Denotational Semantics; Divergence; Runtime Errors

CS 536: Science of Programming, Fall 2019

A. Why

- Our simple programming language is a model for the kind of constructs seen in actual languages.
- Execution of an entire programs can be viewed as a state transformers.
- Infinite loops and runtime errors cause failure of normal program execution.

B. Outcomes

At the end of today, you should know how to

- Use denotational semantics to describe overall execution of programs in our language
- Determine that evaluation of an expression or program fails due to a runtime error.

C. Denotational Semantics Definition and Rules

- In addition to the small step-by-step operational semantics for our programs, we'll also introduce a version of semantics that concentrates only on the beginning and end of the evaluation process (hence he name "large-step" semantics).
- **Definition**: The **denotational semantics** of S in σ is τ if in state σ, program S terminates in τ. (I.e., $\langle S, \sigma\rangle \rightarrow^* \langle E, \tau\rangle$.) Symbolically, we write $M(S, \sigma) = \{\tau\}$.
    - The reason we have a singleton set containing τ instead of just τ is that later, we'll look at non-deterministic computations, which can have more than one possible final state.
- **Notation**: If you slip up and write $M(S, \sigma) = \tau$ instead of $\{\tau\}$, it's not a big deal.
- **Example 1**: Let σ be a state and let $S \equiv \mathtt{x := 1\ ;\ y := 2}$. Since $\langle \mathtt{x := 1\ ;\ y := 2}, \sigma\rangle \rightarrow \langle \mathtt{y := 2}, \sigma[\mathtt{x} \mapsto 1]\rangle \rightarrow \langle E, \sigma[\mathtt{x} \mapsto 1][\mathtt{y} \mapsto 2]\rangle$, we know $M(S, \sigma) = \{\sigma[\mathtt{x} \mapsto 1][\mathtt{y} \mapsto 2]\}$.
- **Notation**: In the literature, some people write hollow square brackets around arguments that are syntactic to emphasize that they are indeed syntactic. Other notations for $M(S, \sigma)$ include $M[\![S]\!](\sigma)$ and $M[\![S]\!]\sigma$ and $M(S)(\sigma)$. In the last two cases, $M[\![S]\!]$ and $M(S)$ are viewed as functions that transform memory state, so $M[\![S]\!](\sigma) = \tau$ means $M[\![S]\!]$ maps σ to τ.

Denotational Semantics Rules

- Since $M(S, \sigma) = \tau$ means $\langle S, \sigma\rangle \rightarrow^* \langle E, \tau\rangle$, we can give specific rules for $M(S, \sigma)$ depending on the kind of S.
- **Skip and Assignment:** These statements complete in only one step, so the operational semantics rules give the denotational semantics immediately.
    - $M(\mathbf{skip}, \sigma) = \{\sigma\}$
    - $M(v := e, \sigma) = \{\sigma[v \mapsto \sigma(e)]\}$

  - $M(\mathtt{b}[e_1] := e, \sigma) = \{\sigma[\mathtt{b}[\alpha] \mapsto \beta]\}$ where $\alpha = \sigma(e_1)$ and $\beta = \sigma(e)$.
- **Composition**: $M(S_1\mathtt{;}\ S_2, \sigma) = M(S_2, \tau)$ where $\{\tau\} = M(S_1, \sigma)$. To justify this, say we have $\langle S_1\mathtt{;}\ S_2, \sigma\rangle \to^* \langle S_2, \tau\rangle \to^* \langle E, \tau'\rangle$. Since $M(S_1, \sigma) = \{\tau\}$, we run S_2 starting in state τ, so $M(S_1\mathtt{;}\ S_2, \sigma) = M(S_2, \tau) = M(S_2, M(S_1, \sigma))$.
  - Note the subscripts in $S_1\mathtt{;}\ S_2$ are 1 then 2 but the subscripts in $M(S_2, M(S_1, \sigma))$ are 2 then 1.
- **Notation**: We'll bend the notation a bit and write $M(S_2, M(S_1, \sigma))$ to mean $M(S_2, \tau)$ where $\{\tau\} = M(S_1, \sigma)$.
- **Conditional**: The meaning of an **`if-else`** statement is either the meaning of the true branch or the meaning of the false branch.
  - If $\sigma(B) = T$, then $M(\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}, \sigma) = M(S_1, \sigma)$
  - If $\sigma(B) = F$, then $M(\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}, \sigma) = M(S_2, \sigma)$
- **Example 2**: Let $S \equiv \mathbf{if}\ \mathtt{y}\ \mathbf{then}\ \mathtt{x} := \mathtt{x+1}\ \mathbf{else}\ \mathtt{z} := \mathtt{x+2}\ \mathbf{fi}$, then
  - If $\sigma(\mathtt{y}) = T$, then $M(S, \sigma) = \{\sigma[\mathtt{x} \mapsto \sigma(\mathtt{x})+1]\}$
  - If $\sigma(\mathtt{y}) = F$, then $M(S, \sigma) = \{\sigma[\mathtt{z} \mapsto \sigma(\mathtt{x})+2]\}$
- **Iterative**: One way to definite the meaning of $W \equiv \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}$ is recursively:
  - If $\sigma(B) = F$ then $M(W, \sigma) = \{\sigma\}$
  - If $\sigma(B) = T$ then $M(W, \sigma) = M(S\mathtt{;}\ W, \sigma) = M(W, M(S, \sigma))$.
  - Unfortunately, this definition is not well-formed if W leads to an infinite loop.
- Another way to characterize $M(W, \sigma)$ involves looking at the series of states in which we evaluate the test.
  - Let $\sigma_0 = \sigma$, and for for $k = 0, 1, \ldots$, let $\sigma_{k+1} = M(S, \sigma_k)$. Then $\sigma_0, \sigma_1, \sigma_2, \ldots$ is the sequence of states seen at successive **while** loop tests: σ_k is the state in effect the k'th time we evaluate the loop test.
  - Then $M(W, \sigma)$ is the (set containing the) first state in this sequence that satisfies $\neg B$, assuming there is such a state. (If there isn't, we have an infinite loop.)
- **Example 3**: Let $W \equiv \mathbf{while}\ \mathtt{x} < \mathtt{n}\ \mathbf{do}\ S\ \mathbf{od}$, where the loop body $S \equiv \mathtt{x} := \mathtt{x+1}\mathtt{;}\ \mathtt{y} := \mathtt{y+y}$. The general case for the behavior of S is (for any τ), $M(S, \tau[\mathtt{x} \mapsto \alpha][\mathtt{y} \mapsto \beta]) = \{\tau[\mathtt{x} \mapsto \alpha+1][\mathtt{y} \mapsto 2\beta]\}$. Say we start execution of W in state $\sigma = \{\mathtt{x} = 0, \mathtt{n} = 3, \mathtt{y} = 1\}$. Our sequence of states is
  - $\sigma_0 = \sigma = \{\mathtt{x} = 0, \mathtt{n} = 3, \mathtt{y} = 1\}$
  - $M(S, \sigma_0) = \{\sigma_1\}$ where $\sigma_1 = \{\mathtt{x} = 1, \mathtt{n} = 3, \mathtt{y} = 2\}$
  - $M(S, \sigma_1) = \{\sigma_2\}$ where $\sigma_2 = \{\mathtt{x} = 2, \mathtt{n} = 3, \mathtt{y} = 4\}$, and
  - $M(S, \sigma_2) = \{\sigma_3\}$ where $\sigma_3 = \{\mathtt{x} = 3, \mathtt{n} = 3, \mathtt{y} = 8\}$.
  - Of this sequence, σ_3 is the first state that satisfies $\mathtt{x} \geq \mathtt{n}$, so $M(W, \sigma) = \{\sigma_3\} = \{\{\mathtt{x} = 3, \mathtt{n} = 3, \mathtt{y} = 8\}\}$.

D. Convergence and Divergence of Loops

- Not all loops terminate. Evaluation of an infinite loop yields an unending path of → steps: Either an infinite sequence of different configurations or a finite-length cycle of configurations. More generally in computer science we can also also have infinite recursion, which we won't study in detail but is treated similarly to infinite iteration.
- **Definition**: Execution of S starting in σ **diverges** if it doesn't converge; i.e., $\langle S, \sigma\rangle \to^* \langle E, \tau\rangle$ for no τ.
- **Notation**: $M(S, \sigma) = \{\perp_d\}$ ("**bottom sub-d**") means S diverges in σ. Note that although we're writing it in a place where you'd expect a memory state, $\perp_d$ is not an actual memory state; we'll call it a **pseudo-state** as apposed to an **actual** or **real** memory state like σ and τ.
    - **Note**: Divergence is one way in which a program doesn't successfully terminate. We'll introduce other flavors of $\perp$ as we look at other ways to not get successful termination.
- To determine when $M(W, \sigma) = \{\perp_d\}$, recall that in the previous section we looked at the series of states $\sigma_0, \sigma_1, \sigma_2, \ldots$ in which we evaluate the loop test. For this sequence, $\sigma_0 = \sigma$, and $\sigma_{k+1} = M(S, \sigma_k)$ for $k \geq 0$. For terminating loops, $M(W, \sigma)$ is the first state in the sequence that satisfies $\neg B$. We can now write $M(W, \sigma) = \{\perp_d\}$ to indicate that no state in the sequence satisfies $\neg B$.
- **Example 4**: Let $W \equiv$ **`while`** `T` **`do skip od`** and σ be any state. Then $\langle W, \sigma\rangle \to \langle \textbf{skip ;}\ W, \sigma\rangle$ but $\langle \textbf{skip ;}\ W, \sigma\rangle \to \langle W, \sigma\rangle$. (As a directed graph, this is a two-node cycle, $\langle W, \sigma\rangle \rightleftarrows \langle \textbf{skip ;}\ W, \sigma\rangle$.) Hence $M(W, \sigma) = \{\perp_d\}$.
- **Example 5**: Let $W \equiv \textbf{while}\ \texttt{x} \neq \texttt{n}\ \textbf{do}\ \texttt{x} := \texttt{x-1}\ \textbf{od}$ and let $\sigma = \{\texttt{x} = -1, \texttt{n} = 0\}$.
    - Let $\sigma_0 = \sigma = \{\texttt{x} = -1, \texttt{n} = 0\}$
    - Let $\{\sigma_1\} = M(\texttt{x} := \texttt{x-1}, \sigma_0) = \{\sigma_0[\texttt{x} \mapsto -2]\} = \{\{\texttt{x} = -2, \texttt{n} = 0\}\}$
    - Let $\{\sigma_2\} = M(\texttt{x} := \texttt{x-1}, \sigma_1) = \{\sigma_1[\texttt{x} \mapsto -2]\} = \{\{\texttt{x} = -3, \texttt{n} = 0\}\}$
    - In general, let $\{\sigma_{k+1}\} = M(\texttt{x} := \texttt{x-1}, \sigma_k) = \{\{\texttt{x} = -k-2, \texttt{n} = 0\}\}$
    - Since every $\sigma_k \models \texttt{x} \neq \texttt{n}$, we have $M(W, \sigma) = \{\perp_d\}$.

E. Expressions With Runtime Errors

- Using $\perp_d$ lets us talk about a program not successfully terminating because it simply doesn't terminate at all.
- Runtime errors cause a program to terminate, but unsuccessfully E.g, in σ, the assignment `z := x/y` fails if σ(y) = 0 because evaluation of σ(`x/y`) fails. There are two notions of failure here: The expression fails, and this causes the statement to fail.
- **Definition**: $\sigma(e) = \perp_e$ means evaluation of e in state σ causes a runtime error.

- Here, $\bot_e$ is used as a pseudo-value of an expression, to indicate an error. It's not a value (though if you want to think of it as one, there's no real harm).
- If e can fail at runtime, then instead of $\sigma(e) \in V$ for some set of values V, we now have $\sigma(e) \in V \cup \{\bot_e\}$. Of course, some expressions never fail: $\sigma(\texttt{2+2}) \in \mathbb{Z} \cup \{\bot_e\}$ but more specifically, $\sigma(\texttt{2+2}) \in \mathbb{Z}$.

- **Primary errors**: The primitive values and operations being supported determines what basic runtime errors can occur. For us, let's include:
  - *Array index out of bounds*: $\sigma(b[e]) = \bot_e$ if $\sigma(e) < 0$ or $\geq \sigma(\texttt{size}(b))$; similar for multiple dimensions.
  - *Division by zero*: $\sigma(e_1 / e_2) = \sigma(e_1 \,\%\, e_2) = \bot_e$ if $\sigma(e_2) = 0$.
  - *Square root of negative number*: $\sigma(\texttt{sqrt}(e)) = \bot_e$ if $\sigma(e) < 0$.
- **Example 6**: `b[-1]`, `n/0`, and `sqrt(-1)` fail for all σ. `b[k]` fails in state {`b` = (2, 3, 5, 8), `k` = 4} but not in state {`b` = (6), `k` = 0}
- **Hereditary Failure**: If evaluating a subexpression fails, then the overall expression fails.
  - If *op* is a unary operator, then $\sigma(op\ e) = \bot_e$ if $\sigma(e) = \bot_e$.
  - If *op* is a binary operator, then $\sigma(e_1\ op\ e_2) = \bot_e$ if $\sigma(e_1)$ or $\sigma(e_2) = \bot_e$.
  - For a conditional expression, $\sigma(\textbf{if}\ B\ \textbf{then}\ e_1\ \textbf{else}\ e_2\ \textbf{fi}) = \bot_e$ if one of the following three situations occurs: (1) $\sigma(B) = \bot_e$ (2) $\sigma(B) = T$ and $\sigma(e_1) = \bot_e$ or (3) $\sigma(B) = F$ and $\sigma(e_2) = \bot_e$. We don't worry about a hypothetical failure of the branch we don't evaluate.
- **Example 7**: $\sigma(\texttt{x/y}) = \bot_e$ when $\sigma(\texttt{y}) = 0$, but $\sigma(\texttt{y = 0 ? 0 : x/y})$ never $= \bot_e$.

F. Statements With Runtime Errors

- An expression that causes a runtime error causes the statement it appears in terminate unsuccessfully. We'll write $\langle S, \sigma\rangle \to \langle E, \bot_e\rangle$ for the operational semantics of such a statement. This use of $\bot_e$ as a (pseudo)-state is different from its use as a pseudo-value ($\sigma(e) = \bot_e$).
- **Definition** (Statements with expressions with runtime errors) If a statement evaluates an expression that causes a runtime error, then the statement terminates unsuccessfully. To the operational semantics, we add:
  - If $\sigma(e) = \bot_e$, then $\langle v := e, \sigma\rangle \to \langle E, \bot_e\rangle$.
  - If $\sigma(e_1)$ or $\sigma(e_2) = \bot_e$, then $\langle b[e_1] := e_2, \sigma\rangle \to \langle E, \bot_e\rangle$
  - If $\sigma(B) = \bot_e$, then $\langle \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}, \sigma\rangle \to \langle E, \bot_e\rangle$
  - If $\sigma(B) = \bot_e$, then $\langle \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}, \sigma\rangle \to \langle E, \bot_e\rangle$
  - If $\langle S_1, \sigma\rangle \to \langle T_1, \bot_e\rangle$ then $\langle S_1\ ;\ S_2, \sigma\rangle \to \langle E, \bot_e\rangle$where T_1 either is a statement or E
- The pseudo-states $\bot_d$ and $\bot_e$ share some properties, so it's helpful to have a more general notation for "error".

- **Notation**: $\perp$ refers generically to $\perp_d$ and/or $\perp_e$ (with the context hopefully making it clear which).
- **Notation**: To make $\perp_d$ more analogous to $\perp_e$, we'll write $\langle S, \sigma\rangle \to^* \langle E, \perp_d\rangle$ to mean that evaluation of S in σ never terminates. Thus $\langle S, \sigma\rangle \to^* \langle E, \perp\rangle$ means that evaluation of S in σ does not terminate successfully.
- **Notation**: $\perp \in M(S, \sigma)$ means $\langle S, \sigma\rangle \to^* \langle E, \perp\rangle$. (Here, $\perp$ can be $\perp_d$ or $\perp_e$.)
  - Since we are writing $\perp$ in some of the places where an actual memory state would appear, it's good to be thorough, look at the other places states appear, and extend those notions or notations.
- Errors are not actual memory states or actual values, so we define
  - $M(S, \perp) = \{\perp\}$
  - $\sigma[v \mapsto \perp] = \perp$. Also, $\perp[v \mapsto \alpha] = \perp$.
  - If $\sigma = \perp$, then $\sigma(e) = \perp$
  - If $\langle S_1, \sigma\rangle \to \langle E, \perp\rangle$, then $\langle S_1\texttt{;}\ S_2, \sigma\rangle \to \langle E, \perp\rangle$ [as we saw above]
- From this last definition, it follows that
  - If $M(S_1, \sigma) = \{\perp\}$, then $M(S_1\texttt{;}\ S_2, \sigma) = M(S_2, M(S_1, \sigma)) = M(S_2, \perp) = \{\perp\}$
  - Also, if $M(S_1, \sigma) = \{\perp\}$, then if $W \equiv \textbf{while}\ B\ \textbf{do}\ S_1\ \textbf{od}$ and $\sigma(B) = \text{T}$, then $M(W, \sigma) = M(S_1\ \texttt{;}\ W, \sigma) = M(W, M(S_1, \sigma)) = M(W, \perp) = \{\perp\}$.
- **Errors and Satisfaction / Validity of predicates**: $\perp$ never satisfies a predicate: $\perp \nvDash p$ for all p, even if $p \equiv$ the constant $\texttt{T}$. In general, we now have three possibilities: $\sigma \vDash p$, $\sigma \vDash \neg p$, or $\sigma = \perp$. So $\sigma \nvDash p$ is now equivalent to ($\sigma \vDash \neg p$ or $\sigma = \perp$), not just $\sigma \vDash \neg p$. We can also have $\sigma \nvDash p$ and $\sigma \nvDash \neg p$ simultaneously (when $\sigma = \perp$).
  - Since $\sigma \vDash \neg p$ is no longer equivalent to $\sigma \nvDash p$, we need a better notion of what $\neg p$ means.
  - The solution is to treat $\neg p$ as shorthand for $p \to \texttt{F}$ where $\texttt{F}$ is the predicate "false".
    - We can define the meaning of F by saying that $\sigma \nvDash \texttt{F}$ for all σ. Defining $\texttt{F} \equiv \texttt{0} \neq \texttt{0}$ is another approach.
    - It's straightforward to show properties like $\sigma \vDash \neg\texttt{F}$ iff $\sigma \neq \perp$.
  - The other problem to worry about is what to do if evaluation of a predicate causes an error?
    - Clearly, we can't allow things like $\{\texttt{y}=0\} \vDash \texttt{y/y} = \texttt{1}$.
  - To handle this, we'll add $\perp$ to the semantics of basic operations and tests:
    - For any *relation* (like less than, etc), we have (α *relation* β) yields* $\perp$ if α or $\beta = \perp$.

* I'm using "yields" here for "semantically evaluates to". Using "equals" or "=" might get confused with "semantically equal to".

  - For any binary *operation* (like addition, etc), we have (α *operation* β) yields $\bot$ if α or $\beta = \bot$.
  - Similarly for a unary operation *op*, we have (*op* α) yields $\bot$ if $\alpha = \bot$.
- Some of the implications of this are reasonably intuitive: ($\bot$ plus one) yields $\bot$.
- But some implications are less intuitive: $\bot$ is also the result of $\bot \neq 2$ (e.g), $\bot < \bot$, $\bot = \bot$, and $\bot \neq \bot$.
- Returning to `y/y = 1`, we still have $\sigma \vDash$ `y/y = 1` iff σ(`y/y`) = σ(`1`) iff (σ(`y`) *divided by* σ(`y`)) = *one*, so
  - If σ(`y`) = some $\alpha \neq 0$, then $\sigma \vDash$ `y/y = 1` iff (α *divided by* α = *one*) iff (*one* = *one*) iff *true*
  - But if σ(`y`) = 0, then $\sigma \vDash$ `y/y = 1` iff (0 *divided by* 0 = *one*) iff ($\bot$ = *one*) iff $\bot$.
  - Thus $\sigma \nvDash$ `y/y = 1` and similarly (since ($\bot \neq$ *one*) yields $\bot$), $\sigma \nvDash$ `y/y ≠ 1`.

Denotational Semantics; Divergence; Runtime Errors

CS 536: Science of Programming, Fall 2019

A. Why

- Our simple programming language is a model for the kind of constructs seen in actual languages.
- Our programs stand for state transformers.
- Runtime errors cause failure of normal program execution.

B. Outcomes

At the end of today, you should be able to

- Give the denotational semantics of a program in a state.
- Say when and how evaluation of an expression or program fails due to a runtime error.

C. Problems

Problems 1–4 are the denotational versions of the similar questions from Activity 5

1. What is
   a. $M(\mathtt{x} := \mathtt{x}+1, \{\mathtt{x}=5\})$?
   b. $M(\mathtt{x} := \mathtt{x}+1, \sigma)$? (Your answer will be symbolic.)
   c. $\langle \mathtt{x} := \mathtt{x}+1\,\mathtt{;}\ \mathtt{y} := 2*\mathtt{x}, \{\mathtt{x}=5\}\rangle$?

2. Let $S \equiv$ **if** $\mathtt{x} > 0$ **then** $\mathtt{x} := \mathtt{x}+1$ **else** $\mathtt{y} := 2*\mathtt{x}$ **fi**.
   a. Let $\sigma(\mathbf{x}) = 8$. What is $M(S,\sigma)$?
   b. Repeat, if $\sigma(\mathbf{x}) = 0$.
   c. Repeat, if we don't know what $\sigma(\mathbf{x})$ is. (Your answer will be symbolic.)

3. Let $S \equiv$ **if** $\mathtt{x} > 0$ **then** $\mathtt{x} := \mathtt{x}/\mathtt{z}$ **fi**.
   a. What is $M(S,\sigma)$ if $\sigma = \{\mathtt{x}=8, \mathtt{z}=3\}$? (Don't forget, integer division truncates)
   b. What is $M(S, \{\mathtt{x}=-2, \mathtt{z}=3\})$?

4. Let $W \equiv$ **while** $\mathtt{x} < 3$ **do** S **od** where $S \equiv \mathtt{x} := \mathtt{x}+1\,\mathtt{;}\ \mathtt{y} := \mathtt{y}*\mathtt{x}$.
   a. Evaluate the body S in an arbitrary state τ and give $M(S,\tau)$.
   b. What is $M(W,\sigma)$ if $\sigma \vDash \mathtt{x} = 4 \wedge \mathtt{y} = 1$?
   c. What is $M(W,\sigma)$ if where $\sigma \vDash \mathtt{x} = 1 \wedge \mathtt{y} = 1$?

5. Let $S \equiv$ `x` := `y/b[x]` and let $\sigma = \{$`b` $= (3, 0, -2, 4)$, `x` $= \alpha$, `y` $= 13\}$. Find all α such that $M(S, \sigma) = \{\perp_e\}$. (Remember, integer division truncates.)

6. Repeat the previous problem on $S \equiv$ `y` := `y / sqrt(b[x])` and $\sigma = \{$`b` $= (-1, 9, 12, 0)$, `x` $= \alpha$, `y` $= 8\}$. Treat `sqrt` as returning the truncated integer square root of its argument. (I.e., `sqrt`(0) = 0, `sqrt` of 1, 2, and 3 all = 1, `sqrt` of 4 through 8 = 2, etc.)

Solution to Activity 6 (Denotational Semantics; Divergence; Runtime Errors)

1. (Calculate meanings of programs)

   a. $M(\mathtt{x:=x+1}, \{\mathtt{x=5}\}) = \{\{\mathtt{x=5}\}[\mathtt{x} \mapsto \{\mathtt{x=5}\}(\mathtt{x+1})]\} = \{\{\mathtt{x = 6}\}\}$

   b. $M(\mathtt{x:=x+1}, \sigma) = \{\sigma[\mathtt{x} \mapsto \sigma(\mathtt{x+1})]\} = \{\sigma[\mathtt{x} \mapsto \sigma(\mathtt{x})+1]\}$

   c. $M(\mathtt{x:=x+1;\ y:=2*x}, \{\mathtt{x=5}\})$
   $= M(\mathtt{y:=2*x},\ M(\mathtt{x:=x+1}, \{\mathtt{x=5}\}))$
   $= M(\mathtt{y:=2*x},\ \{\mathtt{x=6}\})$ [from part (a)]
   $= \{\{\mathtt{x=6}\}[\mathtt{y} \mapsto \beta]\}$ where $\beta = \{\mathtt{x=6}\}(\mathtt{2*x}) = 12$
   $= \{\{\mathtt{x=6},\ \mathtt{y=12}\}\}$

2. Let $S \equiv$ **`if`** $\mathtt{x>0}$ **`then`** $\mathtt{x:=x+1}$ **`else`** $\mathtt{y:=2*x}$ **`fi`**.

   a. If $\sigma(\mathtt{x}) = 8$, then $\sigma(\mathtt{x>0}) = T$, so $M(S, \sigma) = M(\mathtt{x:=x+1}, \sigma) = \{\sigma[\mathtt{x} \mapsto \sigma(\mathtt{x+1})]\} = \{\sigma[\mathtt{x} \mapsto 9]\}$

   b. If $\sigma(\mathtt{x}) = 0$, then $\sigma(\mathtt{x>0}) = F$, so $M(S, \sigma) = M(\mathtt{y:=2*x}, \sigma) = \{\sigma[\mathtt{y} \mapsto \sigma(\mathtt{2*x})]\} = \{\sigma[\mathtt{y} \mapsto 0]\}$

   c. If $\sigma(\mathtt{x})>0$ then $M(S, \sigma) = M(\mathtt{x:=x+1}, \sigma) = \{\sigma[\mathtt{x} \mapsto \sigma(\mathtt{x})+1]\}$
   If $\sigma(\mathtt{x})\le 0$ then $M(S, \sigma) = M(\mathtt{y:=2*x}, \sigma) = \{\sigma[\mathtt{y} \mapsto 2\times\sigma(\mathtt{x})]\}$

3. Let $S \equiv$ **`if`** $\mathtt{x>0}$ **`then`** $\mathtt{x:=x/z}$ **`fi`** $\equiv$ **`if`** $\mathtt{x>0}$ **`then`** $\mathtt{x:=x/z}$ **`else skip fi`**

   a. If $\sigma=\{\mathtt{x=8},\mathtt{z=3}\}$, then $\sigma(\mathtt{x>0}) = T$, so $M(S,\sigma) = M(\mathtt{x:=x/z}, \sigma) = \{\sigma[\mathtt{x} \mapsto \alpha]\}$ where $\alpha = \sigma(\mathtt{x/z}) =$
   $\sigma[\mathtt{x} \mapsto 8/3] = \sigma[\mathtt{x} \mapsto 2]$, since integer division truncates.

   b. If $\sigma=\{\mathtt{x=-2},\mathtt{z=3}\}$ then $\sigma(\mathtt{x>0}) = F$, so $M(S,\sigma) =$ so $M(\textbf{skip},\sigma) = \{\sigma\}$.

4. Let $W \equiv$ **`while`** $\mathtt{x<3}$ **`do`** S **`od`** where $S \equiv \mathtt{x:=x+1;\ y:=y*x}$.

   a. For arbitrary τ,
   $M(S,\tau) = M(\mathtt{x:=x+1;\ y:=y*x}, \tau)$
   $= M(\mathtt{y:=y*x}, \tau[\mathtt{x}\mapsto\tau(\mathtt{x})+1])$
   $= \{\tau[\mathtt{x}\mapsto\tau(\mathtt{x})+1][\mathtt{y}\mapsto\alpha]\}$ where $\alpha=\tau[\mathtt{x}\mapsto\tau(\mathtt{x})+1](\mathtt{y*x}) = \tau(\mathtt{y})\times(\tau(\mathtt{x})+1)$

   b. If $\sigma\vDash \mathtt{x=4} \wedge \mathtt{y=1}$, then $\sigma(\mathtt{x<3}) = F$ so $M(W,\sigma) = \{\sigma\}$.

   c. If $\sigma\vDash \mathtt{x=1} \wedge \mathtt{y=1}$, then $\sigma(\mathtt{x<3}) = T$ so we have at least one iteration to do. Let $\sigma_0 = \sigma$,
   let $\sigma_1 = M(S, \sigma_0) = \sigma_0(\mathtt{y})\times(\sigma_0(\mathtt{x})+1)$, and let $\sigma_2 = M(S, \sigma_1) = \sigma_1(\mathtt{y})\times(\sigma_1(\mathtt{x})+1)$. Then
   $\sigma_0 = \sigma[\mathtt{x} \mapsto 1][\mathtt{y} \mapsto 1]$
   $\sigma_1 = M(S, \sigma_0) = \sigma_0[\mathtt{x}\mapsto\sigma_0(\mathtt{x})+1][\mathtt{y}\mapsto\sigma_0(\mathtt{y})\times(\sigma_0(\mathtt{x})+1)] = \sigma[\mathtt{x} \mapsto 2][\mathtt{y} \mapsto 2]$
   $\sigma_2 = M(S, \sigma_1) = \sigma_1[\mathtt{x}\mapsto 2+1][\mathtt{y}\mapsto 2\times(2+1)] = \sigma[\mathtt{x} \mapsto 3][\mathtt{y} \mapsto 6]$
   Since σ_0 and $\sigma_1 \vDash \mathtt{x<3}$ but $\sigma_2 \vDash \mathtt{x\ge 3}$, we have $M(W, \sigma) = \{\sigma_2\} = \{\sigma[\mathtt{x} \mapsto 3][\mathtt{y} \mapsto 6]\}$.

5. $M(S, \sigma) = M(\mathtt{x} := \mathtt{y/b[x]}, \sigma) = \{\sigma[\mathtt{x} \mapsto \gamma]\}$ where $\gamma = \sigma(\mathtt{y/b[x]}) = 13 / \sigma(\mathtt{b})(\alpha) = \bot_e$

iff $\sigma(\mathtt{b})(\alpha) = \bot_e$ or $\sigma(\mathtt{b})(\alpha) = 0$

iff (α is out of range for $\sigma(\mathtt{b})$) or ($\sigma(\mathtt{b})(\alpha) = 0$) (b[x] fails if x is out of range)

iff ($\alpha < 0$ or $\alpha \geq 4$) or ($\sigma(\mathtt{b})(\alpha) = 0$) ($\sigma$(b) has size 4)

iff ($\alpha < 0$ or $\alpha \geq 4$) or ($\alpha = 1$) (b[1] is the only element = 0)

iff $\neg(\alpha = 0, 2, \text{ or } 3)$

6. $M(S, \sigma) = M(\mathtt{y} := \mathtt{y / sqrt(b[x])}, \sigma) = \{\sigma[\mathtt{y} \mapsto \beta]\}$ where $\beta = (\sigma(\mathtt{y}) / \mathit{sqrt}(\gamma)) = (8 / \mathit{sqrt}(\gamma))$ and $\gamma = \sigma(\mathtt{b})(\sigma(\mathtt{x})) = \sigma(\mathtt{b})(\alpha)$

So $\beta = \bot_e$ (and thus $M(S, \sigma) = \{\sigma[\mathtt{y} \mapsto \bot_e]\} = \{\bot_e\}$)

iff $\gamma = \bot_e$ or $\gamma < 0$ or $\mathit{sqrt}(\gamma) = 0$ (i.e., b[x] fails, b[x] < 0, or sqrt(b[x]]) = 0)

iff (α out of range for $\sigma(\mathtt{b})$] or $\gamma < 0$ or $\mathit{sqrt}(\gamma) = 0$ ($\gamma = \bot_e$ iff b[x] has a bad index)

iff ($\alpha < 0$ or $\alpha \geq 4$) or $\gamma = \sigma(\mathtt{b})(\alpha) < 0$ or $\mathit{sqrt}(\gamma) = 0$ (σ(b) is of size 4)

iff ($\alpha < 0$ or $\alpha \geq 4$) or ($\alpha = 0$) or $\mathit{sqrt}(\gamma) = 0$ (only b[0] < 0)

iff ($\alpha < 0$ or $\alpha \geq 4$) or ($\alpha = 0$) or ($\alpha = 3$) (only sqrt(b[3]) = sqrt(0) = 0)

iff ($\alpha \leq 0$ or ≥ 3) (combining terms)

For next time, add these (from old lecture on total correctness)

1. Give a total correctness precondition p for $\{p\}\ \mathtt{x} := \mathtt{y/b[i]}\ \{\mathtt{x} > 0\}$. Logically simplify if you wish.

1. For $\{p\}\ \mathtt{x} := \mathtt{y/b[i]}\ \{\mathtt{x} > 0\}$, let $p \Leftrightarrow \mathit{wp}(\mathtt{x} := \mathtt{y/b[i]}, \mathtt{x} > 0)$

$\Leftrightarrow D(\mathtt{y/b[i]}) \wedge (\mathtt{y/b[i]} > 0)$.

$\Leftrightarrow (0 \leq \mathtt{i} < \mathtt{size(b)} \wedge \mathtt{b[i]} \neq 0) \wedge (\mathtt{y/b[i]} > 0)$

2. Repeat problem 1 on $\{p_2\}\ \mathtt{j} := \mathtt{x/j};\ \{p_1\}\ \mathtt{y} := e\ \{\mathtt{z} < \mathtt{y}\}$ where $e \equiv \mathtt{sqrt(b[j])}$.

2. For $\{p_2\}\ \mathtt{j} := \mathtt{x/j};\ \{p_1\}\ \mathtt{y} := \mathtt{sqrt(b[j])}\ \{\mathtt{z} < \mathtt{y}\}$,
take $p_1 \Leftrightarrow \mathit{wp}(\mathtt{y} := \mathtt{sqrt(b[j])}, \mathtt{z} < \mathtt{y})$

$\Leftrightarrow D(\mathtt{sqrt(b[j])}) \wedge \mathit{wlp}(\mathtt{y} := \mathtt{sqrt(b[j])}, \mathtt{z} < \mathtt{y})$

$\Leftrightarrow 0 \leq \mathtt{j} < \mathtt{size(b)} \wedge \mathtt{b[j]} \geq 0 \wedge \mathtt{z} < \mathtt{sqrt(b[j])}$

and $p_2 \Leftrightarrow wp(\mathtt{j} := \mathtt{x/j},\, p_1)$

$\Leftrightarrow D(\mathtt{x/j}) \wedge wlp(\mathtt{j} := \mathtt{x/j},\, p_1)$

$\Leftrightarrow \mathtt{j} \neq 0 \wedge (0 \leq \mathtt{j} < \mathtt{size}(\mathtt{b}) \wedge \mathtt{b[j]} \geq 0 \wedge \mathtt{z} < \mathtt{sqrt}(\mathtt{b[j]}))[\mathtt{x/j\ /\ j}]$

$\Leftrightarrow \mathtt{j} \neq 0 \wedge 0 \leq \mathtt{x/j} < \mathtt{size}(\mathtt{b}) \wedge \mathtt{b[x/j]} \geq 0 \wedge \mathtt{z} < \mathtt{sqrt}(\mathtt{b[x/j]})$.