Bonus Notes:
WHILE and WHILE3ADDR Reference

15-819O: Program Analysis *(Spring 2016)*
Claire Le Goues
clegoues@cs.cmu.edu

As of week four of class, we've gone over or produced syntax/semantics for both WHILE and WHILE3ADDR several times, including in class (on the board, in slides, in notes) and on the homework. For clarity, this document summarizes the syntactic and semantic rules for WHILE and WHILE3ADDR that we have covered together (i.e., those that were not part of an assignment as a question for you to answer). Most of this material is redundant (for example, you should refer to the first set of lecture notes for more complete explanations of the material), but organized in a more straightforward way for use as a reference.

Rule of thumb: Generally, when we discuss new language features, we use the rules for WHILE; when we engage in proofs about the properties of an analysis, we use the rules for WHILE3ADDR, a language we introduced explicitly to simplify elements of analysis.

1 WHILE

1.1 Syntax

These metavariables describe the syntactic categories that comprise the WHILE language:

S	statements
a	arithmetic expressions (AExp)
x, y	program variables
n	number literals
P	boolean predicates (BExp)

The syntax for WHILE is given as follows:

$$
\begin{array}{rcl@{\qquad}rcl}
S & ::= & x := a & a & ::= & x \\
 & | & \text{skip} & & | & n \\
 & | & S_1;\ S_2 & & | & a_1\ op_a\ a_2 \\
 & | & \text{if } P \text{ then } S_1 \text{ else } S_2 & op_a & ::= & + \mid - \mid * \mid / \\
 & | & \text{while } P \text{ do } S & & & \\
P & ::= & \text{true} & op_b & ::= & \text{and} \mid \text{or} \\
 & | & \text{false} & op_r & ::= & < \mid \leq \mid = \mid > \mid \geq \\
 & | & \text{not } P & & & \\
 & | & P_1\ op_b\ P_2 & & & \\
 & | & a_1\ op_r\ a_2 & & &
\end{array}
$$

1.2 Big step semantics

We use E to range over $\mathcal{E}$, denoting program state: $\mathcal{E} \in \mathit{Var} \to \mathbb{Z}$. We use $E(x)$ to mean "the value of x as stored in E" and $E[x \mapsto n]$ to mean "E with the change that x now maps to n."

$\Downarrow$ denotes a big step *judgement* over language constructs in WHILE. All language constructs are evaluated in the context of a program state E:[1]

- $e \in a$ evaluate to integers: $\langle e, E\rangle \Downarrow n$
- $p \in P$ evaluate to booleans:[2] $\langle p, E\rangle \Downarrow b$
- $S \in S$ evaluate to new states: $\langle S, E\rangle \Downarrow E'$

Rules we have seen for arithmetic expressions include:[3]

$$\frac{}{\langle n, E\rangle \Downarrow n}\ ints \qquad \frac{}{\langle x, E\rangle \Downarrow E(x)}\ vars$$

$$\frac{\langle e_1, E\rangle \Downarrow n_1 \quad \langle e_2, E\rangle \Downarrow n_2}{\langle e_1 + e_2, E\rangle \Downarrow n_1 + n_2}\ add$$

Rules we have seen for boolean expressions are:[4]

$$\frac{}{\langle \text{true}, E\rangle \Downarrow \text{true}}\ \textit{true} \qquad \frac{}{\langle \text{false}, E\rangle \Downarrow \text{false}}\ \textit{false}$$

$$\frac{\langle e_1, E\rangle \Downarrow n_1 \quad \langle e_2, E\rangle \Downarrow n_2}{\langle e_1 \leq e_2, E\rangle \Downarrow n_1 \leq n_2}\ \leq$$

$$\frac{\langle b_2, E\rangle \Downarrow \text{true} \quad \langle b_2, E\rangle \Downarrow \text{true}}{\langle b_1 \wedge b_2, E\rangle \Downarrow \texttt{false}}\ \textit{and1} \qquad \frac{\langle b_1, E\rangle \Downarrow \text{false}}{\langle b_1 \wedge b_2, E\rangle \Downarrow \text{false}}\ \textit{and2} \qquad \frac{\langle b_2, E\rangle \Downarrow \text{false}}{\langle b_1 \wedge b_2, E\rangle \Downarrow \text{false}}\ \textit{and3}$$

$$\frac{\langle b_1, E\rangle \Downarrow \text{true} \quad \langle b_2, E\rangle \Downarrow \text{true}}{\langle b_1 \vee b_2, E\rangle \Downarrow \text{true}}\ \textit{or1} \qquad \frac{\langle b_1, E\rangle \Downarrow false \quad \langle b_2, E\rangle \Downarrow b'}{\langle b_1 \vee b_2, E\rangle \Downarrow b'}\ \textit{or2}$$

We have options here, of course. For example, an alternative way to specify and is using the following two rules in place of the two I gave above:

$$\frac{\langle b_1, E\rangle \Downarrow \text{true} \quad \langle b_2, E\rangle \Downarrow b'}{\langle b_1 \wedge b_2, E\rangle \Downarrow b'}\ \text{and-alt1} \qquad \frac{\langle b_1, E\rangle \Downarrow \text{false}}{\langle b_1 \wedge b_2, E\rangle \Downarrow \text{false}}\ \text{and-alt2}$$

Rules we have seen for statements are:

$$\frac{}{\langle \texttt{skip}, E\rangle \Downarrow E}\ skip \qquad \frac{\langle e, E\rangle \Downarrow n}{\langle x := e, E\rangle \Downarrow E[x \mapsto n]}\ assignment$$

$$\frac{\langle s_1, E\rangle \Downarrow E' \quad \langle s_2, E'\rangle \Downarrow E''}{\langle s_1; s_2, E\rangle \Downarrow E''}\ seq$$

[1] We typically do not differentiate notationally between judgements over expressions and those over statements, except in terms of the results, e.g., arithmetic expressions evaluate to integers while statements evaluate to new states.

[2] We also sometimes use b or even e for particular boolean expressions in our judgements, when the context is clear.

[3] I omit multiplication and subtraction because they are blindingly obvious once you've seen addition.

[4] I only provide one relational operator rule, because the rest follow obviously.

$$\frac{\langle b, E\rangle \Downarrow \texttt{true} \quad \langle s_1, E\rangle \Downarrow E'}{\langle \texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2, E\rangle \Downarrow E'}\ \textit{if-true} \qquad \frac{\langle b, E\rangle \Downarrow \texttt{false} \quad \langle s_2, E\rangle \Downarrow E'}{\langle \texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2, E\rangle \Downarrow E'}\ \textit{if-false}$$

$$\frac{\langle b, E\rangle \Downarrow \texttt{true} \quad \langle s;\ \texttt{while}\ b\ \texttt{do}\ s, E\rangle \Downarrow E'}{\langle \texttt{while}\ b\ \texttt{do}\ s, E\rangle \Downarrow E'}\ \textit{while-true} \qquad \frac{\langle b, E\rangle \Downarrow \texttt{false}}{\langle \texttt{while}\ b\ \texttt{do}\ s, E\rangle \Downarrow E}\ \textit{while-false}$$

1.3 Small step semantics

(We didn't do a full specification of the small step semantics for WHILE *together, and focused on statements rather than expressions. I thus give only one example for expressions; the rest follow. If we do the small-step rules for statements other than if and while together at some point, I will update this document. Otherwise, they're still fair game for an assignment or exam. Consider practicing them!)*

$\rightarrow$ denotes the small step judgement over WHILE constructs. It operates over configurations, returning a new configuration. $\langle S, E\rangle \rightarrow \langle S', E'\rangle$ indicates one step of execution; $\langle S, E\rangle \rightarrow^* \langle S', E'\rangle$ indicates zero or more steps of execution. The final configuration in the small-step semantics for WHILE is $\langle \texttt{skip}, E\rangle$. To be complete, we also define auxiliary operators for arithmetic and boolean expressions. The types are thus:

$$\begin{array}{lcl} \rightarrow & : & (\texttt{Stmt} \times E) \rightarrow (\texttt{Stmt} \times E) \\ \rightarrow_a & : & (\texttt{Aexp} \times E) \rightarrow \texttt{Aexp} \\ \rightarrow_b & : & (\texttt{Bexp} \times E) \rightarrow \texttt{Bexp} \end{array}$$

Rules for arithmetic expressions include:[5]

$$\frac{}{< x, E > \rightarrow_a E(x)}\ \textit{vars} \qquad \frac{\langle a_1, E\rangle \rightarrow_a^* n_1 \quad \langle a_2, E\rangle \rightarrow_a^* n_2}{\langle a_1 + a_2, E\rangle \rightarrow_a n_1 + n_2}\ \textit{addition1}$$

Rules for statements include:

$$\frac{\langle a, E\rangle \rightarrow_a^* n}{\langle x := a, E\rangle \rightarrow \langle x := n, E\rangle}\ \textit{assignment1} \qquad \frac{}{\langle x := n, E\rangle \rightarrow \langle \text{skip}, E[x \mapsto n]\rangle}\ \textit{assignment2}$$

$$\frac{\langle b, E\rangle \rightarrow_b\ b'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2, E\rangle \rightarrow \langle \texttt{if}\ b'\ \texttt{then}\ S_1\ \texttt{else}\ S_2, E\rangle}\ \textit{ifGen}$$

$$\frac{}{\langle \texttt{if true then}\ s_1\ \texttt{else}\ s_2, E\rangle \rightarrow \langle s_1, E'\rangle}\ \textit{ifT}$$

$$\frac{}{\langle \texttt{if false then}\ s_1\ \texttt{else}\ s_2, E\rangle \rightarrow \langle s_2, E'\rangle}\ \textit{ifF}$$

$$\frac{\langle b, E\rangle \rightarrow_b b'}{\langle \texttt{while b do}\ S, E\rangle \rightarrow \langle \texttt{while}\ b'\ \texttt{do}S, E\rangle}\ \textit{whileGen}$$

$$\frac{\langle b, E\rangle \rightarrow_b \texttt{true}}{\langle \texttt{while b do}\ S, E\rangle \rightarrow \langle S; \texttt{while}\ b\ \texttt{do}S, E\rangle}\ \textit{whileT}$$

$$\frac{\langle b, E\rangle \rightarrow_b \texttt{false}}{\langle \texttt{while b do}\ S, E\rangle \rightarrow \langle \text{skip}, E\rangle}\ \textit{whileF}$$

[5] We could reduce a_1 and a_2 one at a time in the addition rule; this is a choice of the language designer.

2 WHILE3ADDR

2.1 Syntax

WHILE3ADDR is intended to be equivalent to WHILE, but easier to analyze. The syntax is:

$$
\begin{array}{rcll}
I & ::= & x := n & op ::= + \mid - \mid * \mid / \\
 & \mid & x := y & op_r ::= < \mid = \\
 & \mid & x := y\ op\ z & P \in \mathbb{N} \to I \\
 & \mid & \text{goto } n & \\
 & \mid & \text{if } x\ op_r\ 0 \text{ goto } n &
\end{array}
$$

The configuration of a WHILE3ADDR program includes (usually implicitly) the stored program P, the state environment E, and a program counter n, or $c \in E \times \mathbb{N}$.

2.2 (Small-step) Semantics

We did not formally distinguish between big- and small-step semantics for WHILE3ADDR in class; on Homework 1, I left it to you to sort out something reasonable. Given our definition of a configuration for WHILE3ADDR above, defining a big- vs. small-step semantic distinction for WHILE3ADDR isn't very informative. So, here, I give only a small-step semantics that you should be able to use in any proof of soundness I will ask you for, unless otherwise indicated (if, for example, I ask you to extend the WHILE3ADDR language with a new construct, for example, these will not suffice and you will have to extend the semantics):

$$\frac{P[n] = x := m}{P \vdash < E, n > \rightsquigarrow E[x \mapsto m], n+1}\ \textit{step-const}$$

$$\frac{P[n] = x := y}{P \vdash < E, n > \rightsquigarrow E[x \mapsto E[y]], n+1}\ \textit{step-copy}$$

$$\frac{P[n] = x := y\ op\ z \quad E[y]\ \mathbf{op}\ E[z] = m}{P \vdash < E, n > \rightsquigarrow E[x \mapsto m], n+1}\ \textit{step-arith}$$

$$\frac{P[n] = \text{goto } m}{P \vdash < E, n > \rightsquigarrow < E, m >}\ \textit{step-goto}$$

$$\frac{P[n] = \text{if } x\ op_r\ 0 \text{ goto } m \quad E[x]\ \mathbf{op_r}\ 0 = true}{P \vdash < E, n > \rightsquigarrow < E, m >}\ \textit{step-iftrue}$$

$$\frac{P[n] = \text{if } x\ op_r\ 0 \text{ goto } m \quad E[x]\ \mathbf{op_r}\ 0 = false}{P \vdash E, n \rightsquigarrow E, n+1}\ \textit{step-iffalse}$$