

CS421 Summer 2006 Final Study Guide

CS 421 — Programming Languages and Compilers
Summer Term 2006

1. Type Derivations:

- (a) Provide a full type derivation for the following term. Feel free to write on the paper sideways – the trees tend to be wider than they are tall. You can also write out type environments separately and just reference them in the rules (so you could say $\Gamma_1 = \{x : \text{int}\}$ and then just reference Γ_1 in the derivation) – this is especially useful if you use the same environment in multiple places. Typing rules are on page 11.

`let f = fun a -> (a + 9) in (f 4)`

Here, since we aren't given any starting Γ , we just assume an empty one to start. We will use $\Gamma_1 = \{f : \text{int} \rightarrow \text{int}\}$ to save space (and you can do the same on the final, if you want).

$$\begin{array}{c}
 \frac{}{\{a : \text{int}\} \vdash a : \text{int}} \quad \frac{}{\{a : \text{int}\} \vdash 9 : \text{int}} \\
 \hline
 \frac{}{\{a : \text{int}\} \vdash a + 9 : \text{int}} \\
 \hline
 \frac{}{\{\} \vdash \text{fun } a \rightarrow (a + 9) : \text{int} \rightarrow \text{int}} \\
 \hline
 \frac{}{\{\} \vdash \text{let } f = \text{fun } a \rightarrow (a + 9) \text{ in } (f 4) : \text{int}}
 \end{array}
 \qquad
 \frac{}{\Gamma_1 \vdash f : \text{int} \rightarrow \text{int}} \quad \frac{}{\Gamma_1 \vdash 4 : \text{int}}$$

- (b) Provide a full type derivation for the following term. Feel free to write on the paper sideways – the trees tend to be wider than they are tall. You can also write out type environments separately and just reference them in the rules (so you could say $\Gamma_1 = \{x : \text{int}\}$ and then just reference Γ_1 in the derivation) – this is especially useful if you use the same environment in multiple places. Typing rules are on page 11.

`let rec f = fun a -> (a < 9) in (if (f 4) then 3 else 5)`

Here, since we aren't given any starting Γ , we just assume an empty one to start. We will use $\Gamma_1 = \{f : \text{int} \rightarrow \text{bool}\}$ and $\Gamma_2 = \{f : \text{int} \rightarrow \text{bool}, a : \text{int}\}$ to save space (and you can do the same on the final, if you want). Sorry if you need to zoom the pdf...

$$\begin{array}{c}
 \frac{}{\Gamma_2 \vdash a : \text{int}} \quad \frac{}{\Gamma_2 \vdash 9 : \text{int}} \quad \frac{}{\Gamma_1 \vdash f : \text{int} \rightarrow \text{bool}} \quad \frac{}{\Gamma_1 \vdash 4 : \text{int}} \\
 \frac{}{\Gamma_2 \vdash a < 9 : \text{bool}} \quad \frac{}{\Gamma_1 \vdash f \ 4 : \text{bool}} \quad \frac{}{\Gamma_1 \vdash 3 : \text{int}} \quad \frac{}{\Gamma_1 \vdash 5 : \text{int}} \\
 \frac{}{\Gamma_1 \vdash \text{fun } a \rightarrow (a < 9) : \text{int} \rightarrow \text{bool}} \quad \frac{}{\Gamma_1 \vdash \text{if } (f \ 4) \text{ then } 3 \text{ else } 5 : \text{int}} \\
 \frac{}{\{\} \vdash \text{let rec } f = \text{fun } a \rightarrow (a < 9) \text{ in } (\text{if } (f \ 4) \text{ then } 3 \text{ else } 5) : \text{int}}
 \end{array}$$

2. Lambda Calculus

(a) Reduce the following term using lazy evaluation.

$$(\lambda a b.a b)(\lambda x y.x)((\lambda t.t)(\lambda u.u))(\lambda c.c c c)$$

$$\begin{aligned} &(\lambda a b.a b)(\lambda x y.x)((\lambda t.t)(\lambda u.u))(\lambda c.c c c) \rightarrow^\beta \\ &(\lambda b.(\lambda x y.x)b)((\lambda t.t)(\lambda u.u))(\lambda c.c c c) \rightarrow^\beta \\ &((\lambda x y.x)((\lambda t.t)(\lambda u.u)))(\lambda c.c c c) \rightarrow^\beta \\ &(\lambda y.((\lambda t.t)(\lambda u.u)))(\lambda c.c c c) \rightarrow^\beta \\ &(\lambda t.t)(\lambda u.u) \\ &\lambda u.u \end{aligned}$$

(b) Reduce the same term using eager evaluation.

$$(\lambda a b.a b)(\lambda x y.x)((\lambda t.t)(\lambda u.u))(\lambda c.c c c)$$

$$\begin{aligned} &(\lambda a b.a b)(\lambda x y.x)((\lambda t.t)(\lambda u.u))(\lambda c.c c c) \rightarrow^\beta \\ &(\lambda b.(\lambda x y.x)b)((\lambda t.t)(\lambda u.u))(\lambda c.c c c) \rightarrow^\beta \\ &(\lambda b.(\lambda x y.x)b)(\lambda u.u)(\lambda c.c c c) \rightarrow^\beta \\ &((\lambda x y.x)(\lambda u.u))(\lambda c.c c c) \rightarrow^\beta \\ &(\lambda y.(\lambda u.u))(\lambda c.c c c) \rightarrow^\beta \\ &\lambda u.u \end{aligned}$$

- (c) Recall our definition of pairs in the λ calculus – a pair can be represented as a term $\lambda x.x \ a \ b$ where x is the constructor tag (here we only have one constructor, so we only need one tag) and a and b are the two elements in the pair. Provide a λ abstraction **swappair** that takes a pair and returns a new pair with the element swapped: applying this to pair (a, b) will return pair (b, a) .

Solution: Recall that to get the first element of a pair we use the abstraction $(\lambda x y.x)$, and that to get the second element of a pair we use the abstraction $(\lambda x y.y)$. With these facts in hand, and with the pair constructor, an initial pass would be:

$$\text{swappair} \equiv \lambda p.(\lambda a \ b \ x.x \ a \ b)(p \ (\lambda x \ y.y))(p \ (\lambda x \ y.x))$$

We get the second element of the pair with $(p \ (\lambda x \ y.y))$ and the first element of the pair with $(p \ (\lambda x \ y.x))$; handing them to $(\lambda a \ b \ x.x \ a \ b)$ constructs a new pair. We can reduce this a little further as well:

$$\text{swappair} \equiv \lambda p.(\lambda x.x(p \ (\lambda x \ y.y))(p \ (\lambda x \ y.x)))$$

which is just:

$$\text{swappair} \equiv \lambda p \ x.(p \ (\lambda x \ y.y))(p \ (\lambda x \ y.x))$$

This just “slots in” the two elements of the pair. Note we cannot reduce any further without knowing what p is. Any of these three forms would be an acceptable answer.

3. **Context-Free Grammars, Derivations, and Parse Trees:** The following exercises make use of this grammar for arithmetic expressions. Here, the start symbol is E , id refers to identifiers made up of one lowercase character ($\text{a}, \text{b}, \dots, \text{y}, \text{z}$), and num refers to any integer:

$$\begin{array}{lll} E \rightarrow E + T & T \rightarrow T * F & F \rightarrow \text{id} \\ E \rightarrow E - T & T \rightarrow T / F & F \rightarrow \text{num} \\ E \rightarrow T & T \rightarrow F & F \rightarrow (E) \end{array}$$

- (a) Show a leftmost derivation for the following term:

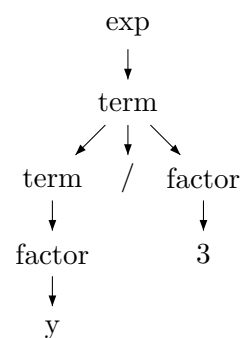
$$y/3$$

$$E \Rightarrow T \Rightarrow T/F \Rightarrow F/F \Rightarrow y/F \Rightarrow y/3$$

- (b) Show a rightmost derivation for the same term:

$$E \Rightarrow T \Rightarrow T/F \Rightarrow T/3 \Rightarrow F/3 \Rightarrow y/3$$

- (c) Provide a parse tree for the same term:



$$\begin{array}{lll}
 E \rightarrow E + T & T \rightarrow T * F & F \rightarrow \text{id} \\
 E \rightarrow E - T & T \rightarrow T / F & F \rightarrow \text{num} \\
 E \rightarrow T & T \rightarrow F & F \rightarrow (E)
 \end{array}$$

(d) Show a leftmost derivation for the following term:

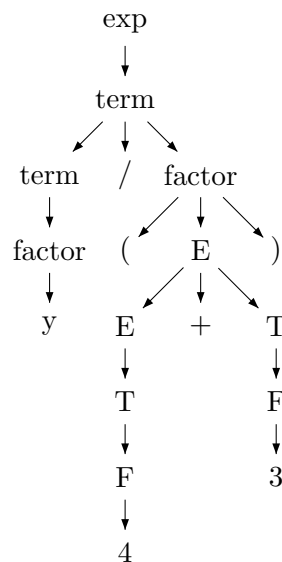
$$y/(4 + 3)$$

$$\begin{aligned}
 E \Rightarrow T \Rightarrow T/F \Rightarrow F/F \Rightarrow y/F \Rightarrow y/(E) \Rightarrow y/(E + T) \Rightarrow y/(T + T) \Rightarrow y/(F + T) \Rightarrow \\
 y/(4 + T) \Rightarrow y/(4 + F) \Rightarrow y/(4 + 3)
 \end{aligned}$$

(e) Show a rightmost derivation for the same term:

$$\begin{aligned}
 E \Rightarrow T \Rightarrow T/F \Rightarrow T/(E) \Rightarrow T/(E + T) \Rightarrow T/(E + F) \Rightarrow T/(E + 3) \Rightarrow T/(T + 3) \Rightarrow \\
 T/(F + 3) \Rightarrow T/(4 + 3) \Rightarrow F/(4 + 3) \Rightarrow y/(4 + 3)
 \end{aligned}$$

(f) Provide a parse tree for the same term:



4. Semantics:

- (a) Derivations: Using natural semantics (rules are on pages 14–15), provide a derivation for the following term:

$$\text{if } (i \leq n) \text{ then } (n := n + 1) \text{ else } (i := i + 1)$$

Assume the starting $\sigma = \{i \mapsto 1, n \mapsto 3\}$. Feel free to turn the page sideways, or to number parts of the derivation and provide them separately (so you can say #1 somewhere in the derivation and then elsewhere provide the definition for the #1 part of the derivation tree).

To save space, we will name the following σ s: $\sigma_0 = \{i \mapsto 1, n \mapsto 3\}$, $\sigma_1 = \{i \mapsto 1, n \mapsto 4\}$

$$\frac{\frac{\langle i, \sigma_0 \rangle \Downarrow 1 \quad \langle n, \sigma_0 \rangle \Downarrow 3 \quad \mathbf{true} = 1 \leq_{int} 3}{\langle i \leq n, \sigma_0 \rangle \Downarrow \mathbf{true}} \quad \frac{\frac{\langle n, \sigma_0 \rangle \Downarrow 3 \quad \langle 1, \sigma_0 \rangle \Downarrow 1 \quad 4 = 3 +_{int} 1}{\langle n + 1, \sigma_0 \rangle \Downarrow 4}}{\langle n := n + 1, \sigma_0 \rangle \Downarrow \sigma_0[4/n]}}{\langle \text{if } (i \leq n) \text{ then } (n := n + 1) \text{ else } (i := i + 1), \sigma_0 \rangle \Downarrow \sigma_1}$$

- (b) Derivations: Using transition semantics (rules are on pages 12–13), provide a derivation for the following term:

$$\mathbf{if\ (i \leq n)\ then\ (n := n + 1)\ else\ (i := i + 1)}$$

Assume the starting $\sigma = \{i \mapsto 1, n \mapsto 3\}$. Feel free to turn the page sideways, or to number parts of the derivation and provide them separately (so you can say #1 somewhere in the derivation and then elsewhere provide the definition for the #1 part of the derivation tree).

To save space, we will name the following σ s: $\sigma_0 = \{i \mapsto 1, n \mapsto 3\}$, $\sigma_1 = \{i \mapsto 1, n \mapsto 4\}$

$$\text{i. } \frac{\frac{\langle i, \sigma_0 \rangle \rightarrow \sigma_0(i) = 1}{\langle i \leq n, \sigma_0 \rangle \rightarrow \langle 1 \leq n, \sigma_0 \rangle}}{\langle \mathbf{if\ (i \leq n)\ then\ (n := n + 1)\ else\ (i := i + 1), \sigma_0} \rangle \rightarrow \langle \mathbf{if\ (1 \leq n)\ then\ (n := n + 1)\ else\ (i := i + 1), \sigma_0} \rangle}$$

$$\text{ii. } \frac{\frac{\langle n, \sigma_0 \rangle \rightarrow \sigma_0(n) = 3}{\langle 1 \leq n, \sigma_0 \rangle \rightarrow \langle 1 \leq 3, \sigma_0 \rangle}}{\langle \mathbf{if\ (1 \leq n)\ then\ (n := n + 1)\ else\ (i := i + 1), \sigma_0} \rangle \rightarrow \langle \mathbf{if\ (1 \leq 3)\ then\ (n := n + 1)\ else\ (i := i + 1), \sigma_0} \rangle}$$

$$\text{iii. } \frac{\langle 1 \leq 3, \sigma_0 \rangle \rightarrow \mathbf{true}}{\langle \mathbf{if\ (1 \leq 3)\ then\ (n := n + 1)\ else\ (i := i + 1), \sigma_0} \rangle \rightarrow \langle \mathbf{if\ true\ then\ (n := n + 1)\ else\ (i := i + 1), \sigma_0} \rangle}$$

$$\text{iv. } \langle \mathbf{if\ true\ then\ (n := n + 1)\ else\ (i := i + 1), \sigma_0} \rangle \rightarrow \langle \mathbf{n := n + 1, \sigma_0} \rangle$$

$$\text{v. } \frac{\frac{\langle n, \sigma_0 \rangle \rightarrow \sigma_0(n) = 3}{\langle n + 1, \sigma_0 \rangle \rightarrow \langle 3 + 1, \sigma_0 \rangle}}{\langle \mathbf{n := n + 1, \sigma_0} \rangle \rightarrow \langle \mathbf{n := 3 + 1, \sigma_0} \rangle}$$

$$\text{vi. } \frac{\langle 3 + 1, \sigma_0 \rangle \rightarrow 4}{\langle \mathbf{n := 3 + 1, \sigma_0} \rangle \rightarrow \langle \mathbf{n := 4, \sigma_0} \rangle}$$

$$\text{vii. } \langle \mathbf{n := 4, \sigma_0} \rangle \rightarrow \sigma_0[4/n] = \sigma_1$$

- (c) Semantics Rules: Say you are given a new language construct for a **for** statement. This adds the following to the grammar (remember, *Com* is a command, something that can change the state):

$$\text{Com } c ::= \mathbf{for } X := a_0 \mathbf{ to } a_1 \mathbf{ do } c_0 \mathbf{ done}$$

This is similar to a **while** statement, but the iteration space is constrained by the initial values specified in the loop and a new name (X) is introduced into scope which can be referenced in the loop body (c_0). The semantics should work as follows:

- i. Expression a_0 is evaluated until it yields a number n_0
- ii. Expression a_1 is evaluated until it yields a number n_1
- iii. Name X is added to the environment and assigned the value n_0
- iv. While the numeric value assigned to X is less than or equal to n_1 , execute command c_0 and then add one to X

The name X should not be available once the loop ends. Define the necessary **transition** semantics rules for this construct. You can reference the transition semantics rules for the IMP language, on pages 12–13, for guidance on how the rules should look.

- i. The first rule handles the first step above – reducing a_0 :

$$\frac{\langle a_0, \sigma \rangle \rightarrow \langle a'_0, \sigma \rangle}{\langle \mathbf{for } X := a_0 \mathbf{ to } a_1 \mathbf{ do } c_0 \mathbf{ done}, \sigma \rangle \rightarrow \langle \mathbf{for } X := a'_0 \mathbf{ to } a_1 \mathbf{ do } c_0 \mathbf{ done}, \sigma \rangle}$$

- ii. The second rule handles the second step – reducing a_1 :

$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle}{\langle \mathbf{for } X := n_0 \mathbf{ to } a_1 \mathbf{ do } c_0 \mathbf{ done}, \sigma \rangle \rightarrow \langle \mathbf{for } X := n_0 \mathbf{ to } a'_1 \mathbf{ do } c_0 \mathbf{ done}, \sigma \rangle}$$

- iii. The third rule handles the final two steps.

$$\langle \mathbf{for } X := n_0 \mathbf{ to } n_1 \mathbf{ do } c_0 \mathbf{ done}, \sigma \rangle \rightarrow \langle \mathbf{while } \mathbf{X} \leq n_1 \mathbf{ do}(c_0; \mathbf{X} := \mathbf{X} + 1), \sigma[n_0/X] \rangle$$

Note that there is one difficulty we do not take care of here – we need to remove this binding of X at the end of the loop and replace it with whatever binding of X (including undefined) was present in σ . If we don't do this we would allow the name bound in the for loop to “escape” into the surrounding code. We need to define a function of some sort to do this, but we don't bother with that here. There will not be any similar difficulties on the final (and in fact I didn't even think of this problem until after I posted this guide).

- (d) Semantics Rules: Define the necessary **natural** semantics rules for the **for** construct defined above. You can reference the natural semantics rules for the IMP language, on pages 14–15, for guidance on how the rules should look.

$$\frac{\langle a_0, \sigma \rangle \Downarrow n_0 \quad \langle a_1, \sigma \rangle \Downarrow n_1 \quad \langle \mathbf{while} \ \mathbf{X} \leq n_1 \ \mathbf{do} \ (c_0; \mathbf{X} := \mathbf{X} + 1), \sigma[n_0/\mathbf{X}] \rangle \Downarrow \sigma'}{\langle \mathbf{for} \ \mathbf{X} := a_0 \ \mathbf{to} \ a_1 \ \mathbf{do} \ c_0 \ \mathbf{done}, \sigma \rangle \Downarrow \sigma''}$$

where σ'' is equal to σ' except for \mathbf{X} , where it is equal to $\sigma(\mathbf{X})$, which may be undefined. I believe in class I had σ' instead of $\sigma[n_1/\mathbf{X}]$, and then σ'' instead of σ' and σ''' instead of σ'' . Both are fine – here it just seems easier to directly write in $\sigma[n_0/\mathbf{X}]$ instead of defining it first as $\sigma' = \sigma[n_0/\mathbf{X}]$.

Rules for type derivations:**Constants**

$$\frac{}{\vdash n : \text{int}} \text{(assuming } n \text{ is an int)}$$

$$\frac{}{\vdash \text{true} : \text{bool}}$$

$$\frac{}{\vdash \text{false} : \text{bool}}$$

Variables

$$\frac{}{\Gamma \vdash x : \tau} \text{if } (x : \tau) \in \Gamma$$

Arithmetic Operators

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \oplus e_2 : \text{int}} \quad (\oplus \in \{+, -, *, /, \dots\})$$

Relational Operators

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \sim e_2 : \text{bool}} \quad (\sim \in \{<, >, \leq, \geq, =, \neq, \dots\})$$

Booleans

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \ \&\& \ e_2 : \text{bool}}$$

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \ || \ e_2 : \text{bool}}$$

$$\frac{\Gamma \vdash e_1 : \text{bool}}{\Gamma \vdash ! e_1 : \text{bool}}$$

If

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \text{if } e_1 \ \text{then } e_2 \ \text{else } e_3 : \tau}$$

Function Abstraction

$$\frac{\Gamma \cup [x : \tau_1] \vdash e : \tau_2}{\Gamma \vdash \text{fun } x \rightarrow e : \tau_1 \rightarrow \tau_2}$$

Function Application

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 \ e_2 : \tau_2}$$

Let

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \cup [x : \tau] \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \ \text{in } e_2 : \tau'}$$

Letrec

$$\frac{\Gamma \cup [x : \tau] \vdash e_1 : \tau \quad \Gamma \cup [x : \tau] \vdash e_2 : \tau'}{\Gamma \vdash \text{let rec } x = e_1 \ \text{in } e_2 : \tau'}$$

Transition Semantics, Dynamic Semantics Rules:

<i>Aexp</i>	$a ::=$	$n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$
<i>Bexp</i>	$b ::=$	true \mid false \mid $a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \wedge b_1 \mid b_0 \vee b_1$
<i>Com</i>	$c ::=$	skip \mid $X := a \mid c_0; c_1 \mid$ if b then c_0 else $c_1 \mid$ while b do c

Identifiers

$$\langle X, \sigma \rangle \rightarrow \sigma(X)$$

Constants

$$\langle n, \sigma \rangle \rightarrow n$$

$$\langle \mathbf{true}, \sigma \rangle \rightarrow \mathbf{true}$$

$$\langle \mathbf{false}, \sigma \rangle \rightarrow \mathbf{false}$$

Arithmetic Operators, $op \in \{+, -, \times\}$

$$\frac{\langle a_0, \sigma \rangle \rightarrow \langle a'_0, \sigma \rangle}{\langle a_0 \text{ op } a_1, \sigma \rangle \rightarrow \langle a'_0 \text{ op } a_1, \sigma \rangle}$$

$$\langle a_0 \text{ op } a_1, \sigma \rangle \rightarrow \langle a'_0 \text{ op } a_1, \sigma \rangle$$

$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle}{\langle n_0 \text{ op } a_1, \sigma \rangle \rightarrow \langle n_0 \text{ op } a'_1, \sigma \rangle}$$

$$\langle n_0 \text{ op } a_1, \sigma \rangle \rightarrow \langle n_0 \text{ op } a'_1, \sigma \rangle$$

$$\langle n_0 \text{ op } n_1, \sigma \rangle \rightarrow n, \text{ where } n = n_0 \text{ op}_{int} n_1$$

Relational Operators, $op \in \{=, \leq\}$

$$\frac{\langle a_0, \sigma \rangle \rightarrow \langle a'_0, \sigma \rangle}{\langle a_0 \text{ op } a_1, \sigma \rangle \rightarrow \langle a'_0 \text{ op } a_1, \sigma \rangle}$$

$$\langle a_0 \text{ op } a_1, \sigma \rangle \rightarrow \langle a'_0 \text{ op } a_1, \sigma \rangle$$

$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle}{\langle n_0 \text{ op } a_1, \sigma \rangle \rightarrow \langle n_0 \text{ op } a'_1, \sigma \rangle}$$

$$\langle n_0 \text{ op } a_1, \sigma \rangle \rightarrow \langle n_0 \text{ op } a'_1, \sigma \rangle$$

$$\langle n_0 \text{ op } n_1, \sigma \rangle \rightarrow b, \text{ where } b = (n_0 \text{ op}_{int} n_1)$$

Logical Operators (\wedge, \vee and \neg are similar)

$$\frac{\langle b_0, \sigma \rangle \rightarrow \langle b'_0, \sigma \rangle}{\langle b_0 \wedge b_1, \sigma \rangle \rightarrow \langle b'_0 \wedge b_1, \sigma \rangle}$$

$$\langle b_0 \wedge b_1, \sigma \rangle \rightarrow \langle b'_0 \wedge b_1, \sigma \rangle$$

$$\frac{\langle b_1, \sigma \rangle \rightarrow \langle b'_1, \sigma \rangle}{\langle \mathbf{true} \wedge b_1, \sigma \rangle \rightarrow \langle b'_1, \sigma \rangle}$$

$$\langle \mathbf{true} \wedge b_1, \sigma \rangle \rightarrow \langle b'_1, \sigma \rangle$$

$$\langle \mathbf{false} \wedge b_1, \sigma \rangle \rightarrow \mathbf{false}$$

Skip

$$\langle \mathbf{skip}, \sigma \rangle \rightarrow \sigma$$

Assignment

$$\frac{\langle a, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle X := a, \sigma \rangle \rightarrow \langle X := a', \sigma \rangle}$$

$$\langle X := n, \sigma \rangle \rightarrow \sigma[n/X]$$

Sequencing

$$\frac{\langle c_0, \sigma \rangle \rightarrow \langle c'_0, \sigma' \rangle}{\langle c_0; c_1, \sigma \rangle \rightarrow \langle c'_0; c_1, \sigma' \rangle}$$

$$\frac{\langle c_0, \sigma \rangle \rightarrow \sigma'}{\langle c_0; c_1, \sigma \rangle \rightarrow \langle c_1, \sigma' \rangle}$$

If

$$\frac{\langle b, \sigma \rangle \rightarrow \langle b', \sigma \rangle}{\langle \text{if } b \text{ then } c_0 \text{ else } c_1, \sigma \rangle \rightarrow \langle \text{if } b' \text{ then } c_0 \text{ else } c_1, \sigma \rangle}$$

$$\langle \text{if true then } c_0 \text{ else } c_1, \sigma \rangle \rightarrow \langle c_0, \sigma \rangle$$

$$\langle \text{if false then } c_0 \text{ else } c_1, \sigma \rangle \rightarrow \langle c_1, \sigma \rangle$$

While

$$\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \langle \text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}, \sigma \rangle$$

Natural Semantics, Dynamic Semantics Rules:

<i>Aexp</i>	$a ::=$	$n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$
<i>Bexp</i>	$b ::=$	$\mathbf{true} \mid \mathbf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \wedge b_1 \mid b_0 \vee b_1$
<i>Com</i>	$c ::=$	$\mathbf{skip} \mid X := a \mid c_0; c_1 \mid \mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1 \mid$ $\mathbf{while } b \mathbf{ do } c$

Identifiers

$$\langle X, \sigma \rangle \Downarrow \sigma(X)$$

Constants

$$\langle n, \sigma \rangle \Downarrow n$$

$$\langle \mathbf{true}, \sigma \rangle \Downarrow \mathbf{true}$$

$$\langle \mathbf{false}, \sigma \rangle \Downarrow \mathbf{false}$$

Arithmetic Operators

$$\frac{\langle a_0, \sigma \rangle \Downarrow n_0 \quad \langle a_1, \sigma \rangle \Downarrow n_1 \quad n = n_0 \text{ op}_{int} n_1}{\langle a_0 \text{ op } a_1, \sigma \rangle \Downarrow n} \quad (op \in \{+, -, \times\})$$

Relational Operators

$$\frac{\langle a_0, \sigma \rangle \Downarrow n_0 \quad \langle a_1, \sigma \rangle \Downarrow n_1 \quad b = n_0 \text{ op}_{int} n_1}{\langle a_0 \text{ op } a_1, \sigma \rangle \Downarrow b} \quad (op \in \{=, \leq\})$$

Logical Operators

$$\frac{\langle b_0, \sigma \rangle \Downarrow \mathbf{false}}{\langle b_0 \wedge b_1, \sigma \rangle \Downarrow \mathbf{false}}$$

$$\langle b_0 \wedge b_1, \sigma \rangle \Downarrow \mathbf{false}$$

$$\frac{\langle b_0, \sigma \rangle \Downarrow \mathbf{true} \quad \langle b_1, \sigma \rangle \Downarrow b}{\langle b_0 \wedge b_1, \sigma \rangle \Downarrow b}$$

$$\langle b_0 \wedge b_1, \sigma \rangle \Downarrow b$$

$$\frac{\langle b_0, \sigma \rangle \Downarrow \mathbf{true}}{\langle b_0 \vee b_1, \sigma \rangle \Downarrow \mathbf{true}}$$

$$\langle b_0 \vee b_1, \sigma \rangle \Downarrow \mathbf{true}$$

$$\frac{\langle b_0, \sigma \rangle \Downarrow \mathbf{false} \quad \langle b_1, \sigma \rangle \Downarrow b}{\langle b_0 \vee b_1, \sigma \rangle \Downarrow b}$$

$$\langle b_0 \vee b_1, \sigma \rangle \Downarrow b$$

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{false}}{\langle \neg b, \sigma \rangle \Downarrow \mathbf{true}}$$

$$\langle \neg b, \sigma \rangle \Downarrow \mathbf{true}$$

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{true}}{\langle \neg b, \sigma \rangle \Downarrow \mathbf{false}}$$

$$\langle \neg b, \sigma \rangle \Downarrow \mathbf{false}$$

Skip

$$\{\mathbf{skip}, \sigma\} \Downarrow \sigma$$

Assignment

$$\frac{\langle a, \sigma \rangle \Downarrow n}{\langle X := a, \sigma \rangle \Downarrow \sigma[n/X]}$$

Sequencing

$$\frac{\langle c_0, \sigma \rangle \Downarrow \sigma' \quad \langle c_1, \sigma' \rangle \Downarrow \sigma''}{\langle c_0; c_1, \sigma \rangle \Downarrow \sigma''}$$

If

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{true} \quad \langle c_0, \sigma \rangle \Downarrow \sigma'}{\langle \mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1, \sigma \rangle \Downarrow \sigma'}$$

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{false} \quad \langle c_1, \sigma \rangle \Downarrow \sigma'}{\langle \mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1, \sigma \rangle \Downarrow \sigma'}$$

While

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{false}}{\langle \mathbf{while } b \mathbf{ do } c, \sigma \rangle \Downarrow \sigma}$$

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{true} \quad \langle c, \sigma \rangle \Downarrow \sigma' \quad \langle \mathbf{while } b \mathbf{ do } c, \sigma' \rangle \Downarrow \sigma''}{\langle \mathbf{while } b \mathbf{ do } c, \sigma \rangle \Downarrow \sigma''}$$