \texttt{true}$ and $\langle S_1, E\rangle \Downarrow E''$ then

By (2) (from above homework description),
$\langle P', E\rangle \Downarrow_b \texttt{true}$ implies $\langle P', E\rangle \to_b^* \texttt{true}$
Since $\langle P, E\rangle \to_b P'$ and $\langle P', E\rangle \to_b^* \texttt{true}$, we get
$\langle P, E\rangle \to_b^* \texttt{true}$
By (2) (from above homework description),
$\langle P, E\rangle \to_b^* \texttt{true}$ implies $\langle P, E\rangle \Downarrow_b \texttt{true}$
Since $\langle P, E\rangle \Downarrow_b \texttt{true}$ and $\langle S_1, E\rangle \Downarrow E''$, we get
$\langle \texttt{if } P \texttt{ then } S_1 \texttt{ else } S_2, E\rangle \Downarrow E''$ by the `big-iftrue` inference rule

If $\langle P', E\rangle \Downarrow_b$ `false` and $\langle S_2, E\rangle \Downarrow E''$ then

[Proof for this subcase is similar to the proof for the $\langle P', E\rangle \Downarrow_b$ `true` subcase]

*** To finish the proof for the inductive cases, each inductive case (WHILE statement inference rule with premise(s)) would need to be proved in a similar fashion as the `small-if-congruence` case.

□

3 Other Homework Help

3.1 Typesetting in LaTeX

You are welcome to use your favorite LaTeX package(s) to typeset your homework 2 submission, but I will accept scanned pdfs of handwritten solutions for homework 2.

If you prefer to use LaTeX and do not have a favorite LaTeX package, you may be interested in `mathpartir`, which you can find on the course website. To see how to write some inference rules,

- Download and unzip the `mathpartir.zip` file on the course website
- Compile the example `mathpartir.tex` in the unzipped folder with, e.g., `pdflatex`

To use it for your assignment, include `mathpartir.sty` in your tex file (i.e., `\usepackage {mathpartir}`). Alternatively, you can also modify `mathpartir.tex` directly.