6.044 Notes 9/16/92

Evaluation assertions for Com:

atomic (*) $\langle \mathrm{skip}, \sigma\rangle \to \sigma$

(**) $$\frac{\langle a,\sigma\rangle \to m}{\langle X := a, \sigma\rangle \to \sigma[m/X]}$$

seq $$\frac{\langle c_0, \sigma\rangle \to \sigma'', \quad \langle c_1, \sigma''\rangle \to \sigma'}{\langle (c_0; c_1), \sigma\rangle \to \sigma'}$$

conditional $$\frac{\langle b,\sigma\rangle \to \mathrm{true}, \quad \langle c_0,\sigma\rangle \to \sigma'}{\langle \mathrm{if}\ b\ \mathrm{then}\ c_0\ \mathrm{else}\ c_1, \sigma\rangle \to \sigma'}$$

like for false

while: $$\frac{\langle b,\sigma\rangle \to \mathrm{false}}{\langle \mathrm{while}\ b\ \mathrm{do}\ c, \sigma\rangle \to \sigma}$$

$$\frac{\langle b,\sigma\rangle \to \mathrm{true}, \quad \langle c,\sigma\rangle \to \sigma'', \quad \langle \mathrm{while}\ldots, \sigma''\rangle \to \sigma'}{\langle \mathrm{while}\ b\ \mathrm{do}\ c, \sigma\rangle \to \sigma'}$$

$$\mathrm{Euclid} \equiv \mathrm{while}\ \overbrace{\neg(M=N)}^{b}\ \mathrm{do}\ \left.\begin{array}{l}\mathrm{if}\ M \le N \\ \quad \mathrm{then}\ N := N - M \\ \quad \mathrm{else}\ M := M - N\end{array}\right\} c$$

6.044

9/16/82

1.2

$\sigma(M) = 6, \ \sigma(N) = 10$

$$\dfrac{\dfrac{\dfrac{\langle M, \sigma\rangle \to 6, \quad \langle N, \sigma\rangle \to 10 \,.}{\langle M = N, \sigma\rangle \to \boxed{\text{false}}_3}}{\langle \neg(M = N), \sigma\rangle \to \boxed{\text{true}}_2} \qquad \dfrac{\dfrac{\vdots\;\ldots}{\langle M \le N, \sigma\rangle \to \boxed{\text{true}}_5} \; , \; \dfrac{\dfrac{\ldots}{\langle N - M, \sigma\rangle \to \boxed{4}_7}}{\langle N := N - M, \sigma\rangle \to \boxed{\sigma[4/N]}_6}}{\langle c, \sigma\rangle \to \boxed{\sigma[4/N]}_4}}{\langle \text{Euclid}, \sigma\rangle \to \boxed{}_1}$$

6.044
9/16/92 p. 3

$\vdots$ $\qquad\qquad\qquad\qquad\qquad\qquad$ $\vdots$

$$\dfrac{\dfrac{\vdots}{\langle \neg(M=N),\ \sigma[4/N] \rangle \to \text{true}} \quad , \quad \dfrac{\vdots}{\langle c,\ \sigma[4/N] \rangle \to \sigma[4/N][2/M]} \quad , \quad \langle \text{Euclid},\ \sigma[4/N][2/M] \rangle \to \sigma[3/N][2/M]}{\langle \text{Euclid},\ \sigma[4/N] \rangle \to \boxed{\sigma[3/N][3/M]}_{8}}$$

6.044 Notes 9/18/92

p.3

~~Sol~~ natural semantics goes in large steps, no bound on size of derivation of $\langle c, \sigma \rangle \to \sigma'$!

~~...~~ And no obvious way to anticipate whether σ' exists (later show this is not computable).

One-step rules are closer to familiar sequential execution of simple commands:

$\sigma(M)=6$, $\sigma(N)=10$

$$\langle Euclid, \sigma \rangle \to_1 \langle (c; Euclid), \sigma \rangle \to_1 \langle (\text{if } \neg(6 \le N) \text{ then } c_0 \text{ else } c_1; Euclid), \sigma \rangle \to_1$$

$$\cdots \le 10 \quad - - - - - - \rangle \to_1$$

$$\langle \text{if } \neg \text{ true then} \quad " \quad \rangle \to_1$$

$$\langle \text{if false then} \quad " \quad \rangle \to_1$$

$$\langle (N := N - M; Euclid), \sigma \rangle \to_1$$

$$\vdots$$

$$\langle (N := 4; Euclid), \sigma \rangle \to_1$$

$$\langle Euclid, \sigma[4/N] \rangle \to_1$$

$$\vdots$$

$$\to_1 \sigma[2/N][2/M]$$

9/16/92

p. 4

Thm: The "evaluates" relation is a function on configurations. That is

$\langle a,\sigma\rangle \to n_1$ and $\langle a,\sigma\rangle \to n_2$ implies $n_1 = n_2$

$\langle b,\sigma\rangle \to t_1$ and $\langle b,\sigma\rangle \to t_2$ " $t_1 = t_2$

$\langle c,\sigma\rangle \to \sigma_1$ and $\langle c,\sigma\rangle \to \sigma_2$ " $\sigma_1 = \sigma_2$

Ended Here 9/16

~~Start 9/18~~

Pf: Aexp an exercise, Bexp like Aexp. Prove for

for 9/23/92

$c \in$ Com, by induction on derivations of $\langle c,\sigma\rangle \to \sigma_1$:

p. 1

~~base + deduction with 1 command rule~~ by cases according to last rule in deduction

easy ~~base~~ case: $c \equiv X := a$

only possible rule is ~~axiom~~ $$\frac{\langle a,\sigma\rangle \to m}{\langle X := a,\sigma\rangle \to \sigma[m/X]}$$

but m is unique by Aexp case. ✓

~~case skip~~

induction

hardest case $c \equiv$ while b do c'

~~Pf Let us say $\langle b,\sigma\rangle \to t$ for unique t~~

~~subcase $\langle b,\sigma\rangle \not\to$ (actually this can't happen, but we don't need this fact)~~

~~then $\langle c,\sigma\rangle \not\to$ so vacuously~~

~~thus~~ subcase $\langle b,\sigma\rangle \to t$ ~~for unique t~~

~~subsubcase: $t \equiv$ true~~

by rule

$$\frac{\langle b,\sigma\rangle \to \text{true},\ \langle c',\sigma\rangle \to \sigma_1',\ \langle c,\sigma_1'\rangle \to \sigma_1}{\langle c,\sigma\rangle \to \sigma_1}$$

likewise for subscript 2.

so

because $\langle b,\sigma\rangle \to$ true is unique, ded. of $\langle c,\sigma\rangle \to \sigma_2$ must end with some instance of same rule.

But $\sigma_1' = \sigma_2'$ by induction, so $\sigma_1 = \sigma_2$ by induction.

Q.E.D.

6.044

9/23/92

Proofs by induction:

Thm: Evalsto relation is a function on command configurations (Winsk, 3.11):

$$\langle c,\sigma\rangle \to \sigma_1 \ \& \ \langle c,\sigma\rangle \to \sigma_2 \ \text{implies} \ \sigma_1 = \sigma_2.$$

<u>Proof:</u> From homework have $\langle a,\sigma\rangle \to n_1$ & $\langle a,\sigma\rangle \to n_2$ implies $n_1 = n_2$ likewise ~~from~~ for $b \in Bexp$. Proof by <u>structural</u> <u>induction</u>, i.e., induction on the size of a. This won't work for c, because ~~of the~~ "while" is an antecedent of "while" in the evalsto rules.

Instead, prove theorem by induction on derivations of the evalsto assertions $\langle c,\sigma\rangle \to \sigma_1$.

That is we ~~want to show that if~~ formulate a property of derivations, $P(d) ::=$

$$\forall c, \sigma, \sigma_1, \sigma_2.\ d \models \langle c,\sigma\rangle \to \sigma_1 \ \& \ \langle c,\sigma\rangle \to \sigma_2 \ \text{implies} \ \sigma_1 = \sigma_2.$$

Use "Course of values induction": for any d, assume P(d') holds for all d' smaller than d, and prove that P(d) holds. Then it's proper to conclude that P(d) holds for all d.

9/23/92

p.2.

So let d be a derivation ~~ends~~. To prove $P(d)$, let $c, \sigma, \sigma_1, \sigma_2$ be such that ~~we are given that~~ $(d \Vdash \langle c,\sigma\rangle \to \sigma_1)$ and

$(\langle c,\sigma\rangle \to \sigma_2$ is derivable$)$

We must show $\sigma_1 = \sigma_2$.

Proceed by cases according to last rule instance of d:

Case: assignment rule:

$$\frac{\langle a,\sigma\rangle \to m_1}{\langle \underbrace{X := a}_{c}, \sigma\rangle \to \underbrace{\sigma[m_1/X]}_{\sigma_1}}$$

Since $\langle c,\sigma\rangle \to \sigma_2$ is derivable, and c is $X := a$, the only rule instance which could end the derivation of $\langle c,\sigma\rangle \to \sigma_2$ would be

$$\frac{\langle a,\sigma\rangle \to m_2}{\langle X := a, \sigma\rangle \to \underbrace{\sigma[m_2/X]}_{\sigma_2}}$$

But $m_1 = m_2$ by theorem for Aexp's, so $\sigma_1 = \sigma_2$.

6.044 9/23/92

p. 3

Case: while-true rule:

$$\frac{\langle b,\sigma\rangle \to true,\ \langle c',\sigma\rangle \to \sigma_1',\ \langle c,\sigma_1'\rangle \to \sigma_1}{\langle \underbrace{\text{while } b \text{ do } c'}_{c},\sigma\rangle \to \sigma_1}$$

Since $\langle c,\sigma\rangle \to \sigma_2$ is derivable and c is a while statement, the last rule in ~~the~~ its derivation must be an instance of while-true or while-false. But while-false would apply only if $\langle b,\sigma\rangle \to false$ was derivable; since we have a derivation of $\langle b,\sigma\rangle \to true$ as part of d, by uniqueness of values for Bexp, we know that $\langle b,\sigma\rangle \to false$ is not derivable, so last rule instance of $\langle c,\sigma\rangle \to \sigma_2$ derivation must be

$$\frac{\langle b,\sigma\rangle \to true,\ \langle c',\sigma\rangle \to \sigma_2',\ \langle c,\sigma_2'\rangle \to \sigma_2}{\langle c,\sigma\rangle \to \sigma_2}$$

for some σ_2'. But a derivation $d' \vdash \langle c',\sigma\rangle \to \sigma_1'$ is a part of d, so P(d') by induction, hence

$$\sigma_1' = \sigma_2'$$

Likewise have $d'' \models \langle c, \sigma_1' \rangle \to \sigma_1$ where d'' part of d,
so $P(d'')$ by induction, and since $\langle c, \sigma_1' \rangle \to \sigma_2$
we have $\sigma_1 = \sigma_2$.

Other cases similar (easier). $\boxtimes$

eval(while b c) $\sigma = \sigma'$ iff

case 1 eval(b) σ = false and $\sigma = \sigma'$

case 2 eval(b) σ = true and
eval(c) $\sigma = \sigma''$ and
eval(while b do c) $\sigma'' = \sigma'$

$\Longrightarrow$ $\mathcal{B}[\![b]\!]\sigma$ = true and
$\mathcal{C}[\![c]\!]\sigma = \sigma''$ and
$\mathcal{C}[\![\text{while } b \text{ do } c]\!]\sigma'' = \sigma'$ by ind on derivation
for some σ''

$\Longleftrightarrow$ " and
" and
$\Gamma^n(\phi)\sigma'' = \sigma'$ for some n, σ''

$\Longleftrightarrow$ $\Gamma^{n+1}(\phi)\sigma = \sigma'$ for some n

$\Longleftrightarrow$ $\mathcal{C}[\![\text{while } b\, c]\!]\sigma = \sigma'$

①

6.044J 10/9/92

Lemma: $eval(a_1 \pm a_2) = eval(a_1) \pm eval(a_2)$

$eval(a_1+a_2)\sigma = n$ iff

Pf: $\langle a_1 \pm a_2, \sigma\rangle \to n$ iff $\langle a_1, \sigma\rangle \to n_1$ and $\langle a_2, \sigma\rangle \to n_2$ for some $n_1+n_2=n$

iff $eval(a_1)\sigma = n_1$ and $eval(a_2)\sigma = n_2$ for some $n_1+n_2=n$

iff $eval(a_1)\sigma + eval(a_2)\sigma = n$

Lemma: $eval(\text{if } b \text{ then } c_1 \text{ else } c_2)\sigma = \begin{cases} eval(c_1)\sigma & \text{if } eval(b)\sigma = true \\ eval(c_2)\sigma & \text{if } eval(b)\sigma = false \end{cases}$

Lemma: $eval(c_1; c_2) = eval(c_2) \circ eval(c_1)$

Pf: $eval(c_1;c_2)\sigma = \sigma'$ iff $\langle c_1, \sigma\rangle \to \sigma''$, $\langle c_2, \sigma''\rangle \to \sigma'$ for some σ''

iff $eval(c_1)\sigma = \sigma''$ and $eval(c_2, \sigma'') = \sigma'$ for some σ''

iff $eval(c_2)\left(eval(c_1)\sigma\right) = \sigma'$

iff $\left(eval(c_2) \circ eval(c_1)\right)\sigma = \sigma'$

But while b do c ~ if b then c; (while b do c) else skip

means eval(while b do c) = eval(if b then c; while b do c else skip)

∴ eval(while b do c) is a <u>fixed point</u> of Γ.

Is it unique? No!

Let b be $X=1$

c be skip

$\gamma(\sigma) = \sigma$ ~~if $\sigma(X) \neq 1$,~~ is also a fixed point

~~while $X=1$ do skip~~ ~

skip ~ if $X=1$ then skip; skip else skip; skip

We will show that

eval (while b do c) = <u>least</u> fixed point of Γ.

$\{\gamma_i\}$ increasing

$$\Gamma(\sqcup \gamma_i) \stackrel{?}{=} \sqcup \Gamma(\gamma_i)$$

case eval(b) = true

$$\cancel{\Gamma(\sqcup \gamma_i)\sigma = (\sqcup \gamma_i)(\mathrm{eval}(c)\sigma)}$$

$$\cancel{= \gamma_j(\mathrm{eval}(c)\sigma) \quad \text{for some } j}$$

$$\Gamma(\sqcup \gamma_i)\sigma = \sigma' \quad \text{iff} \quad (\sqcup \gamma_i)(\mathrm{eval}(c)\sigma) = \sigma'$$

$$\text{iff} \quad \gamma_j(\mathrm{eval}(c)\sigma) = \sigma' \quad \text{some } j$$

$$\text{iff} \quad \bigsqcup_j \gamma_j(\mathrm{eval}(c)\sigma) = \sigma'$$

$$\text{iff} \quad \bigsqcup_j \Gamma(\gamma_j)\sigma = \sigma'$$

---

another way: $\Gamma : C \to C$, $\Gamma = \widehat{R}$ for rules R

if we think of $\gamma \in C$ as a <u>set</u> of ordered pairs,

---

General Machinery for proving continuity:

composition of cont fns is continuous

$$f \circ g(\sqcup d_i) = f(g(\sqcup d_i)) = f(\sqcup g(d_i)) = \sqcup f(g(d_i))$$
$$= \sqcup (f \circ g)(d_i)$$

Let $\text{eval}(c) : \underbrace{\Sigma \to \Sigma}_{C}$ be the function s.t.

$$\text{eval}(c)\sigma = \sigma \quad \text{iff} \quad \langle c, \sigma\rangle \to \sigma'$$

$\text{eval}(a) : \underbrace{\Sigma \to N}_{A}$ by

$$\text{eval}(a)\sigma = n \quad \text{iff} \quad \langle a, \sigma\rangle \to n$$

$\text{eval}(b) : \underbrace{\Sigma \to T}_{B}$ likewise

~~Given while~~ Given b, c define

$$\Gamma : C \to C$$

$$\Gamma(\gamma)\sigma = \begin{cases} \sigma & \text{if } \text{eval}(b)\sigma = \text{false}, \\ \gamma(\text{eval}(c)\sigma) & \text{if } \text{eval}(b)\sigma = \text{true and } \text{eval}(c)\sigma \text{ is defined.} \end{cases}$$

Claim: $\Gamma(\text{eval}(\text{while } b \text{ do } c)) = \text{eval}(\text{if } b \text{ then } c; \text{while } b \text{ do } c \text{ else skip})$

Pf: case: $\text{eval}(b)(\sigma) = \text{false}$ then $\text{eval}(\text{if} \cdots)\sigma = \sigma = \Gamma(\text{any})\sigma$

case $\text{eval}(b)\sigma = \text{true}$ then $\text{eval}(\text{if}\ )\sigma = \text{eval}(c; \text{while})\sigma = \text{eval}(\text{while})(\text{eval}(c)\sigma)$

$= \Gamma(\text{while})\sigma$

6044 9/18/92

p.2

Thm: one-step equiv to natural:

Def. (Reflexive) Transitive closure, C^* of binary relation C on set A:

$$\frac{a \in A}{a\, C^* a} \qquad \frac{a\, C\, a'}{a\, C^* a'}$$

$$\frac{a\, C^* a'', \quad a''\, C\, a'}{a\, C^* a''}.$$

Thm: $\langle \gamma, \sigma \rangle \to \delta$ iff $\langle \gamma, \sigma \rangle \to_1^* \delta$

for config's γ, value $\delta \in (N \cup T \cup \Sigma)$.

Pf: $\Rightarrow$ induction of def'n of $\to$.

$\Leftarrow$ induction on def of $*$.

More discussion next lecture.

~~9/18/92~~

p.5 9/23/92

Remark: evaluates relation is total on Aexp, Bexp and while-free Com, but

~~Lemma~~ Proposition: $\langle \text{while true do } c, \sigma \rangle \not\to$ for all σ

Pf: Consider <u>smallest</u> derivation of assertion of the form

$$\langle \text{while true do } c, \sigma \rangle \to \sigma'$$

must ~~be of the form~~ have had antecedents

$$\langle \text{true}, \sigma \rangle \to \text{true}, \quad \langle c, \sigma \rangle \to \sigma'', \quad \langle \text{while true do } c, \sigma'' \rangle \to \sigma'$$

but then $\langle \text{while true do } c, \sigma'' \rangle \to \sigma'$ has a smaller derivation $\times$.

9/23

What is a derivation anyway?

1. Rule instance (X/y) : X a set, y an element

2. R a set of rule instances

R-derivation of y:

a pair $\langle D, y \rangle$

where $D = \{d_1, \dots, d_n\}$ and d_i is an R-derivation of x_i

and $\{x_1, \dots, x_n\}/y$ is a rule instance in R.

(base case: $D = \emptyset$, rule $\emptyset/y$)

Notation: $\langle D, y \rangle \models_R y$ if it's a derivation. $\models_R y$ means y is derivable

Understood that derivations are finite:

$R = \{(\{a\}/a)\}$. $(\{(\{(\dots a)\}, a)\}, a)$ is not a derivation.

$$I_R = \{x \mid \models_R x\}$$

Rule induction: if $\forall (X, y) \in R.\ \forall x \in X.\ P(x) \Rightarrow P(y)$
$X \subseteq I_R$

then $\forall x \in I_R.\ P(x)$

6.044
9/30/92

$$\hat{R}(A) = \{y \mid X/y \in R \text{ for some } X \subseteq A\}$$

Say $R_0 = \{\{n, n+1\}/n+2 \mid n \in Num\} \cup \{\emptyset/3\} \cup \{\emptyset/4\}$

$$\hat{R}_0(\emptyset) = \{3, 4\}$$

$$\hat{R}_0(\{6, 7, 8, 10\}) = \{3, 4, 8, 9\}$$

$$I_{R_0} = fix(\hat{R}_0) = \text{smallest } \hat{R}_0\text{-closed set} =$$
$$\text{smallest } S \text{ s.t. } \hat{R}_0(S) = S =$$
$$\bigcup_{n \geq 0} \hat{R}_0^n(\emptyset)$$
$$= \{n \mid n \geq 3\}$$

Universe D:
rule instances: X/y for $X \in Pow_{fin}(D)$, $y \in D$

Lemma: $\hat{R}$ is continuous from cpo $(Pow(D), \subseteq)$ to itself.

Pf: We already know $\hat{R}$ is monotone, so must show that if $B_0 \subseteq B_1 \subseteq B_2 \subseteq \ldots$

then $\hat{R}(\bigcup_{n \geq 0} B_n) = \bigcup_{n \geq 0} \hat{R}(B_n)$.

Step: Consider counter-example if B_0, B_1, not increasing, e.g.,
Let $B_n = \{n\}$. So $\bigcup_{n \geq 0} B_n = \omega$ $\quad \hat{R}_0(\omega) = \{3, 4, 5, \ldots\}$

But $\bigcup_{n \geq 0} \hat{R}_0(B_n) = \bigcup_{n \geq 0} \{3, 4\} = \{3, 4\}$.

Proof: ($\supseteq$) since $B_k \subseteq \bigcup_{n\geq 0} B_n$

$\hat{R}(B_k) \subseteq \hat{R}(\bigcup_{n\geq 0} B_n)$ by monotonicity

so $\bigcup_{k\geq 0} \hat{R}(B_k) \subseteq \hat{R}(\bigcup_{n\geq 0} B_n)$ (even if B_n's not increasing)

Remains to show ($\subseteq$)

say $y \in \hat{R}(\bigcup_{n\geq 0} B_n)$

~~then $X/y \in R$ for some X~~

then $\{x_1,\dots,x_k\}/y \in R$ for some $\{x_1,\dots,x_k\} \subseteq \bigcup_{n\geq 0} B_n$

So for each x_i, there is some $n_i \geq 0$ ~~let n_i be least~~ such that $x_i \in B_{n_i}$.

Let $m = \max\{n_i \mid 1\leq i\leq k\}$

(here is where we use $B_n \subseteq B_{n+1}$)

Since $B_{n_i} \subseteq B_m$, for $1 \leq i \leq k$

we conclude $x_i \in B_m$

$\therefore\ x \in \hat{R}(B_m) \subseteq \bigcup_{n\geq 0} \hat{R}(B_n)$

Q.E.D.

A a cpo with least element, $\bot$.

Thm. $f: A \to A$ continuous. Then $\bigsqcup_{n\geq 0} f^n(\bot)$ is least fixed point of f.

Pf: (showed fixed point; skipped "least").

Lecture notes 11/23

Last time: Checkable implies Expressible (Pf in Appendix nicer than in class)

~~Def. Decidable: command c s.t.~~

~~$n \in D$ iff $\mathcal{C}[\![c]\!]\sigma_0[n/X_1] = $~~

~~Def. IMP Computable partial function: $f : N^k \to N$~~

Def. $\{c\}_{\vec{Y}_{n+1}}(\vec{m}_n) = \mathcal{C}[\![c]\!](\sigma_0[\vec{m}_n/\vec{Y}_n])(Y_{n+1})$

$f : N^n \to N$ is IMP computable iff $f = \{c\}_{\vec{Y}_{n+1}}$

Example: $+, -, \times$, projection functions, closed under composition.

Def. $D \subseteq N$ is decidable iff $char_D$ is computable where

$$char_D(n) = \begin{cases} 1 & \text{if } n \in D, \\ 0 & \text{if } n \notin D. \end{cases}$$

Lemma. Decidable implies checkable

Pf: say $char_D = \{c\}_{X_1 Y}$. Then $X_9 := X_1; X_9 := 0; c; \text{if } Y = 0 \text{ then } \underline{diverge}$ is a checker for D.

INSERT PF SKS

Lemma: Decidable sets closed under $\cap, \neg$ (therefore $\cup$)

Pf: $char_{D_i} = \{c_i\}_{X_i Y_i}$ then where $loc(c_1) \cap loc(c_2) = \emptyset$

Let c = $X_2 := X_1$; c_1; if $Y_1 = 0$ then $Y_2 := 0$ else c_2

$char_{D_1 \cap D_2} = \{c\}_{X_1, Y_2}$

~~Lecture~~ 11/23 INSERT PFSYS

function computable ~~Define decidable. $D \subseteq \mathbb{N}$. Lemma: decidable $\Rightarrow$ checkable.~~

Define Proof system with "proofs" $o_0, o_1, o_2, \ldots$
and "is a proof of relation", $:$, between proofs and Assn's,
such that it is decidable whether $o : A$.

(Formally, $\{ n \in \mathbb{N} \mid o_{left(n)} : A_{right(n)}$ ~~$n = mkpair(m, \#A)$ for some m, A such that~~ $\}$
is IMP-decidable)
A is provable iff there is a proof of A.

~~Thm: The set provable assertions~~

(*) Thm: In any proof system the set of provable assertions is checkable.

Def: A is a sentence iff A is closed, location free

Let Provable = the provable sentences

(**) Cor: In any proof system, Provable is a checkable set.

Def: A proof system is sound iff every provable assertion is valid.

Thm: (Incompleteness) In any sound proof system, Provable $\subsetneq$ Truth, i.e., there is a true sentence which is not provable.

Pf: Truth is not expressible, but Provable is expressible, since it is checkable. $\therefore$ Truth $\neq$ Provable. Provable $\subseteq$ Truth by def. of soundness. $\boxtimes$

INSERT CONT'D

11/23

INSERT CONT'D

To prove (*), note that $\{\#A \mid A \text{ is provable}\}$

$$= \text{right}\big(\underbrace{\{n \mid o_{\text{left}(n)} : A_{\text{right}(n)}\}}_{\text{decidable}}\big)$$

So prove

<u>Lemma</u>: If D is decidable, and then right (D) is checkable.

(See classnotes p. 3)

To prove (**): see p. 4

checkable closed under $\wedge$

~~checkable $\wedge$ checkable is ch~~

Sentences ~~are~~ ~~check~~ decidable

notes 11/23

Better proof: $\mathrm{char}_{D_1 \cap D_2} = \mathrm{char}_{D_1} \times \mathrm{char}_{D_2}$

~~So~~ $\cancel{\{\mathrm{char}_{D_1 \cap D_2}\}_X = \{c\}_{X,X}}$

~~where c is Com on X~~ ~~$X := F_1(x) \times F_2(x)$~~

~~where $\rho(F_i) = \mathrm{char}_{D_i}$~~

$\mathrm{char}_{\overline{D}} = 1 - \mathrm{char}_D$

$g(m) = \mu n.\ f(m,n)=0$

$$c \equiv \begin{array}{l} Z:=1 \\ Y:=0 \\ \text{while } Z=1 \text{ do} \\ \quad \text{if } F(X,Y)=0 \text{ then } Z:=0 \\ \quad\quad \text{else if } Y\le 0 \text{ then } Y:=1-Y \\ \quad\quad\quad \text{else } Y:=-Y \end{array}$$

% Z=1 means keep searching

% start searching at 0

% if Y is the answer stop searching

% try the next value for Y.

$g = \{c\}_{X,Y}$

Save for PS.

$g(m) = \mu n.\ f(n)=m \wedge n\in D$ $\quad char_D(n)=1$

$= \mu n\ (f(n)-m)^2 + (char_D(n)-1)^2 = 0$

$dom(g) = f(D)$

C checkable iff $C = dom(f)$ for part comp. f

proof: $\Leftarrow$ Say $f = \{c\}_{X,Y}$

Then $X:=X_1;\ c$ is checker for $dom(f)$

$\Rightarrow$ Say c checks C

Then $C = dom(\{c\}_{X_1,X_1})$

L a list of locations

An L-canonical form ~~for the empty list~~ is an Aexp, defined inductively on L

if $L = \varepsilon$, then $n \in \mathbb{N}$ is an L-canonical form

if $L = X \cdot L'$, then an L-canonical form is

$$C_k \cdot X^k + C_{k-1} X^{k-1} + \cdots + C_1 \cdot X + C_0$$

s.t. $k \geq 0$, (C_i is an L' canonical form for $0 \leq i \leq k$

and if $k > 0$, then C_k is not the number 0.)

Lemma: If $a \in$ Aexp is a nonzero L-canonical form for some L, then $[\![a]\!]\sigma \neq 0$ for some σ.

Pf: Ind on $|L|$. $|L| = 0$ trivial.

Say $L = X \cdot L'$

$$a = \sum_{i=0}^{k} C_i X^i$$

if $k = 0$, then $a \equiv C_0$ an L' form holds by ind,

if $k > 0$ then C_k is nonzero

by induction have $[\![C_k]\!]\sigma \neq 0$ some σ'

so $[\![a]\!]\sigma'[n/x] = \underbrace{\sum_{i=0}^{k} c_i n^i}_{\text{polynomial in } n}$

$c_i \in N$

$c_i = [\![C_k]\!]\sigma$

$c_k \neq 0$

$\therefore$ for large n_0, $[\![a]\!]\sigma'[n_0/x] \neq 0$.

Let σ be $\sigma'[n_0/x]$.

(9) 10/28/92

Thm. (Hoare Logic is Sound): If the antecedents of an instance of a Hoare Logic rule are valid, then so is the consequent.
(Special case: axioms are valid.)

Pf: Already showed assignment axiom is valid.

Do the while-rule (others easier)

$$\models \{A \wedge b\} c \{A\} \quad \text{implies} \quad \models \{A\} \overbrace{\text{while } b \text{ do } c}^{w} \{A \wedge \neg b\}$$

Lemma: $\mathcal{C}[\![w]\!]\sigma = \sigma'$ iff there are $\sigma_0, \sigma_1, \ldots, \sigma_n$ for $n \geq 0$ such that

(1) $\sigma = \sigma_0$

(2) $\sigma' = \sigma_n$ and $\mathcal{B}[\![b]\!]\sigma_n = \text{false}$

(3) $\mathcal{C}[\![c]\!]\sigma_i = \sigma_{i+1}$ for $0 \leq i < n$

(4) $\mathcal{B}[\![b]\!]\sigma_i = \text{true}$ for $0 \leq i < n$

Assume lemma (provable several ways).

~~Suppose $\sigma \models$~~

~~Prove by induction on i that~~

To show consequent is valid, suppose $\sigma \models^I A$ and $\mathcal{C}[\![w]\!]\sigma = \sigma'$

must show $\sigma' \models A \wedge \neg b$.

Prove by induction on i that $i \leq n$ implies $\sigma_i \models^I A$.

$i = 0$ by hypothesis

if $\sigma_i \models^I A$ then $\sigma_i \models A \wedge b$ for $i < n$ by lemma (2)

☺ 10/~~27~~28/92

so $\sigma_{i+1} \models A$ by valid antecedant

$\therefore$ $\sigma_n \models A$ ~~cand~~

also $\sigma_n \models \neg b$ by lemma

so $\sigma_n = \sigma' \models A \wedge \neg b$ ✓ $\boxtimes$

6.044

10/28/92

Hoare Logic:

1) $\{A\}\ \mathrm{skip}\ \{A\}$

2) $\{B[a/x]\}\ X := a\ \{B\}$

3) $$\frac{\{A \wedge b\}\ c_1\ \{B\},\quad \{A \wedge \neg b\}\ c_2\ \{B\}}{\{A\}\ \text{if } b \text{ then } c_1 \text{ else } c_2\ \{B\}}$$

4) $$\frac{\{A\}\ c_1\ \{B\},\quad \{B\}\ c_2\ \{C\}}{\{A\}\ (c_1 ; c_2)\ \{C\}}$$

5) $$\frac{\{A \wedge b\}\ c\ \{A\}}{\{A\}\ \text{while } b \text{ do } c\ \{A \wedge \neg b\}}$$

A is called a "while-invariant"

6) $$\frac{\{A'\}\ c\ \{B'\}}{\{A\}\ c\ \{B\}}$$

providing $\left(\models (A \Rightarrow A') \wedge (B' \Rightarrow A)\right)$

"rule of consequence"

"weakening"

③ 10/28/92

to prove this, only rule is (3), so must prove

$$(*) \qquad \{A \wedge \neg(M=N) \wedge (M \le N)\}\ N:=N-M\ \{A\}$$

and

$$\{\quad " \qquad \neg(M \le N)\}\ M:=N-M\ \{A\}$$

Only way to prove (*) is to ~~weaken~~ use axiom (2). fits, names
weakening so

show ~~and~~ $\models \underbrace{[A \wedge \neg(M=N) \wedge (M \le N)]}_{0<M<N \ \wedge\ gcd(i,j)=gcd(M,N)} \Rightarrow \underbrace{A[{}^{N-M}/_{N}]}_{(M,\ N-M>0)\ \wedge\ gcd(i,j)=gcd(M,N-M)}$

but number theory:

$$\models 0<M<N \Rightarrow gcd(M,N) = gcd(M,N-M)$$

10/28/92 (2)

6.044

Example : Euclid: while $\neg(M \neq N)$ do $\overbrace{\text{if } M \le N \text{ then } N := N - M \text{ else } M := M - N}^{c}$

Prove: PRE: $\{\cancel{h = gcd}\ i = M \wedge j = N \wedge \neg(0 \le M) \wedge \neg(0 \le N)$
$\{\cancel{N = gcd(M,N) \wedge M,N > 0}\}$ Euclid $\{\cancel{M = n}\}$ POST equivalent to $M = gcd(i,j)$

~~description of an Assn, PRE~~

~~Method to apply~~
Method: to reduce verification of Euclid to verification of subparts, only rule (5) will work. to apply rule (5), ~~once~~ only weakening will work. Need $A \in Assn$ s.t.

$$\models (PRE \Rightarrow A) \wedge ((A \wedge \neg\neg(M = N)) \Rightarrow POST)$$

Magic : Choose $A ::= (M,N > 0 \wedge gcd(i,j) = gcd(M,N))$

(check that $\models$?: handwaving appeal to number theory)

Moreover, to apply (5) need

$$\{A \wedge \neg(M = N)\}\ c\ \{A\}$$

$\sigma \models^{I} \forall j.\ A \vee B$ $\qquad$ $j \notin FV(A)$

iff $\sigma \models^{I[n/j]} A \vee B$ for all n

iff $\sigma \models^{I[n/j]} A$ or $\sigma \models^{I[n/j]} B$ for all n

~~iff $\sigma \models^{I} A$ or $\sigma \models^{I[n/j]} B$ for all n~~

~~since $j \notin FV(A)$~~

argue by cases $\sigma \models^{I} A$ or $\sigma \not\models^{I} A$

iff $\sigma \models^{I} A$ or $\sigma \models^{I} \forall j.B$

iff $\sigma \models^{I} A \vee \forall j.B$

$\Longrightarrow$ Case 1. $\sigma \models^{I} A$. ~~then $\sigma \models^{I[n/j]} A$~~ since $I =_{FV(A)} I[n/j]$

Case 2. $\sigma \not\models^{I} A$

then $\sigma \not\models^{I[n/j]} A$ for all n since

so $\sigma \models^{I[n/j]} B$ for all n

so $\sigma \models \forall j.B$

MP: R(P, i)

map: R(-P, i)

6.044
10/21

Defn. B_1 is <u>equivalent</u> to B_2 iff
$\sigma \models^I B_1$ iff $\sigma \models^I B_2$ for all σ, I.

<u>Lemma:</u> $\neg \forall j. B$ equivalent to $\exists j. \neg B$

Pf: $\sigma \models^I \neg \forall j. B$ iff $\sigma \not\models^I \forall j. B$ iff it is not true that $\left(\sigma \models^{I[n/j]} B \text{ for all } n \in N\right)$ iff $\sigma \models^{I[n_0/j]} B$ is not true for some $n_0 \in N$ iff $\sigma \models^{I[n_0/j]} \neg B$ for some $n_0 \in N$
iff $\sigma \models^I \exists j. \neg B$

<u>Lemma.</u> A is equiv. B iff $(A \Rightarrow B) \wedge (B \Rightarrow A)$ is valid

Pf: ($\Rightarrow$ iff) case(1) $\sigma \models^I A$. So $\sigma \models^I B$ by equiv.
So $\sigma \models^I A \Rightarrow B$; likewise $\sigma \models^I B \Rightarrow A$
so $\sigma \models^I (A \Rightarrow B) \wedge (B \Rightarrow A)$

case(2) $\sigma \not\models^I A$. So $\sigma \not\models^I B$. So
$\sigma \models^I B \Rightarrow A$; likewise
$\sigma \models^I A \Rightarrow B$
$\therefore \sigma \models^I (A \Rightarrow B) \wedge (B \Rightarrow A)$

($\Leftarrow$ iff) similar

10/21, p.2

Lemma. $\sigma_1 =_{loc(A)} \sigma_2$ implies $\sigma_1 \models^I A$ iff $\sigma_2 \models^I A$

Lemma. $I_1 =_{FV(A)} I_2$ implies $\sigma \models^{I_1} A$ iff $\sigma \models^{I_2} A$

Pf: By ind. on A:

~~case~~ $I_1 =_{FV(a)} I_2$ implies

sublemma: $\forall a,\ Av[\![a]\!]I_1 = Av[\![a]\!]I_2$

Pf: ind. on a:

(case $a \equiv j$)

$$Av[\![a]\!]I_1 = I_1(j) = I_2(j) = Av[\![a]\!]I_2$$

since $j \in FV(a)$

(case $a \equiv a_1 + a_2$)

$$Av[\![a_1+a_2]\!]I_1 = Av[\![a_1]\!]I_1 + Av[\![a_2]\!]I_1$$

$$= \quad " \; I_2 \qquad " \; I_2$$

because
$FV(a_1) \subseteq FV(a_1+a_2)$
so $I_1 =_{loc(a_1)} I_2$
so induction hold at a_1

10/21, p.3

Syntactic substitution vs. ~~Semantic patching~~ Interpretation Updating

~~Examples $\sigma \models^I X = j+1+i$ iff $\sigma[n/X] \models^I X = Y+i$~~

~~$\sigma(X)=3$, $I(j)=2$~~ ~~where $n = I(j)+1$~~

~~$Av[\![X+j+1]\!]I\sigma = 6 = Av$~~

~~More generally~~

~~$Av[\![$~~

<u>Lemma 8</u>

$$Av[\![subs[a_0, a, X]]\!] I\sigma$$

$$= Av[\![a_0]\!] I(\sigma[n/X])$$

where $n = Av[\![a]\!] I\sigma$

(likewise subs. for **j** and update I)

Example a_0 is $X+X+Y+i$ ~~$\sigma(X)=n$~~

a is $j+1$ $I(j)=3$

$$Av[\![subs(X+X+Y+i, j+1, X)]\!] I\sigma =$$

p.4

$$\mathcal{A}v[\![(j+1)+(j+1)+Y+i]\!] I \sigma$$
$$= \mathcal{A}v[\![X+X+Y+i]\!] I \left(\sigma[4/X]\right).$$

Lemma: For $a \in \text{Aexp}$ (not Aexpv in general)
$B \in \text{Assn}$, $X \in \text{Loc}$

$$\sigma \models^I \text{subs}(B, a, X) \text{ iff } \left(\sigma[n/X]\right) \models^I B$$
where $n = \mathcal{A}[\![a]\!]\sigma$

Application:
$$\sigma \models (Y+X) \times (Y+X+1) = Y+j$$
— subs(B, Y+X, X)

$a \equiv Y+X$

then $\underbrace{\mathcal{C}[\![X := Y+X]\!]\sigma}_{\sigma[n/X]} \models \underbrace{X \times (X+1) = Y+j}_{B}$

where $n = \mathcal{A}[\![Y+X]\!]\sigma$

p. 5

Def of subst: $\underbrace{\text{subs}[a_0, a, i]}_{\text{notation: } a_0[a/i]}$ by ind. $a_0, a \in \text{Aexpv}$, $i \in \text{Intvar}$

ambiguous notation also used for function updating !

$n[a/i] ::= n$

$i[a/i] ::= a$

$X[a/i] ::= X$

$j[a/i] ::= j$ when j not the same Intvar as i.

$(a_0' \text{ op } a_0'')[a/i] ::= a_0'[a/i] \text{ op } a_0''[a/i]$

likewise for $a \in \underline{\text{Aexp}}$ (not Aexp v) and $B \in \text{Assn}$

$B[a/i]$ case $(a_0 = a_1)[a/i] ::= (a_0[a/i] = a_1[a/i])$

$(B_1 \wedge B_2)[a/i] ::= B_1[a/i] \wedge B_2[a/i]$

$(\forall i.\ B_1)[a/i] ::= \forall i.\ B_1$

$(\forall j.\ B_1)[a/i] = \forall j.\ B_1[a/i]$ for $j \not\equiv i$.

Substitution $s : Loc \to Aexp$

patchstate $\tau : Loc \to Num$

~~$\sigma_1 =_{loc(A)} \sigma_2$ $\quad \sigma_1 = \sigma_2[s]$ $\quad dom(s) \cap loc(A) = \phi$~~

~~$\sigma_1 \models^I A$ iff $\sigma_2[s] \models^I A$~~

~~iff $\sigma_2 \models A[$~~

s and τ are consistent at σ iff $dom(s) = dom(\tau)$

and $\mathcal{A}[\![s(X)]\!]\sigma = \tau(\sigma)$ for all $X \in dom(s)$.

s, τ cons at σ implies $\left[\sigma[\tau] \models^I A \text{ iff } \sigma \models^I A[s]\right]$

Pf: By ind. on A

sublemma: $Av[\![a']\!]I(\sigma[\tau])$

$= Av[\![a'[s]]\!]I\sigma$

Cor: $\sigma_1 =_{loc(A)} \sigma_2$ implies $\sigma_1 \models^I A$ iff $\sigma_2 \models^I A$

Pf: Say $\sigma_1 = \sigma_2[\tau]$ where $dom(\tau) \cap loc(A) = \phi$

let $s = \tau$ (so s, τ trivially consistent)

So $\sigma_1 \models A$ iff $\sigma_2[\tau] \models A$ iff $\sigma_2 \models A[s]$

iff $\sigma \models A$

since $A \equiv A[s]$

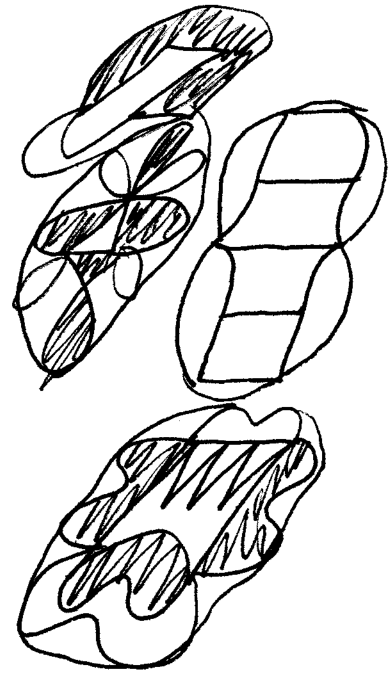
$\models \{A\}\, c_1; c_2\, \{C\}$

choose B equiv. $WP(c_2, C)$

$A \Rightarrow [c_1; c_2]\, C$

$A \Rightarrow [c_1]\,[c_2]\, C$

$\{A\}\, c_1\, \{WP(c_2, C)\}$



6.044 Notes for 11/13/92

$A_{p(m,n)}$ equiv. $A_n[m/i_0]$ $\# i_0 = 307$

$A_n[m/i_0]$ equiv. $\underbrace{\exists i_0.\ i_0 = m \wedge A_n}_{\#(\ )=}$

$p(m,n) =$ mkexists(307, mkconj(mkequality(307, mknum(m)), n))

Now consider $\models T[n/i_0]$ iff $\models A_n$

Then let $F ::= \exists i_1.\ i_1 = p(i_0, i_0) \wedge \neg T[i_1/i_0]$

$F[n/i_0]$ equiv. $\neg T[p(n,n)/i_0]$ equiv. $\neg A_n[n/i_0]$

or $\exists i_1.(i_1 = i_0 \wedge \exists i_0.\ i_0 = p(i_1, i_1) \wedge \neg T)$

6044 11/20/92

Checkable S $= \{ n \mid \mathcal{C}[\![c]\!](\sigma_0[n/X]) \neq \bot \}$

Thm: Checkable implies Expressible.

Let $\vec{Y} = loc(c) - \{X\}$

Pf: $\mathcal{C}[\![c]\!](\sigma_0[n/X]) = \bot$ iff

$$\{X = n \wedge \vec{Y} = 0\}\ c\ \{false\}$$

$$\therefore \neg((X = i_0 \wedge \vec{Y} = 0) \Rightarrow W(c, false))$$

expresses S. $\boxed{\times}$

~~fairly general concept of "proof-system" for Assn's.~~

~~Set of numbered objects called "axioms"~~

~~Set of numbered objects called "theorem~~

~~Set of objects called tuples of numbered objects called "rules"~~

~~$\#(\langle o_1, \ldots, o_n \rangle) =$ mkrule (n) mkpair(#0, mkpair(#o, ...~~

~~numbering of rules so~~

~~1) Decidable set of Assn's called "axioms"~~

~~2) Decidable set of tuples of assertions called "rule-instances"~~

~~An assertion is provable in the proof system iff it is derivable from the axioms and rules.~~

11/20/92

A very general concept of proof-system for Assn's:

A numbered set of objects $o_0, o_1, o_2, \ldots$, called proofs, and a relation, :, between proofs and Assertions such that the $\{\langle n, A\rangle \mid o_n : A\}$ is IMP-~~decidable~~ ~~checkable~~ decidable

(This means ~~$\{m \mid m = mkpair(n, \#A)$ for some $n \geq 0$, and $A \in Assn$ such that $o_n : A\}$~~

$$\{mkpair(n, \#A) \mid A \in Assn \text{ and } o_n : A\}$$

is decidable.)

~~Claim~~ A is provable iff $o : A$ for some proof o.

Thm: The provable Assn's are checkable

Pf: $F := 1;\ Z = 0$

while $F = 1$ do

$C(Z, Y)$;

if $right(Z) = X \wedge Y = 1$ then $F := 0$

else $Z := Z + 1$

$c(Z, Y)$ sets $Y = \begin{cases} 0 \\ 1 \text{ if } o_{left(Z)} : A_{right(Z)} \end{cases}$

$f(1) = \; \text{if}$

$\$371$ Naomi

$c_1 \text{ or } c_2$

local $X := 0;$
$(X:=0 \,|\, X:=1);$ if $X=0$ do c_1
else c_2 end

$(c_0,\sigma_0)\ (c_1,\sigma_0')\ (c_1,\sigma_1)\ (c_2,\sigma_1')\ (c_2,\sigma_2)$

$(\sigma_0,\sigma_0')\ (\sigma_1,\sigma_1') \cdots (\sigma_n,\sigma_n')$

$$\frac{M \xrightarrow{1} M'}{M|N \xrightarrow{1} M'|N}$$

s.t. $\langle c_i,\sigma_i\rangle \xrightarrow{*}_1 \langle c_{i+1},\sigma_i'\rangle$

$\langle c_k,\sigma_k\rangle \xrightarrow{*}_1 \sigma_k'$ some $k \le n$

$X:=X \sim \text{skip}$?

$$\frac{\langle M,\sigma\rangle \to \langle M',\sigma'\rangle}{} \qquad \frac{\langle M,\sigma\rangle \to \sigma'}{\langle M|N,\sigma\rangle \to \langle N,\sigma'\rangle}$$

$M|N$ — $(\sigma_1,\sigma_2)\ (\sigma_2,\sigma_3)$

— (σ_1,σ_3) —

mumbling

$\cancel{X:=1;\text{skip}}$

$\sigma(X)=1$

$\text{skip} \,|\, X:=0$

$X=0$

(σ_1,σ_2)

$X:=X \,|\, X:=0$

(σ,σ) stuttering

$\xrightarrow{1} X:=1 \,|\, X:=0$

fairness! (σ_n,σ_n')

$\to X:=1$

$$L;L' = \text{mumble}\left(\{\, l\cdot l' \mid l \in L,\ l' \in L' \,\}\right)$$

$$[\![X:=0]\!] = (\sigma, \sigma[0/X])$$

$$\frac{\cancel{\langle B,\sigma\rangle \to \langle B',\sigma\rangle} \quad \langle b,\sigma\rangle \to \text{true},\ \langle c,\sigma\rangle \to \sigma'}{\langle \text{await } b \text{ atomic } c,\sigma\rangle \to_1 \sigma'}$$

$$H = \{ c \mid \{c\}_{X_1, X_1}(\#c) \downarrow \}$$

Thm: $\overline{H}$ is not checkable

- Suppose ~~suppose~~ c_1 checker for $\overline{H}$

so for all $c \in Com$

$\{c_1\}(\#c)\downarrow$ iff $\#c \in \overline{H}$ iff $\{c\}(\#c) \uparrow$

Let c be c_1, get contradiction

Cor: H and $\overline{H}$ are undecidable. (decidable closed under $\overline{\ }$; decid $\Rightarrow$ checkable)

The Halting Problem $= \{ \langle c, n \rangle \mid \{c\}(n) \downarrow \}$

Cor: The Halting Problem is undecidable.

Pf: If $d \in IMP$ was a decider for halting problem then

$X_1 := mkpair(X_1, X_1)$; d

would be a decider for H.

Big Thm: Universal Machine Theorem: There is a $SIM \in Com$ s.t. $\{SIM\}(n) = \#(\{c\}\sigma)\restriction loc(c))$ where $left(n) = \#c$ and $right(n) = \#(\sigma \restriction loc(c))$

Cor: Halting Problem and self-halting problem are ~~r.e.~~ checkable

(Cor: checkable not closed under complement)

$H_0 = \{c \mid c \text{ halts in input } 0\}$

Thm: $\overline{H_0}$ is not checkable

Let c_H be checker for H

Pf: For $n \in N$, let $d_n \in Com$ be

$X_1 := n;\ c_H$

$n \in H \Rightarrow d_n$ ~~deep~~ halts in all states $\Rightarrow d_n \in H_0$

$n \notin H \Rightarrow d_n$ never halts $\Rightarrow d_n \notin H_0$

so $n \in \overline{H}$ iff $d_n \in \overline{H_0}$

Let $f(n) = \# d_n$ So to ~~check~~ check $n \in \overline{H}$, run $\overline{H_0}$ checker on $f(n)$.

That is, if $c_{\overline{H_0}}$ was an $\overline{H_0}$ checker, then

Then $X_1 := F(X_1);\ c_{\overline{H_0}}$ would be an $\overline{H}$ checker $\times$

List of Handouts

1. Course info
2. Diagnostic quiz
3. Step-by-step rewrite rules for **IMP**
4. Problem set 1
5. Instructions for Problem Sets
6. Diagnostic quiz solutions
7. Problem set 2
8. Problem set 1 solutions
9. Addendum to ps1, problem 2b solution
10. (replaced by handout 12 below)
11. Problem set 3
12. Equivalence of natural and one-step semantics
13. Lecture outline (see handout 43)
14. Step-by-step rewrite rules for $\mathbf{IMP}_r$
15. Last year's quiz 1, with solutions
16. Quiz 1
17. Quiz 1 Solutions
18. Problem set 4
19. Quiz 1 grading statistics
20. Problem set 5
21. Problem set 2 solutions
22. Problem set 3 solutions
23. Problem set 4 solutions
24. Last year's quiz 2, with solutions
25. Last year's quiz 3, with solutions
26. Problem set 5 solutions
27. Quiz 2
28. Problem set 6
29. Quiz 2 grading statistics
30. Quiz 2 Solutions
31. Problem set 7
32. Notes on Expressiveness

33. Quiz 3 review material
34. Quiz 3
35. Problem set 6 solutions
36. The Four Squares theorem
37. Quiz 3 grading statistics
38. Problem set 8
39. Semantics by translation: translating $\mathbf{IMP}_r$ into **IMP**, and **IMP**into counter machines
40. Notes on expressibility, checkability, decidability
41. Problem set 9 (withdrawn because of bug in prob 1(b))
42. Problem set 9, revised
43. Lecture outline
44. Problem set 7 solutions
45. Problem set 8 solutions
46. Problem set 9 solutions
47. Quiz 4
48. Quiz 4 solutions
49. Quiz 4 grading statistics

    Uri Treisman article

Course Information

Staff.

Lecturer:	Prof. Albert R. Meyer `meyer@theory.lcs.mit.edu`	NE43-315	x3-6024
Teaching Assistant:	David Wald `wald@theory.lcs.mit.edu`	NE43-334	x3-5866
Secretary:	David Jones `6044-secretary@theory.lcs.mit.edu`	NE43-316	x3-5936

Lectures and Tutorials. Class meets MWF from 1:00–2:00 PM in 2-146. There will be no recitation sections, but tutorial/review sessions may be organized in response to requests. The TA will have one regularly scheduled office hour to be announced the first day of class. Further meetings with the TA or instructor can be scheduled by appointment.

Prerequisites. The official requirement for the course is either 18.063 *Introduction to Algebraic Systems*, or 18.310 *Principles of Applied Mathematics.* If you know the basic vocabulary of mathematics and how to do elementary proofs, then you may take this course with the permission of the instructor.

Contrarequisites. There will be less overlap between 6.045J/18.400J and this course than in previous terms, so Course 6 students gung-ho for theory will no longer be discouraged from taking both courses. There will also be a smaller overlap with 6.840J/18.404J; students, especially Math majors, *may* routinely take both this course and 6.840J/18.404J.

Textbook. The required text for the course is *Introduction to the Formal Semantics of Programming Languages* by Glynn Winskel. The book is in manuscript form and copies of the portions covered in the course will be handed out in class. Students will be minimally charged for reproduction costs.

Grading. There will be regular problem sets, and four one hour quizzes—two in class, one evening quiz, and one during final exam period. The problem sets and quizzes *count about equally* toward the final grade. The grading is nonlinear: ace the homework *or* the quizzes and you get an A, but shortchanging homework is imprudent.

Problem Sets. There will be likely be six to eight problem sets. Homework will usually be assigned on a Friday and due 7–10 days later.

Handouts and Notebook. You may find it useful to get a loose-leaf notebook for use with the course, since all handouts and homework will be on standard three-hole punched paper. If you fail to obtain a handout in lecture, you can get a copy from the file cabinet to the right of the door to room NE43-311. If you take the last copy of a handout, please inform the course secretary, and get instructions on making more copies.

Handouts will also be available on-line in the 6.044 directory. To access this directory from Athena, type

```
attach -m /theory/6.044 -e theory.lcs.mit.edu:/pub/ftp/pub/6.044
source /theory/6.044/.athena_startup
```

(We recommend adding these lines to your `.environment` file, causing them to be executed every time you log in.) You will get a warning that "theory.lcs.mit.edu isn't registered with kerberos," which is entirely accurate but irrelevant. This will make the 6.044 directory available to you as `/theory/6.044` and tell LaTeX where to find the additional files it needs. All handouts are written in LaTeX.

If all else fails, the handouts can be retrieved via anonymous ftp or by mail from Theory. To retrieve these files by ftp, run `ftp theory.lcs.mit.edu`, supplying "anonymous" as the name (account) and "guest" as the password. Files may then be fetched by first typing "`cd pub/6.044`" to change directories and then typing "`get` *filename*". If you get the files in this way, you will also need to get the files `6.044.sty`, `handout.sty` and `handouts-6044-fall-92.tex` from `pub/6.044/input` in order to run LaTeX on the handout files. To find out about retrieving files by mail, send mail to `archive-server@theory.lcs.mit.edu` containing the single word "help" in the body of the message.

Electronic mail. All students are encouraged to subscribe to the course mail list by sending email to `6044-secretary@theory.lcs.mit.edu`; other administrative requests should also be directed to this address.

To facilitate communication in the class, there are three electronic mail addresses:

```
6044-secretary@theory.lcs.mit.edu
6044-forum@theory.lcs.mit.edu
6044-staff@theory.lcs.mit.edu
```

The **`6044-forum`** mailing list is for general communication by students, the instructor, and the TA to the class; a message sent here will automatically be distributed to those on the mailing list. Students are strongly encouraged to use **`6044-forum`** to arrange study sessions, discuss ambiguities and problems with homework, and send comments to the whole class. The TA and instructor may also post bugs and corrections to homeworks and handouts to **`6044-forum`**.

Messages to the instructor, TA, or grader should be sent to **`6044-staff`**.

Pictures. You can help us learn who you are by giving us your photograph with your name on it. This is especially helpful if you later need a recommendation.

Diagnostic Quiz

You will not be graded on this quiz. Take it sometime after class, and return it in class on Monday, September 14. Be sure to indicate your name, the date and "6.044 Diagnostic Quiz" on your answer sheet.

Problem 1. Describe the function which is the composition of the integer successor function with itself (successor(x) = $x + 1$).

Problem 2. How many strings of length 4 are there over the alphabet $\{a, b, c\}$?

Problem 3.

(a) Which of these says that a mapping is *injective* (a synonym for "injective" is "one-to-one" or "monomorphism")?

1. Each element is mapped onto some element, possibly itself.
2. Each element is mapped onto an element different from itself.
3. No two elements are mapped onto the same element.
4. Every element has some element that maps onto it.

(b) What sets have the property that there is *no* injection from the set into a *proper* subset of itself?

Problem 4. Define a binary relation, $\preceq$, between sets A, B as follows:

$$A \preceq B \text{ iff } (\exists f : A \to B)(f \text{ is injective}).$$

That is, $A \preceq B$ means that the cardinality of A is less than or equal to the cardinality of B.

(a) What is the definition of "uncountable set"? Now express the definition in terms of $\preceq$. Give an example of an uncountable set.

(b) Which of the following properties does the relation $\preceq$ have? For those properties it fails, describe some simple sets $A, B, \ldots$ which provide a counterexample.

1. reflexivity
2. symmetricity
3. transitivity

Problem 5. Two Boolean formulas, $F_i(x_1, \ldots, x_n)$ for $i = 1, 2$, are *equivalent* iff they yield the same truth value for all truth assignments to the variables $x_1, \ldots, x_n$. (Electrical engineers typically use $\{0, 1\}$, but in this course we use $\{\mathbf{true}, \mathbf{false}\}$ as the set of truth values.)

(a) The Boolean binary operation *conjunction* (and), which our text writes as "&", is commutative, namely x_1 & x_2 is equivalent to x_2 & x_1. Describe a Boolean binary operation which is not commutative.

(b) Describe an infinite set of equivalent Boolean formulas.

(c) Explain why "equivalence" is actually an equivalence relation on formulas.

(d) Explain why there are only a finite number of equivalence classes of formulas with (at most) variables $x_1, \ldots, x_n$.

(e) How many?

Problem 6. For each of the following properties, describe a partial order of the set $\{1, 2, 3, 4, 5\}$ with the property that

(a) it is a total order,

(b) glb's ("greatest lower bounds" or "meets") exist for every pair of elements, but the lub (least upper bound) of 4 and 5 does not exist,

(c) glb's and lub's exist for all pairs, but the order is not total.

Problem 7. For any set, A, let $\mathcal{P}\text{ow}(A)$ be the powerset of A, namely, the set of all subsets of A. Exhibit the members of $\mathcal{P}\text{ow}(\mathcal{P}\text{ow}(\mathcal{P}\text{ow}(\emptyset)))$.

One-Step Rewriting Rules

Throughout this document, we will use **op** to range over syntactic operator symbols, and *op* to range over corresponding arithmetic or Boolean operations.

1 Rules for Arithmetic Expresssions, Aexp

$$\langle X, \sigma \rangle \rightarrow_1 \langle \sigma(X), \sigma \rangle$$

$$\frac{\langle a_0, \sigma \rangle \rightarrow_1 \langle a_0', \sigma \rangle}{\langle a_0 \ \mathbf{op}\ a_1, \sigma \rangle \rightarrow_1 \langle a_0' \ \mathbf{op}\ a_1, \sigma \rangle}$$

$$\frac{\langle a_1, \sigma \rangle \rightarrow_1 \langle a_1', \sigma \rangle}{\langle n \ \mathbf{op}\ a_1, \sigma \rangle \rightarrow_1 \langle n \ \mathbf{op}\ a_1', \sigma \rangle}$$

$$\langle n \ \mathbf{op}\ m, \sigma \rangle \rightarrow_1 \langle n \ op\ m, \sigma \rangle$$

op	*op*
$+$	the sum function
$-$	the subtraction function
$\times$	the multiplication function

Notice that

$$\langle 5 + 7, \sigma \rangle \rightarrow_1 \langle 12, \sigma \rangle$$

is an instance of the rule $\langle n \ \mathbf{op}\ m, \sigma \rangle \rightarrow_1 \langle n \ op\ m, \sigma \rangle$, but that

$$\langle 5 + 7, \sigma \rangle \rightarrow_1 \langle 5 + 7, \sigma \rangle$$

is *not* derivable at all.

2 Rules for Boolean Expressions, Bexp

$$\frac{\langle a_0, \sigma \rangle \rightarrow_1 \langle a_0', \sigma \rangle}{\langle a_0 \ \mathbf{op}\ a_1, \sigma \rangle \rightarrow_1 \langle a_0' \ \mathbf{op}\ a_1, \sigma \rangle}$$

$$\frac{\langle a_1, \sigma \rangle \rightarrow_1 \langle a_1', \sigma \rangle}{\langle n \ \mathbf{op}\ a_1, \sigma \rangle \rightarrow_1 \langle n \ \mathbf{op}\ a_1', \sigma \rangle}$$

$$\langle n \ \mathbf{op}\ m, \sigma \rangle \rightarrow_1 \langle n \ op\ m, \sigma \rangle$$

op	*op*
$=$	the equality predicate
$\leq$	the less than or equal to predicate

We next have the rules for Boolean negation:

$$\frac{\langle b, \sigma\rangle \rightarrow_1 \langle b', \sigma\rangle}{\langle \neg b, \sigma\rangle \rightarrow_1 \langle \neg b', \sigma\rangle}$$

$$\langle \neg\, \mathbf{true}, \sigma\rangle \rightarrow_1 \langle \mathbf{false}, \sigma\rangle$$

$$\langle \neg\, \mathbf{false}, \sigma\rangle \rightarrow_1 \langle \mathbf{true}, \sigma\rangle$$

Finally we have the rules for binary Boolean operators. We use **op** and *op* to range over the symbols and functions in the chart following the rules. We let $t, t_0, t_1, \ldots$ range over the set $\mathbf{T} = \{\mathbf{true}, \mathbf{false}\}$.

$$\frac{\langle b_0, \sigma\rangle \rightarrow_1 \langle b_0', \sigma\rangle}{\langle b_0 \;\mathbf{op}\; b_1, \sigma\rangle \rightarrow_1 \langle b_0' \;\mathbf{op}\; b_1, \sigma\rangle}$$

$$\frac{\langle b_1, \sigma\rangle \rightarrow_1 \langle b_1', \sigma\rangle}{\langle t_0 \;\mathbf{op}\; b_1, \sigma\rangle \rightarrow_1 \langle t_0 \;\mathbf{op}\; b_1', \sigma\rangle}$$

$$\langle t_0 \;\mathbf{op}\; t_1, \sigma\rangle \rightarrow_1 \langle t_0 \;op\; t_1, \sigma\rangle$$

op	*op*
$\wedge$	the conjunction operation (Boolean AND)
$\vee$	the disjunction operation (Boolean OR)

3 Rules for Commands, Com

Atomic Commands:

$$\langle \mathbf{skip}, \sigma\rangle \rightarrow_1 \sigma$$

$$\frac{\langle a, \sigma\rangle \rightarrow_1 \langle a', \sigma\rangle}{\langle X := a, \sigma\rangle \rightarrow_1 \langle X := a', \sigma\rangle}$$

$$\langle X := n, \sigma\rangle \rightarrow_1 \sigma[n/X]$$

Sequencing:

$$\frac{\langle c_0, \sigma\rangle \rightarrow_1 \langle c_0', \sigma'\rangle}{\langle (c_0; c_1), \sigma\rangle \rightarrow_1 \langle (c_0'; c_1), \sigma'\rangle}$$

$$\frac{\langle c_0, \sigma\rangle \rightarrow_1 \sigma'}{\langle (c_0; c_1), \sigma\rangle \rightarrow_1 \langle c_1, \sigma'\rangle}$$

Conditionals:

$$\frac{\langle b, \sigma\rangle \rightarrow_1 \langle b', \sigma\rangle}{\langle \textbf{if}\, b\, \textbf{then}\, c_0\, \textbf{else}\, c_1, \sigma\rangle \rightarrow_1 \langle \textbf{if}\, b'\, \textbf{then}\, c_0\, \textbf{else}\, c_1, \sigma\rangle}$$

$$\langle \textbf{if true then}\, c_0\, \textbf{else}\, c_1, \sigma\rangle \rightarrow_1 \langle c_0, \sigma\rangle$$

$$\langle \textbf{if false then}\, c_0\, \textbf{else}\, c_1, \sigma\rangle \rightarrow_1 \langle c_1, \sigma\rangle$$

While-loops:

$$\langle \textbf{while}\, b\, \textbf{do}\, c, \sigma\rangle \rightarrow_1 \langle \textbf{if}\, b\, \textbf{then} (c; \textbf{while}\, b\, \textbf{do}\, c)\, \textbf{else}\, \textbf{skip}, \sigma\rangle$$

Problem Set 1

Reading assignment. Winskel Chapters 1–2.

Due: 21 September 1992.

Problem 1. Winskel, Exercise 1.5.

Problem 2. Let Intsqrt $\in$ **Com** be the **IMP** command

$$\mathbf{while}\, \neg(N \leq M * M)\, \mathbf{do}\, M := M + 1,$$

and let $\sigma_0 \in \Sigma$ be a state such that $\sigma_0(M) = 1$ and $\sigma_0(N) = 9$.

2(a) Using the evaluation rules in Chapter 2 for the "natural semantics" of **IMP**, exhibit a derivation tree for an evaluation assertion of the form

$$\langle \text{Intsqrt}, \sigma_0 \rangle \rightarrow \sigma_0[m/M]$$

for some (necessarily unique) integer m.

2(b) Using the one-step rules of Handout 3, exhibit a sequence of one-step evaluation assertions which demonstrate that

$$\langle \text{Intsqrt}, \sigma_0 \rangle \rightarrow_1^* \sigma_0[m/M].$$

Problem 3. Prove that for any arithmetic expression $a_0 \in$ **Aexp** and state $\sigma \in \Sigma$ there is exactly one integer $n \in \mathbf{N}$ such that $\langle a_0, \sigma \rangle \rightarrow n$. (*Hint*: Induction on the definition of **Aexp**, *i.e.*, structural induction.)

Instructions for Problem Sets

1 Form of Solutions

Each problem is to be done on a *separate sheet* of *three-hole punched paper.* If a problem requires more than one sheet, staple these sheets together, but keep each problem separate. Do not use red ink. Mark the top of the paper with:

- your name,
- "6.044J/18.423J",
- the assignment number,
- the problem number, and
- the date.

Try to be as clear and precise as possible in your presentations. Problem grades are based not only on getting the right answer or otherwise demonstrating that you understand how a solution goes, but also on your ability to explain the solution or proof in a way helpful to a reader.

If you have doubts about the way your homework has been graded, first see the TA. Other questions and suggestions will be welcomed by both the instructor and the TA.

Problem sets will be collected at the beginning of class; graded problem sets will be returned at the end of class. Solutions will generally be available with the graded problem sets, one week after their submission.

2 Collaboration and References

You must write your own problem solutions and other assigned course work in your own words and entirely alone. On the other hand, you are encouraged to discuss the problems with one or two classmates before you write your solutions. If you do so, please be sure to

> *indicate the members of your discussion group*

on your solution.

Similarly, you are welcome to use other texts and references in doing homework, but if you find that a solution to an assigned problem has been given in such a reference, you should nevertheless rewrite the solution in your own words and *cite your source.*

3 Late Policy

Late homeworks should be submitted to the TA. If they can be graded without inconvenience, they will be. Late homeworks that are not graded will be kept for reference until after the final. No homework will be accepted after the solutions have been given out.

Diagnostic Quiz Solutions

Problem 1. Describe the function which is the composition of the integer successor function with itself (successor$(x) = x + 1$).

The composition of the successor function of itself is the "add two" function, add2, because

$$\begin{aligned}(\text{successor} \circ \text{successor})(x) &= \text{successor}(\text{successor}(x)) \\ &= \text{successor}(x+1) \\ &= (x+1)+1 \\ &= x+2 \\ &= \text{add2}(x)\end{aligned}$$

Problem 2. How many strings of length 4 are there over the alphabet $\{a, b, c\}$?

With three possibilities in each of four positions, there are $3^4 = 81$ possible strings.

Problem 3.

(a) A mapping, f, is "injective" or "one-to-one" or a "monomorphism on sets" if $f(x) = f(y)$ implies that $x = y$. Another way to say this is:

3. No two elements are mapped onto the same element.

Note that a mapping from a set A to a set B is *total* if every element of A is mapped onto some element (of B, of course). A mapping from a set A to a set B such that for every element of B, there is some element of A which maps onto it, is *surjective* or *onto*. A mapping that is both injective and surjective is a *bijection*.

(b) What sets have the property that there is *no* injection from the set into a *proper* subset of itself?

The finite sets. (Note that an infinite set *must* have an injection into a proper subset of itself; in fact, a set is infinite iff it has such an injection.)

Problem 4. Define a binary relation, $\preceq$, between sets A, B as follows:

$$A \preceq B \text{ iff } (\exists f : A \rightarrow B)(f \text{ is injective}).$$

That is, $A \preceq B$ means that the cardinality of A is less than or equal to the cardinality of B.

(a) What is the definition of "uncountable set"? Now express the definition in terms of $\preceq$. Give an example of an uncountable set.

A set A is uncountable iff there is no one-to-one and onto (bijective) correspondence between A and a subset of the integers. Thus, if $\mathbf{N}$ is the set of integers, then an uncountable set is any set A such that $A \not\preceq \mathbf{N}$. The real numbers, $\mathbb{R}$, are a well-known example of an uncountable set, as is the powerset of any infinite set.

(b) The relation $\preceq$ is reflexive ($A \preceq A$ for all sets A) and transitive (if $A \preceq B$ and $B \preceq C$ then $A \preceq C$), but not symmetric (there are sets A and B with $A \preceq B$ but $B \not\preceq A$, for example $\mathbf{N}$ and $\mathbb{R}$).

Problem 5. Two Boolean formulas, $F_i(x_1, \ldots, x_n)$ for $i = 1, 2$, are *equivalent* iff they yield the same truth value for all truth assignments to the variables $x_1, \ldots, x_n$. (Electrical engineers typically use $\{0, 1\}$, but in this course we use $\{\mathbf{true}, \mathbf{false}\}$ as the set of truth values.)

(a) Describe a Boolean binary operation which is not commutative.

The "implies" operation, often written $\Rightarrow$, is non-commutative: $\mathbf{false} \Rightarrow \mathbf{true}$, but $\mathbf{true} \not\Rightarrow \mathbf{false}$, so $x_1 \Rightarrow x_2$ and $x_2 \Rightarrow x_1$ are not equivalent.

(b) Describe an infinite set of equivalent Boolean formulas.

Define the formulas F_i as $F_0 \stackrel{\text{def}}{=} x_0$ and $F_{i+1} \stackrel{\text{def}}{=} (F_i \mathbin{\&} x_0)$ for all $i \geq 0$. These formulas are all equivalent, since for any truth assignment they all have the same truth value as x_0.

(c) Explain why "equivalence" is actually an equivalence relation on formulas.

An equivalence relation is a binary relation on a set A, that is, $R \subseteq A \times A$, which is reflexive ($a\ R\ a$ for all $a \in A$), transitive (if $a\ R\ a'$ and $a'\ R\ a''$ then $a\ R\ a''$), and symmetric ($a\ R\ a'$ iff $a'\ R\ a$).

Take any three formulas F_0, F_1 and F_2. Given any truth assignment to $x_1, \ldots, x_n$:

- F_0 yields the same truth value as itself, so "equivalence" is reflexive.
- If F_0 yields the same truth value as F_1, then F_1 yields the same truth value as F_0, so "equivalence" is symmetric.
- If F_0 yields the same truth value as F_1, and F_1 yields the same truth value as F_2, then clearly F_0 yields the same truth value as F_2, so "equivalence" is transitive.

Thus, "equivalence" is an equivalence relation on formulas.

(d) Explain why there are only a finite number of equivalence classes of formulas with (at most) variables $x_1, \ldots, x_n$.

(e) How many?

Formulas are equivalent iff their truth tables agree, so there are as many equivalence classes as there are truth tables. Given n variables, there are 2^{2^n} possible truth tables. To see this, think of a truth table as having a row for each possible truth assignment. A truth assignment consists of a **true** or **false** value for each variable, so there are (size of $\{\mathbf{true}, \mathbf{false}\})^n = 2^n$ possible truth assignments. Then, a truth table consists of an assignment of **true** or **false** to each truth assignment, so with 2^n truth assignments there are 2^{2^n} possible truth tables, giving us 2^{2^n} equivalence classes of formulas.

Problem 6. For each of the following properties, describe a partial order of the set $\{1, 2, 3, 4, 5\}$ with the property that

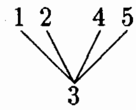
(a) it is a total order,

The ordering $(A, \leq)$ (in other words, $1 \leq 2 \leq 3 \leq 4 \leq 5$) is a total order: every pair of elements in the set is ordered.

(b) glb's ("greatest lower bounds" or "meets") exist for every pair of elements, but the lub (least upper bound) of 4 and 5 does not exist,

A lower bound of two elements a and b in a partial order is an element c (possibly equal to either a or b) which is less than or equal to both a and b. A greatest lower bound is a lower bound which is greater than or equal to any other lower bound. An upper bound and a least upper bound are defined analogously.

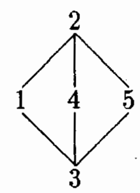
Thus, take the ordering in which 3 is less than or equal to any other number, but all other numbers are incomparable:



Then every pair of distinct numbers has a unique lower bound in the ordering, namely 3. Since there is only one lower bound possible, it is clearly the greatest lower bound. However, 4 and 5 have no upper bound at all, let alone a least upper bound.

(c) glb's and lub's exist for all pairs, but the order is not total.

A slight variant, in which we leave 3 as the least element but make 2 the greatest, gives us an ordering in which every pair of elements has both a least upper bound and a greatest lower bound, but the ordering is still not total.



Problem 7. Exhibit the members of $\mathcal{P}\mathrm{ow}(\mathcal{P}\mathrm{ow}(\mathcal{P}\mathrm{ow}(\emptyset)))$.

Given the empty set, $\emptyset$, with no members, the only subset is the (non-proper) subset $\emptyset$. Thus, if $\mathcal{P}\mathrm{ow}(\emptyset)$ is the set of all subsets of $\emptyset$, then

$$\mathcal{P}\mathrm{ow}(\emptyset) = \{\emptyset\}.$$

Moving along, the subsets of $\mathcal{P}\mathrm{ow}(\emptyset)$ are $\emptyset$ and $\{\emptyset\}$, so

$$\mathcal{P}\mathrm{ow}(\mathcal{P}\mathrm{ow}(\emptyset)) = \{\emptyset, \{\emptyset\}\}.$$

Then, $\mathcal{P}\mathrm{ow}(\mathcal{P}\mathrm{ow}(\emptyset))$ has subsets $\emptyset$, $\{\emptyset\}$, $\{\{\emptyset\}\}$, and $\{\emptyset, \{\emptyset\}\}$, so

$$\mathcal{P}\mathrm{ow}(\mathcal{P}\mathrm{ow}(\mathcal{P}\mathrm{ow}(\emptyset))) = \{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \{\emptyset, \{\emptyset\}\}\}$$

Problem Set 2

Due: 28 September 1992.

In this problem set, we examine an extension of **IMP** by a further expression construct

$$c\,\mathbf{resultis}\,a$$

where c is a command and a is an arithmetic expression. To evaluate such an expression, the command c is first executed and then a evaluated in the changed state.

We call the extended language $\mathbf{IMP}_r$. In contrast to **IMP**, the expression evaluation in $\mathbf{IMP}_r$ now has side effects—the evaluation may change the state. To model this in natural operational semantics, we modify evaluation assertions for expressions to be of the form

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

where σ' is the state that results from the evaluation of a in original state σ. The evaluation relation, $\rightarrow_r$, for $\mathbf{IMP}_r$ is given at the end of this handout.

Problem 1. Recall that for $a_0, a_1 \in \mathbf{Aexp}$ of **IMP**

$$a_0 \sim a_1 \quad \text{iff} \quad (\forall n \in \mathbf{N}.\forall \sigma \in \Sigma.\ \langle a_0, \sigma\rangle \rightarrow n \text{ iff } \langle a_1, \sigma\rangle \rightarrow n).$$

The corresponding notion of $\sim$ for $\mathbf{IMP}_r$ is

$$a_0 \sim a_1 \quad \text{iff} \quad (\forall n \in \mathbf{N}.\forall \sigma, \sigma' \in \Sigma.\ \langle a_0, \sigma\rangle \rightarrow \langle n, \sigma'\rangle \text{ iff } \langle a_1, \sigma\rangle \rightarrow \langle n, \sigma'\rangle).$$

(a) Prove that $a + a \sim 2 \times a$ for $a \in \mathbf{Aexp}$ of **IMP**.

(b) Exhibit a simple $a \in \mathbf{Aexp}_r$ of $\mathbf{IMP}_r$ for which this fails.

Problem 2. Note that **IMP** is a subset of $\mathbf{IMP}_r$, in the sense that any command $c \in \mathbf{IMP}$ can be evaluated using either the natural semantics of **IMP** or of $\mathbf{IMP}_r$. Prove that these semantics agree, *i.e.*, prove that for any $c \in \mathbf{IMP}$, $\langle c, \sigma\rangle \rightarrow_r \sigma'$ iff $\langle c, \sigma\rangle \rightarrow \sigma'$. (*Hint*: Formulate a similar if-and-only-if connection between **IMP** and $\mathbf{IMP}_r$ natural semantics for **Aexp**'s and **Bexp**'s, and prove all three "iff" statements simultaneously by induction on derivations, rule induction, or some similar method.)

Problem 3. Describe a one-step semantics for $\mathbf{IMP}_r$ in the same style and with similar properties to that for **IMP**. In particular, it should be the case that $\rightarrow_{r,1}$ is a total function on configurations and that one-step and natural semantics are "equivalent" in the precise sense that:

$$\langle c, \sigma \rangle \rightarrow_{r,1}^{*} \sigma' \text{ iff } \langle c, \sigma \rangle \rightarrow_r \sigma'$$

Problem 4. Briefly describe how to extend to $\mathbf{IMP}_r$ the proof of the equivalence of natural and one-step operational semantics of **IMP**. Note in particular which induction hypotheses have to be modified and how.

Evaluation Rules for $\mathbf{IMP}_r$

Formally, we'll define the following $\rightarrow_r$ relation on a language $\mathbf{IMP}_r$, which is **IMP** extended with the **resultis** construction:

Rules for $\mathbf{Aexp}_r$

$$\langle n, \sigma\rangle \rightarrow_r \langle n, \sigma\rangle$$

$$\langle X, \sigma\rangle \rightarrow_r \langle \sigma(n), \sigma\rangle$$

$$\frac{\langle a_0, \sigma\rangle \rightarrow_r \langle n_0, \sigma''\rangle, \quad \langle a_1, \sigma''\rangle \rightarrow_r \langle n_1, \sigma'\rangle}{\langle a_0 + a_1, \sigma\rangle \rightarrow_r \langle n, \sigma'\rangle}$$

where n is the sum of n_0 and n_1.

There are similar rules for $\times$ and $-$.

$$\frac{\langle c, \sigma\rangle \rightarrow_r \sigma'', \quad \langle a, \sigma''\rangle \rightarrow_r \langle n, \sigma'\rangle}{\langle c\, \mathbf{resultis}\, a, \sigma\rangle \rightarrow_r \langle n, \sigma'\rangle}$$

Rules for $\mathbf{Bexp}_r$

$$\langle t, \sigma\rangle \rightarrow_r \langle t, \sigma\rangle$$

$$\frac{\langle a_0, \sigma\rangle \rightarrow_r \langle n_0, \sigma''\rangle, \quad \langle a_1, \sigma''\rangle \rightarrow_r \langle n_1, \sigma'\rangle}{\langle a_0 = a_1, \sigma\rangle \rightarrow_r \langle t, \sigma'\rangle}$$

where $t = \mathbf{true}$ if n_0 and n_1 are equal, otherwise $t = \mathbf{false}$.

There is a similar rule for $\leq$.

$$\frac{\langle b, \sigma\rangle \rightarrow_r \langle t, \sigma'\rangle}{\langle \neg b, \sigma\rangle \rightarrow_r \langle t', \sigma'\rangle}$$

where t' is the negation of t.

$$\frac{\langle b_0, \sigma\rangle \rightarrow_r \langle t_0, \sigma''\rangle, \quad \langle b_1, \sigma''\rangle \rightarrow_r \langle t_1, \sigma'\rangle}{\langle b_0 \wedge b_1, \sigma\rangle \rightarrow_r \langle t, \sigma'\rangle}$$

where t is **true** if $t_0 = \mathbf{true}$ and $t_1 = \mathbf{true}$, and is **false** otherwise.

There is a similar rule for $\vee$.

Rules for $\mathbf{Com}_r$

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

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

$$\frac{\langle c_0, \sigma\rangle \rightarrow_r \sigma'', \quad \langle c_1, \sigma''\rangle \rightarrow_r \sigma'}{\langle (c_0; c_1), \sigma\rangle \rightarrow_r \sigma'}$$

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

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

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

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

Solutions to Problem Set 1

Problem 1. Assume that we have sets X and Y, and that Y has at least two distinct elements y_0 and y_1. We can extend Cantor's argument about powersets to demonstrate that there is no one-to-one correspondence between X and $(X \to Y)$.

First, assume that there *is* some one-to-one correspondence, θ, between a set X and the set of functions $(X \to Y)$. Thus, for each element $x \in X$ there is a corresponding function $\theta(x)$, and for all functions $f : X \to Y$ there is some $x' \in X$ such that $\theta(x') = f$. Consider then the function $f : X \to Y$ defined by

$$f(x) = \begin{cases} y_0 & \text{if } (\theta(x))(x) \neq y_0, \\ y_1 & \text{otherwise.} \end{cases}$$

Now, by this definition, $f(x) \neq (\theta(x))(x)$ for all $x \in X$. But $f = \theta(x')$ for some $x' \in X$, so for all $x \in X$, $(\theta(x'))(x) \neq (\theta(x))(x)$. Now let $x = x'$, and we obtain a contradiction.

Problem 2. For typesetting convenience, let

$$S \stackrel{\text{def}}{=} \text{Intsqrt} = \textbf{while}\,\neg(N \leq (M \times M))\,\textbf{do}\,M := M + 1$$
$$C \stackrel{\text{def}}{=} M := M + 1$$

Thus, we have $S = \textbf{while}\,\neg(N \leq (M \times M))\,\textbf{do}\,C$. This will let us shrink the following derivation trees a little. They need it.

2(a) Here we have a "natural semantics" derivation for $\langle \text{Intsqrt}, \sigma_0 \rangle$. A derivation of this sort forms a single large tree, often with several preconditions for any given conclusion. Due to the size of the derivation tree, the tree has been broken in two places. The connections are indicated by the boldface numbers.

$$\dfrac{\dfrac{\dfrac{\langle N, \sigma_0[3/M]\rangle \to 9 \qquad \dfrac{\langle M, \sigma_0[3/M]\rangle \to 3}{\langle M \times M, \sigma_0[3/M]\rangle \to 9}}{\langle N \leq (M \times M), \sigma_0[3/M]\rangle \to \textbf{true}}}{\langle \neg(N \leq (M \times M)), \sigma_0[3/M]\rangle \to \textbf{false}}}{\langle S, \sigma_0[3/M]\rangle \to \sigma_0[3/M] \qquad \textbf{(1)}}$$

$$\dfrac{\dfrac{\dfrac{\langle N, \sigma_0[2/M]\rangle \rightarrow 9 \quad \dfrac{\langle M, \sigma_0[2/M]\rangle \rightarrow 2}{\langle M \times M, \sigma_0[2/M]\rangle \rightarrow 4}}{\langle N \leq (M \times M), \sigma_0[2/M]\rangle \rightarrow \textbf{false}}}{\langle \neg(N \leq (M \times M)), \sigma_0[2/M]\rangle \rightarrow \textbf{true}} \quad \dfrac{\dfrac{\langle M, \sigma_0[2/M]\rangle \rightarrow 2 \quad \langle 1, \sigma_0[2/M]\rangle \rightarrow 1}{\langle M+1, \sigma_0[2/M]\rangle \rightarrow 3}}{\langle C, \sigma_0[2/M]\rangle \rightarrow \sigma_0[3/M]} \quad (1)}{\langle S, \sigma_0[2/M]\rangle \rightarrow \sigma_0[3/M] \quad (2)}$$

$$\dfrac{\dfrac{\dfrac{\langle N, \sigma_0\rangle \rightarrow 9 \quad \dfrac{\langle M, \sigma_0\rangle \rightarrow 1}{\langle M \times M, \sigma_0\rangle \rightarrow 1}}{\langle N \leq (M \times M), \sigma_0\rangle \rightarrow \textbf{false}}}{\langle \neg(N \leq (M \times M)), \sigma_0\rangle \rightarrow \textbf{true}} \quad \dfrac{\dfrac{\langle M, \sigma_0\rangle \rightarrow 1 \quad \langle 1, \sigma_0\rangle \rightarrow 1}{\langle M+1, \sigma_0\rangle \rightarrow 2}}{\langle C, \sigma_0\rangle \rightarrow \sigma_0[2/M]} \quad (2)}{\langle S, \sigma_0\rangle \rightarrow \sigma_0[3/M]}$$

Thus, according to this derivation, we wind up in a state identical to the original one, but with M having the value 3.

2(b) $\langle \text{Intsqrt}, \sigma_0\rangle$ evaluates to $\sigma_0[3/M]$ in 31 "steps," shown here without their individual derivation trees:

$$
\begin{aligned}
\langle S, \sigma_0\rangle &\rightarrow_1 \langle \textbf{if}\, \neg(N \leq (M \times M))\, \textbf{then}(C; S)\, \textbf{else}\, \textbf{skip}, \sigma_0\rangle && (1)\\
&\rightarrow_1 \langle \textbf{if}\, \neg(9 \leq (M \times M))\, \textbf{then}(C; S)\, \textbf{else}\, \textbf{skip}, \sigma_0\rangle && (2)\\
&\rightarrow_1 \langle \textbf{if}\, \neg(9 \leq (1 \times M))\, \textbf{then}(C; S)\, \textbf{else}\, \textbf{skip}, \sigma_0\rangle && (3)\\
&\rightarrow_1 \langle \textbf{if}\, \neg(9 \leq (1 \times 1))\, \textbf{then}(C; S)\, \textbf{else}\, \textbf{skip}, \sigma_0\rangle && (4)\\
&\rightarrow_1 \langle \textbf{if}\, \neg(9 \leq 1)\, \textbf{then}(C; S)\, \textbf{else}\, \textbf{skip}, \sigma_0\rangle && (5)\\
&\rightarrow_1 \langle \textbf{if}\, \neg\, \textbf{false}\, \textbf{then}(C; S)\, \textbf{else}\, \textbf{skip}, \sigma_0\rangle && (6)\\
&\rightarrow_1 \langle \textbf{if}\, \textbf{true}\, \textbf{then}(C; S)\, \textbf{else}\, \textbf{skip}, \sigma_0\rangle && (7)\\
&\rightarrow_1 \langle (M := M + 1; S), \sigma_0\rangle && (8)\\
&\rightarrow_1 \langle (M := 1 + 1; S), \sigma_0\rangle && (9)\\
&\rightarrow_1 \langle (M := 2; S), \sigma_0\rangle && (10)\\
&\rightarrow_1 \langle S, \sigma_0[2/M]\rangle && (11)\\
&\rightarrow_1 \langle \textbf{if}\, \neg(N \leq (M \times M))\, \textbf{then}(C; S)\, \textbf{else}\, \textbf{skip}, \sigma_0[2/M]\rangle && (12)\\
&\rightarrow_1 \langle \textbf{if}\, \neg(9 \leq (M \times M))\, \textbf{then}(C; S)\, \textbf{else}\, \textbf{skip}, \sigma_0[2/M]\rangle && (13)\\
&\rightarrow_1 \langle \textbf{if}\, \neg(9 \leq (2 \times M))\, \textbf{then}(C; S)\, \textbf{else}\, \textbf{skip}, \sigma_0[2/M]\rangle && (14)\\
&\rightarrow_1 \langle \textbf{if}\, \neg(9 \leq (2 \times 2))\, \textbf{then}(C; S)\, \textbf{else}\, \textbf{skip}, \sigma_0[2/M]\rangle && (15)\\
&\rightarrow_1 \langle \textbf{if}\, \neg(9 \leq 4)\, \textbf{then}(C; S)\, \textbf{else}\, \textbf{skip}, \sigma_0[2/M]\rangle && (16)\\
&\rightarrow_1 \langle \textbf{if}\, \neg\, \textbf{false}\, \textbf{then}(C; S)\, \textbf{else}\, \textbf{skip}, \sigma_0[2/M]\rangle && (17)\\
&\rightarrow_1 \langle \textbf{if}\, \textbf{true}\, \textbf{then}(C; S)\, \textbf{else}\, \textbf{skip}, \sigma_0[2/M]\rangle && (18)
\end{aligned}
$$

$$\rightarrow_1 \langle (M := M + 1; S), \sigma_0[2/M] \rangle \tag{19}$$
$$\rightarrow_1 \langle (M := 2 + 1; S), \sigma_0[2/M] \rangle \tag{20}$$
$$\rightarrow_1 \langle (M := 3; S), \sigma_0[2/M] \rangle \tag{21}$$
$$\rightarrow_1 \langle S, \sigma_0[3/M] \rangle \tag{22}$$
$$\rightarrow_1 \langle \mathbf{if}\, \neg(N \leq (M \times M))\, \mathbf{then}(C; S)\, \mathbf{else\, skip}, \sigma_0[3/M] \rangle \tag{23}$$
$$\rightarrow_1 \langle \mathbf{if}\, \neg(9 \leq (M \times M))\, \mathbf{then}(C; S)\, \mathbf{else\, skip}, \sigma_0[3/M] \rangle \tag{24}$$
$$\rightarrow_1 \langle \mathbf{if}\, \neg(9 \leq (3 \times M))\, \mathbf{then}(C; S)\, \mathbf{else\, skip}, \sigma_0[3/M] \rangle \tag{25}$$
$$\rightarrow_1 \langle \mathbf{if}\, \neg(9 \leq (3 \times 3))\, \mathbf{then}(C; S)\, \mathbf{else\, skip}, \sigma_0[3/M] \rangle \tag{26}$$
$$\rightarrow_1 \langle \mathbf{if}\, \neg(9 \leq 9)\, \mathbf{then}(C; S)\, \mathbf{else\, skip}, \sigma_0[3/M] \rangle \tag{27}$$
$$\rightarrow_1 \langle \mathbf{if}\, \neg\, \mathbf{true\, then}(C; S)\, \mathbf{else\, skip}, \sigma_0[3/M] \rangle \tag{28}$$
$$\rightarrow_1 \langle \mathbf{if\, false\, then}(C; S)\, \mathbf{else\, skip}, \sigma_0[3/M] \rangle \tag{29}$$
$$\rightarrow_1 \langle \mathbf{skip}, \sigma_0[3/M] \rangle \tag{30}$$
$$\rightarrow_1 \sigma_0[3/M] \tag{31}$$

Problem 3. See Proposition 3.3, on page 30 of Winskel's text.

6.044J/18.423J: Computability, Programming, and Logic — Handout 9
Massachusetts Institute of Technology — 23 September 1992

Full Derivation Trees for 2b on Problem Set 1

Though it wasn't required for the problem set, we thought it might be helpful to see the derivations for each step in the solution to problem 2b. As in Handout 8, we've introduced the following abbreviations:

$$S \stackrel{\text{def}}{=} \text{Intsqrt} = \textbf{while}\,\neg(N \le (M \times M))\,\textbf{do}\,M := M + 1$$
$$C \stackrel{\text{def}}{=} M := M + 1$$

The derivations are then as follows:

1. $$\dfrac{}{\langle S, \sigma_0\rangle \to_1 \langle \textbf{if}\,\neg(N \le (M\times M))\,\textbf{then}(C;S)\,\textbf{else}\,\textbf{skip}, \sigma_0\rangle}$$

2. $$\dfrac{\dfrac{\langle N, \sigma_0\rangle \to_1 9}{\langle N \le (M\times M), \sigma_0\rangle \to_1 \langle 9 \le (M\times M), \sigma_0\rangle}}{\begin{array}{l}\langle \textbf{if}\,\neg(N \le (M\times M))\,\textbf{then}(C;S)\,\textbf{else}\,\textbf{skip}, \sigma_0\rangle \to_1 \\ \quad \langle \textbf{if}\,\neg(9 \le (M\times M))\,\textbf{then}(C;S)\,\textbf{else}\,\textbf{skip}, \sigma_0\rangle\end{array}}$$

3. $$\dfrac{\dfrac{\dfrac{\langle M, \sigma_0\rangle \to_1 1}{\langle M\times M, \sigma_0\rangle \to_1 \langle 1\times M, \sigma_0\rangle}}{\langle 9 \le (M\times M), \sigma_0\rangle \to_1 \langle 9 \le (1\times M), \sigma_0\rangle}}{\begin{array}{l}\langle \textbf{if}\,\neg(9 \le (M\times M))\,\textbf{then}(C;S)\,\textbf{else}\,\textbf{skip}, \sigma_0\rangle \to_1 \\ \quad \langle \textbf{if}\,\neg(9 \le (1\times M))\,\textbf{then}(C;S)\,\textbf{else}\,\textbf{skip}, \sigma_0\rangle\end{array}}$$

4. $$\dfrac{\dfrac{\dfrac{\langle M, \sigma_0\rangle \to_1 1}{\langle 1\times M, \sigma_0\rangle \to_1 \langle 1\times 1, \sigma_0\rangle}}{\langle 9 \le (1\times M), \sigma_0\rangle \to_1 \langle 9 \le (1\times 1), \sigma_0\rangle}}{\begin{array}{l}\langle \textbf{if}\,\neg(9 \le (1\times M))\,\textbf{then}(C;S)\,\textbf{else}\,\textbf{skip}, \sigma_0\rangle \to_1 \\ \quad \langle \textbf{if}\,\neg(9 \le (1\times 1))\,\textbf{then}(C;S)\,\textbf{else}\,\textbf{skip}, \sigma_0\rangle\end{array}}$$

5. $$\dfrac{\dfrac{\langle 1\times 1, \sigma_0\rangle \to_1 1}{\langle 9 \le (1\times 1), \sigma_0\rangle \to_1 \langle 9 \le 1, \sigma_0\rangle}}{\begin{array}{l}\langle \textbf{if}\,\neg(9 \le (1\times 1))\,\textbf{then}(C;S)\,\textbf{else}\,\textbf{skip}, \sigma_0\rangle \to_1 \\ \quad \langle \textbf{if}\,\neg(9 \le 1)\,\textbf{then}(C;S)\,\textbf{else}\,\textbf{skip}, \sigma_0\rangle\end{array}}$$

6. $$\dfrac{\langle 9 \le 1, \sigma_0\rangle \to_1 \langle \textbf{false}, \sigma_0\rangle}{\langle \textbf{if}\,\neg(9 \le 1)\,\textbf{then}(C;S)\,\textbf{else}\,\textbf{skip}, \sigma_0\rangle \to_1 \langle \textbf{if}\,\neg\,\textbf{false}\,\textbf{then}(C;S)\,\textbf{else}\,\textbf{skip}, \sigma_0\rangle}$$

7. $$\frac{\langle \neg\,\mathbf{false}, \sigma_0\rangle \to_1 \langle \mathbf{true}, \sigma_0\rangle}{\langle \mathbf{if}\ \neg\,\mathbf{false}\ \mathbf{then}(C;S)\ \mathbf{else}\ \mathbf{skip}, \sigma_0\rangle \to_1 \langle \mathbf{if}\ \mathbf{true}\ \mathbf{then}(C;S)\ \mathbf{else}\ \mathbf{skip}, \sigma_0\rangle}$$

8. $$\frac{}{\langle \mathbf{if}\ \mathbf{true}\ \mathbf{then}(C;S)\ \mathbf{else}\,\mathbf{skip}, \sigma_0\rangle \to_1 \langle (C;S), \sigma_0\rangle}$$

9. $$\dfrac{\dfrac{\dfrac{\langle M, \sigma_0\rangle \to_1 1}{\langle M+1, \sigma_0\rangle \to_1 \langle 1+1, \sigma_0\rangle}}{\langle M := M+1, \sigma_0\rangle \to_1 \langle M := 1+1, \sigma_0\rangle}}{\langle (C;S), \sigma_0\rangle \to_1 \langle M := 1+1; S, \sigma_0\rangle}$$

10. $$\dfrac{\dfrac{\langle 1+1, \sigma_0\rangle \to_1 \langle 2, \sigma_0\rangle}{\langle M := 1+1, \sigma_0\rangle \to_1 \langle M := 2, \sigma_0\rangle}}{\langle (M := 1+1; S), \sigma_0\rangle \to_1 \langle M := 2; S, \sigma_0\rangle}$$

11. $$\frac{\langle M := 2, \sigma_0\rangle \to_1 \sigma_0[2/M]}{\langle (M := 2; S), \sigma_0\rangle \to_1 \langle S, \sigma_0[2/M]\rangle}$$

12. $$\frac{}{\langle S, \sigma_0[2/M]\rangle \to_1 \langle \mathbf{if}\ \neg(N \le (M \times M))\ \mathbf{then}(C;S)\ \mathbf{else}\ \mathbf{skip}, \sigma_0[2/M]\rangle}$$

13. $$\dfrac{\dfrac{\langle N, \sigma_0[2/M]\rangle \to_1 9}{\langle N \le (M \times M), \sigma_0[2/M]\rangle \to_1 \langle 9 \le (M \times M), \sigma_0[2/M]\rangle}}{\begin{array}{l}\langle \mathbf{if}\ \neg(N \le (M \times M))\ \mathbf{then}(C;S)\ \mathbf{else}\,\mathbf{skip}, \sigma_0[2/M]\rangle \to_1 \\ \quad \langle \mathbf{if}\ \neg(9 \le (M \times M))\ \mathbf{then}(C;S)\ \mathbf{else}\ \mathbf{skip}, \sigma_0[2/M]\rangle\end{array}}$$

14. $$\dfrac{\dfrac{\dfrac{\langle M, \sigma_0[2/M]\rangle \to_1 2}{\langle M \times M, \sigma_0[2/M]\rangle \to_1 \langle 2 \times M, \sigma_0[2/M]\rangle}}{\langle 9 \le (M \times M), \sigma_0[2/M]\rangle \to_1 \langle 9 \le (2 \times M), \sigma_0[2/M]\rangle}}{\begin{array}{l}\langle \mathbf{if}\ \neg(9 \le (M \times M))\ \mathbf{then}(C;S)\ \mathbf{else}\,\mathbf{skip}, \sigma_0[2/M]\rangle \to_1 \\ \quad \langle \mathbf{if}\ \neg(9 \le (2 \times M))\ \mathbf{then}(C;S)\ \mathbf{else}\,\mathbf{skip}, \sigma_0[2/M]\rangle\end{array}}$$

15. $$\dfrac{\dfrac{\dfrac{\langle M, \sigma_0[2/M]\rangle \to_1 2}{\langle 2 \times M, \sigma_0[2/M]\rangle \to_1 \langle 2 \times 2, \sigma_0[2/M]\rangle}}{\langle 9 \le (2 \times M), \sigma_0[2/M]\rangle \to_1 \langle 9 \le (2 \times 2), \sigma_0[2/M]\rangle}}{\begin{array}{l}\langle \mathbf{if}\ \neg(9 \le (2 \times M))\ \mathbf{then}(C;S)\ \mathbf{else}\ \mathbf{skip}, \sigma_0[2/M]\rangle \to_1 \\ \quad \langle \mathbf{if}\ \neg(9 \le (2 \times 2))\ \mathbf{then}(C;S)\ \mathbf{else}\ \mathbf{skip}, \sigma_0[2/M]\rangle\end{array}}$$

16. $$\dfrac{\dfrac{\langle 2\times 2, \sigma_0[2/M]\rangle \longrightarrow_1 4}{\langle 9 \le (2\times 2), \sigma_0[2/M]\rangle \longrightarrow_1 \langle 9 \le 4, \sigma_0[2/M]\rangle}}{\begin{array}{c}\langle \textbf{if}\ \neg(9 \le (2\times 2))\ \textbf{then}(C;S)\ \textbf{else}\ \textbf{skip}, \sigma_0[2/M]\rangle \longrightarrow_1 \\ \langle \textbf{if}\ \neg(9 \le 4)\ \textbf{then}(C;S)\ \textbf{else}\ \textbf{skip}, \sigma_0[2/M]\rangle\end{array}}$$

17. $$\dfrac{\langle 9 \le 4, \sigma_0[2/M]\rangle \longrightarrow_1 \langle \textbf{false}, \sigma_0[2/M]\rangle}{\begin{array}{c}\langle \textbf{if}\ \neg(9 \le 4)\ \textbf{then}(C;S)\ \textbf{else}\ \textbf{skip}, \sigma_0[2/M]\rangle \longrightarrow_1 \\ \langle \textbf{if}\ \neg\,\textbf{false}\ \textbf{then}(C;S)\ \textbf{else}\ \textbf{skip}, \sigma_0[2/M]\rangle\end{array}}$$

18. $$\dfrac{\langle \neg\,\textbf{false}, \sigma_0[2/M]\rangle \longrightarrow_1 \langle \textbf{true}, \sigma_0[2/M]\rangle}{\begin{array}{c}\langle \textbf{if}\ \neg\,\textbf{false}\ \textbf{then}(C;S)\ \textbf{else}\,\textbf{skip}, \sigma_0[2/M]\rangle \longrightarrow_1 \\ \langle \textbf{if}\ \textbf{true}\ \textbf{then}(C;S)\ \textbf{else}\,\textbf{skip}, \sigma_0[2/M]\rangle\end{array}}$$

19. $$\dfrac{}{\langle \textbf{if}\ \textbf{true}\ \textbf{then}(C;S)\ \textbf{else}\,\textbf{skip}, \sigma_0[2/M]\rangle \longrightarrow_1 \langle (C;S), \sigma_0[2/M]\rangle}$$

20. $$\dfrac{\dfrac{\dfrac{\langle M, \sigma_0[2/M]\rangle \longrightarrow_1 2}{\langle M+1, \sigma_0[2/M]\rangle \longrightarrow_1 \langle 2+1, \sigma_0[2/M]\rangle}}{\langle M := M+1, \sigma_0[2/M]\rangle \longrightarrow_1 \langle M := 2+1, \sigma_0[2/M]\rangle}}{\langle (C;S), \sigma_0[2/M]\rangle \longrightarrow_1 \langle M := 2+1; S, \sigma_0[2/M]\rangle}$$

21. $$\dfrac{\dfrac{\langle 2+1, \sigma_0[2/M]\rangle \longrightarrow_1 \langle 3, \sigma_0[2/M]\rangle}{\langle M := 2+1, \sigma_0[2/M]\rangle \longrightarrow_1 \langle M := 3, \sigma_0[2/M]\rangle}}{\langle (M := 2+1; S), \sigma_0[2/M]\rangle \longrightarrow_1 \langle M := 3; S, \sigma_0[2/M]\rangle}$$

22. $$\dfrac{\langle M := 3, \sigma_0[2/M]\rangle \longrightarrow_1 \sigma_0[3/M]}{\langle (M := 3; S), \sigma_0[2/M]\rangle \longrightarrow_1 \langle S, \sigma_0[3/M]\rangle}$$

23. $$\dfrac{}{\langle S, \sigma_0[3/M]\rangle \longrightarrow_1 \langle \textbf{if}\ \neg(N \le (M\times M))\ \textbf{then}(C;S)\ \textbf{else}\,\textbf{skip}, \sigma_0[3/M]\rangle}$$

24. $$\dfrac{\dfrac{\langle N, \sigma_0[3/M]\rangle \longrightarrow_1 9}{\langle N \le (M\times M), \sigma_0[3/M]\rangle \longrightarrow_1 \langle 9 \le (M\times M), \sigma_0[3/M]\rangle}}{\begin{array}{c}\langle \textbf{if}\ \neg(N \le (M\times M))\ \textbf{then}(C;S)\ \textbf{else}\,\textbf{skip}, \sigma_0[3/M]\rangle \longrightarrow_1 \\ \langle \textbf{if}\ \neg(9 \le (M\times M))\ \textbf{then}(C;S)\ \textbf{else}\,\textbf{skip}, \sigma_0[3/M]\rangle\end{array}}$$

25. $$\dfrac{\dfrac{\dfrac{\langle M, \sigma_0[3/M]\rangle \longrightarrow_1 3}{\langle M\times M, \sigma_0[3/M]\rangle \longrightarrow_1 \langle 2\times M, \sigma_0[3/M]\rangle}}{\langle 9 \le (M\times M), \sigma_0[3/M]\rangle \longrightarrow_1 \langle 9 \le (3\times M), \sigma_0[3/M]\rangle}}{\begin{array}{c}\langle \textbf{if}\ \neg(9 \le (M\times M))\ \textbf{then}(C;S)\ \textbf{else}\,\textbf{skip}, \sigma_0[3/M]\rangle \longrightarrow_1 \\ \langle \textbf{if}\ \neg(9 \le (3\times M))\ \textbf{then}(C;S)\ \textbf{else}\,\textbf{skip}, \sigma_0[3/M]\rangle\end{array}}$$

26.
$$\dfrac{\dfrac{\dfrac{\langle M, \sigma_0[3/M]\rangle \rightarrow_1 3}{\langle 3\times M, \sigma_0[3/M]\rangle \rightarrow_1 \langle 3\times 3, \sigma_0[3/M]\rangle}}{\langle 9 \le (3\times M), \sigma_0[3/M]\rangle \rightarrow_1 \langle 9 \le (3\times 3), \sigma_0[3/M]\rangle}}{\begin{array}{l}\langle \mathbf{if}\,\neg(9 \le (3\times M))\,\mathbf{then}(C;S)\,\mathbf{else\,skip}, \sigma_0[3/M]\rangle \rightarrow_1 \\ \quad \langle \mathbf{if}\,\neg(9 \le (3\times 3))\,\mathbf{then}(C;S)\,\mathbf{else\,skip}, \sigma_0[3/M]\rangle\end{array}}$$

27.
$$\dfrac{\dfrac{\langle 3\times 3, \sigma_0[3/M]\rangle \rightarrow_1 9}{\langle 9 \le (3\times 3), \sigma_0[3/M]\rangle \rightarrow_1 \langle 9 \le 9, \sigma_0[3/M]\rangle}}{\begin{array}{l}\langle \mathbf{if}\,\neg(9 \le (3\times 3))\,\mathbf{then}(C;S)\,\mathbf{else\,skip}, \sigma_0[3/M]\rangle \rightarrow_1 \\ \quad \langle \mathbf{if}\,\neg(9 \le 9)\,\mathbf{then}(C;S)\,\mathbf{else\,skip}, \sigma_0[3/M]\rangle\end{array}}$$

28.
$$\dfrac{\langle 9 \le 9, \sigma_0[3/M]\rangle \rightarrow_1 \langle \mathbf{true}, \sigma_0[3/M]\rangle}{\begin{array}{l}\langle \mathbf{if}\,\neg(9 \le 9)\,\mathbf{then}(C;S)\,\mathbf{else\,skip}, \sigma_0[3/M]\rangle \rightarrow_1 \\ \quad \langle \mathbf{if}\,\neg\,\mathbf{true}\,\mathbf{then}(C;S)\,\mathbf{else\,skip}, \sigma_0[3/M]\rangle\end{array}}$$

29.
$$\dfrac{\langle \neg\,\mathbf{true}, \sigma_0[3/M]\rangle \rightarrow_1 \langle \mathbf{false}, \sigma_0[3/M]\rangle}{\begin{array}{l}\langle \mathbf{if}\,\neg\,\mathbf{true}\,\mathbf{then}(C;S)\,\mathbf{else\,skip}, \sigma_0[3/M]\rangle \rightarrow_1 \\ \quad \langle \mathbf{if\,false}\,\mathbf{then}(C;S)\,\mathbf{else\,skip}, \sigma_0[3/M]\rangle\end{array}}$$

30.
$$\dfrac{}{\langle \mathbf{if\,false}\,\mathbf{then}(C;S)\,\mathbf{else\,skip}, \sigma_0[3/M]\rangle \rightarrow_1 \langle \mathbf{skip}, \sigma_0[3/M]\rangle}$$

31.
$$\dfrac{}{\langle \mathbf{skip}, \sigma_0[3/M]\rangle \rightarrow_1 \sigma_0[3/M]}$$

Thus, after 31 "steps," we are able to conclude that $\langle \text{Intsqrt}, \sigma_0\rangle \rightarrow_1^* \sigma_0[3/M]$.

Equivalence of $\rightarrow_1$ and $\rightarrow$

The goal is to prove

Theorem 1. For all $c \in \mathbf{Com}$ and $\sigma, \sigma' \in \Sigma$,

$$\langle c, \sigma \rangle \rightarrow_1^* \sigma' \text{ iff } \langle c, \sigma \rangle \rightarrow \sigma'.$$

The proof can proceed in three stages:

(A) Prove $\langle a, \sigma \rangle \rightarrow_1^* \langle n, \sigma \rangle$ iff $\langle a, \sigma \rangle \rightarrow n$;

(B) Prove $\langle b, \sigma \rangle \rightarrow_1^* \langle t, \sigma \rangle$ iff $\langle b, \sigma \rangle \rightarrow t$, using **(A)**;

(C) Prove $\langle c, \sigma \rangle \rightarrow_1^* \sigma'$ iff $\langle c, \sigma \rangle \rightarrow \sigma'$, using **(A)**and **(B)**.

Let's assume that **(A)** and **(B)** are proven. Then, there are two "directions" to prove:

"Only if" ($\Rightarrow$). We want to show

$$\langle c, \sigma \rangle \rightarrow_1^* \sigma' \Rightarrow \langle c, \sigma \rangle \rightarrow \sigma'.$$

Proof: For this stage, we will need the following lemma:

Lemma 1. If $\langle c, \sigma \rangle \rightarrow_1 \langle c, \sigma \rangle''$ and $\langle c', \sigma'' \rangle \rightarrow \sigma'$ then $\langle c, \sigma \rangle \rightarrow \sigma'$.

Intuitively, this means that we can prepend a single step evaluation to a natural evaluation, and get a derivable natural evaluation.

Proof: The proof of the lemma is by induction on the (natural) derivation of $\langle c', \sigma'' \rangle \rightarrow \sigma'$. We can break the problem down into cases, based on the value of c.

First, we note that c cannot be either **skip** or $X := n$, with $n \in \mathbf{N}$. In either of these cases, $\langle c, \sigma \rangle \rightarrow_1 \sigma'$, $\sigma' \not\rightarrow$, and the condition of the lemma is not satisfied. For the other cases:

[$c \equiv X := a$] (where a is not a numeral.) We have $\langle X := a, \sigma\rangle \rightarrow_1 \langle X := a', \sigma\rangle$ and $\langle X := a', \sigma\rangle \rightarrow \sigma'$. There is only one possible derivation with the conclusion $\langle X := a', \sigma\rangle \rightarrow \sigma'$, and that is of the form

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

for some $n \in \mathbf{N}$. By case **(A)**, $\langle a', \sigma\rangle \rightarrow n$ implies $\langle a', \sigma\rangle \rightarrow_1^* \langle n, \sigma\rangle$, so

$$\langle a, \sigma\rangle \rightarrow_1 \langle a', \sigma\rangle \rightarrow_1{}^* \langle n, \sigma\rangle,$$

so $\langle a, \sigma\rangle \rightarrow_1^* \langle n, \sigma\rangle$. Again, by **(A)**, this implies $\langle a, \sigma\rangle \rightarrow n$, which gives us the derivation

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

[$c \equiv c_0; c_1$] Here there are two subcases

[$\langle c_0, \sigma\rangle \rightarrow_1 \sigma''$] We start off with $\langle c_1, \sigma''\rangle \rightarrow \sigma'$, by assumption. We also know from the rules for $\rightarrow_1$ that c_0 must be either **skip** or $X := n$, for $n \in \mathbf{N}$. In the former case, $\sigma'' = \sigma$, and in the latter $\sigma'' = \sigma[n/X]$, giving us either

$$\frac{\langle \mathbf{skip}, \sigma\rangle \rightarrow \sigma, \quad \langle c_1, \sigma\rangle \rightarrow \sigma'}{\langle \mathbf{skip}; c_1, \sigma\rangle \rightarrow \sigma'}$$

or

$$\frac{\langle X := n, \sigma\rangle \rightarrow \sigma[n/X], \quad \langle c_1, \sigma[n/X]\rangle \rightarrow \sigma'}{\langle X := n; c_1, \sigma\rangle \rightarrow \sigma'}$$

[$\langle c_0, \sigma\rangle \rightarrow_1 \langle c_0', \sigma''\rangle$] We have $d \Vdash \langle c_0', c_1, \sigma''\rangle \rightarrow \sigma'$, for some d. Then d is of the form

$$\frac{\begin{array}{cc} \vdots & \vdots \\ \langle c_0', \sigma''\rangle \rightarrow \sigma''' & \langle c_1, \sigma'''\rangle \rightarrow \sigma' \end{array}}{\langle c_0', c_1, \sigma''\rangle \rightarrow \sigma'}$$

Now, we have derivations of $\langle c_0, \sigma\rangle \rightarrow_1 \langle c_0', \sigma''\rangle$ and $\langle c_0', \sigma''\rangle \rightarrow \sigma'''$. The derivation of the latter is a subderivation of d, so, by induction on derivations, we have $\langle c_0, \sigma\rangle \rightarrow \sigma'''$. But that gives us

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

[$c \equiv \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1$] Again there are two subcases:

[$b \in \{\mathbf{true}, \mathbf{false}\}$] Assume for now that $b = \mathbf{true}$. Then, we have

$$\langle c, \sigma \rangle \rightarrow_1 \langle c_0, \sigma \rangle \qquad \text{and} \qquad \langle c_0, \sigma \rangle \rightarrow \sigma',$$

And we get the derivation

$$\frac{\langle \mathbf{true}, \sigma \rangle \rightarrow \mathbf{true}, \quad \langle c_0, \sigma \rangle \rightarrow \sigma'}{\langle c, \sigma \rangle \rightarrow \sigma'}$$

The case for **false** is analagous.

[$b \notin \{\mathbf{true}, \mathbf{false}\}$] In this case, we have derivations

$$\frac{\langle b, \sigma \rangle \rightarrow_1 \langle b', \sigma \rangle}{\langle c, \sigma \rangle \rightarrow_1 \langle \mathbf{if}\, b'\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma \rangle}$$

and

$$\frac{\langle b', \sigma \rangle \rightarrow t, \quad \langle c_i, \sigma \rangle \rightarrow \sigma'}{\langle c, \sigma \rangle \rightarrow \sigma'}$$

for some $t \in \{\mathbf{true}, \mathbf{false}\}$ and $i \in \{0, 1\}$. By **(B)**, $\langle b', \sigma \rangle \rightarrow t$ implies $\langle b', \sigma \rangle \rightarrow_1^* \langle t, \sigma \rangle$, so

$$\langle b, \sigma \rangle \rightarrow_1 \langle b', \sigma \rangle \rightarrow_1{}^* \langle t, \sigma \rangle$$

which, again by **(B)**, implies $\langle b, \sigma \rangle \rightarrow t$, so we have

$$\frac{\langle b, \sigma \rangle \rightarrow t, \quad \langle c_i, \sigma \rangle \rightarrow \sigma'}{\langle c, \sigma \rangle \rightarrow \sigma'}$$

[$c \equiv \mathbf{while}\, b\, \mathbf{do}\, c'$] Here, there is only one one-step derivation:

$$\langle c, \sigma \rangle \rightarrow_1 \langle \mathbf{if}\, b\, \mathbf{then}\, c'; c\, \mathbf{else}\, \mathbf{skip}, \sigma \rangle$$

The two cases are

[$\langle b, \sigma \rangle \rightarrow \mathbf{false}$] Here, the derivation is easy, since the natural derivation must be

$$\frac{\langle b, \sigma \rangle \rightarrow \mathbf{false}}{\langle \mathbf{if}\, b\, \mathbf{then}\, c'; c\, \mathbf{else}\, \mathbf{skip}, \sigma \rangle \rightarrow \sigma}$$

meaning that $\sigma' = \sigma$, so the derivation we are looking for is just

$$\frac{\langle b, \sigma \rangle \rightarrow \mathbf{false}}{\langle c, \sigma \rangle \rightarrow \sigma}$$

[$\langle b, \sigma\rangle \rightarrow \mathbf{true}$] The derivation of $\langle \mathbf{if}\, b\, \mathbf{then}\, c'; c\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle \rightarrow \sigma'$ must be of the form

$$\dfrac{\begin{matrix}\vdots \\ \langle b, \sigma\rangle \rightarrow \mathbf{true}\end{matrix} \qquad \dfrac{\begin{matrix}\vdots \\ \langle c', \sigma\rangle \rightarrow \sigma''\end{matrix} \quad \begin{matrix}\vdots \\ \langle c, \sigma''\rangle \rightarrow \sigma'\end{matrix}}{\langle c'; c, \sigma\rangle \rightarrow \sigma'}}{\langle \mathbf{if}\, b\, \mathbf{then}\, c'; c\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle \rightarrow \sigma'}$$

Now, look at this tree. We have a derivation of

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

a derivation of

$$\langle c', \sigma\rangle \rightarrow \sigma'',$$

and a derivation of

$$\langle \mathbf{while}\, b\, \mathbf{do}\, c', \sigma''\rangle \rightarrow \sigma'.$$

But these are enough to give the following derivation:

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

as desired.

■

Now that we have our lemma, the actual proof is relatively simple. We're trying to prove that

$$\forall c, \sigma, \sigma'.\ \langle c, \sigma\rangle \rightarrow_1^* \sigma' \Rightarrow \langle c, \sigma\rangle \rightarrow \sigma'.$$

Now, we can prove this by induction on the length of $\rightarrow_1{}^*$:

Base case: Assume $\langle c, \sigma\rangle \rightarrow_1 \sigma'$. Then either $\langle \mathbf{skip}, \sigma\rangle \rightarrow_1 \sigma$, in which case we have $\langle \mathbf{skip}, \sigma\rangle \rightarrow \sigma$; or $\langle X := n, \sigma\rangle \rightarrow_1 \sigma[n/X]$, in which case we get $\langle X := n, \sigma\rangle \rightarrow \sigma[n/X]$.

Induction: Assume the implication holds for all $\rightarrow_1{}^*$ chains of length n. Then, if $\langle c, \sigma\rangle \rightarrow_1^* \sigma'$ by an evaluation of length $n+1$, we know from the definition of $\rightarrow_1{}^*$ that

$$\langle c, \sigma\rangle \rightarrow_1 \langle c', \sigma''\rangle \rightarrow_1{}^* \sigma'$$

for some c' and σ'', where $\langle c', \sigma''\rangle \rightarrow_1^* \sigma'$ by an evaluation of length n. But, since the implication we're proving is assumed to hold for evaluations of length n, we have

$$\langle c', \sigma''\rangle \rightarrow \sigma'.$$

But now, since $\langle c, \sigma\rangle \rightarrow_1 \langle c', \sigma''\rangle$, the lemma applies, which gives us

$$\langle c, \sigma''\rangle \rightarrow \sigma'.$$

■

"If" ($\Leftarrow$). We're now ready to prove the other direction:

$$\forall c, \sigma, \sigma'. \ \langle c, \sigma\rangle \rightarrow_1^* \sigma' \Leftarrow \langle c, \sigma\rangle \rightarrow \sigma'.$$

Proof: Assume we have some derivation $d \Vdash \langle c, \sigma\rangle \rightarrow \sigma'$. Our proof then proceeds, as above, by induction on the derivation d. We have the following cases:

$[c \equiv \mathbf{skip}]$ Then $\sigma' = \sigma$, and, by the definition of $\rightarrow_1$ we have $\langle \mathbf{skip}, \sigma\rangle \rightarrow_1 \sigma$.

$[c \equiv X := a]$ The derivation d must be of the form

$$\frac{\begin{array}{c}\vdots \\ \langle a, \sigma\rangle \rightarrow n\end{array}}{\langle X := a, \sigma\rangle \rightarrow \sigma[n/X]}$$

for some n. By **(A)**, $\langle a, \sigma\rangle \rightarrow n$ implies $\langle a, \sigma\rangle \rightarrow_1^* \langle n, \sigma\rangle$. It then requires a trivial induction on the length of $\rightarrow_1{}^*$ to show that

$$\langle X := a, \sigma\rangle \rightarrow_1^* \langle X := n, \sigma\rangle.$$

By the definition of $\rightarrow_1$, $\langle X := n, \sigma\rangle \rightarrow_1 \sigma[n/X]$, so we have

$$\langle X := a, \sigma\rangle \rightarrow_1^* \sigma[n/X]$$

$[c \equiv c_0; c_1]$ Then d must be of the form

$$\frac{\begin{array}{c}\vdots \\ \langle c_0, \sigma\rangle \rightarrow \sigma''\end{array} \quad \begin{array}{c}\vdots \\ \langle c_1, \sigma''\rangle \rightarrow \sigma'\end{array}}{\langle c_0; c_1, \sigma\rangle \rightarrow \sigma'}$$

Let d_0 and d_1 be the left and right subderivations above. Then, by induction, from d_0 we have $\langle c_0, \sigma\rangle \rightarrow_1^* \sigma''$, and from d_1 we have $\langle c_1, \sigma''\rangle \rightarrow_1^* \sigma'$. We now need the following lemma:

Lemma 2. For all $\sigma, \sigma', \sigma'' \in \Sigma$,

$$\langle c_0, \sigma\rangle \rightarrow_1^* \sigma'' \ \& \ \langle c_1, \sigma''\rangle \rightarrow_1^* \sigma' \Rightarrow \langle c_0; c_1, \sigma\rangle \rightarrow_1^* \sigma'$$

Proof: Again by induction on the definition of $\rightarrow_1{}^*$ as the transitive closure of $\rightarrow_1$ The basis step ($\rightarrow_1{}^*$ has length 0) holds trivially.

For the induction step. Suppose $\langle c_0, \sigma\rangle \rightarrow_1 \gamma$ and $\gamma \rightarrow_1{}^* \sigma''$. Moreover, suppose $\langle c_1, \sigma''\rangle \rightarrow_1 \sigma'$. There are now two cases:

- γ is of the form $\langle c_0', \sigma_0\rangle$. Then, by the induction hypothesis (applied to $\langle c_0', \sigma_0\rangle$), $\langle c_0'; c_1, \sigma_0\rangle \rightarrow_1^* \sigma'$. But since $\langle c_0, \sigma\rangle \rightarrow_1 \langle c_0', \sigma_0\rangle$ we have by the definition of $\rightarrow_1$ that $\langle c_0; c_1, \sigma\rangle \rightarrow_1 \langle c_0'; c_1, \sigma_0\rangle$. Combining things gives us $\langle c_0; c_1, \sigma\rangle \rightarrow_1^* \sigma'$.
- γ is of the form σ''. The result then holds trivially, since $\langle c_0, \sigma\rangle \rightarrow_1 \sigma''$ implies that $\langle c_0; c_1, \sigma\rangle \rightarrow_1 \langle c_1, \sigma''\rangle$. Combining this with our original presumption of $\langle c_1, \sigma''\rangle \rightarrow_1^* \sigma'$ gives us our desired result.

■

Now, by this lemma, since we have $\langle c_0, \sigma\rangle \rightarrow_1^* \sigma''$, and $\langle c_1, \sigma''\rangle \rightarrow_1^* \sigma'$, we also have

$$\langle c_0; c_1, \sigma\rangle \rightarrow_1^* \sigma'.$$

$[c \equiv \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1]$ The derivation d is of one of the forms

$$\frac{\begin{matrix}\vdots \\ \langle b, \sigma\rangle \rightarrow \mathbf{true}\end{matrix} \quad \begin{matrix}\vdots \\ \langle c_0, \sigma\rangle \rightarrow \sigma'\end{matrix}}{\langle c, \sigma\rangle \rightarrow \sigma'} \quad \text{or} \quad \frac{\begin{matrix}\vdots \\ \langle b, \sigma\rangle \rightarrow \mathbf{false}\end{matrix} \quad \begin{matrix}\vdots \\ \langle c_1, \sigma\rangle \rightarrow \sigma'\end{matrix}}{\langle c, \sigma\rangle \rightarrow \sigma'}$$

Let us call the right-hand subderivation of either derivation, d'.

Now, assume the **true** case. Then, by **(B)**, since we have a derivation for $\langle b, \sigma\rangle \rightarrow \mathbf{true}$, we also have one for $\langle b, \sigma\rangle \rightarrow_1^* \langle \mathbf{true}, \sigma\rangle$. Again, by induction on the length of $\rightarrow_1{}^*$, we have

$$\langle \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_1^* \langle \mathbf{if}\, \mathbf{true}\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle$$

Then, by the induction hypothesis (applied to d') we have $\langle c_0, \sigma\rangle \rightarrow_1^* \sigma'$, so we have

$$\begin{aligned}\langle \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_1{}^* &\langle \mathbf{if}\, \mathbf{true}\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \\ \rightarrow_1\ &\langle c_0, \sigma\rangle \\ \rightarrow_1{}^*\ &\sigma'\end{aligned}$$

The **false** case is analagous.

$[c \equiv \mathbf{while}\, b\, \mathbf{do}\, c']$ We will treat the simpler case first: If d is of the form

$$\frac{\begin{matrix}\vdots \\ \langle b, \sigma\rangle \rightarrow \mathbf{false}\end{matrix}}{\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma\rangle \rightarrow \sigma}$$

Then $\sigma' = \sigma$, and, by **(B)**, we have

$$\langle b, \sigma\rangle \rightarrow_1^* \langle b, \sigma\rangle$$

Once again, by induction on the length of $\rightarrow_1{}^*$, this gives us

$$\langle \mathbf{if}\, b\, \mathbf{then}(c'; c)\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle \rightarrow_1^* \langle \mathbf{if}\, \mathbf{false}\, \mathbf{then}(c'; c)\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle.$$

Thus we have

$$\begin{aligned}\langle \mathbf{while}\, b\, \mathbf{do}\, c', \sigma\rangle &\rightarrow_1 \langle \mathbf{if}\, b\, \mathbf{then}(c'; c)\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle \\ &\rightarrow_1{}^* \langle \mathbf{if}\, \mathbf{false}\, \mathbf{then}(c'; c)\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle \\ &\rightarrow_1 \langle \mathbf{skip}, \sigma\rangle \\ &\rightarrow_1 \sigma\end{aligned}$$

On the other hand, if $\langle b, \sigma\rangle \rightarrow \mathbf{true}$, then d must be of the form

$$\frac{\begin{array}{ccc}\vdots & \vdots & \vdots \\ \langle b, \sigma\rangle \rightarrow \mathbf{true} & \langle c', \sigma\rangle \rightarrow \sigma'' & \langle c, \sigma''\rangle \rightarrow \sigma'\end{array}}{\langle c, \sigma\rangle \rightarrow \sigma'}$$

As in the previous cases, from the subderivations we can conclude

$$\begin{aligned}&\langle b, \sigma\rangle \rightarrow_1{}^* \langle \mathbf{true}, \sigma\rangle \\ &\langle c', \sigma\rangle \rightarrow_1{}^* \sigma'' \\ &\langle c, \sigma''\rangle \rightarrow_1{}^* \sigma\end{aligned}$$

by **(B)** and by induction on derivations.

Then, by Lemma 2, we can conclude

$$\langle c'; c, \sigma\rangle \rightarrow_1^* \sigma'$$

Putting the pieces together with the definition of $\rightarrow_1$, this gives us

$$\begin{aligned}\langle c, \sigma\rangle &\rightarrow_1 \langle \mathbf{if}\, b\, \mathbf{then}(c'; c)\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle \\ &\rightarrow_1{}^* \langle \mathbf{if}\, \mathbf{true}\, \mathbf{then}(c'; c)\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle \\ &\rightarrow_1 \langle (c'; c), \sigma\rangle \\ &\rightarrow_1{}^* \sigma'\end{aligned}$$

thus proving the final case in the theorem.

■

Problem Set 3

Due: 5 October 1992.

Reading assignment. Winskel Chapters 3–4, except §3.3.

Problem 1. Winskel's text §2.5 proves that a simple "unwinding" of a while-loop preserves natural semantics, namely,

$$\textbf{while}\, b\, \textbf{do}\, c \sim \textbf{if}\, b\, \textbf{then}(c; \textbf{while}\, b\, \textbf{do}\, c)\, \textbf{else}\, \textbf{skip}\,.$$

Prove that this equivalence also holds in $\mathbf{IMP}_r$, the extension of **IMP** with a $c\,\textbf{resultis}\,a$ expression construct (*cf.*, Problem Set 2). Indicate, without repeating them, those portions of the proof for **IMP** which apply unchanged to $\mathbf{IMP}_r$.

Problem 2. The operational behavior of **IMP** expressions and commands depends only on the locations occurring in them. In this problem we give a precise definition of this dependence and verify it. (It may be helpful to look at Winskel §3.5 and Prop. 4.7 for related ideas.)

For $L \subseteq \mathbf{Loc}$ define a relation $=_L$ between states by

$$\sigma_1 =_L \sigma_2 \text{ iff } \sigma_1(X) = \sigma_2(X) \text{ for all } X \in L.$$

Define a relation, $\sim_L$, between commands by

$$\begin{aligned} c_1 \sim_L c_2 \text{ iff } (\sigma_1 =_L \sigma_2 \Rightarrow \\ &((\langle c_1, \sigma_1\rangle \to \sigma_1' \text{ for some } \sigma_1' \text{ iff } \langle c_2, \sigma_2\rangle \to \sigma_2' \text{ for some } \sigma_2') \;\&\\ &(((\langle c_1, \sigma_1\rangle \to \sigma_1' \;\&\; \langle c_2, \sigma_2\rangle \to \sigma_2') \Rightarrow \sigma_1' =_L \sigma_2')).\end{aligned}$$

(a) Give a structural inductive definition of $\mathrm{loc}(a)$, the set of locations occurring in $a \in \mathbf{Aexp}$. Do likewise for $\mathrm{loc}(b)$ and $\mathrm{loc}(c)$ (*cf.* Winskel 3.5).

(b) Prove that $c \sim_{\mathrm{loc}(c)} c$ for all $c \in \mathbf{Com}$.

(c) We referred to the natural semantics and the one-step semantics of **IMP** as "operational," but since **Loc** is an infinite set, and therefore so is each state, these "operational" semantics involve copying and updating infinite objects in derivations. Briefly explain how the result of part 2(b) leads to a more truly "operational" version of natural and one-step semantics using finite portions of states. Then say precisely how the original versions of operational semantics can be retrieved from the versions using finite portions of states.

Problem 3. The *natural evaluation control states* of a command c are defined to be the commands c' such that $\langle c', \sigma_3 \rangle \rightarrow \sigma_4$ occurs in a derivation of $\langle c, \sigma_1 \rangle \rightarrow \sigma_2$ for some states $\sigma_1, \sigma_2, \sigma_3, \sigma_4$. Likewise for the *one-step control states*.

(a) List the natural evaluation and the one-step control states of the command Intsqrt from Problem Set 1.

(b) Prove that $\mathbf{IMP}_r$, so *a fortiori* also **IMP**, has *finite-state control*, *i.e.*, for any $c \in \mathbf{Com}_r$, there are only finitely many natural evaluation control states of c.

(c) Show the same result for the one-step control states.

(d) Briefly discuss how this observation could be used to improve the efficiency of an interpreter for $\mathbf{IMP}_r$ based on the natural or one-step operational semantics.

One-step vs. Natural Evaluation

This is a revised, more polished version of Handout 10, "Equivalence of $\rightarrow_1$ and $\rightarrow$." We prove

Theorem. For all $c \in \mathbf{Com}$ and $\sigma, \sigma' \in \Sigma$,

$$\langle c, \sigma \rangle \rightarrow_1^* \sigma' \text{ iff } \langle c, \sigma \rangle \rightarrow \sigma'.$$

The proof proceeds in three stages:

(A) Prove $\langle a, \sigma \rangle \rightarrow_1^* \langle n, \sigma \rangle$ iff $\langle a, \sigma \rangle \rightarrow n$;

(B) Prove $\langle b, \sigma \rangle \rightarrow_1^* \langle t, \sigma \rangle$ iff $\langle b, \sigma \rangle \rightarrow t$, using **(A)**;

(C) Prove $\langle c, \sigma \rangle \rightarrow_1^* \sigma'$ iff $\langle c, \sigma \rangle \rightarrow \sigma'$, using **(A)** and **(B)**.

The proofs of all three parts are similar, except that where structural induction serves to prove a version of Lemma 1 below for expressions, induction on derivations is needed to prove it for commands. We will henceforth assume that **(A)** and **(B)** are proven, and present **(C)** only.

We first prove **(C)** from left to right, namely

$$\langle c, \sigma \rangle \rightarrow_1^* \sigma' \quad \Rightarrow \quad \langle c, \sigma \rangle \rightarrow \sigma'.$$

The proof is based on

Lemma 1. If $\langle c, \sigma \rangle \rightarrow_1 \langle c', \sigma'' \rangle$ and $\langle c', \sigma'' \rangle \rightarrow \sigma'$, then $\langle c, \sigma \rangle \rightarrow \sigma'$.

Assuming this lemma, a simple induction on the definition of the transitive closure, $\rightarrow_1{}^*$, of the one-step evaluation relation, $\rightarrow_1$, completes the proof of **(C)** $\Rightarrow$:

Base case: Suppose $\langle c, \sigma \rangle \rightarrow_1^* \sigma'$ because $\langle c, \sigma \rangle \rightarrow_1 \sigma'$. Then either c is **skip** and σ' is σ, or c is $X := n$ and σ' is $\sigma[n/X]$. In either case, $\langle c, \sigma \rangle \rightarrow \sigma'$ follows immediately by a natural evaluation axiom.

Induction: Suppose $\langle c, \sigma \rangle \rightarrow_1^* \sigma'$ because $\langle c, \sigma \rangle \rightarrow_1 \langle c', \sigma'' \rangle$ for some c', σ'' such that $\langle c', \sigma'' \rangle \rightarrow_1^* \sigma'$. Then by induction, we have that $\langle c', \sigma'' \rangle \rightarrow \sigma'$. But now Lemma 1 immediately implies $\langle c, \sigma \rangle \rightarrow \sigma'$, as required.

So we need only prove Lemma 1, which we do by induction on the derivation of $\langle c', \sigma'' \rangle \to \sigma'$. The induction breaks into cases according to the form of c.

First, we note that c cannot be either **skip** or $X := n$ where $n \in \mathbf{N}$, because in both these cases, $\langle c, \sigma \rangle$ moves in one step to a state rather than a command configuration $\langle c', \sigma'' \rangle$, and the condition of the lemma is not satisfied. The other cases are:

•[c is of the form $X := a$ where $a \notin \mathbf{Num}$] There is a unique one-step rule by which $\langle X := a, \sigma \rangle \to_1 \langle c', \sigma'' \rangle$ could be derived, and by this rule c' must be $X := a'$ where $\langle a, \sigma \rangle \to_1 \langle a', \sigma \rangle$, and σ'' must be σ. We are given that $\langle c', \sigma'' \rangle \to \sigma'$, and there is only one possible derivation with this conclusion:

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

We conclude that

$$\sigma' \text{ must be } \sigma[n/X] \text{ for some } n \text{ such that } \langle a', \sigma \rangle \to n.$$

But $\langle a', \sigma \rangle \to n$ implies $\langle a', \sigma \rangle \to_1^* \langle n, \sigma \rangle$, by **(A)**. Thus, we have

$$\langle a, \sigma \rangle \to_1 \langle a', \sigma \rangle \to_1{}^* \langle n, \sigma \rangle.$$

By **(A)** again, we conclude that $\langle a, \sigma \rangle \to n$. Now the assignment rule for natural evaluation gives the derivation

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

So indeed, $\langle c, \sigma \rangle \to \sigma'$.

•[c is $(c_0; c_1)$] There are two ways that the given condition $\langle c, \sigma \rangle \to_1 \langle c', \sigma'' \rangle$ could be derived, namely,

[$\langle c_0, \sigma \rangle \to_1 \sigma''$ and c' is c_1] Since $\langle c_0, \sigma \rangle \to_1 \sigma''$, the rules for $\to_1$ imply that either c_0 must be **skip** and $\sigma'' = \sigma$, or else c_0 is $X := n$ and $\sigma'' = \sigma[n/X]$. So

$$\frac{\langle \mathbf{skip}, \sigma \rangle \to \sigma, \quad \langle c_1, \sigma \rangle \to \sigma'}{\langle (\mathbf{skip}; c_1), \sigma \rangle \to \sigma'}$$

or

$$\frac{\langle X := n, \sigma \rangle \to \sigma[n/X], \quad \langle c_1, \sigma[n/X] \rangle \to \sigma'}{\langle (X := n; c_1), \sigma \rangle \to \sigma'}$$

is a derivation of $\langle c, \sigma \rangle \to \sigma'$, as required.

[$\langle c_0, \sigma\rangle \rightarrow_1 \langle c_0', \sigma''\rangle$ and c' is $(c_0'; c_1)$] We have

$$d \Vdash \langle (c_0'; c_1), \sigma''\rangle \rightarrow \sigma',$$

for some d which must be of the form

$$\frac{\begin{matrix}\vdots \\ \langle c_0', \sigma''\rangle \rightarrow \sigma'''\end{matrix} \quad \begin{matrix}\vdots \\ \langle c_1, \sigma'''\rangle \rightarrow \sigma'\end{matrix}}{\langle (c_0'; c_1), \sigma''\rangle \rightarrow \sigma'}$$

Thus we have a derivation of $\langle c_0', \sigma''\rangle \rightarrow \sigma'''$. This derivation is a subderivation of d, and we are given that $\langle c_0, \sigma\rangle \rightarrow_1 \langle c_0', \sigma''\rangle$, so by induction we deduce

$$\langle c_0, \sigma\rangle \rightarrow \sigma'''.$$

But that gives

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

•[c is $\textbf{if}\, b\, \textbf{then}\, c_0\, \textbf{else}\, c_1$] There are two subcases:

[$b \in \mathbf{T}$ and c' is c_i] Then we have

$$\langle c, \sigma\rangle \rightarrow_1 \langle c_i, \sigma\rangle \qquad \text{and} \qquad \langle c_i, \sigma\rangle \rightarrow \sigma',$$

and we get the derivation

$$\frac{\langle b, \sigma\rangle \rightarrow b, \quad \langle c_i, \sigma\rangle \rightarrow \sigma'}{\langle c, \sigma\rangle \rightarrow \sigma'}$$

[$b \notin \mathbf{T}$] Then we have derivations

$$\frac{\langle b, \sigma\rangle \rightarrow_1 \langle b', \sigma\rangle}{\langle c, \sigma\rangle \rightarrow_1 \langle \textbf{if}\, b'\, \textbf{then}\, c_0\, \textbf{else}\, c_1, \sigma\rangle}$$

and

$$\frac{\langle b', \sigma\rangle \rightarrow t, \quad \langle c_i, \sigma\rangle \rightarrow \sigma'}{\langle \textbf{if}\, b'\, \textbf{then}\, c_0\, \textbf{else}\, c_1, \sigma\rangle \rightarrow \sigma'}$$

for some $t \in \mathbf{T}$ and $i \in \{0, 1\}$. By (**B**), $\langle b', \sigma\rangle \rightarrow t$ implies $\langle b', \sigma\rangle \rightarrow_1^* \langle t, \sigma\rangle$, so

$$\langle b, \sigma\rangle \rightarrow_1 \langle b', \sigma\rangle \rightarrow_1{}^* \langle t, \sigma\rangle$$

which, again by **(B)**, implies $\langle b, \sigma\rangle \rightarrow t$. So we have

$$\frac{\langle b, \sigma\rangle \rightarrow t, \quad \langle c_i, \sigma\rangle \rightarrow \sigma'}{\langle c, \sigma\rangle \rightarrow \sigma'}$$

•[c is $\mathbf{while}\, b\, \mathbf{do}\, c''$] There is a unique one-step derivation

$$\langle c, \sigma\rangle \rightarrow_1 \langle \mathbf{if}\, b\, \mathbf{then}(c''; c)\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle,$$

so c' is $\mathbf{if}\, b\, \mathbf{then}(c''; c)\, \mathbf{else}\, \mathbf{skip}$ and σ'' is σ. But we know (Winskel Prop. 2.8) that $c \sim c'$, and are given that $\langle c', \sigma\rangle \rightarrow \sigma'$, so we conclude $\langle c, \sigma\rangle \rightarrow \sigma'$.

This completes the proof of Lemma 1. ■

It remains to prove the converse implication:

$$\langle c, \sigma\rangle \rightarrow \sigma' \quad \Rightarrow \quad \langle c, \sigma\rangle \rightarrow_1^* \sigma'.$$

The proof requires some simple facts about $\rightarrow_1{}^*$.

Lemma 2.

1. $\langle a, \sigma\rangle \rightarrow_1^* \langle a', \sigma'\rangle \quad \Rightarrow \quad \langle X := a, \sigma\rangle \rightarrow_1^* \langle X := a', \sigma'\rangle,$
2. $\langle b, \sigma\rangle \rightarrow_1^* \langle b', \sigma'\rangle \Rightarrow \langle \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_1^* \langle \mathbf{if}\, b'\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma'\rangle,$
3. $[\langle c_0, \sigma\rangle \rightarrow_1^* \sigma'' \;\&\; \langle c_1, \sigma''\rangle \rightarrow_1^* \sigma'] \quad \Rightarrow \quad \langle (c_0; c_1), \sigma\rangle \rightarrow_1^* \sigma'.$

Proof: All three parts follow by straightforward induction on the definition of the transitive closure, $\rightarrow_1{}^*$, of $\rightarrow_1$. We omit 1 and 2, and do only the slightly more complicated part 3. In particular, we prove 3 by induction on the derivation of $\langle c_0, \sigma\rangle \rightarrow_1^* \sigma''$.

Such a derivation must consist of $\langle c_0, \sigma\rangle \rightarrow_1 \gamma$ and a derivation of $\gamma \rightarrow_1{}^* \sigma''$ for some state or configuration γ. So there are two cases:

[γ is a state σ''] The result then holds easily, with $\langle c_0, \sigma\rangle \rightarrow_1 \sigma''$ implying that $\langle (c_0; c_1), \sigma\rangle \rightarrow_1 \langle c_1, \sigma''\rangle$. Combining this with our original presumption of $\langle c_1, \sigma''\rangle \rightarrow_1^* \sigma'$ gives the desired result.

[γ is a configuration $\langle c_0', \sigma_0\rangle$] Then, we can apply the induction hypothesis to the evaluation $\langle c_0', \sigma_0\rangle \rightarrow_1^* \sigma''$, and we conclude that $\langle (c_0'; c_1), \sigma_0\rangle \rightarrow_1^* \sigma'$. But since $\langle c_0, \sigma\rangle \rightarrow_1 \langle c_0', \sigma_0\rangle$, we have by the definition of $\rightarrow_1$ that

$$\langle (c_0; c_1), \sigma\rangle \rightarrow_1 \langle (c_0'; c_1), \sigma_0\rangle \rightarrow_1^* \sigma'.$$

So $\langle (c_0; c_1), \sigma\rangle \rightarrow_1^* \sigma'$ as required.

■

Proof: We now prove the converse implication of **(C)** by induction on the derivation d of $\langle c, \sigma\rangle \to \sigma'$. The induction breaks into cases according to the form of c.

•[c is **skip**] By the definition of $\to_1$ we have $\langle \mathbf{skip}, \sigma\rangle \to_1 \sigma$, so $\sigma' = \sigma$, and hence $\langle \mathbf{skip}, \sigma\rangle \to_1^* \sigma'$, as required.

•[c is of the form $X := a$] The derivation of $\langle c, \sigma\rangle \to \sigma'$ must be of the form

$$\frac{\begin{array}{c}\vdots\\ \langle a, \sigma\rangle \to n\end{array}}{\langle X := a, \sigma\rangle \to \sigma[n/X]}$$

so σ' is $\sigma[n/X]$. By **(A)**, $\langle a, \sigma\rangle \to n$ implies

$$\langle a, \sigma\rangle \to_1^* \langle n, \sigma\rangle.$$

But then by Lemma 2.1

$$\langle X := a, \sigma\rangle \to_1^* \langle X := n, \sigma\rangle.$$

Now $\langle X := n, \sigma\rangle \to_1 \sigma[n/X]$ is an $\to_1$ rule, so we have

$$\langle X := a, \sigma\rangle \to_1^* \sigma[n/X] = \sigma'.$$

•[c is $(c_0; c_1)$] Then d must be of the form

$$\frac{\begin{array}{c}\vdots\\ \langle c_0, \sigma\rangle \to \sigma''\end{array} \quad \begin{array}{c}\vdots\\ \langle c_1, \sigma''\rangle \to \sigma'\end{array}}{\langle (c_0; c_1), \sigma\rangle \to \sigma'}$$

Let d_0 and d_1 be the left and right subderivations above. Then, by induction we have $\langle c_0, \sigma\rangle \to_1^* \sigma''$ and $\langle c_1, \sigma''\rangle \to_1^* \sigma'$. Now by Lemma 2.3 we conclude

$$\langle (c_0; c_1), \sigma\rangle \to_1^* \sigma'.$$

•[c is $\mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1$] The derivation d is of the form

$$\frac{\begin{array}{c}\vdots\\ \langle b, \sigma\rangle \to t\end{array} \quad \begin{array}{c}\vdots\\ \langle c_i, \sigma\rangle \to \sigma'\end{array}}{\langle c, \sigma\rangle \to \sigma'}$$

for $t \in \mathbf{T}$. Let us call the right-hand subderivation d'. Since $\langle b, \sigma\rangle \to t$, we know by **(B)** that $\langle b, \sigma\rangle \to_1^* \langle t, \sigma\rangle$, and hence by Lemma 2.2

$$\langle \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \to_1^* \langle \mathbf{if}\, t\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle.$$

Then, by the induction hypothesis (applied to d') we have $\langle c_i, \sigma\rangle \to_1^* \sigma'$, so

$$\begin{aligned}\langle \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle &\to_1{}^* \langle \mathbf{if}\, t\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \\ &\to_1 \langle c_i, \sigma\rangle \\ &\to_1{}^* \sigma'\end{aligned}$$

•[c is $\mathbf{while}\, b\, \mathbf{do}\, c'$] The simpler subcase is when d is of the form

$$\frac{\begin{array}{c}\vdots\\ \langle b, \sigma\rangle \to \mathbf{false}\end{array}}{\langle \mathbf{while}\, b\, \mathbf{do}\, c', \sigma\rangle \to \sigma}$$

Then $\sigma' = \sigma$, and, by **(B)**, we have

$$\langle b, \sigma\rangle \to_1^* \langle \mathbf{false}, \sigma\rangle,$$

so by Lemma 2.2

$$\langle \mathbf{if}\, b\, \mathbf{then}(c'; c)\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle \to_1^* \langle \mathbf{if}\, \mathbf{false}\, \mathbf{then}(c'; c)\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle.$$

Thus,

$$\begin{aligned}\langle \mathbf{while}\, b\, \mathbf{do}\, c', \sigma\rangle &\to_1 \langle \mathbf{if}\, b\, \mathbf{then}(c'; c)\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle \\ &\to_1{}^* \langle \mathbf{if}\, \mathbf{false}\, \mathbf{then}(c'; c)\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle \\ &\to_1 \langle \mathbf{skip}, \sigma\rangle \\ &\to_1 \sigma.\end{aligned}$$

The other subcase is when d is of the form

$$\frac{\begin{array}{c}\vdots\\ \langle b, \sigma\rangle \to \mathbf{true}\end{array} \quad \begin{array}{c}\vdots\\ \langle c', \sigma\rangle \to \sigma''\end{array} \quad \begin{array}{c}\vdots\\ \langle c, \sigma''\rangle \to \sigma'\end{array}}{\langle c, \sigma\rangle \to \sigma'}$$

As in the previous cases, from the subderivations we can conclude

$$\begin{aligned}\langle b, \sigma\rangle &\to_1{}^* \langle \mathbf{true}, \sigma\rangle \\ \langle c', \sigma\rangle &\to_1{}^* \sigma'' \\ \langle c, \sigma''\rangle &\to_1{}^* \sigma\end{aligned}$$

by **(B)** and induction hypothesis. Then, by Lemma 2.3, we can conclude

$$\langle (c'; c), \sigma \rangle \rightarrow_1^* \sigma'.$$

Putting the pieces together with the definition of $\rightarrow_1$, this gives us

$$\begin{aligned}
\langle c, \sigma \rangle &\rightarrow_1 \langle \mathbf{if}\, b\, \mathbf{then} (c'; c)\, \mathbf{else}\, \mathbf{skip}, \sigma \rangle \\
&\rightarrow_1{}^* \langle \mathbf{if}\, \mathbf{true}\, \mathbf{then} (c'; c)\, \mathbf{else}\, \mathbf{skip}, \sigma \rangle \\
&\rightarrow_1 \langle (c'; c), \sigma \rangle \\
&\rightarrow_1{}^* \sigma',
\end{aligned}$$

thus proving the final case in the theorem.

■

Outline: Lectures 1–19

1. (Fri, 9/11) Administrivia. Sample **IMP** while-program, Euclid, p.34; brief sketch of partial correctness and termination.

2. (Mon, 9/14) Syntax of **IMP** and "natural" evaluation semantics of for **Aexp**. Derivation tree for $\langle (M+N) \times N, \sigma[10/N][6/M]\rangle \rightarrow 160$.

3. (Wed, 9/16) Natural eval rules for **Com**. Derivation tree for $\langle \text{Euclid}, \sigma[10/\text{N}][6/\text{M}]\rangle \rightarrow \sigma[2/M][2/N]$. Uniqueness of derivation tree for each configuration; exists for **Aexp**, **Bexp**, and *while-free* **Com**, but $\langle \mathbf{while}\,\mathbf{true}\,\mathbf{do}\, c, \sigma\rangle \not\rightarrow$ for all c, σ. No proofs.

4. (Fri, 9/18) One-step rules. Example $\langle \mathit{Euclid}, \sigma[10/N][6/M]\rangle \rightarrow_1^* \sigma[2/M][2/N]$. Remark: $\rightarrow_1$ is total, functional, computable relation. Inductive def of transitive closure. Statement of "equivalence" of one-step and natural rules: $\gamma {\rightarrow_1}^* \delta$ iff $\gamma \rightarrow \delta$ for all configurations γ and values $\delta \in \mathbf{N} \cup \mathbf{T} \cup \Sigma$.

5. (Mon, 9/21) Proof of equiv of natural and one-step semantics.

6. (Wed, 9/23) Proof by induction on deriv. of functionality of command evaluation (Winsk, 3.11). Proof by minimum principle that $\langle \mathbf{while}\,\mathbf{true}\,\mathbf{do}\, c, \sigma\rangle \not\rightarrow$ (Winsk. 3.12).

7. (Fri, 9/25) Formal def of derivations, and induction on them (§3.4). Set R of rule instances determines a monotone, continuous, operator $\hat{R}$ on sets (§4.4) with derivable elements $= \mathit{fix}(\hat{R})$.

8. (Mon, 9/28) (Winskel §5.4) Def and examples of cpo's, monotone and continuous functions. Contrast with usual (epsilon-delta) continuity.

9. (Wed, 9/30) Proof that $\hat{R}$ is continuous. Monotone functions are a cpo under pointwise order. Least fixed points of continuous functions on cpo's. Motivation for fixed points as explanation of recursion: While-loops as fixed points of command mappings.

10. (Fri, 10/2) QUIZ 1, IN CLASS, on lectures 1–8

11. (Mon, 10/5) Function def by structural induction, eg, length and depth of a derivation, def of loc_L (§3.5) and statement w/o proof: c only effects $\mathrm{loc}_L(c)$ (Winskel 4.7). Discussion of well-formed and non-wellformed recursive function def's, eg, $e(x) = e(x+1)$, $f(x) = f(x+1)+1$, for g, h functions on ω^+: $g(x+y) = g(x) + g(y)$, $h(x+y) = h(x) + 2h(y)$.

12. (Wed, 10/7) Meanings of expressions will be total functions from states to **Num** or **T**. Command meanings will be partial functions $\in \Sigma \rightharpoonup \Sigma$. Statement of equivalence of denotational and natural semantics. Then define denotational semantics by structural induction assuming $\Gamma_{\mathbf{while}}$ (Winskel p.62) is continuous.

13. (Fri, 10/9) Prove that $\Gamma_{\mathbf{while}}$ (Winskel p.62) is continuous: product cpo's and continuity of functions of several arguments. Continuity of command operators, and closure of continuous operators under composition.

    (Mon, 10/12) COLUMBUS DAY

14. (Wed, 10/14) Proof of equivalence of natural and denotational semantics (Winskel Thm. 5.7).

15. (Fri, 10/16) First-order arithmetic: **Assn**'s and their meaning.

16. (Mon, 10/19) Valid assertions and sound inference rules.

17. (Wed, 10/21) Hoare logic and examples.

18. (Fri, 10/23) Soundness of Hoare rules.

19. (Mon, 10/26) Expressiveness of **Assn**'s and relative completeness of Hoare logic.
    EVENING QUIZ 2, Mon, 10/26, on lectures 9, 11–18
20. (Wed, 10/28)
21. (Fri, 10/30)
22. (Mon, 11/2)
23. (Wed, 11/4)
24. (Fri, 11/6)
25. (Mon, 11/9)
    (Wed, 11/11) VETERAN'S DAY
26. (Fri, 11/13)
27. (Mon, 11/16) QUIZ 3, IN CLASS, on lectures 19–25
28. (Wed, 11/18)
29. (Fri, 11/20) DROP DATE
30. (Mon, 11/23)
31. (Wed, 11/25)
    (Fri, 11/27) THANKSGIVING HOLIDAY
32. (Mon, 11/30)
33. (Wed, 12/2)
34. (Fri, 12/4)
35. (Mon, 12/7)
36. (Wed, 12/9)
    (Exam Period) QUIZ 4 2 hours, on lectures 26, 28–36

One-Step Rewriting Rules for $\mathbf{IMP}_r$

Throughout this document, we will use **op** to range over syntactic operator symbols, and *op* to range over corresponding arithmetic or Boolean operations. The only completely new rules are for $\mathbf{Aexp}_r$'s of the form $c\,\mathbf{resultis}\,a$, but most have changed to account for the presence of side effects in $\mathbf{Aexp}_r$'s.

1 Rules for Arithmetic Expressions, $\mathbf{Aexp}_r$

$$\langle X, \sigma\rangle \rightarrow_{r,1} \langle \sigma(X), \sigma\rangle$$

$$\frac{\langle c, \sigma\rangle \rightarrow_{r,1} \langle c', \sigma'\rangle}{\langle c\,\mathbf{resultis}\,a, \sigma\rangle \rightarrow_{r,1} \langle c'\,\mathbf{resultis}\,a, \sigma'\rangle}$$

$$\frac{\langle c, \sigma\rangle \rightarrow_{r,1} \sigma'}{\langle c\,\mathbf{resultis}\,a, \sigma\rangle \rightarrow_{r,1} \langle a, \sigma'\rangle}$$

$$\frac{\langle a_0, \sigma\rangle \rightarrow_{r,1} \langle a_0', \sigma'\rangle}{\langle a_0\ \mathbf{op}\ a_1, \sigma\rangle \rightarrow_{r,1} \langle a_0'\ \mathbf{op}\ a_1, \sigma'\rangle}$$

$$\frac{\langle a_1, \sigma\rangle \rightarrow_{r,1} \langle a_1', \sigma'\rangle}{\langle n\ \mathbf{op}\ a_1, \sigma\rangle \rightarrow_{r,1} \langle n\ \mathbf{op}\ a_1', \sigma'\rangle}$$

$$\langle n\ \mathbf{op}\ m, \sigma\rangle \rightarrow_{r,1} \langle n\ op\ m, \sigma\rangle$$

op	*op*
$+$	the sum function
$-$	the subtraction function
$\times$	the multiplication function

2 Rules for Boolean Expressions, $\mathbf{Bexp}_r$

$$\frac{\langle a_0, \sigma\rangle \rightarrow_{r,1} \langle a_0', \sigma'\rangle}{\langle a_0\ \mathbf{op}\ a_1, \sigma\rangle \rightarrow_{r,1} \langle a_0'\ \mathbf{op}\ a_1, \sigma'\rangle}$$

$$\frac{\langle a_1, \sigma\rangle \rightarrow_{r,1} \langle a_1', \sigma'\rangle}{\langle n\ \mathbf{op}\ a_1, \sigma\rangle \rightarrow_{r,1} \langle n\ \mathbf{op}\ a_1', \sigma'\rangle}$$

$$\langle n\ \mathbf{op}\ m, \sigma\rangle \rightarrow_{r,1} \langle n\ op\ m, \sigma\rangle$$

op	*op*
$=$	the equality predicate
$\leq$	the less than or equal to predicate

We next have the rules for Boolean negation:

$$\frac{\langle b, \sigma\rangle \rightarrow_{r,1} \langle b', \sigma'\rangle}{\langle \neg b, \sigma\rangle \rightarrow_{r,1} \langle \neg b', \sigma'\rangle}$$

$$\langle \neg\, \mathbf{true}, \sigma\rangle \rightarrow_{r,1} \langle \mathbf{false}, \sigma\rangle$$

$$\langle \neg\, \mathbf{false}, \sigma\rangle \rightarrow_{r,1} \langle \mathbf{true}, \sigma\rangle$$

Finally we have the rules for binary Boolean operators. We let $t, t_0, t_1, \ldots$ range over the set $\mathbf{T} = \{\mathbf{true}, \mathbf{false}\}$.

$$\frac{\langle b_0, \sigma\rangle \rightarrow_{r,1} \langle b_0', \sigma'\rangle}{\langle b_0 \ \mathbf{op}\ b_1, \sigma\rangle \rightarrow_{r,1} \langle b_0' \ \mathbf{op}\ b_1, \sigma'\rangle}$$

$$\frac{\langle b_1, \sigma\rangle \rightarrow_{r,1} \langle b_1', \sigma'\rangle}{\langle t_0 \ \mathbf{op}\ b_1, \sigma\rangle \rightarrow_{r,1} \langle t_0 \ \mathbf{op}\ b_1', \sigma'\rangle}$$

$$\langle t_0 \ \mathbf{op}\ t_1, \sigma\rangle \rightarrow_{r,1} \langle t_0 \ op\ t_1, \sigma\rangle$$

op	*op*
$\wedge$	the conjunction operation (Boolean AND)
$\vee$	the disjunction operation (Boolean OR)

3 Rules for Commands, Com$_r$

Atomic Commands:

$$\langle \mathbf{skip}, \sigma\rangle \rightarrow_{r,1} \sigma$$

$$\frac{\langle a, \sigma\rangle \rightarrow_{r,1} \langle a', \sigma'\rangle}{\langle X := a, \sigma\rangle \rightarrow_{r,1} \langle X := a', \sigma'\rangle}$$

$$\langle X := n, \sigma\rangle \rightarrow_{r,1} \sigma[n/X]$$

Sequencing:

$$\frac{\langle c_0, \sigma\rangle \rightarrow_{r,1} \langle c_0', \sigma'\rangle}{\langle (c_0; c_1), \sigma\rangle \rightarrow_{r,1} \langle (c_0'; c_1), \sigma'\rangle}$$

$$\frac{\langle c_0, \sigma\rangle \rightarrow_{r,1} \sigma'}{\langle (c_0; c_1), \sigma\rangle \rightarrow_{r,1} \langle c_1, \sigma'\rangle}$$

Conditionals:

$$\frac{\langle b, \sigma\rangle \rightarrow_{r,1} \langle b', \sigma'\rangle}{\langle \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_{r,1} \langle \mathbf{if}\, b'\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma'\rangle}$$

$$\langle \mathbf{if\ true\ then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_{r,1} \langle c_0, \sigma\rangle$$

$$\langle \mathbf{if\ false\ then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_{r,1} \langle c_1, \sigma\rangle$$

While-loops:

$$\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma\rangle \rightarrow_{r,1} \langle \mathbf{if}\, b\, \mathbf{then} (c; \mathbf{while}\, b\, \mathbf{do}\, c)\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle$$

Last Year's Quiz #1, and Solutions

Instructions. For your reference, there is an appendix listing the definitions of the "evaluates to" relation $\rightarrow$, and the one-step rewriting relation $\rightarrow_1$ on configurations of the language **IMP**.

For Problems 1 and 2, let w be the **IMP** command

$$\mathbf{while}\,45 \leq X\,\mathbf{do}\,X := X - 3;\ Y := X - 1;\ X := Y - 1$$

and let σ be a state such that $\sigma(X) = 1000$ and $\sigma(Y) = 2000$.

Problem 1 [10 points]. According to the inductive definition of evaluation, the assertion

$$\langle w, \sigma\rangle \rightarrow \sigma[40/X][41/Y]$$

has a unique derivation. How many instances of the sequencing rule scheme (seq $\rightarrow$) given below appear in this derivation? $\boxed{384}$

$$\frac{\langle c_0, \sigma\rangle \rightarrow \sigma'', \quad \langle c_1, \sigma''\rangle \rightarrow \sigma'}{\langle (c_0; c_1), \sigma\rangle \rightarrow \sigma'} \qquad (\text{seq} \rightarrow)$$

Note: The quiz did not ask for any explanation. One will be asked for on problem set 4.

Problem 2 [15 points]. By definition, $\langle w, \sigma\rangle \rightarrow_1^* \sigma[40/X][41/Y]$ because there is a (unique) sequence of the form:

$$\langle w, \sigma\rangle \rightarrow_1 \langle c_1, \sigma_1\rangle \rightarrow_1 \langle c_2, \sigma_2\rangle \rightarrow_1 \cdots \rightarrow_1 \langle c_n, \sigma_n\rangle \rightarrow_1 \sigma[40/X][41/Y]$$

where n happens to be 2500.

Notice that σ_1 must equal σ, and c_1 must be

$$\mathbf{if}\,45 \leq X\,\mathbf{then}\,(X := X - 3;\ Y := X - 1;\ X := Y - 1;\ w)\,\mathbf{else}\,\mathbf{skip}.$$

2(a) [6 points]. What are

c_2?	$\mathbf{if}\, 45 \leq 1000\, \mathbf{then}\, (X := X - 3;\ Y := X - 1;\ X := Y - 1;\ w\, \mathbf{else\, skip}$
σ_2?	σ
c_3?	$\mathbf{if\, true\, then}\, (X := X - 3;\ Y := X - 1;\ X := Y - 1;\ w)\, \mathbf{else\, skip}$
σ_3?	σ
c_n?	**skip**
σ_n?	$\sigma[40/X][41/Y]$

The answers for c_3 and σ_3 were graded relative to the answers for c_2 and σ_2.

2(b) [4 points]. How many c_i's are of the form $\mathbf{while}\, b\, \mathbf{do}\, c$? $\boxed{193}$ *Actually the correct answer is really 192. We forgot that the first configuration in the chain ($\langle w, \sigma \rangle$) was not explicitly described to be c_0. Thus the first* **while** *in the chain does not really contribute to the count. If we had let $\langle c_0, \sigma_0 \rangle = \langle w, \sigma \rangle$ then there would not have been a problem. Full credit was given for either answer, unless it was clear that 192 was arrived at via a mistake (and thus two wrongs making a right).*

2(c) [5 points]. There are k times as many c_i's which are of the form

$$\mathbf{if}\, b'\, \mathbf{then}\, c\, \mathbf{else}\, c'$$

than are of the form

$$\mathbf{while}\, b''\, \mathbf{do}\, c''.$$

What is k? $\boxed{3}$ *Due to the slight miscounting in the preceding answer the correct answer is really* $(193 * 3)/192 \approx 3.0156$. *Credit was given for either answer.*

Problem 3 [20 points]. It was noted in class that every **Aexp** configuration evaluates to a number. Likewise, one can prove by *structural induction on* **Bexp** that every **Bexp** configuration evaluates to a truth value, namely,

for all $\langle b, \sigma \rangle$, there is a $t \in \{\mathbf{true}, \mathbf{false}\}$ such that $\langle b, \sigma \rangle \rightarrow t$.

3(a) [10 points]. List the cases of the structural induction and indicate what must be shown for each case.

The base cases are:

$[b \equiv t \in \{\mathbf{true}, \mathbf{false}\}]$ *We must show that, under no additional assumptions, there exists a* $t' \in \{\mathbf{true}, \mathbf{false}\}$ *such that* $\langle t, \sigma \rangle \rightarrow t'$.

$[b \equiv a_0 = a_1]$ *We must show that there exists a* $t \in \{\mathbf{true}, \mathbf{false}\}$ *such that* $\langle a_0 = a_1, \sigma \rangle \rightarrow t$. *To do so, we may use the analogous property for* **Aexp**, viz. *to assume that there exist* n_0 *and* n_1 *such that* $\langle a_0, \sigma \rangle \rightarrow n_0$ *and* $\langle a_1, \sigma \rangle \rightarrow n_1$.

$[b \equiv a_0 \leq a_1]$ *Similar to the preceding case.*

The non-base cases are:

$[b \equiv \neg b']$ *Under the assumption that there exists a* $t \in \{\mathbf{true}, \mathbf{false}\}$ *such that* $\langle b', \sigma \rangle \rightarrow t$, *we must show that there exists a* $t' \in \{\mathbf{true}, \mathbf{false}\}$ *such that* $\langle \neg b', \sigma \rangle \rightarrow t'$.

$[b \equiv b_0 \wedge b_1]$ *Under the assumption that there exist* $t_0, t_1 \in \{\mathbf{true}, \mathbf{false}\}$ *such that* $\langle b_0, \sigma \rangle \rightarrow t_0$ *and* $\langle b_1, \sigma \rangle \rightarrow t_1$, *we must show that there exists a* $t \in \{\mathbf{true}, \mathbf{false}\}$ *such that* $\langle b_0 \wedge b_1, \sigma \rangle \rightarrow t$.

$[b \equiv b_0 \vee b_1]$ *Similar to the preceding case.*

3(b) [10 points]. Pick a non-base case and prove it!

We pick the non base-case $[b \equiv b_0 \wedge b_1]$.

Under the inductive assumption that there exist $t_0, t_1 \in \{\mathbf{true}, \mathbf{false}\}$ *such that* $\langle b_0, \sigma \rangle \rightarrow t_0$ *and* $\langle b_1, \sigma \rangle \rightarrow t_1$, *we must show that there exists a* $t \in \{\mathbf{true}, \mathbf{false}\}$ *such that* $\langle b_0 \wedge b_1, \sigma \rangle \rightarrow t$. *But by rule (and* $\rightarrow$*), there is such a* t, *namely the conjunction of* t_0 *and* t_1.

Problem 4 [25 points]. We define a "parallel evals to" relation, $\hookrightarrow$, which is a variation of the "evals to" relation, $\to$. The rules to define $\hookrightarrow$ are obtained from the rules defining $\to$ by replacing all occurrences of "$\to$" by "$\hookrightarrow$". In addition, there is one further "parallel if" rule:

$$\frac{\langle c_0, \sigma\rangle \hookrightarrow \sigma', \quad \langle c_1, \sigma\rangle \hookrightarrow \sigma'}{\langle \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \hookrightarrow \sigma'} \qquad \text{(par-if} \hookrightarrow)$$

4(a) [5 points]. Give a simple example of a command, c, such that $\langle c, \sigma\rangle \hookrightarrow \sigma$ has more than one derivation for any state σ.

$\mathbf{if}\, b\, \mathbf{then}\, c'\, \mathbf{else}\, c'$, *for any* c'

Although the *definition* of $\hookrightarrow$ differs from that of $\to$, it turns out to specify the *same relation* on configurations as $\to$. The nontrivial direction of this remark is the implication

$$\langle c, \sigma\rangle \hookrightarrow \sigma' \;\Rightarrow\; \langle c, \sigma\rangle \to \sigma'.$$

This implication can be proved by induction on the definition of $\hookrightarrow$ (that is by rule induction on the rules for $\hookrightarrow$).

4(b) [10 points]. Briefly explain what the cases of the induction are, and why there is only one non-trivial case.

There is one case for each of the inference rules of $\hookrightarrow$ on **Com**-*configurations (or on* **Aexp**, **Bexp**, *and* **Com** *configurations. This was slightly ambiguous but unimportant, since either reading gave the same definition of $\hookrightarrow$).*

So there are the base cases for the **Com**-*configuration rules: (skip $\hookrightarrow$), (assign $\hookrightarrow$).*

The inductive cases are for the rules: (seq $\hookrightarrow$), (if-true $\hookrightarrow$), (if-false $\hookrightarrow$) (while-false $\hookrightarrow$), (while-false $\hookrightarrow$), and finally a case for the new rule (par-if $\hookrightarrow$).

The only non-trivial case is for the new rule (par-if $\hookrightarrow$), because the other rules for $\hookrightarrow$ are the same as the corresponding rules for $\to$. In particular, if $\langle c, \sigma\rangle \hookrightarrow \sigma'$ follows from some ($\hookrightarrow$)-rule, R, other than (par-if $\to$), then the antecedents if any, of R which involve $\hookrightarrow$, each implies by induction, the corresponding antecedent with "$\hookrightarrow$" replaced by "$\to$", so $\langle c, \sigma\rangle \to \sigma'$ follows trivially by the $\to$-version of R.

4(c) [10 points]. Prove the non-trivial case. (You may assume the results mentioned in Problem 3.)

So, we suppose that $\langle c, \sigma\rangle \hookrightarrow \sigma'$ because of the rule (par-if $\hookrightarrow$).

Then c must be of the form $\textbf{if}\ b\ \textbf{then}\ c_0\ \textbf{else}\ c_1$, *where* $\langle c_0, \sigma\rangle \hookrightarrow \sigma'$ *and* $\langle c_1, \sigma\rangle \hookrightarrow \sigma'$ *(so by induction, we may assume that* $\langle c_0, \sigma\rangle \rightarrow \sigma'$ *and* $\langle c_1, \sigma\rangle \rightarrow \sigma'$*).*

By Problem 3, we know that there exists a $t \in \{\textbf{true}, \textbf{false}\}$ *such that* $\langle b, \sigma\rangle \rightarrow t$*. This gives us two cases based on* t*.*

Suppose $t \equiv$ **true**. *Since* $\langle b, \sigma\rangle \rightarrow$ **true**, *rule (if-true →) applies, and so then* $\langle c, \sigma\rangle \rightarrow \sigma'$.

The case of $t \equiv$ **false** *works similarly. Since* $\langle b, \sigma\rangle \rightarrow$ **false**, *rule (if-false →) applies, and so then* $\langle c, \sigma\rangle \rightarrow \sigma'$.

A Appendix

We use n, sometimes with subscripts as in n_0, n_1, to denote arbitrary elements of **Num**. Similarly, we assume $X, Y \in$ **Loc**; $a \in$ **Aexp**, $t \in \{\textbf{true}, \textbf{false}\}$; $b \in$ **Bexp**; $c \in$ **Com**; and $\sigma \in$ the set of states.

A.1 "Evals to" Rules for IMP

Notice that we give a name for each rule in parentheses to its right.

A.1.1 Aexp Rules

$$\langle n, \sigma \rangle \rightarrow n \qquad (\text{num} \rightarrow)$$

$$\langle X, \sigma \rangle \rightarrow \sigma(n) \qquad (\text{loc} \rightarrow)$$

$$\frac{\langle a_0, \sigma \rangle \rightarrow n_0, \quad \langle a_1, \sigma \rangle \rightarrow n_1}{\langle a_0 + a_1, \sigma \rangle \rightarrow n} \qquad (\text{plus} \rightarrow)$$

where n is the sum of n_0 and n_1.

Similarly, there are rules (times $\rightarrow$) and (minus $\rightarrow$).

A.1.2 Bexp Rules

$$\langle t, \sigma \rangle \rightarrow t \qquad (\text{bool} \rightarrow)$$

$$\frac{\langle a_0, \sigma \rangle \rightarrow n_0, \quad \langle a_1, \sigma \rangle \rightarrow n_1}{\langle a_0 = a_1, \sigma \rangle \rightarrow t} \qquad (\text{equal} \rightarrow)$$

where $t \equiv$ **true** if n_0 and n_1 are equal, otherwise $t \equiv$ **false**.

Similarly, there is a rule ($\leq \rightarrow$).

$$\frac{\langle b, \sigma \rangle \rightarrow t}{\langle \neg b, \sigma \rangle \rightarrow t'} \qquad (\text{not} \rightarrow)$$

where t' is the negation of t.

$$\frac{\langle b_0, \sigma \rangle \rightarrow t_0, \quad \langle b_1, \sigma \rangle \rightarrow t_1}{\langle b_0 \wedge b_1, \sigma \rangle \rightarrow t} \qquad (\text{and} \rightarrow)$$

where t is **true** if $t_0 \equiv$ **true** and $t_1 \equiv$ **true**, and is **false** otherwise.

Similarly, there is a rule (or $\rightarrow$).

A.1.3 Com Rules

$$\langle \mathbf{skip}, \sigma\rangle \to \sigma \qquad \text{(skip} \to)$$

$$\frac{\langle a, \sigma\rangle \to n}{\langle X := a, \sigma\rangle \to \sigma[n/X]} \qquad \text{(assign} \to)$$

$$\frac{\langle c_0, \sigma\rangle \to \sigma'', \quad \langle c_1, \sigma''\rangle \to \sigma'}{\langle (c_0; c_1), \sigma\rangle \to \sigma'} \qquad \text{(seq} \to)$$

$$\frac{\langle b, \sigma\rangle \to \mathbf{true}, \quad \langle c_0, \sigma\rangle \to \sigma'}{\langle \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \to \sigma'} \qquad \text{(if-true} \to)$$

$$\frac{\langle b, \sigma\rangle \to \mathbf{false}, \quad \langle c_1, \sigma\rangle \to \sigma'}{\langle \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \to \sigma'} \qquad \text{(if-false} \to)$$

$$\frac{\langle b, \sigma\rangle \to \mathbf{false}}{\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma\rangle \to \sigma} \qquad \text{(while-false} \to)$$

$$\frac{\langle b, \sigma\rangle \to \mathbf{true}, \quad \langle c, \sigma\rangle \to \sigma'', \quad \langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma''\rangle \to \sigma'}{\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma\rangle \to \sigma'} \qquad \text{(while-true} \to)$$

A.2 Rewriting rules for IMP

A.2.1 Aexp Rules

$$\langle X, \sigma\rangle \to_1 \langle \sigma(X), \sigma\rangle \qquad \text{(loc} \to_1)$$

$$\frac{\langle a_0, \sigma\rangle \to_1 \langle a_0', \sigma\rangle}{\langle a_0 + a_1, \sigma\rangle \to_1 \langle a_0' + a_1, \sigma\rangle} \qquad \text{(plus-left} \to_1)$$

$$\frac{\langle a, \sigma\rangle \to_1 \langle a', \sigma\rangle}{\langle n + a, \sigma\rangle \to_1 \langle n + a', \sigma\rangle} \qquad \text{(plus-right} \to_1)$$

$$\langle n_0 + n_1, \sigma\rangle \to_1 \langle n, \sigma\rangle \qquad \text{(plus-num} \to_1)$$

where n is the sum of n_0 and n_1.

Similarly, there are rules (times-left $\to_1$), (times-right $\to_1$), (times-num $\to_1$), (minus-left $\to_1$), (minus-right $\to_1$), and (minus-num $\to_1$).

A.2.2 Bexp Rules

$$\frac{\langle a_0, \sigma\rangle \rightarrow_1 \langle a_0', \sigma\rangle}{\langle a_0 = a_1, \sigma\rangle \rightarrow_1 \langle a_0' = a_1, \sigma\rangle} \qquad \text{(equal-left} \rightarrow_1)$$

$$\frac{\langle a, \sigma\rangle \rightarrow_1 \langle a', \sigma\rangle}{\langle n = a, \sigma\rangle \rightarrow_1 \langle n = a', \sigma\rangle} \qquad \text{(equal-right} \rightarrow_1)$$

$$\langle n_0 = n_1, \sigma\rangle \rightarrow_1 \langle t, \sigma\rangle \qquad \text{(equal-num} \rightarrow_1)$$

where $t \equiv$ **true** if n_0 and n_1 are equal, otherwise $t \equiv$ **false**.

Similarly, there are rules ($\leq$-left $\rightarrow_1$), ($\leq$-right $\rightarrow_1$), and ($\leq$-num $\rightarrow_1$).

$$\frac{\langle b, \sigma\rangle \rightarrow_1 \langle b', \sigma\rangle}{\langle \neg b, \sigma\rangle \rightarrow_1 \langle \neg b', \sigma\rangle} \qquad \text{(not-eval-arg} \rightarrow_1)$$

$$\langle \neg t, \sigma\rangle \rightarrow_1 \langle t', \sigma\rangle \qquad \text{(not-bool} \rightarrow_1)$$

where t' is the negation of t.

$$\frac{\langle b_0, \sigma\rangle \rightarrow_1 \langle b_0', \sigma\rangle}{\langle b_0 \wedge b_1, \sigma\rangle \rightarrow_1 \langle b_0' \wedge b_1, \sigma\rangle} \qquad \text{(and-left} \rightarrow_1)$$

$$\frac{\langle b, \sigma\rangle \rightarrow_1 \langle b', \sigma\rangle}{\langle t \wedge b, \sigma\rangle \rightarrow_1 \langle t \wedge b', \sigma\rangle} \qquad \text{(and-right} \rightarrow_1)$$

$$\langle t_0 \wedge t_1, \sigma\rangle \rightarrow_1 \langle t, \sigma\rangle \qquad \text{(and-bool} \rightarrow_1)$$

where $t \equiv$ **true** if $t_0 \equiv$ **true** and $t_1 \equiv$ **true**, otherwise $t \equiv$ **false**.

Similarly there are rules (or-left $\rightarrow_1$), (or-right, $\rightarrow_1$) and (or-bool $\rightarrow_1$).

A.2.3 Com Rules

$$\langle \mathbf{skip}, \sigma\rangle \rightarrow_1 \sigma \qquad \text{(skip} \rightarrow_1)$$

$$\frac{\langle a, \sigma\rangle \rightarrow_1 \langle a', \sigma\rangle}{\langle X := a, \sigma\rangle \rightarrow_1 \langle X := a', \sigma\rangle} \qquad \text{(assign-eval-arg} \rightarrow_1)$$

$$\langle X := n, \sigma\rangle \rightarrow_1 \sigma[n/X] \qquad \text{(assign-num} \rightarrow_1)$$

$$\frac{\langle c_0, \sigma\rangle \rightarrow_1 \langle c_0', \sigma'\rangle}{\langle (c_0; c_1), \sigma\rangle \rightarrow_1 \langle (c_0'; c_1), \sigma'\rangle} \qquad \text{(seq-start } \rightarrow_1\text{)}$$

$$\frac{\langle c_0, \sigma\rangle \rightarrow_1 \sigma'}{\langle (c_0; c_1), \sigma\rangle \rightarrow_1 \langle c_1, \sigma'\rangle} \qquad \text{(seq-finish } \rightarrow_1\text{)}$$

$$\frac{\langle b, \sigma\rangle \rightarrow_1 \langle b', \sigma\rangle}{\langle \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_1 \langle \mathbf{if}\, b'\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle} \qquad \text{(if-eval-guard } \rightarrow_1\text{)}$$

$$\langle \mathbf{if\, true\, then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_1 \langle c_0, \sigma\rangle \qquad \text{(if-true } \rightarrow_1\text{)}$$

$$\langle \mathbf{if\, false\, then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_1 \langle c_1, \sigma\rangle \qquad \text{(if-false } \rightarrow_1\text{)}$$

$$\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma\rangle \rightarrow_1 \langle \mathbf{if}\, b\, \mathbf{then} (c; \mathbf{while}\, b\, \mathbf{do}\, c)\, \mathbf{else\, skip}, \sigma\rangle \qquad \text{(while } \rightarrow_1\text{)}$$

Quiz 1

Instructions. This is a **closed book** exam; no notes either. There are four (4) problems, worth 25 points each, on pages 2–7 of this booklet. Write your solutions for all problems on this exam sheet in the spaces provided, including your ***name on each sheet.*** Don't accidentally skip a page. Ask for further blank sheets if you need them.

For your reference, there are appendices listing the definitions of the "evaluates to" relation $\rightarrow_r$ and the one-step rewriting relation $\rightarrow_{r,1}$ on configurations of the language $\mathbf{IMP}_r$.

GOOD LUCK!

NAME

problem	*points*	*score*
1	25	
2	25	
3	25	
4	25	
Total	100	

Recall from Problem Set 2 that $\mathbf{IMP}_r$ is the extension of **IMP** by a further expression construct, $c\,\mathbf{resultis}\,a$, where c is a command and a is an arithmetic expression. On this quiz, arithmetic expressions, $a, a_0, \ldots$, Boolean expressions, $b, b_0, \ldots$, and commands, $c, c_0, \ldots$, are understood to be those of $\mathbf{IMP}_r$. The natural evaluation rules ($\rightarrow_r$) and one-step ($\rightarrow_{r,1}$) rules of $\mathbf{IMP}_r$ are attached in an appendix to this quiz. To avoid clutter, the subscript "r" on the arrows will henceforth be dropped.

In $\mathbf{IMP}_r$, every command is equivalent to an assignment statement, namely

$$c \quad \sim \quad X := c\,\mathbf{resultis}\,X$$

(Recall that $c_1 \sim c_2$ means that for all σ, σ',

$$\langle c_1, \sigma\rangle \rightarrow \sigma' \text{ iff } \langle c_1, \sigma\rangle \rightarrow \sigma'.)$$

Problem 1 [25 points]. One way to prove this equivalence would be by appeal to the one-step semantics. Thus, if

$$\langle c, \sigma\rangle \rightarrow_1 \langle c_1, \sigma_1\rangle \rightarrow_1 \cdots \rightarrow_1 \langle c_n, \sigma_n\rangle \rightarrow_1 \sigma'$$

for some sequence of configurations $\langle c_i, \sigma_i\rangle$ and $n \geq 1$, then

$$\langle X := c\,\mathbf{resultis}\,X, \sigma\rangle \rightarrow_1 \langle c'_1, \sigma'_1\rangle \rightarrow_1 \cdots \rightarrow_1 \langle c'_n, \sigma'_n\rangle \rightarrow_1 \cdots \rightarrow_1 \langle c'_{n+k}, \sigma'_{n+k}\rangle \rightarrow_1 \sigma'$$

for some sequence of configurations $\langle c'_i, \sigma'_i\rangle$ and $k \geq 1$.

Note that c'_{n+k} is an assignment of the form $X := m$.

1(a) [18 points]. Which of the following correctly describes m? (**circle** *all* those which are correct):

1. n	6. $\sigma'(X)$
2. $n+1$	7. $\sigma(X)+k$
3. $n+k$	8. $\sigma'(X)+k$
4. $n+k+1$	9. $\sigma'_n(X)$
5. $\sigma(X)$	10. $\sigma'_{n+k}(X)$

1(b) [7 points]. What is k? $k = \boxed{}$

Problem 2 [25 points]. There is another, direct way to prove this equivalence: describe how to find a derivation of

2(a) [12 points].

$$\langle X := c\,\mathbf{resultis}\,X, \sigma\rangle \to \sigma'$$

from a derivation of

$$\langle c, \sigma\rangle \to \sigma',$$

2(b) [13 points]. and vice-versa.

Problem 3 [25 points]. We define four (4) sets of rule instances over the numbers:

$$R_1 ::= \quad \frac{}{7} \quad \frac{n}{n+2} \quad \frac{n, n+1}{n+2} \quad \text{for } n \in \textbf{Num}$$

$$R_2 ::= \quad \frac{}{7} \quad \frac{}{8} \quad \frac{n}{n+2} \quad \frac{n, n+1}{n+2} \quad \text{for } n \in \textbf{Num}$$

$$R_3 ::= \quad \frac{}{1} \quad \frac{n}{n+2} \quad \frac{1, 3, 5, \ldots, 2n+1}{2n} \quad \text{for } n \in \textbf{Num}$$

$$R_4 ::= \quad \frac{}{4} \quad \frac{}{6} \quad \frac{n, m}{n+m} \quad \text{for } n \in \textbf{Num}$$

3(a) [16 points]. For $i = 1, 2, 3, 4$ give a simple description of I_{R_i}, the set of numbers derivable from R_i.

$I_{R_1} =$ ______

$I_{R_2} =$ ______

$I_{R_3} =$ ______

$I_{R_4} =$ ______

3(b) [9 points]. Which i satisfy the property that every number in I_{R_i} has a unique R_i-derivation? For the others, what is the smallest integer with two derivations?

	unique derivation (yes/no)	If "no," smallest integer
R_1		
R_2		
R_3		
R_4		

Problem 4 [25 points]. Say that $b \in \mathbf{Bexp}_r$ is *side-effect free* iff, for all states σ_1, σ_2,

$$\langle b, \sigma_1 \rangle \rightarrow \langle t, \sigma_2 \rangle \quad \Rightarrow \quad \sigma_1 = \sigma_2.$$

4(a) [20 points]. Suppose b is side-effect free. Carefully prove that

$$\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma \rangle \rightarrow \sigma' \quad \Rightarrow \quad \langle b, \sigma' \rangle \rightarrow \langle \mathbf{false}, \sigma' \rangle.$$

(Part 4(b) is on the next page.)

4(b) [5 points]. Give a simple b (with side-effects) which is a counterexample to the implication of part 4(a).

Appendix A: Natural Evaluation Rules for IMP$_r$

As mentioned earlier in the quiz, we'll be omitting the r subscripts to reduce clutter. Otherwise, these definitions are the same as those given in Problem Set 2.

Rules for Aexp

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

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

$$\frac{\langle a_0, \sigma \rangle \rightarrow \langle n_0, \sigma'' \rangle, \quad \langle a_1, \sigma'' \rangle \rightarrow \langle n_1, \sigma' \rangle}{\langle a_0 + a_1, \sigma \rangle \rightarrow \langle n, \sigma' \rangle}$$

where n is the sum of n_0 and n_1.

There are similar rules for $\times$ and $-$.

$$\frac{\langle c, \sigma \rangle \rightarrow \sigma'', \quad \langle a, \sigma'' \rangle \rightarrow \langle n, \sigma' \rangle}{\langle c\, \mathbf{resultis}\, a, \sigma \rangle \rightarrow \langle n, \sigma' \rangle}$$

Rules for Bexp

$$\langle t, \sigma \rangle \rightarrow \langle t, \sigma \rangle$$

$$\frac{\langle a_0, \sigma \rangle \rightarrow \langle n_0, \sigma'' \rangle, \quad \langle a_1, \sigma'' \rangle \rightarrow \langle n_1, \sigma' \rangle}{\langle a_0 = a_1, \sigma \rangle \rightarrow \langle t, \sigma' \rangle}$$

where $t = \mathbf{true}$ if n_0 and n_1 are equal, otherwise $t = \mathbf{false}$.

There is a similar rule for $\leq$.

$$\frac{\langle b, \sigma \rangle \rightarrow \langle t, \sigma' \rangle}{\langle \neg b, \sigma \rangle \rightarrow \langle t', \sigma' \rangle}$$

where t' is the negation of t.

$$\frac{\langle b_0, \sigma \rangle \rightarrow \langle t_0, \sigma'' \rangle, \quad \langle b_1, \sigma'' \rangle \rightarrow \langle t_1, \sigma' \rangle}{\langle b_0 \wedge b_1, \sigma \rangle \rightarrow \langle t, \sigma' \rangle}$$

where t is **true** if $t_0 = \mathbf{true}$ and $t_1 = \mathbf{true}$, and is **false** otherwise.

There is a similar rule for $\vee$.

Rules for Com

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

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

$$\frac{\langle c_0, \sigma\rangle \to \sigma'', \quad \langle c_1, \sigma''\rangle \to \sigma'}{\langle (c_0; c_1), \sigma\rangle \to \sigma'}$$

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

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

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

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

Appendix B: One-Step Evaluation Rules for $\mathbf{IMP}_r$

As in previous handouts, we will use **op** to range over syntactic operator symbols, and *op* to range over corresponding arithmetic or Boolean operations. Also, as elsewhere in the quiz, we will be dropping the r subscripts to avoid clutter. The language involved is still $\mathbf{IMP}_r$, however.

Rules for Arithmetic Expressions, Aexp

$$\langle X, \sigma\rangle \rightarrow_1 \langle \sigma(X), \sigma\rangle$$

$$\frac{\langle c, \sigma\rangle \rightarrow_1 \langle c', \sigma'\rangle}{\langle c\,\mathbf{resultis}\,a, \sigma\rangle \rightarrow_1 \langle c'\,\mathbf{resultis}\,a, \sigma'\rangle}$$

$$\frac{\langle c, \sigma\rangle \rightarrow_1 \sigma'}{\langle c\,\mathbf{resultis}\,a, \sigma\rangle \rightarrow_1 \langle a, \sigma'\rangle}$$

$$\frac{\langle a_0, \sigma\rangle \rightarrow_1 \langle a_0', \sigma'\rangle}{\langle a_0\ \mathbf{op}\ a_1, \sigma\rangle \rightarrow_1 \langle a_0'\ \mathbf{op}\ a_1, \sigma'\rangle}$$

$$\frac{\langle a_1, \sigma\rangle \rightarrow_1 \langle a_1', \sigma'\rangle}{\langle n\ \mathbf{op}\ a_1, \sigma\rangle \rightarrow_1 \langle n\ \mathbf{op}\ a_1', \sigma'\rangle}$$

$$\langle n\ \mathbf{op}\ m, \sigma\rangle \rightarrow_1 \langle n\ op\ m, \sigma\rangle$$

op	*op*
$+$	the sum function
$-$	the subtraction function
$\times$	the multiplication function

Rules for Boolean Expressions, Bexp

$$\frac{\langle a_0, \sigma\rangle \rightarrow_1 \langle a_0', \sigma'\rangle}{\langle a_0\ \mathbf{op}\ a_1, \sigma\rangle \rightarrow_1 \langle a_0'\ \mathbf{op}\ a_1, \sigma'\rangle}$$

$$\frac{\langle a_1, \sigma\rangle \rightarrow_1 \langle a_1', \sigma'\rangle}{\langle n\ \mathbf{op}\ a_1, \sigma\rangle \rightarrow_1 \langle n\ \mathbf{op}\ a_1', \sigma'\rangle}$$

$$\langle n\ \mathbf{op}\ m, \sigma\rangle \rightarrow_1 \langle n\ op\ m, \sigma\rangle$$

op	*op*
$=$	the equality predicate
$\leq$	the less than or equal to predicate

We next have the rules for Boolean negation:

$$\frac{\langle b, \sigma\rangle \rightarrow_1 \langle b', \sigma'\rangle}{\langle \neg b, \sigma\rangle \rightarrow_1 \langle \neg b', \sigma'\rangle}$$

$$\langle \neg\, \mathbf{true}, \sigma\rangle \rightarrow_1 \langle \mathbf{false}, \sigma\rangle$$

$$\langle \neg\, \mathbf{false}, \sigma\rangle \rightarrow_1 \langle \mathbf{true}, \sigma\rangle$$

Finally we have the rules for binary Boolean operators. We let $t, t_0, t_1, \ldots$ range over the set $\mathbf{T} = \{\mathbf{true}, \mathbf{false}\}$.

$$\frac{\langle b_0, \sigma\rangle \rightarrow_1 \langle b_0', \sigma'\rangle}{\langle b_0 \;\mathbf{op}\; b_1, \sigma\rangle \rightarrow_1 \langle b_0' \;\mathbf{op}\; b_1, \sigma'\rangle}$$

$$\frac{\langle b_1, \sigma\rangle \rightarrow_1 \langle b_1', \sigma'\rangle}{\langle t_0 \;\mathbf{op}\; b_1, \sigma\rangle \rightarrow_1 \langle t_0 \;\mathbf{op}\; b_1', \sigma'\rangle}$$

$$\langle t_0 \;\mathbf{op}\; t_1, \sigma\rangle \rightarrow_1 \langle t_0 \;op\; t_1, \sigma\rangle$$

op	*op*
$\wedge$	the conjunction operation (Boolean AND)
$\vee$	the disjunction operation (Boolean OR)

Rules for Commands, Com

Atomic Commands:

$$\langle \mathbf{skip}, \sigma\rangle \rightarrow_1 \sigma$$

$$\frac{\langle a, \sigma\rangle \rightarrow_1 \langle a', \sigma'\rangle}{\langle X := a, \sigma\rangle \rightarrow_1 \langle X := a', \sigma'\rangle}$$

$$\langle X := n, \sigma\rangle \rightarrow_1 \sigma[n/X]$$

Sequencing:

$$\frac{\langle c_0, \sigma\rangle \rightarrow_1 \langle c_0', \sigma'\rangle}{\langle (c_0; c_1), \sigma\rangle \rightarrow_1 \langle (c_0'; c_1), \sigma'\rangle}$$

$$\frac{\langle c_0, \sigma\rangle \rightarrow_1 \sigma'}{\langle (c_0; c_1), \sigma\rangle \rightarrow_1 \langle c_1, \sigma'\rangle}$$

Conditionals:

$$\frac{\langle b, \sigma\rangle \rightarrow_1 \langle b', \sigma'\rangle}{\langle \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_1 \langle \mathbf{if}\, b'\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma'\rangle}$$

$$\langle \mathbf{if\, true\, then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_1 \langle c_0, \sigma\rangle$$

$$\langle \mathbf{if\, false\, then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_1 \langle c_1, \sigma\rangle$$

While-loops:

$$\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma\rangle \rightarrow_1 \langle \mathbf{if}\, b\, \mathbf{then}(c; \mathbf{while}\, b\, \mathbf{do}\, c)\, \mathbf{else\, skip}, \sigma\rangle$$

Quiz 1 Solutions

(This was a **closed book** exam; no notes either. There were four (4) problems, worth 25 points each.)

Recall from Problem Set 2 that $\mathbf{IMP}_r$ is the extension of $\mathbf{IMP}$ by a further expression construct, $c\,\mathbf{resultis}\,a$, where c is a command and a is an arithmetic expression. On this quiz, arithmetic expressions, $a, a_0, \ldots$, Boolean expressions, $b, b_0, \ldots$, and commands, $c, c_0, \ldots$, are understood to be those of $\mathbf{IMP}_r$. The natural evaluation rules ($\rightarrow_r$) and one-step ($\rightarrow_{r,1}$) rules of $\mathbf{IMP}_r$ were attached in an appendix to this quiz. To avoid clutter, the subscript "r" on the arrows will henceforth be dropped.

In $\mathbf{IMP}_r$, every command is equivalent to an assignment statement, namely

$$c \quad \sim \quad X := c\,\mathbf{resultis}\,X$$

(Recall that $c_1 \sim c_2$ means that for all σ, σ',

$$\langle c_1, \sigma\rangle \rightarrow \sigma' \text{ iff } \langle c_1, \sigma\rangle \rightarrow \sigma'.)$$

(Note that the second command should have been "c_2".)

Problem 1 [25 points]. One way to prove this equivalence would be by appeal to the one-step semantics. Thus, if

$$\langle c, \sigma\rangle \rightarrow_1 \langle c_1, \sigma_1\rangle \rightarrow_1 \cdots \rightarrow_1 \langle c_n, \sigma_n\rangle \rightarrow_1 \sigma'$$

for some sequence of configurations $\langle c_i, \sigma_i\rangle$ and $n \geq 1$, then

$$\langle X := c\,\mathbf{resultis}\,X, \sigma\rangle \rightarrow_1 \langle c'_1, \sigma'_1\rangle \rightarrow_1 \cdots \rightarrow_1 \langle c'_n, \sigma'_n\rangle \rightarrow_1 \cdots \rightarrow_1 \langle c'_{n+k}, \sigma'_{n+k}\rangle \rightarrow_1 \sigma'$$

for some sequence of configurations $\langle c'_i, \sigma'_i\rangle$ and $k \geq 1$.

Note that c'_{n+k} is an assignment of the form $X := m$.

1(a) [18 points]. Which of the following correctly describes m? (**circle** ***all*** those which are correct):

1. n	6. $\sigma'(X)$
2. $n+1$	7. $\sigma(X)+k$
3. $n+k$	8. $\sigma'(X)+k$
4. $n+k+1$	9. $\sigma'_n(X)$
5. $\sigma(X)$	10. $\sigma'_{n+k}(X)$

1(b) [7 points]. What is k? $k = \boxed{2}$

From the evaluation rules for $\rightarrow_1$*, we know that every* $\langle c'_i, \sigma'_i\rangle$ *must be exactly* $\langle X := c_i\, \mathbf{resultis} X, \sigma_i\rangle$*, for* $i \leq n$*. Further, since* $\langle c_n, \sigma_n\rangle \rightarrow_1 \sigma'$*, we know that* c_n *is one of* **skip**, $X := m$*, or* $Y := m'$ *(for some numeral* m'*), where* Y *is a different location from* X*. This gives us three possible classes of evaluations, starting with* $\langle c'_n, \sigma'_n\rangle$*:*

$$\begin{aligned}\langle X := (\mathbf{skip}\,\mathbf{resultis} X), \sigma_n\rangle &\rightarrow_1 \langle X := X, \sigma_n\rangle \\ &\rightarrow_1 \langle X := m, \sigma_n\rangle \\ &\rightarrow_1 \sigma_n\end{aligned}$$

or

$$\begin{aligned}\langle X := (X := m\,\mathbf{resultis} X), \sigma_n\rangle &\rightarrow_1 \langle X := X, \sigma_n[m/X]\rangle \\ &\rightarrow_1 \langle X := m, \sigma_n[m/X]\rangle \\ &\rightarrow_1 \sigma_n[m/X]\end{aligned}$$

or

$$\begin{aligned}\langle X := (Y := m'\,\mathbf{resultis} X), \sigma_n\rangle &\rightarrow_1 \langle X := X, \sigma_n[m'/Y]\rangle \\ &\rightarrow_1 \langle X := m, \sigma_n[m'/Y]\rangle \\ &\rightarrow_1 \sigma_n[m'/Y]\end{aligned}$$

Note, first, that all of these are of the form

$$\langle c'_n, \sigma'_n\rangle \rightarrow_1 \langle c'_{n+1}, \sigma'_{n+1}\rangle \rightarrow_1 \langle c'_{n+2}, \sigma'_{n+2}\rangle \rightarrow_1 \sigma',$$

so $k = 2$*. Next, each* c'_{n+2} *is of the form* $X := \sigma'_{n+2}(X)$*, so answer* 10 *applies. Finally, note that in each case* $\sigma' = \sigma'_{n+2}$*, so answer* 6 *applies as well. Answer* 9 *is true in some cases (like the first and third), but not true in general, as the second case shows.*

Problem 2 [25 points]. There is another, direct way to prove this equivalence: describe how to find a derivation of

2(a) [12 points].

$$\langle X := c\,\mathbf{resultis}\,X, \sigma\rangle \to \sigma'$$

from a derivation of

$$\langle c, \sigma\rangle \to \sigma',$$

Given a derivation $d \Vdash \langle c, \sigma\rangle \to \sigma'$, *we have the following derivation:*

$$\cfrac{\cfrac{d \qquad \langle X, \sigma'\rangle \to \langle \sigma'(X), \sigma'\rangle}{\langle c\,\mathbf{resultis}\,X, \sigma\rangle \to \langle \sigma'(X), \sigma'\rangle}}{\langle X := c\,\mathbf{resultis}\,X, \sigma\rangle \to \sigma'[\sigma'(X)/X]}$$

Since $\sigma'[\sigma'(X)/X] = \sigma'$, *we have a derivation of*

$$\langle X := c\,\mathbf{resultis}\,X, \sigma\rangle \to \sigma'.$$

2(b) [13 points]. and vice-versa.

Given a derivation $d \Vdash \langle X := c\,\mathbf{resultis}\,X, \sigma\rangle \to \sigma'$, *we know that* d *must end with the rule*

$$\frac{\langle c\,\mathbf{resultis}\,X, \sigma\rangle \to \langle m, \sigma''\rangle}{\langle X := c\,\mathbf{resultis}\,X, \sigma\rangle \to \sigma'}$$

where $\sigma' = \sigma''[m/X]$.

Then, the derivation of $\langle c\,\mathbf{resultis}\,X, \sigma\rangle \to \langle m, \sigma''\rangle$ *must end with the rule*

$$\frac{\langle c, \sigma\rangle \to \sigma''' \quad \langle X, \sigma'''\rangle \to \langle m, \sigma''\rangle}{\langle c\,\mathbf{resultis}\,X, \sigma\rangle \to \langle m, \sigma''\rangle}$$

Now,

$$\langle X, \sigma'''\rangle \to \langle m, \sigma''\rangle$$

is an axiom; but for this axiom to hold, it must be the case that $\sigma'''(X) = m$ *and* $\sigma'' = \sigma'''$. *But this means that*

$$\sigma''' = \sigma'''[m/X] = \sigma''[m/X] = \sigma'.$$

Thus, we know that d is of the form:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \langle c,\sigma\rangle \rightarrow \sigma'\end{array} \qquad \langle X,\sigma'\rangle \rightarrow \langle \sigma'(X),\sigma'\rangle}{\langle c\,\mathbf{resultis}\,X,\sigma\rangle \rightarrow \langle \sigma'(X),\sigma'\rangle}}{\langle X := c\,\mathbf{resultis}\,X,\sigma\rangle \rightarrow \sigma'}$$

which contains a derivation for $\langle c,\sigma\rangle \rightarrow \sigma'$ as a subderivation.

Problem 3 [25 points]. We define four (4) sets of rule instances over the numbers:

$$R_1 ::= \quad \frac{}{7} \quad \frac{n}{n+2} \quad \frac{n, n+1}{n+2} \qquad \text{for } n \in \mathbf{Num}$$

$$R_2 ::= \quad \frac{}{7} \quad \frac{}{8} \quad \frac{n}{n+2} \quad \frac{n, n+1}{n+2} \qquad \text{for } n \in \mathbf{Num}$$

$$R_3 ::= \quad \frac{}{1} \quad \frac{n}{n+2} \quad \frac{1,3,5,\ldots,2n+1}{2n} \qquad \text{for } n \in \mathbf{Num}$$

$$R_4 ::= \quad \frac{}{4} \quad \frac{}{6} \quad \frac{n, m}{n+m} \qquad \text{for } n \in \mathbf{Num}$$

3(a) [16 points]. For $i = 1,2,3,4$ give a simple description of I_{R_i}, the set of numbers derivable from R_i.

$I_{R_1} =$	*The odd integers* ≥ 7
$I_{R_2} =$	*The integers* ≥ 7
$I_{R_3} =$	*The integers* ≥ 1 (or ≥ 0)
$I_{R_4} =$	*The even integers* ≥ 4

(The set **Num** *in the above definitions was really intended to be the natural numbers* $\omega = \{0, 1, 2, \ldots\}$ *rather than all the integers, and both* n *and* m *were intended to range over* ω.*) There was an unintended ambiguity in the definition of* R_3*: it was not clear from the definition whether the set of rules refered to by*

$$\frac{1,3,5,\ldots,2n+1}{2n} \qquad \textit{for } n \in \mathbf{Num}$$

was

$$\frac{1,3}{2}, \frac{1,3,5}{4}, \frac{1,3,5,7}{6}, \ldots$$

or

$$\frac{1}{0}, \frac{1,3}{2}, \frac{1,3,5}{4}, \frac{1,3,5,7}{6}, \ldots$$

The second set of rules makes 0 *a member of* I_{R_3}, *while the first set does not. We accepted either answer, and graded the answer to part 3(b) accordingly.*

3(b) [9 points]. Which i satisfy the property that every number in I_{R_i} has a unique R_i-derivation? For the others, what is the smallest integer with two derivations?

	unique derivation (yes/no)	If "no," smallest integer
R_1	*Yes*	
R_2	*No*	9
R_3	*No*	4 (or 2)
R_4	*No*	12

If the description of I_{R_3} in part 3(a) included 0*, then the answer for R_3 should be* 2*; otherwise the answer is* 4.

Problem 4 [25 points]. Say that $b \in \mathbf{Bexp}_r$ is *side-effect free* iff, for all states σ_1, σ_2,

$$\langle b, \sigma_1 \rangle \to \langle t, \sigma_2 \rangle \quad \Rightarrow \quad \sigma_1 = \sigma_2.$$

4(a) [20 points]. Suppose b is side-effect free. Carefully prove that

$$\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma \rangle \to \sigma' \quad \Rightarrow \quad \langle b, \sigma' \rangle \to \langle \mathbf{false}, \sigma' \rangle.$$

This is a very simple induction on derivations.

Suppose $d \Vdash \langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma \rangle \to \sigma'$. *Then:*

Base case: *If the last rule of d is*

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

then, since b is side-effect free, $\sigma = \sigma'$, *so we have* $\langle b, \sigma' \rangle \to \langle \mathbf{false}, \sigma' \rangle$.

Induction: *If the last rule of d is not the while-false rule, then it must be while-true:*

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

But, since the derivation of $\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma''' \rangle \to \sigma'$ *is a subderivation of d, by induction we have*

$$\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma''' \rangle \to \sigma' \quad \Rightarrow \quad \langle b, \sigma' \rangle \to \langle \mathbf{false}, \sigma' \rangle.$$

So $\langle b, \sigma' \rangle \to \langle \mathbf{false}, \sigma' \rangle$.

4(b) [5 points]. Give a simple b (with side-effects) which is a counterexample to the implication of part 4(a). $\boxed{(\textbf{if}\, X = 0\, \textbf{then}\, X := 1\, \textbf{else}\, X := 0\, \textbf{resultis}\, X) = 0}$

It was required to find a **Bexp**, *b, such that*

$$\langle \textbf{while}\, b\, \textbf{do}\, c, \sigma\rangle \to \sigma' \qquad \textit{and} \qquad \langle b, \sigma'\rangle \not\to \langle \textbf{false}, \sigma'\rangle$$

for some c, σ *and* σ'. *The easiest way is simply to ensure that b always changes the state; thus, if b is* $(X := X + 1\, \textbf{resultis}\, 0) = 1$, *then, if* $\sigma(X) = 0$ *we have*

$$\langle \textbf{while}\, b\, \textbf{do}\, \textbf{skip}, \sigma\rangle \to \sigma[1/X] \qquad \textit{and} \qquad \langle b, \sigma[1/X]\rangle \to \langle \textbf{false}, \sigma[2/X]\rangle$$

The expression we have given in the answer box is slightly more complex, and actually yields **true** *as well as changing the state. With b as in the answer box and* $\sigma(X) = 0$:

$$\langle \textbf{while}\, b\, \textbf{do}\, \textbf{skip}, \sigma\rangle \to \sigma[1/X] \qquad \textit{and} \qquad \langle b, \sigma[1/X]\rangle \to \langle \textbf{true}, \sigma[0/X]\rangle$$

Yet another possibility would be for $\langle b, \sigma'\rangle$ *to fail to evaluate to anything in state* σ', *as with* $((\textbf{while}\, X = 1\, \textbf{do}\, \textbf{skip}); X := X + 1\, \textbf{resultis}\, 0) = 1$.

Finally, there is the possibility of a **Bexp**, *b, such that*

$$\langle b, \sigma'\rangle \to \langle \textbf{true}, \sigma'\rangle.$$

We leave this as an extra-credit exercise.

Problem Set 4

Due: 14 October 1993

Reading assignment. Winskel Chapter 5, §1–4.

Problem 1. (Diagnostic, no credit) Indicate how you have fulfilled the prerequisites for 6.044J/18.423J, *e.g.*, prior course and grade, concurrent course, "permission of instructor", ...

Problem 2. Let S be a set, and P be $\mathcal{P}\mathrm{ow}(S)$ with the set containment relation ($\subseteq$).

2(a) Prove that P is a partial order in which the lub of a family $\mathcal{S}$ of subsets of S exists and in fact equals $\bigcup \mathcal{S}$. Likewise for glb's and intersections ($\bigcap$). Conclude that $\mathcal{P}\mathrm{ow}(S)$ is a cpo.

2(b) Why isn't $\mathcal{P}\mathrm{ow}_f(S)$, the family of ***finite*** subsets of S, a cpo under containment?

2(c) Prove that in P, the binary operation lub is continuous in each argument, *i.e.*, for any fixed $p \in P$, the function $f_{p,\mathrm{lub}} : P \to P$ is continuous where $f_{p,\mathrm{lub}}(x) = \mathrm{lub}(x, p) = x \cup p$. Likewise for glb.

Problem 3. Let $U = [0, 1]$ be the closed unit interval with the usual order. Since U is a totally ordered cpo, a function $f : U \to U$ may be order-continuous as well as real-continuous in the usual (ϵ-δ) sense. For each of the f_i's below, indicate whether it is monotone, real-continuous, and/or order-continuous.

3(a) $f_0(x) = 0.0$ for $x < 1/2$, $f_0(x) = 0.1$ for $x \geq 1/2$.

3(b) $f_1(x) = 0.0$ for $x \leq 1/2$, $f_1(x) = 0.1$ for $x > 1/2$.

3(c) $f_2(x) = 1/(x + 1)$.

(over)

Problem 4. Let $(P, \leq)$ be a poset. Define $f : P \to \mathcal{P}ow(P)$ by $f(x) = \{y \in P \mid y \leq x\}$.

4(a) Prove $x \leq y$ iff $f(x) \subseteq f(y)$.

4(b) Suppose P is a cpo. Prove that f is continuous iff there is *no* strictly increasing infinite chain of elements in P.

In the following two problems we consider denotational semantics for $\mathbf{IMP}_r$, the extension of **IMP** considered in previous problem sets.

Problem 5. Define the denotational semantics of $\mathbf{IMP}_r$ by structural induction, omitting the case of while-loops. (Notation: let $A_r = \Sigma \rightharpoonup (\mathbf{Num} \times \Sigma)$ be the domain of values of $\mathbf{Aexp}_r$, likewise B_r for $\mathbf{Bexp}_r$, and $C = \Sigma \rightharpoonup \Sigma$ for $\mathbf{Com}_r$.)

Problem 6. Let $w \in \mathbf{Com}_r$ be $\mathbf{while}\, b\, \mathbf{do}\, c$.

6(a) Carefully define the function $\Gamma_w : C \to C$ such that $\mathcal{C}[\![w]\!] = \mathit{fix}(\Gamma_w)$.

6(b) Prove that Γ_w is continuous.

Grade Statistics for Quiz 1

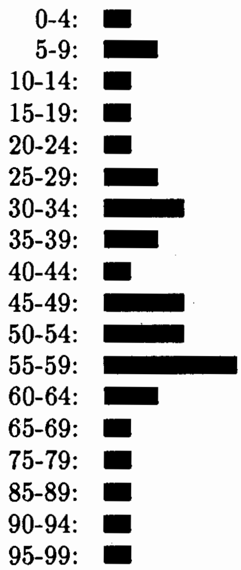
Number of quizzes taken: 32

Grade range: 2–98

Mean: 46

Median: 47

Histogram:



Problem Set 5

Due: 21 October 1993

Reading assignment. Winskel Chapter 6.

Problem 1. Let $C = \Sigma \rightharpoonup \Sigma$ be the "command meanings," namely, partial functions from Σ to Σ. For $f \in C$, let graph(f) be $\{(\sigma, \sigma') \mid f(\sigma) = \sigma'\}$. Partially order C by the rule that $f_1 \leq_C f_2$ iff $\text{graph}(f_1) \subseteq \text{graph}(f_2)$.

Prove that C is a cpo.

Problem 2. Let $\Sigma_\perp$ be the set of states, Σ, along with a "fresh" object, $\perp$, called the "bottom state." Partially order $\Sigma_\perp$ by the rule that

$$\sigma_1 \ \leq_{\Sigma_\perp} \ \sigma_2 \text{ iff } [(\sigma_1 = \perp) \text{ or } (\sigma_1 = \sigma_2)].$$

Let S be the set of *total* functions $g : \Sigma_\perp \to \Sigma_\perp$ such that $g(\perp) = \perp$ (such functions are called "strict"). Partially order S by the rule that $g_1 \leq_S g_2$ iff $g_1(\sigma) \leq_{\Sigma_\perp} g_2(\sigma)$ for all $\sigma \in \Sigma_\perp$.

2(a) Describe a *bijection* taking any $f \in C$ to an $f^s \in S$ such that

$$f_1 \leq_C f_2 \text{ iff } f_1^s \leq_S f_2^s.$$

2(b) Conclude that S is a cpo.

In the next two problems we explore the expressive power of formulas in **Assn**, culminating in construction of a formula $POW \in$ **Assn** which means "$i = k^n$." The key technical trick on which the construction rests is the coding of any finite sequence of numbers into a single number, in such a way that the relationship between the code and the numbers it codes can be expressed by a formula in **Assn**. Lemma 7.3 of Winskel describes one ingenious way of expressing such a sequence-coding formula using the Chinese Remainder Theorem of number theory. We illustrate an alternative approach based on treating nonnegative numbers as strings of digits and expressing string concatentation by **Assn**'s.

For $n \geq 0, p \geq 2$ we write $(n)_p$ to denote the string of digits (between 0 and $p-1$) representing n in base p notation, and we use $\cdot$ to denote the concatenation of strings. For example:

$$\begin{aligned}
(5)_3 &= \text{“12”} \\
(21)_3 &= \text{“210”} \\
(5)_3 \cdot (21)_3 &= \text{“12210”} \\
&= (1 \times 3^4 + 2 \times 3^3 + 2 \times 3^2 + 1 \times 3^1 + 0 \times 3^0)_3 \\
&= (156)_3
\end{aligned}$$

Problem 3.

3(a) Write a formula $POWP \in \textbf{Assn}$ with free variables p, i which means "p is prime and i is a power of p".

Hint: i is a power of p iff $i \geq 1$ and any divisor of i other than 1 must itself have p as a divisor.

3(b) Write a formula $LEN \in \textbf{Assn}$ with free variables i, j, p, such that LEN means "p is prime and $j = p^l$ where l is the length of the base p representation of i."

Hint: j is the largest power of p with a certain relation to i.

3(c) Write a formula $CONCAT \in \textbf{Assn}$ with free variables p, i, j, k which means: "p is prime and $(i)_p \cdot (j)_p = (k)_p$."

Hint: $k = p^{\text{length}((j)_p)} \times i + j$.

Problem 4.

4(a) For $k, n \geq 1$, let $s(k, n)$ be the string of numbers $0\,1\,k\,0\,2\,k^2 \ldots 0\,n\,k^n$. Describe a formula $KNS \in \textbf{Assn}$ with free variables j, k, n which means "$(j)_p = s(k, n)$ for some p."

Hint: Use several copies of *CONCAT*, with some variables renamed, as subformulas of *KNS*.

4(b) Describe a formula $POW \in \textbf{Assn}$ with free variables i, k, n which means "$i = k^n$."

Hint: Use *KNS*.

Problem Set 2 Solutions

Problem 1.

(a) We want to show that $a + a \sim 2 \times a$ for $a \in \mathbf{Aexp}$ of **IMP**.

Pick an arbitrary state σ_0. Then, as we have previously shown, for all $a \in \mathbf{Aexp}$ there is a unique integer n such that $\langle a, \sigma_0 \rangle \to n$. But then, by the rules for $\to$:

$$\frac{\langle a, \sigma_0 \rangle \to n}{\langle a + a, \sigma_0 \rangle \to n + n} \quad \text{and} \quad \frac{\langle 2, \sigma_0 \rangle \to 2, \quad \langle a, \sigma_0 \rangle \to n}{\langle 2 \times a, \sigma_0 \rangle \to 2n}$$

so, since $n + n = 2n$, and evaluations are deterministic, we have

$$\langle a + a, \sigma_0 \rangle \to m \text{ iff } \langle 2 \times a, \sigma_0 \rangle \to m.$$

But, since we have proven this with no assumptions about σ_0, this proof holds for *all* σ_0, so

$$a + a \sim 2 \times a.$$

(b) We know from the proof above that if $a + a \not\sim 2 \times a$ then a must be an expression in $\mathbf{Aexp}_r$ that is not in $\mathbf{Aexp}$; *i.e.*, it must contain at least one **resultis** expression. A simple example is

$$a \equiv (X := X + 1 \,\mathbf{resultis}\, X).$$

To see that $a + a \not\sim 2 \times a$ in $\mathbf{IMP}_r$, take a state σ such that $\sigma(X) = 0$. Then

$$\frac{\begin{array}{c}\vdots \\ \langle X := X + 1, \sigma \rangle \to_r \sigma[1/X]\end{array} \quad \langle X, \sigma[1/X] \rangle \to_r \langle 1, \sigma[1/X] \rangle}{\langle a, \sigma \rangle \to_r \langle 1, \sigma[1/X] \rangle}$$

and

$$\frac{\begin{array}{c}\vdots \\ \langle X := X + 1, \sigma[1/X] \rangle \to_r \sigma[2/X]\end{array} \quad \langle X, \sigma[2/X] \rangle \to_r \langle 2, \sigma[2/X] \rangle}{\langle a, \sigma[1/X] \rangle \to_r \langle 2, \sigma[2/X] \rangle}$$

so

$$\frac{\langle a, \sigma \rangle \to_r \langle 1, \sigma[1/X] \rangle \quad \langle a, \sigma[1/X] \rangle \to_r \langle 2, \sigma[2/X] \rangle}{\langle a + a, \sigma \rangle \to_r \langle 3, \sigma[2/X] \rangle}$$

but

$$\dfrac{\langle 2,\sigma\rangle \rightarrow_r \langle 2,\sigma\rangle \qquad \dfrac{\dfrac{\vdots}{\langle X := X+1,\sigma\rangle \rightarrow_r \sigma[1/X]} \qquad \langle X,\sigma[1/X]\rangle \rightarrow_r \langle 1,\sigma[1/X]\rangle}{\langle a,\sigma\rangle \rightarrow_r \langle 1,\sigma[1/X]\rangle}}{\langle 2\times a,\sigma\rangle \rightarrow_r \langle 2,\sigma[1/X]\rangle}$$

Problem 2. (As noted in an email message to **6044-forum**, the hint given for this problem was slightly misleading. While it was helpful to do the proof separately for **Aexp** then **Bexp** then **Com**, it was not necessary to prove the statements simultaneously by induction. The latter part of this hint more properly applied to problem 4.)

The proof can proceed in three stages:

(A) Prove $\langle a,\sigma\rangle \rightarrow_r \langle n,\sigma\rangle$ iff $\langle a,\sigma\rangle \rightarrow n$;

(B) Prove $\langle b,\sigma\rangle \rightarrow_r \langle t,\sigma\rangle$ iff $\langle b,\sigma\rangle \rightarrow t$, using **(A)**;

(C) Prove $\langle c,\sigma\rangle \rightarrow_r \sigma'$ iff $\langle c,\sigma\rangle \rightarrow \sigma'$, using **(A)** and **(B)**.

Note that cases **(A)** and **(B)** not only imply that **Aexp's** and **Bexp's** evaluate to the same number or truth value under $\rightarrow$ and $\rightarrow_r$, but also that the evaluation leaves the state unchanged. This condition is important for proving case **(C)**.

(A) This case can be proven by induction on the structure of a term $a \in$ **Aexp**.

Base cases

***a* is a numeral *m*:** This case follows immediately from the rules for **IMP** and **IMP$_r$**:

$$\dfrac{}{\langle m,\sigma\rangle \rightarrow_r \langle m,\sigma\rangle} \qquad \text{and} \qquad \dfrac{}{\langle m,\sigma\rangle \rightarrow m}.$$

***a* is a location *X*:** Similarly,

$$\dfrac{}{\langle X,\sigma\rangle \rightarrow_r \langle \sigma(X),\sigma\rangle} \qquad \text{and} \qquad \dfrac{}{\langle X,\sigma\rangle \rightarrow \sigma(X)}.$$

Inductive case

Assume $a \equiv a_0$ **op** a_1 and $a \in$ **Aexp**. First of all, a_0 and a_1 are in **Aexp** by the definition of **Aexp**. Then, if $\langle a,\sigma\rangle \rightarrow_r \langle n,\sigma'\rangle$, we know by the rules for $\rightarrow_r$ that

$$\langle a_0,\sigma\rangle \rightarrow_r \langle n_0,\sigma''\rangle \qquad \text{and} \qquad \langle a_1,\sigma''\rangle \rightarrow_r \langle n_1,\sigma'\rangle$$

for some state σ'', with $n = n_0\ op\ n_1$ (where op is the operation corresponding to the symbol **op**). But, by induction, since a_0 and a_1 are subterms of a, we know that $\sigma = \sigma'' = \sigma'$, and that $\langle a_0, \sigma\rangle \to n_0$ and $\langle a_1, \sigma\rangle \to n_1$. Then, by the rules for **IMP**,

$$\frac{\langle a_0,\sigma\rangle \to n_0 \quad \langle a_1,\sigma\rangle \to n_1}{\langle a,\sigma\rangle \to n_0\ op\ n_1 = n}$$

Conversely, if $\langle a_0\ \mathbf{op}\ a_1, \sigma\rangle \to n$, then $\langle a_0, \sigma\rangle \to n_0$ and $\langle a_1, \sigma\rangle \to n_1$. Again, by structural induction we have

$$\langle a_0,\sigma\rangle \to_r \langle n_0,\sigma\rangle \qquad \text{and} \qquad \langle a_1,\sigma\rangle \to_r \langle n_1,\sigma\rangle$$

yielding

$$\frac{\langle a_0,\sigma\rangle \to_r \langle n_0,\sigma\rangle \quad \langle a_1,\sigma\rangle \to_r \langle n_1,\sigma\rangle}{\langle a,\sigma\rangle \to_r n_0\ op\ n_1 = n}$$

(B) The case for **Bexp**'s is proved similarly, using case **(A)**.

(C) The case for **Com**'s must, as usual, be proved by induction on derivations (or rule induction), but the proof is straightforward, now that we have shown **Aexp**'s and **Bexp**'s to be side-effect free.

Base cases

c **is skip:** The only derivations are

$$\frac{}{\langle c,\sigma\rangle \to_r \sigma} \qquad \text{and} \qquad \frac{}{\langle c,\sigma\rangle \to \sigma}$$

c **is** $X := a$**:** We know from case **(A)** that

$$\langle a,\sigma\rangle \to_r \langle n,\sigma\rangle \text{ iff } \langle a,\sigma\rangle \to n.$$

Thus, the derivations must be

$$\frac{\langle a,\sigma\rangle \to_r \langle n,\sigma\rangle}{\langle c,\sigma\rangle \to_r \sigma[n/X]} \qquad \text{and} \qquad \frac{\langle a,\sigma\rangle \to_r n}{\langle c,\sigma\rangle \to \sigma[n/X]}$$

Note how we made use of the absence of **Aexp** side-effect effects in the second case, ensuring that an assignment would evaluate to the same state under either set of rules.

Inductive cases

Most of the other cases follow straightforwardly by induction, as long as we take care to keep track of when the state does and cannot change. A typical case is that for conditionals. Take

$$c \equiv \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1.$$

From case **(B)** we know that $\langle b,\sigma\rangle \rightarrow_r \langle \mathbf{true},\sigma\rangle$ iff $\langle b,\sigma\rangle \rightarrow \mathbf{true}$, and similarly for **false**. Take the case for $\langle b,\sigma\rangle \rightarrow \mathbf{true}$. Then, the derivations are

$$\frac{\langle b,\sigma\rangle \rightarrow_r \langle \mathbf{true},\sigma\rangle \quad \langle c_0,\sigma\rangle \rightarrow_r \sigma'}{\langle c,\sigma\rangle \rightarrow_r \sigma'}$$

and

$$\frac{\langle b,\sigma\rangle \rightarrow \mathbf{true} \quad \langle c_0,\sigma\rangle \rightarrow \sigma''}{\langle c,\sigma\rangle \rightarrow \sigma''}$$

By induction, we can conclude that $\sigma' = \sigma''$, since the derivation of $\langle c_0,\sigma\rangle \rightarrow \sigma''$ is a subderivation of the derivation of $\langle c,\sigma\rangle \rightarrow \sigma''$; we have thus shown $\langle c,\sigma\rangle \rightarrow_r \sigma'$ iff $\langle c,\sigma\rangle \rightarrow \sigma''$. The **false** case is analagous.

Problem 3. We'll use **op** to range over syntactic operator symbols, and *op* to range over corresponding arithmetic or Boolean operations. The only completely new rules are for $\mathbf{Aexp}_r$'s of the form $c\,\mathbf{resultis}\,a$, but many other rules have been changed to account for the presence of side effects in $\mathbf{Aexp}_r$'s.

Rules for $\mathbf{Aexp}_r$

$$\langle X,\sigma\rangle \rightarrow_{r,1} \langle \sigma(X),\sigma\rangle$$

$$\frac{\langle c,\sigma\rangle \rightarrow_{r,1} \langle c',\sigma'\rangle}{\langle c\,\mathbf{resultis}\,a,\sigma\rangle \rightarrow_{r,1} \langle c'\,\mathbf{resultis}\,a,\sigma'\rangle}$$

$$\frac{\langle c,\sigma\rangle \rightarrow_{r,1} \sigma'}{\langle c\,\mathbf{resultis}\,a,\sigma\rangle \rightarrow_{r,1} \langle a,\sigma'\rangle}$$

$$\frac{\langle a_0,\sigma\rangle \rightarrow_{r,1} \langle a_0',\sigma'\rangle}{\langle a_0\ \mathbf{op}\ a_1,\sigma\rangle \rightarrow_{r,1} \langle a_0'\ \mathbf{op}\ a_1,\sigma'\rangle}$$

$$\frac{\langle a_1,\sigma\rangle \rightarrow_{r,1} \langle a_1',\sigma'\rangle}{\langle n\ \mathbf{op}\ a_1,\sigma\rangle \rightarrow_{r,1} \langle n\ \mathbf{op}\ a_1',\sigma'\rangle}$$

$$\langle n\ \mathbf{op}\ m,\sigma\rangle \rightarrow_{r,1} \langle n\ op\ m,\sigma\rangle$$

op	*op*
$+$	the sum function
$-$	the subtraction function
$\times$	the multiplication function

Rules for $\mathbf{Bexp}_r$

$$\frac{\langle a_0, \sigma\rangle \rightarrow_{r,1} \langle a_0', \sigma'\rangle}{\langle a_0 \ \mathbf{op}\ a_1, \sigma\rangle \rightarrow_{r,1} \langle a_0' \ \mathbf{op}\ a_1, \sigma'\rangle}$$

$$\frac{\langle a_1, \sigma\rangle \rightarrow_{r,1} \langle a_1', \sigma'\rangle}{\langle n \ \mathbf{op}\ a_1, \sigma\rangle \rightarrow_{r,1} \langle n \ \mathbf{op}\ a_1', \sigma'\rangle}$$

$$\langle n \ \mathbf{op}\ m, \sigma\rangle \rightarrow_{r,1} \langle n \ op\ m, \sigma\rangle$$

op	*op*
$=$	the equality predicate
$\leq$	the less than or equal to predicate

We next have the rules for Boolean negation:

$$\frac{\langle b, \sigma\rangle \rightarrow_{r,1} \langle b', \sigma'\rangle}{\langle \neg b, \sigma\rangle \rightarrow_{r,1} \langle \neg b', \sigma'\rangle}$$

$$\langle \neg\, \mathbf{true}, \sigma\rangle \rightarrow_{r,1} \langle \mathbf{false}, \sigma\rangle$$

$$\langle \neg\, \mathbf{false}, \sigma\rangle \rightarrow_{r,1} \langle \mathbf{true}, \sigma\rangle$$

Finally we have the rules for binary Boolean operators. We let $t, t_0, t_1, \ldots$ range over the set $\mathbf{T} = \{\mathbf{true}, \mathbf{false}\}$.

$$\frac{\langle b_0, \sigma\rangle \rightarrow_{r,1} \langle b_0', \sigma'\rangle}{\langle b_0 \ \mathbf{op}\ b_1, \sigma\rangle \rightarrow_{r,1} \langle b_0' \ \mathbf{op}\ b_1, \sigma'\rangle}$$

$$\frac{\langle b_1, \sigma\rangle \rightarrow_{r,1} \langle b_1', \sigma'\rangle}{\langle t_0 \ \mathbf{op}\ b_1, \sigma\rangle \rightarrow_{r,1} \langle t_0 \ \mathbf{op}\ b_1', \sigma'\rangle}$$

$$\langle t_0 \ \mathbf{op}\ t_1, \sigma\rangle \rightarrow_{r,1} \langle t_0 \ op\ t_1, \sigma\rangle$$

op	*op*
$\wedge$	the conjunction operation (Boolean AND)
$\vee$	the disjunction operation (Boolean OR)

Rules for $\mathbf{Com}_r$

Atomic Commands:

$$\langle \mathbf{skip}, \sigma\rangle \rightarrow_{r,1} \sigma$$

$$\frac{\langle a, \sigma\rangle \rightarrow_{r,1} \langle a', \sigma'\rangle}{\langle X := a, \sigma\rangle \rightarrow_{r,1} \langle X := a', \sigma'\rangle}$$

$$\langle X := n, \sigma\rangle \rightarrow_{r,1} \sigma[n/X]$$

Sequencing:

$$\frac{\langle c_0, \sigma\rangle \rightarrow_{r,1} \langle c_0', \sigma'\rangle}{\langle (c_0; c_1), \sigma\rangle \rightarrow_{r,1} \langle (c_0'; c_1), \sigma'\rangle}$$

$$\frac{\langle c_0, \sigma\rangle \rightarrow_{r,1} \sigma'}{\langle (c_0; c_1), \sigma\rangle \rightarrow_{r,1} \langle c_1, \sigma'\rangle}$$

Conditionals:

$$\frac{\langle b, \sigma\rangle \rightarrow_{r,1} \langle b', \sigma'\rangle}{\langle \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_{r,1} \langle \mathbf{if}\, b'\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma'\rangle}$$

$$\langle \mathbf{if}\, \mathbf{true}\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_{r,1} \langle c_0, \sigma\rangle$$

$$\langle \mathbf{if}\, \mathbf{false}\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1, \sigma\rangle \rightarrow_{r,1} \langle c_1, \sigma\rangle$$

While-loops:

$$\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma\rangle \rightarrow_{r,1} \langle \mathbf{if}\, b\, \mathbf{then} (c; \mathbf{while}\, b\, \mathbf{do}\, c)\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle$$

Problem 4. As noted in the solution to problem 2, the hint that appeared there is more applicable to this problem. The difficulty with doing this proof for $\mathbf{IMP}_r$ is that $\mathbf{Aexp}_r$ and $\mathbf{Bexp}_r$ are no longer independent of $\mathbf{Com}_r$. Since the evaluation of a $\mathbf{Com}_r$ configuration cannot be determined by structural induction, neither can those for $\mathbf{Aexp}_r$ and $\mathbf{Bexp}_r$. All must be handled simultaneously by induction on derivations.

(A simpler problem, which several people noted, is that several statements in the proof must be changed to account for possible changes in state due to the evaluation of $\mathbf{Aexp}_r$'s or $\mathbf{Bexp}_r$'s.)

The simplest way to adapt the proof in handout 12 to the case of $\mathbf{IMP}_r$, then, is to reproduce the proof for the **Com** case, replacing each proof of a property for **Com**'s with a proof of a corresponding property for *all* terms in $\mathbf{IMP}_r$. Thus, where we formerly would prove, say,

$$\langle c, \sigma \rangle \rightarrow_1^* \sigma' \text{ iff } \langle c, \sigma \rangle \rightarrow \sigma'$$

by induction on the derivation of $\langle c, \sigma \rangle \rightarrow \sigma'$, we now prove the (single) property

$$\begin{gathered}\langle a, \sigma \rangle \rightarrow_{r,1}^* \langle n, \sigma' \rangle \text{ iff } \langle a, \sigma \rangle \rightarrow_r \langle n, \sigma' \rangle; \text{ and} \\ \langle b, \sigma \rangle \rightarrow_{r,1}^* \langle t, \sigma' \rangle \text{ iff } \langle b, \sigma \rangle \rightarrow_r \langle t, \sigma' \rangle; \text{ and} \\ \langle c, \sigma \rangle \rightarrow_{r,1}^* \sigma' \text{ iff } \langle c, \sigma \rangle \rightarrow_r \sigma'.\end{gathered}$$

using induction on the derivation of *any* term, where the cases now correspond to *all* the rules in $\mathbf{IMP}_r$, including those for $\mathbf{Aexp}_r$'s, $\mathbf{Bexp}_r$'s and $\mathbf{Com}_r$'s.

Otherwise, the structure of the proof for **Com** can remain the same. There are no longer any separate lemmas about $\mathbf{Aexp}_r$'s and $\mathbf{Bexp}_r$'s needed, since these cases are all taken care of by a general induction on derivations.

6.044J/18.423J: Computability, Programming, and Logic — Handout 22
Massachusetts Institute of Technology — 21 October 1992

Problem Set 3 Solutions

Problem 1. Winskel's text §2.5 proves that a simple "unwinding" of a while-loop preserves natural semantics, namely,

$$\mathbf{while}\, b\, \mathbf{do}\, c \sim \mathbf{if}\, b\, \mathbf{then}(c; \mathbf{while}\, b\, \mathbf{do}\, c)\, \mathbf{else}\, \mathbf{skip}\,.$$

Prove that this equivalence also holds in $\mathbf{IMP}_r$, the extension of IMP with a $c\,$resultis$\,a$ expression construct (*cf.*, Problem Set 2). Indicate, without repeating them, those portions of the proof for IMP which apply unchanged to $\mathbf{IMP}_r$.

Winskel's proof goes through pretty much intact for $\mathbf{IMP}_r$, except that all constructions must account for the possible changes in state resulting from evaluating b. Thus, instead of using a derivation ending in

$$\frac{\begin{array}{c}\vdots\\ \langle b,\sigma\rangle \to \mathbf{true}\end{array}\qquad \begin{array}{c}\vdots\\ \langle c,\sigma\rangle\to\sigma''\end{array}\qquad \begin{array}{c}\vdots\\ \langle w,\sigma''\rangle\to\sigma'\end{array}}{\langle w,\sigma\rangle\to\sigma'}$$

to construct one ending in

$$\frac{\begin{array}{c}\vdots\\ \langle b,\sigma\rangle \to \mathbf{true}\end{array}\qquad \dfrac{\begin{array}{c}\vdots\\ \langle c,\sigma\rangle\to\sigma''\end{array}\qquad \begin{array}{c}\vdots\\ \langle w,\sigma''\rangle\to\sigma'\end{array}}{\langle c; w,\sigma\rangle\to\sigma'}}{\langle \mathbf{if}\, b\, \mathbf{then}\, c; w\, \mathbf{else}\, \mathbf{skip},\sigma\rangle\to\sigma'}$$

we now take one ending in

$$\frac{\begin{array}{c}\vdots\\ \langle b,\sigma\rangle \to_r \langle\mathbf{true},\sigma''\rangle\end{array}\qquad \begin{array}{c}\vdots\\ \langle c,\sigma''\rangle\to_r\sigma'''\end{array}\qquad \begin{array}{c}\vdots\\ \langle w,\sigma'''\rangle\to_r\sigma'\end{array}}{\langle w,\sigma\rangle\to_r\sigma'}$$

and similarly construct

$$\frac{\begin{array}{c}\vdots\\ \langle b,\sigma\rangle \to_r \langle\mathbf{true},\sigma''\rangle\end{array}\qquad \dfrac{\begin{array}{c}\vdots\\ \langle c,\sigma''\rangle\to_r\sigma'''\end{array}\qquad \begin{array}{c}\vdots\\ \langle w,\sigma'''\rangle\to_r\sigma'\end{array}}{\langle c; w,\sigma''\rangle\to_r\sigma'}}{\langle \mathbf{if}\, b\, \mathbf{then}\, c; w\, \mathbf{else}\, \mathbf{skip},\sigma\rangle\to_r\sigma'}$$

The changes can be done uniformly, and the constructions and proof are otherwise identical.

Problem 2. The operational behavior of IMP expressions and commands depends only on the locations occurring in them. In this problem we give a precise definition of this dependence and verify it. (It may be helpful to look at Winskel §3.5 and Prop. 4.7 for related ideas.)

For $L \subseteq \mathbf{Loc}$ define a relation $=_L$ between states by

$$\sigma_1 =_L \sigma_2 \text{ iff } \sigma_1(X) = \sigma_2(X) \text{ for all } X \in L.$$

Define a relation, $\sim_L$, between commands by

$$\begin{aligned} c_1 \sim_L c_2 \text{ iff } (\sigma_1 =_L \sigma_2 \Rightarrow \\ &(\langle c_1, \sigma_1\rangle \to \sigma_1' \text{ for some } \sigma_1' \text{ iff } \langle c_2, \sigma_2\rangle \to \sigma_2' \text{ for some } \sigma_2') \ \& \\ &((\langle c_1, \sigma_1\rangle \to \sigma_1' \ \& \ \langle c_2, \sigma_2\rangle \to \sigma_2') \Rightarrow \sigma_1' =_L \sigma_2')). \end{aligned}$$

2(a) Give a structural inductive definition of $\mathrm{loc}(a)$, the set of locations occurring in $a \in \mathbf{Aexp}$. Do likewise for $\mathrm{loc}(b)$ and $\mathrm{loc}(c)$.

By structural induction, we can define the function as follows:

For **Aexp**:

$$\begin{aligned} \mathrm{loc}(n) &\stackrel{\text{def}}{=} \emptyset \qquad \text{for } n \in \mathbf{N} \\ \mathrm{loc}(X) &\stackrel{\text{def}}{=} \{X\} \qquad \text{for } X \in \mathbf{Loc} \\ \mathrm{loc}(a_0 \ \mathbf{op} \ a_1) &\stackrel{\text{def}}{=} \mathrm{loc}(a_0) \cup \mathrm{loc}(a_1) \end{aligned}$$

For **Bexp**:

$$\begin{aligned} \mathrm{loc}(t) &\stackrel{\text{def}}{=} \emptyset \qquad \text{for } t \in \mathbf{T} \\ \mathrm{loc}(a_0 \ \mathbf{op} \ a_1) &\stackrel{\text{def}}{=} \mathrm{loc}(a_0) \cup \mathrm{loc}(a_1) \\ \mathrm{loc}(\neg b) &\stackrel{\text{def}}{=} \mathrm{loc}(b) \\ \mathrm{loc}(b_0 \ \mathbf{op} \ b_1) &\stackrel{\text{def}}{=} \mathrm{loc}(b_0) \cup \mathrm{loc}(b_1) \end{aligned}$$

For **Com**:

$$\begin{aligned} \mathrm{loc}(\mathbf{skip}) &\stackrel{\text{def}}{=} \emptyset \\ \mathrm{loc}(X := a) &\stackrel{\text{def}}{=} \{X\} \cup \mathrm{loc}(a) \\ \mathrm{loc}(c_0; c_1) &\stackrel{\text{def}}{=} \mathrm{loc}(c_0) \cup \mathrm{loc}(c_1) \\ \mathrm{loc}(\mathbf{if}\, b \,\mathbf{then}\, c_0 \,\mathbf{else}\, c_1) &\stackrel{\text{def}}{=} \mathrm{loc}(b) \cup \mathrm{loc}(c_0) \cup \mathrm{loc}(c_1) \\ \mathrm{loc}(\mathbf{while}\, b \,\mathbf{do}\, c) &\stackrel{\text{def}}{=} \mathrm{loc}(b) \cup \mathrm{loc}(c) \end{aligned}$$

2(b) Prove that $c \sim_{\mathrm{loc}(c)} c$ for all $c \in$ Com.

As is frequently the case, we will divide the proof into three stages:

(A) For all $a \in \mathbf{Aexp}$, $\sigma_1 =_{\mathrm{loc}(a)} \sigma_2 \Rightarrow \exists n \in \mathbf{N}.\ \langle a, \sigma_1 \rangle \to n\ \&\ \langle a, \sigma_2 \rangle \to n$

(B) For all $b \in \mathbf{Bexp}$, $\sigma_1 =_{\mathrm{loc}(b)} \sigma_2 \Rightarrow \exists t \in \mathbf{T}.\ \langle b, \sigma_1 \rangle \to t\ \&\ \langle b, \sigma_2 \rangle \to t$

(C) For all $c \in \mathbf{Com}$, $c \sim_{\mathrm{loc}(c)} c$.

Now, case **(A)** can be proved by a simple structural induction. Assume that $\sigma_1 =_{\mathrm{loc}(a)} \sigma_2$ for some $a \in \mathbf{Aexp}$. Then, the cases are

$a \equiv n$ for $n \in \mathbf{N}$:

Trivially, $\langle a, \sigma_1 \rangle \to n$ and $\langle a, \sigma_2 \rangle \to n$ for any σ_1 and σ_2.

$a \equiv X$ for $X \in$ Loc:

Here, $\mathrm{loc}(a) = \{X\}$, so $\sigma_1(X) = \sigma_2(X)$. Then we have $\langle a, \sigma_1 \rangle \to \sigma_1(X)$ and $\langle a, \sigma_2 \rangle \to \sigma_2(X)$, so this case holds.

$a \equiv a_0$ op a_1:

Note, first of all, that if $\sigma_1 =_{X \cup Y} \sigma_2$ for any sets of locations X and Y, then it must be true that $\sigma_1 =_X \sigma_2$, since if σ_1 and σ_2 agree on all locations in X and Y, then they clearly agree on all locations in X alone.

Therefore, if $\sigma_1 =_{\mathrm{loc}(a)} \sigma_2$, then $\sigma_1 =_{\mathrm{loc}(a_0)} \sigma_2$ and $\sigma_1 =_{\mathrm{loc}(a_1)} \sigma_2$. Thus, we can assume by induction that $\langle a_0, \sigma_1 \rangle \to n_0$ and $\langle a_0, \sigma_2 \rangle \to n_0$ for the same n_0, and similarly for a_1 and some n_1.

Thus, we have:

$$\frac{\langle a_0, \sigma_1 \rangle \to n_0, \quad \langle a_1, \sigma_1 \rangle \to n_1}{\langle a, \sigma_1 \rangle \to n_0\ op\ n_1} \quad \text{and} \quad \frac{\langle a_0, \sigma_2 \rangle \to n_0, \quad \langle a_1, \sigma_2 \rangle \to n_1}{\langle a, \sigma_2 \rangle \to n_0\ op\ n_1}$$

Case **(B)** follows similarly, again by structural induction.

Case **(C)**, as usual, will be proven by induction on derivations.

$c \equiv$ skip:

Trivially, $\langle c, \sigma_1 \rangle \to \sigma_1$ and $\langle c, \sigma_2 \rangle \to \sigma_2$ for any σ_1 and σ_2.

$c \equiv X := a$

As above, we know that if $\sigma_1 =_{\mathrm{loc}(c)} \sigma_2$ then $\sigma_1 =_{\mathrm{loc}(a)} \sigma_2$ by the definition of loc(c). Thus, by case **(A)**, there is some $n \in \mathbf{N}$ such that $\langle a, \sigma_1 \rangle \to n$ and $\langle a, \sigma_2 \rangle \to n$, which gives us

$$\frac{\langle a, \sigma_1 \rangle \to n}{\langle c, \sigma_1 \rangle \to \sigma_1[n/X]} \quad \text{and} \quad \frac{\langle a, \sigma_2 \rangle \to n}{\langle c, \sigma_2 \rangle \to \sigma_2[n/X]}$$

Now, if $\sigma_1 =_{\mathrm{loc}(c)} \sigma_2$, then, for any $Y \in \mathrm{loc}(c)$, if $Y \neq X$, then

$$\sigma_1[n/X](Y) = \sigma_1(Y) = \sigma_2(Y) = \sigma_2[n/X](Y),$$

and

$$\sigma_1[n/X](X) = n = \sigma_2[n/X](X),$$

so

$$\sigma_1[n/X] =_{\mathrm{loc}(c)} \sigma_2[n/X].$$

Note that in neither of these cases did we address the possibility that $\langle c, \sigma_1\rangle$ didn't evaluate to anything, since we know that c will always evaluate to something in the base cases. The other cases follow by induction, assuming we consider the possibility of non-evaluation (non-termination). An interesting case is:

$c \equiv$ **while** b **do** c'

Once again, if $\sigma_1 =_{\mathrm{loc}(c)} \sigma_2$ then $\sigma_1 =_{\mathrm{loc}(b)} \sigma_2$ and $\sigma_1 =_{\mathrm{loc}(c')} \sigma_2$.

We first need to verify that $\langle c, \sigma_1\rangle \to \sigma_1'$ for some σ_1' iff $\langle c, \sigma_2\rangle \to \sigma_2'$ for some $\sigma_2' =_{\mathrm{loc}(c)} \sigma_1'$. Assume that $\langle c, \sigma_1\rangle \to \sigma_1'$. Then we have some derivation

$$\frac{\langle b, \sigma_1\rangle \to \mathbf{false}}{\langle c, \sigma_1\rangle \to \sigma_1' = \sigma_1}$$

or

$$\frac{\langle b, \sigma_1\rangle \to \mathbf{true} \quad \langle c', \sigma_1\rangle \to \sigma_1'' \quad \langle c, \sigma_1''\rangle \to \sigma_1'}{\langle c, \sigma_1\rangle \to \sigma_1'}$$

In the first case, case (**B**) gives us

$$\frac{\langle b, \sigma_2\rangle \to \mathbf{false}}{\langle c, \sigma_2\rangle \to \sigma_2}$$

and in the second, case (**B**) together with induction on derivations gives us $\langle b, \sigma_2\rangle \to \mathbf{true}$ and $\langle c', \sigma_2\rangle \to \sigma_2''$ for some $\sigma_2'' =_{\mathrm{loc}(c')} \sigma_1''$. We now invoke the following lemma:

Lemma 1. If $Y \in \mathbf{Loc}$, then for all commands c and states σ and σ',

$$Y \notin \mathrm{loc}(c) \;\&\; \langle c, \sigma\rangle \to \sigma' \Rightarrow \sigma(Y) = \sigma'(Y).$$

(Note that this is just the loc version of Winskel's Proposition 4.7.)

Proof: Note that, by induction on the definitions of loc and loc_L, $\mathrm{loc}_L(c) \subseteq \mathrm{loc}(c)$ for all c. Then, if $Y \notin \mathrm{loc}(c)$ then $Y \notin \mathrm{loc}_L(c)$, so, by Proposition 4.7, $\sigma(Y) = \sigma'(Y)$. ■

From this lemma, we can conclude that if $\sigma_2'' =_{\mathsf{loc}(c')} \sigma_1''$ then, in fact, $\sigma_2'' =_{\mathsf{loc}(c)} \sigma_1''$. Then, by induction, we have $\langle c, \sigma_2'' \rangle \rightarrow \sigma_2'$ for some σ_2' such that $\sigma_2' =_{\mathsf{loc}(c)} \sigma_1'$, which gives us

$$\frac{\langle b, \sigma_2 \rangle \rightarrow \mathbf{true} \quad \langle c', \sigma_2 \rangle \rightarrow \sigma_2'' \quad \langle c, \sigma_2'' \rangle \rightarrow \sigma_2'}{\langle c, \sigma_2 \rangle \rightarrow \sigma_2'}$$

Symmetrically, if $\langle c, \sigma_2 \rangle \rightarrow \sigma_2'$ then $\langle c, \sigma_1 \rangle \rightarrow \sigma_1'$, again with $\sigma_1' =_{\mathsf{loc}(c)} \sigma_2'$.

2(c) We referred to the natural semantics and the one-step semantics of IMP as "operational," but since Loc is an infinite set, and therefore so is each state, these "operational" semantics involve copying and updating infinite objects in derivations. Briefly explain how the result of part 2(b) leads to a more truly "operational" version of natural and one-step semantics using finite portions of states. Then say precisely how the original versions of operational semantics can be retrieved from the versions using finite portions of states.

If we actually want to interpret programs according to the natural or one-step semantics, we need some way of representing the states. Using the observations about loc(c), we can see that only those states in loc(c) need be accounted for when evaluating a program, so we can represent the states in the execution as (finite) tables $T_0, T_1 \ldots$ mapping elements of loc(c) to $\mathbf{N}$.

Now, if we want to verify an evaluation assertion $\langle c, \sigma \rangle \rightarrow \sigma'$ involving infinite σ', σ [illegible], we can do so by finding a derivation of $\langle c, T \rangle \rightarrow T'$ such that $\sigma =_{\mathsf{loc}(c)} T$ and $\sigma' =_{\mathsf{loc}(c)} T'$.

Problem 3. The *natural evaluation control states* of a command c are defined to be the commands c' such that $\langle c', \sigma_3 \rangle \rightarrow \sigma_4$ occurs in a derivation of $\langle c, \sigma_1 \rangle \rightarrow \sigma_2$ for some states $\sigma_1, \sigma_2, \sigma_3, \sigma_4$. Likewise for the *one-step control states.*

3(a) List the natural evaluation and the one-step control states of the command Intsqrt from Problem Set 1.

Recall that Intsqrt is $\mathbf{while}\, \neg(N \leq M \times M)\, \mathbf{do}\, M := M + 1$. Thus, the control states, as they appear in any natural evaluation derivation, are simply

1. $\mathbf{while}\, \neg(N \leq M \times M)\, \mathbf{do}\, M := M + 1$,
2. $M := M + 1$.

No other commands can appear anywhere in any derivation tree.

3(b) Prove that $\mathbf{IMP}_r$, so *a fortiori* also IMP, has *finite-state control*, *i.e.*, for any $c \in \mathbf{Com}_r$, there are only finitely many natural evaluation control states of c.

First, extend the notion of a natural evaluation control state to cover $a \in \mathbf{Aexp}_r$ and $b \in \mathbf{Bexp}_r$: let the natural evaluation control states of an $\mathbf{Aexp}_r$, a, be the set of *commands* c such that $\langle c, \sigma_3 \rangle \rightarrow_r \sigma_4$ appears in the derivation of $\langle a, \sigma_1 \rangle \rightarrow_r \langle n, \sigma_2 \rangle$ for some n, σ_1 and σ_2; and similarly for $b \in \mathbf{Bexp}_r$. (Note that we're only keeping track of commands that appear in the derivation, not any other kinds of subexpressions.)

We can now show that all control states of an expression in $\mathbf{Aexp}_r$, $\mathbf{Bexp}_r$ or $\mathbf{Com}_r$ are (not necessarily proper) subexpressions of that expression, and thus form a finite set.

Proof: By structural induction on $a \in \mathbf{Aexp}_r$, $b \in \mathbf{Bexp}_r$ and $c \in \mathbf{Com}_r$ (simultaneously):

Base Cases:

If a is a numeral or location, or b is a truth value, then there are no control states in any derivation of $\langle a, \sigma \rangle \rightarrow_r \langle n, \sigma \rangle$ or $\langle b, \sigma \rangle \rightarrow_r \langle t, \sigma \rangle$.

For any state σ, if c is **skip**, then the only control state in any derivation of $\langle c, \sigma \rangle \rightarrow_r \sigma$ is c.

Inductive Cases:

Assume that the proposition holds for any command a', b' or c' that is a subexpression of a, b, or c. Then, in any of the following cases

- a is a_0 **op** a_1
- a is c_0 **resultis** a_0
- b is $\neg b_0$
- b is a_0 **op** a_1
- b is b_0 **op** b_1
- c is $X := a_0$
- c is $c_0 ; c_1$
- c is **if** b_0 **then** c_0 **else** c_1

in any derivation d leading to an evaluation of a, b or c in some state, all control states appearing in d must either appear in the derivations of the given subexpressions, or, in the case of c, must be c itself. By induction, all possible control states of each subexpression must be subexpressions of that subexpression, and thus subexpressions of a, b or c as appropriate, so the proposition holds.

If c is **while** b' **do** c' then we can prove the proposition by the minimum principle: pick a σ and σ' such that the derivation $d \Vdash \langle c, \sigma\rangle \rightarrow_r \sigma'$ contains a control state that is not a subexpression of c, and d is a minimal such derivation (*i.e.*, d contains no subderivations of $\langle c, \sigma_0\rangle \rightarrow_r \sigma_1$ which violate the proposition). Derivation d must end with either

$$\frac{\langle b', \sigma\rangle \rightarrow_r \langle \mathbf{false}, \sigma'\rangle}{\langle c, \sigma\rangle \rightarrow_r \sigma'}$$

or

$$\frac{\langle b', \sigma\rangle \rightarrow_r \langle \mathbf{true}, \sigma''\rangle \quad \langle c', \sigma''\rangle \rightarrow_r \sigma''' \quad \langle c, \sigma'''\rangle \rightarrow_r \sigma'}{\langle c, \sigma\rangle \rightarrow_r \sigma'}$$

Since b' and c' are subexpressions of c, the proposition applies to them by induction. Thus, if d contains any control states that are not subexpressions of c, d must end with the second (while-true) rule, and these control states must appear in the derivation of $\langle c, \sigma'''\rangle \rightarrow_r \sigma'$. But that means that there is a subderivation of d that violates the proposition, so d is not minimal, which is a contradiction. Thus c satisfies the proposition, and all control states of c are subexpressions of c.

■

3(c) Show the same result for the one-step control states.

This problem was a blunder on our part, since the obvious "same result" about the control states of $\rightarrow_{r,1}$ *is, in fact, false; consider the one-step control states of* $X := X$. *For any state* σ, *we can derive* $\langle X := X, \sigma\rangle \rightarrow_{r,1} \langle X := \sigma(X), \sigma\rangle$. *But* $\sigma(X)$ *can be any integer, so* $X := X$ *has a one-step control state* $X := n$ *for every* $n \in \mathbf{N}$.

There is, in fact, a notion of finite-state control to be found here, but the definitions we've given won't get to it.

3(d) Briefly discuss how this observation could be used to improve the efficiency of an interpreter for IMP$_r$ based on the natural or one-step operational semantics.

If we know that a given program will only involve finitely many control states in its execution then we can, in some sense, compile it into a list of those states, together with rules for evaluating any given control state. For example, if we know we're evaluating a given control state, we know in advance which other control states we might need to evaluate, and under which conditions we will need to evaluate them. All this information can be compiled into a compact form before we attempt the evaluation.

This is an instance of a common, simple model of computation, in which programs are considered to be finite objects, including only a text (*i.e.*, a list of control states) and a "program counter" pointing to the next portion of text to be evaluated. Memory, however large, can then be considered a separate, and in a sense much simpler and more uniform, object.

Problem Set 4 Solutions

Problem 2. Let S be a set, and P be $\mathcal{P}\text{ow}(S)$ with the set containment relation ($\subseteq$).

2(a) Prove that P is a partial order in which the lub of a family $\mathcal{S}$ of subsets of S exists and in fact equals $\bigcup\mathcal{S}$. Likewise for glb's and intersections ($\bigcap$). Conclude that $\mathcal{P}\text{ow}(S)$ is a cpo.

For P to be a partial order, the ordering relation must be reflexive, transitive and anti-symmetric. In excruciating detail, then:

- Trivially, for any set $A \in P$, every $a \in A$ is an element of A, so $A \subseteq A$, so $\subseteq$ is reflexive.
- For any sets $A, B, C \in P$ with $A \subseteq B \subseteq C$, if $a \in A$ then $a \in B$, so $a \in C$, so $A \subseteq C$, so $\subseteq$ is transitive.
- For any two *distinct* sets $A, B \in P$, assume $A \subseteq B$ and $B \subseteq A$. If A and B are distinct, then there is some element in one that is not in the other, but this will violate either $A \subseteq B$ or $B \subseteq A$. Thus $\subseteq$ is anti-symmetric.

Now, consider a family of sets $\mathcal{S} \subseteq P$ and their union, $\bigcup\mathcal{S}$. Then, for any set $A \in \mathcal{S}$, if $a \in A$ then $a \in \bigcup\mathcal{S}$ (by the definition of the union), so $A \subseteq \bigcup\mathcal{S}$, so $\bigcup\mathcal{S}$ is an upper bound on $\mathcal{S}$. Now, consider any other upper bound U on $\mathcal{S}$. For any $a \in \bigcup\mathcal{S}$, we know $a \in A$ for some $A \in \mathcal{S}$, so, since $A \subseteq U$, $a \in U$. Thus $\bigcup\mathcal{S} \subseteq U$, so $\bigcup\mathcal{S}$ is a least upper bound on $\mathcal{S}$.

Consider now $\bigcap\mathcal{S}$. Then, for any set $A \in \mathcal{S}$, if $a \in \bigcap\mathcal{S}$ then $a \in A$ (by the definition of the intersection), so $\bigcap\mathcal{S} \subseteq A$ for all $A \in \mathcal{S}$, so $\bigcap\mathcal{S}$ is a lower bound on $\mathcal{S}$. Now, consider any other lower bound L on $\mathcal{S}$. For any $a \in L$, we know $a \in A$ for all $A \in \mathcal{S}$, but then $a \in \bigcap\mathcal{S}$. Thus $L \subseteq \bigcap\mathcal{S}$, so $\bigcap\mathcal{S}$ is a greatest lower bound on $\mathcal{S}$.

2(b) Why isn't $\mathcal{P}\text{ow}_f(S)$, the family of *finite* subsets of S, a cpo under containment?

For $\mathcal{P}\text{ow}_f(S)$ to be a cpo, it would have to include greatest lower bounds for all nondecreasing ω-chains in it. For a quick counterexample to this, let S be ω, the set of natural numbers. Then, for each $i \geq 0$, let $A_i = \{j \mid 0 \leq j < i\}$. Each A_i is in $\mathcal{P}\text{ow}_f(\omega)$, and, in fact, the A_i's form an increasing ω-chain under $\subseteq$. However, there is no finite subset B of ω such that $A_i \subseteq B$ for all A_i, so this chain has no upper bound (let alone a least one) in $\mathcal{P}\text{ow}_f(\omega)$.

2(c) Prove that in P, the binary operation lub is continuous in each argument, *i.e.*, for any fixed $p \in P$, the function $f_{p,\text{lub}} : P \to P$ is continuous where $f_{p,\text{lub}}(x) = \text{lub}(x,p) = x \cup p$. Likewise for glb.

Take $f = f_{p,\text{lub}}$ for some arbitrary $p \in P$. For f to be continuous, it must be true that, for any nondecreasing ω-chain $A_0, A_1, \ldots$, $\bigsqcup_i f(A_i) = f(\bigsqcup_i A_i)$. In the case of P and f, this means we have to show that $\bigcup_i (p \cup A_i) = p \cup \bigcup_i A_i$. In detail, then, consider any $a \in \bigcup_i (p \cup A_i)$. By the definition of the union, a must be in $p \cup A_i$ for some i, so a must be in either p or A_i; but if $a \in A_i$ then $a \in \bigcup_i A_i$, so in either case $a \in p \cup \bigcup_i A_i$.

Similarly, if $a \in p \cup \bigcup_i A_i$ then either $a \in p$ or $a \in A_i$ for some A_i. Thus $a \in p \cup A_i$ for some A_i, so $a \in \bigcup_i (p \cup A_i)$. Thus $\bigcup_i (p \cup A_i) = p \cup \bigcup_i A_i$.

Problem 3. Let $U = [0,1]$ be the closed unit interval with the usual order. Since U is a totally ordered cpo, a function $f : U \to U$ may be order-continuous as well as real-continuous in the usual (ϵ-δ) sense. For each of the f_i's below, indicate whether it is monotone, real-continuous, and/or order-continuous.

3(a) $f_0(x) = 0.0$ for $x < 1/2$, $f_0(x) = 0.1$ for $x \geq 1/2$.

- If $y \geq x$ then $f_0(y) \geq f_0(x)$, so f_0 is monotone.
- f_0 is not real-continuous (look at $x = 1/2$).
- Take an increasing ω-chain $a_0, a_1, \ldots$ with least upper bound $1/2$ (say, the sequence $0, 1/4, 3/8, 7/16, \ldots$). $f_0(a_i) = 0.0$ for any i in this chain, so $\bigsqcup_i f_0(a_i) = 0.0$, but $f_0(\bigsqcup_i a_i) = f_0(1/2) = 0.1 \neq \bigsqcup_i f_0(a_i)$, so f_0 is not order-continuous.

3(b) $f_1(x) = 0.0$ for $x \leq 1/2$, $f_1(x) = 0.1$ for $x > 1/2$.

- As before, f_1 is monotone.
- Again, f_1 is discontinuous at $1/2$.
- This time, if any chain has a least upper bound which is greater than $1/2$ (thus causing $\bigsqcup_i f_1(a_i) = 0.1$), then it must have some element greater than $1/2$ (since otherwise $1/2$ would be a smaller upper bound). Thus, for any chain $\{a_i\}$, if $\bigsqcup_i a_i > 1/2$ then $\bigsqcup_i f_1(a_i) = 0.1 = f_1(\bigsqcup_i a_i)$, and otherwise $\bigsqcup_i f_1(a_i) = 0.0 = f_1(\bigsqcup_i a_i)$, so f_1 is order continuous.

3(c) $f_2(x) = 1/(x+1)$.

- f_2 is not monotone, since, *e.g.*, $f(0) = 1 > 1/2 = f(1)$.
- f_2 is real-continuous on the interval [0,1] (though not necessarily outside it).
- Since f_2 is not monotone, it can't be order-continuous.

Problem 4. Let $(P, \leq)$ be a poset. Define $f : P \to \mathcal{P}\mathrm{ow}(P)$ by $f(x) = \{y \in P \mid y \leq x\}$.

4(a) Prove $x \leq y$ iff $f(x) \subseteq f(y)$.

Assume $x \leq y$. Now, $z \in f(x)$ iff $z \leq x$, but then $z \leq y$, so $z \in f(y)$. Conversely, assume $f(x) \subseteq f(y)$. Then, for all $z \leq x$, $z \leq y$; in particular, since $x \leq x$, we have $x \leq y$.

4(b) Suppose P is a cpo. Prove that f is continuous iff there is *no* strictly increasing infinite chain of elements in P.

Assume that there is a strictly increasing infinite chain $a_0, a_1, \ldots$ in P. Then, since P is a cpo, $\{a_i\}$ has a least upper bound. Let $a = \bigsqcup_i a_i$. Since $\{a_i\}$ is strictly increasing and infinite, we know that $a \not\leq a_i$ for any a_i (since, otherwise, we would have a_{i+1} strictly greater than a, which would be a contradiction). Now, since $a \not\leq a_i$ for any i, $a \notin f(a_i)$ for any i, so $a \notin \bigcup_i f(a_i)$; but by definition, $a \in f(a) = f(\bigsqcup_i a_i)$, so $f(\bigsqcup_i a_i) \neq \bigsqcup_i f(a_i)$, so f is not continuous.

On the other hand, assume there is no such chain. Then, for any nondecreasing chain $\{a_i\}$ in P, $\bigsqcup_i a_i = a_j$ for some j (in other words, every nondecreasing chain eventually reaches its maximum value). By 4(a), then, $f(a_i) \subseteq f(a_j)$ for all i, so $\bigcup_i f(a_i) = f(a_j)$, which gives us

$$f(\bigsqcup_i a_i) = f(a_j) = \bigcup_i f(a_i)$$

so f is continuous.

Problem 5. Define the denotational semantics of IMP$_r$ by structural induction, omitting the case of while-loops. (Notation: let $A_r = \Sigma \to (\mathrm{Num} \times \Sigma)$ be the domain of values of Aexp$_r$, likewise B_r for Bexp$_r$, and $C = \Sigma \to \Sigma$ for Com$_r$.)

Assuming **while**-loops are taken care of, we can extend $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ to $\mathcal{A}_r$, $\mathcal{B}_r$ and $\mathcal{C}_r$ as follows:

$$\mathcal{A}_r[\![n]\!]\sigma = (n, \sigma)$$

$$\mathcal{A}_r[\![X]\!]\sigma = (\sigma(X), \sigma)$$

$$\mathcal{A}_r[\![a_0 \textbf{ op } a_1]\!]\sigma = \begin{cases} (n_0 \; op \; n_1, \sigma') & \text{if } \mathcal{A}_r[\![a_0]\!]\sigma = (n_0, \sigma'') \;\& \\ & \quad \mathcal{A}_r[\![a_1]\!]\sigma'' = (n_1, \sigma') \text{ for some } \sigma'' \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathcal{A}_r[\![c \textbf{ resultis } a]\!] = \mathcal{A}_r[\![a]\!] \circ \mathcal{C}_r[\![c]\!]$$

$$\mathcal{B}_r[\![\textbf{true}]\!]\sigma = (\textbf{true}, \sigma)$$

$$\mathcal{B}_r[\![\textbf{false}]\!]\sigma = (\textbf{false}, \sigma)$$

$$\mathcal{B}_r[\![\neg b]\!]\sigma = \begin{cases} (\neg t, \sigma') & \text{if } \mathcal{B}_r[\![b]\!]\sigma = (t, \sigma') \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathcal{B}_r[\![a_0 \textbf{ op } a_1]\!]\sigma = \begin{cases} (n_0 \; op \; n_1, \sigma') & \text{if } \mathcal{A}_r[\![a_0]\!]\sigma = (n_0, \sigma'') \;\& \\ & \quad \mathcal{A}_r[\![a_1]\!]\sigma'' = (n_1, \sigma') \text{ for some } \sigma'' \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathcal{B}_r[\![b_0 \textbf{ op } b_1]\!]\sigma = \begin{cases} (t_0 \; op \; t_1, \sigma') & \text{if } \mathcal{B}_r[\![b_0]\!]\sigma = (t_0, \sigma'') \;\& \\ & \quad \mathcal{B}_r[\![b_1]\!]\sigma'' = (t_1, \sigma') \text{ for some } \sigma'' \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathcal{C}_r[\![\textbf{skip}]\!]\sigma = \sigma$$

$$\mathcal{C}_r[\![X := a]\!]\sigma = \begin{cases} \sigma'[n/X] & \text{if } \mathcal{A}_r[\![a]\!]\sigma = (n, \sigma') \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathcal{C}_r[\![c_0; c_1]\!] = \mathcal{C}_r[\![c_1]\!] \circ \mathcal{C}_r[\![c_0]\!]$$

$$\mathcal{C}_r[\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!]\sigma = \begin{cases} \mathcal{C}_r[\![c_0]\!]\sigma' & \text{if } \mathcal{B}_r[\![b]\!]\sigma = (\textbf{true}, \sigma') \\ \mathcal{C}_r[\![c_1]\!]\sigma' & \text{if } \mathcal{B}_r[\![b]\!]\sigma = (\textbf{false}, \sigma') \\ \text{undefined} & \text{otherwise} \end{cases}$$

Problem 6. Let $w \in \textbf{Com}_r$ be $\textbf{while}\, b \,\textbf{do}\, c$.

6(a) Carefully define the function $\Gamma_w : C \to C$ such that $\mathcal{C}[\![w]\!] = \mathit{fix}(\Gamma_w)$.

Following the text, we can define Γ_w as

$$\Gamma_w(\varphi)(\sigma) = \begin{cases} \sigma' & \text{if } \mathcal{B}_r[\![b]\!]\sigma = (\textbf{true}, \sigma'') \;\&\; (\varphi \circ \mathcal{C}_r[\![c]\!])(\sigma'') = \sigma' \\ \sigma' & \text{if } \mathcal{B}_r[\![b]\!]\sigma = (\textbf{false}, \sigma') \\ \text{undefined} & \text{otherwise} \end{cases}$$

6(b) Prove that Γ_w is continuous.

Take some chain of meanings $\gamma_0 \leq \gamma_1 \leq \cdots$, where each γ_i is an element of C (*i.e.*, a partial function in $\Sigma \rightharpoonup \Sigma$). $\gamma \leq \gamma'$ if $\gamma(\sigma) = \sigma' \Rightarrow \gamma'(\sigma) = \sigma'$. We must show that $\Gamma_w(\bigsqcup_i \gamma_i) = \bigsqcup_i \Gamma_w(\gamma_i)$.

Let $\gamma = \bigsqcup_i \gamma_i$. For any $\sigma \in \Sigma$, we have to show that $\Gamma_w(\gamma)\sigma$ is defined iff $\bigsqcup_i \Gamma_w(\gamma_i)$ is defined, and if both are defined then they are equal.

- If $\mathcal{B}_r[\![b]\!]\sigma$ is undefined, then so is $\Gamma_w(\varphi)(\sigma)$ for any φ; therefore, so are $\Gamma_w(\gamma)(\sigma)$ and $\Gamma_w(\gamma_i)(\sigma)$ for all γ_i, and thus so is $(\bigsqcup_i \Gamma_w(\gamma_i))(\sigma)$
- If $\mathcal{B}_r[\![b]\!]\sigma = (\mathbf{false}, \sigma')$ for some σ', then $\Gamma_w(\varphi)(\sigma) = \sigma'$ for all φ, so $\Gamma_w(\gamma)(\sigma) = \Gamma_w(\gamma_i)(\sigma)$ for all for any γ_i, so $\Gamma_w(\gamma)(\sigma) = (\bigsqcup_i \Gamma_w(\gamma_i))(\sigma)$.
- If $\mathcal{B}_r[\![b]\!]\sigma = (\mathbf{true}, \sigma'')$ for some σ'', then $\Gamma_w(\varphi)(\sigma) = \varphi \circ \mathcal{C}_r[\![c]\!](\sigma'')$, for any φ. Now, if $\mathcal{C}_r[\![c]\!](\sigma'')$ is undefined, then $\Gamma_w(\varphi)(\sigma)$ is undefined for all φ.

  On the other hand, if $\mathcal{C}_r[\![c]\!](\sigma'') = \sigma'''$ for some σ', then $\Gamma_w(\gamma)(\sigma) = \gamma(\sigma''')$. By the ordering on partial functions, it must be true that if $\gamma(\sigma''')$ is undefined then so are all $\gamma_i(\sigma''')$, and if $\gamma(\sigma''') = \sigma'$ for some σ' then, for some i, $\gamma_i(\sigma''') = \sigma'$, so $\Gamma_w(\gamma_i)(\sigma''') = \sigma'$, which means that $\bigsqcup_i(\Gamma_w(\gamma_i))(\sigma) = \sigma'$. Similarly, if $\bigsqcup_i(\Gamma_w(\gamma_i))(\sigma) = \sigma'$, $\mathcal{B}_r[\![b]\!]\sigma = (\mathbf{true}, \sigma'')$ and $\mathcal{C}_r[\![c]\!]\sigma'' = \sigma'''$, then $\gamma_i(\sigma''') = \sigma'$, so $\gamma(\sigma''') = \sigma'$, so $\Gamma_w(\gamma)(\sigma) = \sigma'$.

  Thus, $\Gamma_w(\gamma)(\sigma)$ is defined and has value σ' iff $\bigsqcup_i(\Gamma_w(\gamma_i))$ is also defined with value σ'.

Thus

$$\Gamma_w(\bigsqcup_i \gamma_i) = \bigsqcup_i \Gamma_w(\gamma_i)$$

Last Year's Quiz #2, and Solutions

The original exam had two appendices for reference: Appendix A provided the complete definition of the denotational semantics for the language **IMP**, Appendix B provided the syntax and the definition the "evaluates to" relation $\rightarrow_r$ for the language $\mathbf{IMP}_r$.

Problems 1–3 concern the language $\mathbf{IMP}_r$.

We now describe the denotational semantics of $\mathbf{IMP}_r$. The semantic functions

$$\mathcal{A}_r : \mathbf{Aexp}_r \rightarrow (\Sigma \rightarrow (\mathbf{Num} \times \Sigma))$$
$$\mathcal{B}_r : \mathbf{Bexp}_r \rightarrow (\Sigma \rightarrow (\mathbf{T} \times \Sigma))$$
$$\mathcal{C}_r : \mathbf{Com}_r \rightarrow (\Sigma \rightarrow \Sigma)$$

are defined by structural induction on the expressions *and* the commands of $\mathbf{IMP}_r$ simultaneously.

As usual, we use n, sometimes with subscripts as in n_0, n_1, to denote arbitrary elements of **Num**. Similarly, we assume $X, Y \in \mathbf{Loc}$, $a \in \mathbf{Aexp}_r$, $t \in \mathbf{T} = \{\mathbf{true}, \mathbf{false}\}$, $b \in \mathbf{Bexp}_r$, $c \in \mathbf{Com}_r$, and $\sigma \in \Sigma =$ the set of states.

$$\mathcal{A}_r[\![n]\!]\sigma = (n, \sigma)$$
$$\mathcal{A}_r[\![X]\!]\sigma = (\sigma(X), \sigma)$$
$$\mathcal{A}_r[\![a_0 + a_1]\!]\sigma = \begin{cases} (n_0 + n_1, \sigma') & \text{if } \mathcal{A}_r[\![a_0]\!]\sigma = (n_0, \sigma'')\ \& \\ & \quad \mathcal{A}_r[\![a_1]\!]\sigma'' = (n_1, \sigma'), \\ \text{undefined} & \text{otherwise.} \end{cases}$$
$$\mathcal{A}_r[\![c\,\mathbf{resultis}\,a]\!]\sigma = \begin{cases} \mathcal{A}_r[\![a]\!]\sigma'' & \text{if } \mathcal{C}_r[\![c]\!]\sigma = \sigma'', \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The definitions of $\mathcal{A}_r[\![a_0 - a_1]\!]$ and $\mathcal{A}_r[\![a_0 \times a_1]\!]$ are similar to the definition of $\mathcal{A}_r[\![a_0 + a_1]\!]$. Some other selected cases follow.

$$\mathcal{B}_r[\![\neg b]\!]\sigma = \begin{cases} (\neg t, \sigma') & \text{if } \mathcal{B}_r[\![b]\!]\sigma = (t, \sigma'), \\ \text{undefined} & \text{otherwise.} \end{cases}$$
$$\mathcal{C}_r[\![\mathbf{skip}]\!]\sigma = \sigma$$
$$\mathcal{C}_r[\![c_0; c_1]\!]\sigma = \mathcal{C}_r[\![c_1]\!](\mathcal{C}_r[\![c_0]\!]\sigma)$$
$$\mathcal{C}_r[\![\mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1]\!]\sigma = \begin{cases} \mathcal{C}_r[\![c_0]\!]\sigma'' & \text{if } \mathcal{B}_r[\![b]\!]\sigma = (\mathbf{true}, \sigma''), \\ \mathcal{C}_r[\![c_1]\!]\sigma'' & \text{if } \mathcal{B}_r[\![b]\!]\sigma = (\mathbf{false}, \sigma''), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Problem 1 [25 points]. Supply definitions for the following cases:

1(a) [12 points]. $\mathcal{B}_r[\![a_0 = a_1]\!]$

Solution:

$$\mathcal{B}_r[\![a_0 = a_1]\!]\sigma = \begin{cases} (\mathbf{true}, \sigma') & \text{if } \exists \sigma'', n_0, n_1 . \mathcal{A}_r[\![a_0]\!]\sigma = (n_0, \sigma'') \ \& \\ & \qquad \mathcal{A}_r[\![a_1]\!]\sigma'' = (n_1, \sigma') \ \& \ n_0 \neq n_1, \\ (\mathbf{false}, \sigma') & \text{if } \exists \sigma'', n_0, n_1 . \mathcal{A}_r[\![a_0]\!]\sigma = (n_0, \sigma'') \ \& \\ & \qquad \mathcal{A}_r[\![a_1]\!]\sigma'' = (n_1, \sigma') \ \& \ n_0 = n_1, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

1(b) [13 points]. $\mathcal{C}_r[\![X := a]\!]$.

Solution:

$$\mathcal{C}_r[\![X := a]\!]\sigma = \begin{cases} \sigma'[n/X] & \text{if } \mathcal{A}_r[\![a]\!]\sigma = (n, \sigma'), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Problem 2 [25 points]. We define

$$\mathcal{C}_r[\![\mathbf{while}\, b\, \mathbf{do}\, c]\!] = \mathit{fix}(\Gamma_r)$$

where $\Gamma_r : (\Sigma \rightharpoonup \Sigma) \to (\Sigma \rightharpoonup \Sigma)$ is chosen to have the property that

$$\Gamma_r(\mathcal{C}_r[\![c']\!]) = \mathcal{C}_r[\![\mathbf{if}\, b\, \mathbf{then}\, c; c'\, \mathbf{else}\, \mathbf{skip}]\!].$$

In particular, Γ_r is defined by:

$$\Gamma_r(\varphi)\sigma = \begin{cases} \sigma' & \text{if } \mathcal{B}_r[\![b]\!]\sigma = (\mathbf{false}, \sigma'), \\ \varphi(\mathcal{C}_r[\![c]\!]\sigma'') & \text{if } \mathcal{B}_r[\![b]\!]\sigma = (\mathbf{true}, \sigma''), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Since a partial function $\varphi : \Sigma \rightharpoonup \Sigma$ can be regarded as a subset of $\Sigma \times \Sigma$, the function Γ_r can also be thought of as a function taking a set of state-pairs to another set of state-pairs. We can then conclude, just as was done in class for **IMP**, that Γ_r is continuous and $\mathit{fix}(\Gamma_r)$ is well-defined because $\Gamma = \widehat{R}$ for a certain set of rules R.

Give a set of rules R with the property that $\Gamma_r = \widehat{R}$.

Hint: (For the exam, the set of rules used in the corresponding case for **IMP** were repeated in Appendix A.)

$$R = \{(\emptyset/(\sigma, \sigma')) \mid \mathcal{B}[\![b]\!]\sigma = (\mathbf{false}, \sigma')\} \cup \\ \left\{ \left(\{(\sigma''', \sigma')\}/(\sigma, \sigma') \right) \mid \exists \sigma''.\, \mathcal{B}[\![b]\!]\sigma = (\mathbf{true}, \sigma'') \ \& \ \mathcal{C}[\![c]\!]\sigma'' = \sigma''' \right\}$$

Problem 3 [25 points]. A proof that evaluation behavior determines denotational meaning for $\mathbf{IMP}_r$ can be done by rule induction on the full set of rules simultaneously defining $\rightarrow_r$ for expressions and commands. The induction hypothesis $P(\beta)$, where β is an "evaluation tuple," is defined to be the conjunction ("and") of the following three assertions:

$$\begin{array}{l} \text{if } \beta \equiv (a, \sigma, n, \sigma'), \text{ then } \mathcal{A}_r[\![a]\!]\sigma = (n, \sigma'); \\ \text{if } \beta \equiv (b, \sigma, t, \sigma'), \text{ then } \mathcal{B}_r[\![b]\!]\sigma = (t, \sigma'); \\ \text{if } \beta \equiv (c, \sigma, \sigma'), \text{ then } \mathcal{C}_r[\![c]\!]\sigma = \sigma'. \end{array}$$

Give the subcase of the inductive proof for the rule (if-true $\rightarrow_r$):

We must show that $P(\beta)$ holds because $\beta \equiv (c, \sigma, \sigma')$, and $\langle c, \sigma\rangle \rightarrow_r \sigma'$ due to the rule (if-true $\rightarrow_r$). So, we may assume that $c \equiv$ $\mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1$, and there exists a σ'' such that:

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

Since we are doing a rule induction, we may now assume $P((c_0, \sigma'', \sigma'))$ and $P((b, \sigma, \mathbf{true}, \sigma''))$.

Our goal is to show that $\mathcal{C}_r[\![\mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1]\!]\sigma = \sigma'$. We look back to the introduction and see the definition:

$$\mathcal{C}_r[\![\mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1]\!]\sigma = \begin{cases} \mathcal{C}_r[\![c_0]\!]\sigma'' & \textit{if } \mathcal{B}_r[\![b]\!]\sigma = (\mathbf{true}, \sigma''), \\ \mathcal{C}_r[\![c_1]\!]\sigma'' & \textit{if } \mathcal{B}_r[\![b]\!]\sigma = (\mathbf{false}, \sigma''), \\ \textit{undefined} & \textit{otherwise.} \end{cases}$$

Since $P((b, \sigma, \mathbf{true}, \sigma''))$, $\mathcal{B}_r[\![B]\!]\sigma = (\mathbf{true}, \sigma'')$, so

$$\mathcal{C}_r[\![\mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, C_1]\!]\sigma = \mathcal{C}_r[\![c_0]\!]\sigma''.$$

Since $P((c_0, \sigma'', \sigma'))$, we have

$$\mathcal{C}_r[\![c_0]\!]\sigma'' = \sigma',$$

thus giving $\mathcal{C}_r[\![\mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1]\!]\sigma = \sigma'$, exactly as required.

Problem 4 [25 points].

4(a) [8 points]. Describe a cpo C and a continuous function $g : C \to C$ such that g has no fixed point. You need not prove any of the properties of your example.

Hint: C cannot have a least element.

Let C be $\{\mathbf{true}, \mathbf{false}\}$ with the discrete partial order, so all functions from C to C are continuous, and let g be "negate."

Another example is to let C be the negative integers ordered as usual (by value), and g be the predecessor function.

4(b) [17 points]. Let D be a cpo, $d \in D$, and $f : D \to D$ be a continuous function such that $d \sqsubseteq_D f(d)$. Prove that f has a fixed point d' such that $d \sqsubseteq_D d'$.

Hint: Let $d' = \bigsqcup_{n \geq 0} f^{(n)}(d)$.

This problem was essentially Theorem 16 from page 64 of Winskel in disguise.

We must prove that d' exists. But since $d \sqsubseteq f(d)$, we have by monotonicity that $f(d) \sqsubseteq f(f(d))$ and $f(f(d)) \sqsubseteq f(f(f(d)))$, ... so, $d, f(d), f^{(2)}(d), \ldots$ is an ascending chain, and therefore has an lub d' in the cpo D.

We then need to prove that d' has the following two properties:

- $d \sqsubseteq_D d'$
- *d' is a fixed point of f.* viz. $f(d') = d'$.

To argue $d \sqsubseteq_D d'$. We recall that a fundamental property of $\bigsqcup$ is:

$$\forall n \geq 0. f^{(n)}(d) \sqsubseteq_D \bigsqcup_{n \geq 0} f^{(n)}(d)$$

So specifically, $d = f^{(0)}(d) \sqsubseteq d'$.

We argue that d' is a fixed point as follows:

$$
\begin{aligned}
f(d') &= f(\bigsqcup_{n\geq 0} f^{(n)}(d)) && \text{(by definition of } d'\text{)}\\
&= \bigsqcup_{n\geq 0} f(f^{(n)}(d)) && \text{(by continuity of } f\text{)}\\
&= \bigsqcup_{n\geq 0} f^{(n+1)}(d) && \text{(by definition of } f^{(n)}\text{)}\\
&= \bigsqcup_{n>0} f^{(n)}(d) && \text{(by change of } n+1 \text{ to } n\text{)}\\
&= \bigsqcup_{n>0} f^{(n+1)}(d) \sqcup \{d\} && \text{(since } d \sqsubseteq f(d) = f^{(1)}(d)\text{)}\\
&= \bigsqcup_{n>0} f^{(n)}(d) \sqcup \{f^{(0)}(d)\} && \text{(since } d = f^{(0)}(d)\text{)}\\
&= \bigsqcup_{n\geq 0} f^{(n)}(d) \\
&= d' && \text{(by definition of } d'\text{)}
\end{aligned}
$$

Last Year's Quiz #3, and Solutions

The original exam had one appendix, giving the syntax and the definition the "evaluates to" relation $\rightarrow_r$ for the language $\mathbf{IMP}_r$.

Problem 1 [20 points].

1(a) [5 points]. Define a formula $DIVIDES \in \mathbf{Assn}$ with free integer variables i, j, which means "j divides i," that is, we require that:

$$\sigma \models^I DIVIDES \quad \text{iff} \quad I(j) \text{ divides } I(i).$$

Solution: $\exists k. j \times k = i$

1(b) [7 points]. Define a formula $PRIME \in \mathbf{Assn}$ with free integer variable i, which means "i is a prime." You may assume the result of problem 1(a).

Hint: A prime is a number larger than 1, whose only divisor greater than 1 is itself.

Solution: $2 \leq i \wedge \forall j.((2 \leq j \wedge DIVIDES) \Rightarrow j = i)$

1(c) [8 points]. There is a while-invariant of the form

$$n_1 \times i + n_2 \times Y + n_3 \times X = 0$$

appropriate for a Hoare logic proof of the partial correctness assertion:

$$\{X = 0 \wedge 2 \times Y = i\}c\{X = i\}$$

where c is the command:

$$\begin{array}{l} \textbf{while } (Y = 0) \textbf{ do} \\ \quad Y := Y - 1; \\ \quad X := X + 1; \\ \quad X := X + 1 \end{array}$$

What are the values of n_1, n_2, and n_3 (Partial credit may be awarded, please show your work)?

Solution: The expected invariant is $i = 2 \times Y + X$, so $n_1 = -1$, $n_2 = 2$ and $n_3 = 1$ (or any multiples thereof).

Problem 2 [15 points].

2(a) [5 points]. State the definition of $\sigma \models^I \{A\}c\{B\}$.

Solution: if $\sigma \models^I A$, and if $\mathcal{C}[\![c]\!]\sigma$ is defined, then $\mathcal{C}[\![c]\!]\sigma \models^I B$.

A *weakest precondition* of an assertion B under a command c is any logical formula, W, such that $\sigma \models^I W$ iff

$$\text{if } \mathcal{C}[\![c]\!]\sigma \text{ is defined, then } \mathcal{C}[\![c]\!]\sigma \models^I B.$$

Note that, by definition, all weakest preconditions of B under c are equivalent logical formulas.

2(b) [10 points]. Exhibit an $A \in$ **Assn** such that A is a weakest precondition of B under c where:

$$c \equiv \textbf{if}\, X \le Y \times Y \,\textbf{then}\, Y := Y + Y \,\textbf{else}\,\textbf{skip}$$
$$B \equiv Y \times Y \le Z + 1$$

Solution: The best way to solve this problem was to chug through the definition of the formula $W(c, B)$ given in class. So:

$$\begin{aligned} W(c,B) &= (W(Y := Y+Y, B) \wedge X \le Y \times Y) \\ &= \vee(W(\textbf{skip}, B) \wedge \neg(X \le Y \times Y)) \\ W(Y := Y+Y, B) &= B[Y+Y/Y] \quad \equiv \quad (Y+Y) \times (Y+Y) \le Z+1 \\ W(\textbf{skip}, B) &= B \end{aligned}$$

and so anything logically equivalent to:

$$((Y+Y)\times(Y+Y) \le Z+1 \wedge X \le Y \times Y) \vee (Y \times Y \le Z+1) \wedge \neg(X \le Y \times Y))$$

is acceptable.

Problem 3 [20 points]. Although primality is easy to express with an arithmetic first-order formula, other familiar number-theoretic functions, *e.g.*, exponentiation, are not so straightforwardly expressible as **Assn**'s. But our study of expressiveness implies that exponentiation and indeed every function which can be computed by, or even "checked" by, an **IMP** command, is expressible by **Assn**'s.

More precisely, we shall say that a binary relation, R, on numbers is called **IMP**-*checkable* iff there is an **IMP** command which halts when run on precisely those states σ for which $R(\sigma(X_1), \sigma(X_2))$.

3(a) [8 points]. Explain why the relation $R(n,m)$ defined by $(n = 2^m)$ is **IMP**-checkable.

Because the following **IMP** command c halts when run on precisely those states σ for which $R(\sigma(X_1), \sigma(X_2))$. The key to an acceptable solution is a convincing argument that such a command does exist. Clearly exhibiting one will do the job.

$X_3 := 1;$
while $1 \le X_2$ **do**
  $X_3 := X_3 \times 2;$
  $X_2 := X_2 - 1;$
if $X_1 = X_3$ **then skip else(while true do skip)**

3(b) [12 points]. Show that for any **IMP**-checkable relation R, there is an $A_R \in$ **Assn**, such that

$$\sigma \models^I A_R \text{ iff } R(\sigma(X_1), \sigma(X_2))$$

Hint: Expressiveness.

Solution:

$$A_R ::= \neg W(c_R, \mathbf{false})$$

will have the required properties, where c_R is a command which checks R.

A weakest precondition of **false** under c_R, will be true in precisely those states σ in which $R(\sigma(X_1), \sigma(X_2))$ *does not* hold, the negation of $W_r(c_R, \mathbf{false})$ is satisfied by the desired set of states.

The next problems concern a Hoare logic for the language $\mathbf{IMP}_r$, obtained by extending **IMP** with a **resultis** construct, as in Quiz 2. Recall that $\mathbf{IMP}_r$ evaluation contrasts with **IMP** evaluation because $\mathbf{IMP}_r$ expressions have side effects and so return *both* states as well as values.

This is sufficient to determine the denotational semantics, since:

$$\begin{array}{lll} & \langle a, \sigma\rangle \to_r \langle n, \sigma'\rangle & \text{iff} \quad \mathcal{A}[\![a]\!]\sigma = (n, \sigma') \\ \text{and} & \langle c, \sigma\rangle \to_r \sigma' & \text{iff} \quad \mathcal{C}[\![a]\!]\sigma = \sigma' \end{array}$$

and similarly for $\mathbf{Bexp}_r$'s.

As for ordinary Hoare logic, we will need to prove the expressiveness of **Assn** for $\mathbf{IMP}_r$. (We will NOT change the definition of **Assn**! It is precisely as it was for **IMP**; so there are no commands embedded within **Assn**'s.) To do this we will need a notion of weakest precondition for *expressions*, referring to both the value and the state after evaluation. We define a ***weakest precondition*** for a number n and assertion B, with respect to an expression $a \in \mathbf{Aexp_r}$, to be a logical formula W which means "if a successfully evaluates, then its value is n and the final state satisfies B." More formally we have $\sigma \models^I W$ iff

$$\mathcal{A}[\![a]\!]\sigma = (n, \sigma') \quad \Rightarrow \quad (I(i) = n \ \&\ \sigma' \models^I B), \text{ for all } n \in \mathbf{Num}, \sigma' \in \Sigma.$$

We can define **Assn**'s $W_r(a, i, B)$ expressing weakest preconditions for $\mathbf{Aexp_r}$'s and likewise **Assn**'s $W_r(c, B)$ for commands (and similarly for $\mathbf{Bexp_r}$'s, which we omit) by structural induction simultaneously on expressions and commands.

For example, some cases in the definition of $W_r(a, i, B)$ and $W_r(c, B) \in \mathbf{Assn}$ are:

$$\begin{aligned} W_r(n, i, B) &::= (n = i) \wedge B \\ W_r(a_1 + a_2, i, B) &::= \exists i_1 . \exists i_2 . (i = i_1 + i_2) \wedge W_r(a_1, i_1, (W_r(a_2, i_2, B))) \\ &\qquad \text{where } i_1 \text{ and } i_2 \text{ are "fresh."} \\ W_r(\mathbf{skip}, B) &::= B \end{aligned}$$

Problem 4 [25 points]. Supply definitions for the following cases, assuming by structural induction the existence of **Assn**'s $W_r(\ldots)$ for subexpressions and subcommands:

4(a) [5 points]. $W_r(X, i, B)$

Solution: $i = X \wedge B$

4(b) [10 points]. $W_r(c\, \mathbf{resultis}\, a, i, B)$

Solution: $W_r(c, W_r(a, i, B))$

4(c) [10 points]. $W_r(X := a, B)$

Hint: A straightforward version is of the form $Qi.W_r(a, i, B[\cdot/\cdot])$, where Q is one of $\forall$ or $\exists$, and of course the dots need to be filled in.

Solution: $\exists i.W_r(a, i, B[i/X])$, where i is "fresh."

Problem 5 [20 points]. In $\mathbf{IMP}_r$, all of Hoare logic is essentially embodied in the assignment axiom because c and $X := c\,\mathbf{resultis}\;X$ are equivalent commands. So we content ourselves with defining a Hoare logic just for $\mathbf{IMP}_r$ assignment statements.

We observed in the previous problem that for any $\mathbf{IMP}_r$ command c and $B \in$ **Assn**, there is a formula $W_r(c, B) \in$ **Assn** which is a weakest precondition. (The present problem does *not* depend on the correctness of your answer to the Problem 4(c).) The provability relation, $\vdash_r$, of the logic is determined by the following two rules:

Rule for assignments:

$$\{W_r(X := a, B)\}X := a\{B\}$$

Rule of consequence:

$$\frac{\models (A \Rightarrow A') \quad \{A'\}c\{B'\} \quad \models (B' \Rightarrow B)}{\{A\}c\{B\}}$$

Show that $\vdash_r$ is complete for partial correctness assertions about assignments. In other words, show that

$$\models \{A\}X := a\{B\} \quad \Rightarrow \quad \vdash_r \{A\}X := a\{B\}.$$

Hint: You may use the fact that the assertion $A \Rightarrow W_r(c, B)$ is equivalent to the partial correctness assertion $\{A\}c\{B\}$.

Solution: By the assignment rule:

$$\vdash_r \{W_r(X := a, B)\}X := a\{B\}$$

By assumption, $\models \{A\}X := a\{B\}$. Since $\{A\}X := a\{B\}$, and $A \Rightarrow W_r(X := a, B)$ are equivalent, it is also the case that $\models A \Rightarrow W_r(X := a, B)$. Obviously, $\models B \Rightarrow B$, so by the rule of consequence (with $c ::= X := a$, $A' ::= W_r(X := a, B)$, and $B' ::= B$):

$$\vdash_r \{A\}X := a\{B\}$$

Problem Set 5 Solutions

Problem 1. Let $C = \Sigma \rightharpoonup \Sigma$ be the "command meanings," namely, partial functions from Σ to Σ. For $f \in C$, let

$$\mathbf{graph}(f) = \{(\sigma, \sigma') \mid f(\sigma) = \sigma'\}.$$

Partially order C by the rule that $f_1 \leq_C f_2$ iff $\mathbf{graph}(f_1) \subseteq \mathbf{graph}(f_2)$.

Prove that C is a cpo.

We will prove that, for any ω-chain $f_0, f_1, \ldots$, the union $G = \bigcup_i \text{graph}(f_i)$ is the graph of a function f, and f is the least upper bound of the chain $\{f_i\}$.

Assume G is not the graph of a function. Then, there are $(\sigma, \sigma') \in G$ and $(\sigma, \sigma'') \in G$ with $\sigma' \neq \sigma''$. Since G is the union of the graphs of the functions $\{f_i\}$, there must be f_j and f_k such that $(\sigma, \sigma') \in \text{graph}(f_j)$ and $(\sigma, \sigma'') \in \text{graph}(f_k)$. Since f_j and f_k are members of the same chain, they must be ordered. Without loss of generality, assume $f_j \leq_C f_k$. But then, $\text{graph}(f_j) \subseteq \text{graph}(f_k)$, so both (σ, σ') and (σ, σ'') are in $\text{graph}(f_k)$, contradicting the fact that f_k is a function.

Thus, G is the graph of some function g. Now, by definition, $f_i \leq_C g$ for all i, so g is an upper bound on $\{f_i\}$. Suppose there is another h such that $f_i \leq_C h$ for all i; that is, $\text{graph}(f_i) \subseteq \text{graph}(h)$ for all i. Then, $G = \bigcup_i \text{graph}(f_i) \subseteq \text{graph}(h)$, so $g \leq_C h$ and g is a least upper bound of $\{f_i\}$.

Thus, all ω-chains have least upper bounds, so C is a cpo.

Problem 2. Let $\Sigma_\perp$ be the set of states, Σ, along with a "fresh" object, $\perp$, called the "bottom state." Partially order $\Sigma_\perp$ by the rule that

$$\sigma_1 \ \leq_{\Sigma_\perp} \ \sigma_2 \text{ iff } [(\sigma_1 = \perp) \text{ or } (\sigma_1 = \sigma_2)].$$

Let S be the set of *total* functions $g : \Sigma_\perp \to \Sigma_\perp$ such that $g(\perp) = \perp$ (such functions are called "strict"). Partially order S by the rule that $g_1 \leq_S g_2$ iff $g_1(\sigma) \leq_{\Sigma_\perp} g_2(\sigma)$ for all $\sigma \in \Sigma_\perp$.

2(a) Describe a *bijection* taking any $f \in C$ to an $f^s \in S$ such that

$$f_1 \leq_C f_2 \text{ iff } f_1^s \leq_S f_2^s.$$

Let $H : C \rightarrow S$ be defined by $H(f) = f^s$, where

$$f^s(\sigma) = \begin{cases} f(\sigma) & \text{if } \sigma \neq \perp \text{ and } f(\sigma) \text{ is defined} \\ \perp & \text{otherwise} \end{cases}$$

To see that H is injective, assume there are $f, g \in C$ with $H(f) = H(g)$. Then, for all $\sigma \in \Sigma_\perp$, $H(f)(\sigma) = H(g)(\sigma)$. In particular, this is true for all $\sigma \neq \perp$; but by the definition of H, this means that f and g agree on all $\sigma \in \Sigma$, so $f = g$. To see that H is surjective, pick any $s \in S$. Now, take the function $f \in C$ such that $f(\sigma) = s(\sigma)$ if $s(\sigma) \neq \perp$, with f undefined otherwise. Now, $H(f) = s$, so, since every $s \in S$ has such an f, H is surjective. Thus, H is a bijection.

Further, if $f \leq_C g$, then, whenever $f(\sigma) = \sigma'$, we have $g(\sigma) = \sigma'$. By the definition of H, then, $H(f) \leq_S H(g)$. Conversely, if $H(f) \leq_S H(g)$, then, if $f(\sigma) = \sigma'(\neq \perp)$, $H(f)(\sigma) = \sigma'$, so $H(g)(\sigma) = \sigma'$, so, by the definition of H, it must be that $g(\sigma) = \sigma'$, so $f \leq_C g$.

2(b) Conclude that S is a cpo.

Let H be as above. Since H is a bijection, it has an inverse, H^{-1}, which, by 2(a) is also order-preserving (monotonic). Now, any nondecreasing ω-chain in S has a least upper bound: consider such an ω-chain $f_0^s \leq_S f_1^s \leq_S \cdots$ in S. Since H^{-1} is monotonic, the $H^{-1}(f_i^s)$'s form a nondecreasing ω-chain in C. Since C is a cpo, the chain has a least upper bound, say g. Since $H^{-1}(f_i^s) \leq_C g$ for all i, and h is monotonic, $f_i^s = H \circ H^{-1}(f_i^s) \leq_S H(g)$ for all i, so $\{f_i^s\}$ has an upper bound. Similarly, suppose there is some other upper bound, $g_1^s \in S$. Then, $H^{-1}(g_1^s)$ is an upper bound on the $H^{-1}(f_i^s)$'s, so $g \leq_C H^{-1}(g_1^s)$, since g is a least upper bound in C, so $H(g) \leq_S H \circ H^{-1}(g_1^s) = g_1^s$, so $H(g)$ is a least upper bound on $\{f_i^s\}$ in S.

Since all nondecreasing ω-chains have least upper bounds, S is a cpo.

Problem 3. In the answers to the this and the next problem, we will introduce the following convention for referring to predicates. Say that, as in class, we define a predicate

$$PRIME(p) \equiv (\neg(p \leq 1) \wedge \forall i_0.\forall j_0.((i_0 \times j_0 = p \wedge \neg(i_0 \leq 1)) \Rightarrow j_0 = 1))$$

If we then, in the course of defining another predicate, refer to $PRIME(k)$, this will be taken to mean the above predicate, *with k substituted for p.* (To avoid scoping problems, we make sure to choose k to be an integer variable different from i and j.) We will also allow k to be an integer, *e.g.* $PRIME(5)$.

3(a) Write a formula $POWP \in$ Assn with free variables p, i which means "p is prime and i is a power of p".

Given $PRIME(p)$ as above, we can define $POWP(p, i)$ to be true when p is prime, $i \geq 1$, and every $j \neq 1$ which divides i is in turn a multiple of p.

$$\begin{aligned} POWP(p,i) \equiv\ & (PRIME(p) \wedge (1 \leq i) \wedge \\ & \forall j_1.(\neg(j_1 = 1) \wedge \exists k_1.j_1 \times k_1 = i) \Rightarrow (\exists k_1.p \times k_1 = j_1)) \end{aligned}$$

3(b) Write a formula $LEN \in$ Assn with free variables i, j, p, such that LEN means "p is prime and $j = p^l$ where l is the length of the base p representation of i."

If $i \geq 1$, then $j = p^l$ should be the least power of p greater than i. If $i = 0$, then $l = 1$, so $j = p$.

$$\begin{aligned} LEN(p,i,j) \equiv\ & ((i = 0 \Rightarrow j = p) \wedge \\ & (1 \leq i \Rightarrow (POWP(p,j) \wedge (i + 1 \leq j) \wedge \\ & \quad \forall k_2.(POWP(p,k_2) \wedge (i + 1 \leq k_2) \Rightarrow (j \leq k_2))))) \end{aligned}$$

3(c) Write a formula $CONCAT \in$ Assn with free variables p, i, j, k which means: "p is prime and $(i)_p \cdot (j)_p = (k)_p$."

We want $k = p^l \times i + j$ where $l = \text{length}((j)_p)$.

$$CONCAT(p,i,j,k) \equiv (\exists \ell_0.(LEN(p,j,\ell_0) \wedge k = i \times \ell_0 + j))$$

Problem 4.

4(a) For $k, n \geq 1$, let $s(k, n)$ be the string of numbers

$$0\,1\,k\,0\,2\,k^2 \ldots 0\,n\,k^n.$$

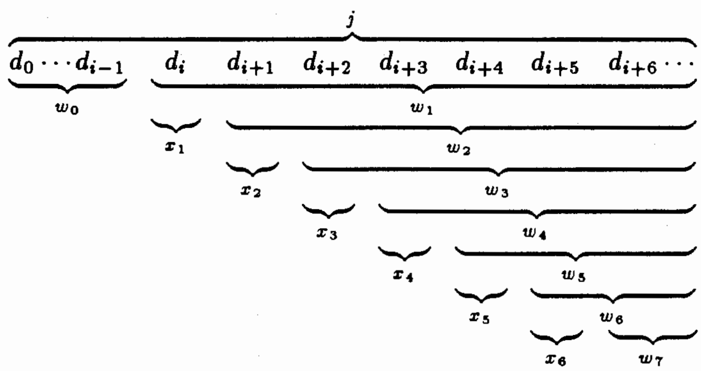
Describe a formula $KNS \in$ Assn with free variables j, k, n which means "$(j)_p = s(k, n)$ for some p."

This problem was made more complicated than necessary by the presence of the leading 0 in $s(k, n)$. Since the eventual goal is simply to represent the sequence $k^1 \ldots k^n$, we will here slightly change the definition of the problem, placing the first 0 at the end. From now on, let $s(k, n)$ be the string of numbers

$$1\,k\,0\,2\,k^2\,0 \ldots n\,k^n\,0$$

Now, an essential idea is that there must be some prime p large enough that every number in the string $s(k, n)$ is less than p, and thus can be represented as a single base-p digit.

We will first define some other useful predicates. We will start by defining $WINDOW(p, j, x_1, x_2, x_3, x_4, x_5, x_6)$ to be true if x_1 through x_6 are consecutive digits in the base-p representation of j. This can be done by using $CONCAT$ to say that j can be split into an initial segment w_0 (which we will then ignore) and a remainder w_1, and then assert that each w_i can be split into x_i and w_{i+1}. Visually, if d_i are the base-p digits of j from left to right, and all the x_i are less than p, then we're splitting j as follows:



$$
\begin{aligned}
WINDOW(p, j, x_1, x_2, x_3, x_4, x_5, x_6) \equiv\ & (\exists w_0.\exists w_1.\exists w_2.\exists w_3.\exists w_4.\exists w_5.\exists w_6.\exists w_7.\\
& CONCAT(p, w_0, w_1, j)\wedge\\
& CONCAT(p, x_1, w_2, w_1)\wedge\\
& CONCAT(p, x_2, w_3, w_2)\wedge\\
& CONCAT(p, x_3, w_4, w_3)\wedge\\
& CONCAT(p, x_4, w_5, w_4)\wedge\\
& CONCAT(p, x_5, w_6, w_5)\wedge\\
& CONCAT(p, x_6, w_7, w_6))
\end{aligned}
$$

Now, using $WINDOW$, we can write an intermediate predicate $CONSISTENT$, which checks that, if the string of base-p digits $ia0$ appears in the base-p representation of j, then it is followed by $(i+1)(a \times k)0$

$$
\begin{aligned}
& CONSISTENT(p, j, i, k) \equiv\\
& \quad (\forall y_0.\forall y_1.\forall y_2.\forall y_3.(WINDOW(p, j, i, y_0, 0, y_1, y_2, y_3)\wedge\\
& \qquad (0 \le y_0 \wedge y_0 \le p-1) \wedge (0 \le y_1 \wedge y_1 \le p-1)\wedge\\
& \qquad (0 \le y_2 \wedge y_2 \le p-1) \wedge (0 \le y_3 \wedge y_3 \le p-1))\\
& \qquad \Rightarrow (y_1 = i+1 \wedge y_2 = y_0 \times k \wedge y_3 = 0))
\end{aligned}
$$

Predicates (with free variables p, j, k and p, j, k, n, w) to check that k will fit in a single base-p digit and the beginning and end of j are in the form $(0)_p \cdot (1)_p \cdot (k)_p$

and $(0)_p \cdot (n)_p \cdot (w)_p$ is

$$\begin{aligned}
START(p,j,k) \equiv (k \le p-1) \wedge \ & (\exists z_0.\exists z_1.\exists z_2.CONCAT(p,1,z_0,j) \wedge \\
& \quad CONCAT(p,k,z_1,z_0) \wedge \\
& \quad CONCAT(p,0,z_2,z_1)) \\
FINISH(p,j,n,w) \equiv \ & (\exists z_0.\exists z_1.\exists z_2.CONCAT(p,z_0,z_1,j) \wedge \\
& \quad CONCAT(p,n,z_2,z_1) \wedge \\
& \quad CONCAT(p,w,0,z_2))
\end{aligned}$$

A final intermediate predicate, which we'll need for the next subproblem, is one that, given p, checks the beginning, end, and each intermediate step of $(j)_p$.

$$\begin{aligned}
KNS'(p,j,k,n) \equiv \ & (\exists w.(1 \le w \wedge w \le p-1 \wedge \\
& \quad START(p,j,k) \wedge FINISH(p,j,n,w)) \wedge \\
& \forall i_1.(1 \le i_1 \wedge i_1 \le n-1) \Rightarrow CONSISTENT(p,j,i_1,k))
\end{aligned}$$

Then,

$$KNS(j,k,n) \equiv \exists p.KNS'(p,j,k,n).$$

4(b) Describe a formula $POW \in$ Assn with free variables i, k, n which means "$i = k^n$."

We need only say that there are a p and a j such that $(j)_p$ is equal to $s(k,n)$ and the second-to-last digit of $(j)_p$ is i.

$$POW(i,k,n) \equiv (\exists p.\exists j.KNS'(p,j,k,n) \wedge FINISH(p,j,n,i))$$

Quiz 2

Instructions. This is a **closed book** exam; no notes either. There are eight (8) problems worth 10–20 points each as indicated on pages 2–3 of this booklet. Following the quiz, there are several appendices containing selected definitions cited in the problems.

Write your solutions for all problems in the examination books provided, including your *name on each examination book*. Be sure to indicate *clearly* in your examination books which answer is associated with which problem. Ask for further books if you need them.

GOOD LUCK!

Problem 1 [20 points]. Explain why it follows directly from the definition of denotational semantics of **IMP** (repeated in Appendix A), and the equivalence of natural evaluation and denotational semantics, that

1(a) [10 points].

$$\text{if } c_1 \sim c_1' \text{ and } c_2 \sim c_2',\ \text{then } (c_1; c_2) \sim (c_1'; c_2'),$$

1(b) [10 points]. and that

$$\text{if } c \sim c',\ \text{then } \mathbf{while}\, b\, \mathbf{do}\, c \sim \mathbf{while}\, b\, \mathbf{do}\, c'.$$

Problem 2 [15 points]. Let A, B, C be cpo's, and $f : B \to C$, $g : A \to B$ be continuous total functions. Prove that $f \circ g : A \to C$ is continuous.

Problem 3 [10 points]. We define an extension $\mathbf{IMP}_{\mathsf{val}}$ of the language **IMP** by adding a new **Aexp** construct "$c\, \mathbf{valis}\, a$" which is a side-effect free version of the construct $c\, \mathbf{resultis}\, a$ considered previously. The natural evaluation semantics for **valis** is given by

$$\frac{\langle c, \sigma\rangle \to \sigma', \langle a, \sigma'\rangle \to n}{\langle c\, \mathbf{valis}\, a, \sigma\rangle \to n}$$

The clauses defining the denotational semantics of the commands and expressions other than **valis** are the same for $\mathbf{IMP}_{\mathsf{val}}$ as for **IMP** (except that, because expressions no longer always terminate, the set A of **Aexp** meanings becomes the *partial* functions $\Sigma \rightharpoonup \mathbf{Num}$ instead of the total functions $\Sigma \to \mathbf{Num}$; likewise for $B = \Sigma \rightharpoonup \mathbf{T}$).

Write the remaining denotational semantics clause for $\mathcal{A}[\![c\, \mathbf{valis}\, a]\!]$.

Problem 4 [10 points]. Write an **Assn** with free integer variables i and j which means "j is greater than twice the absolute value of i."

Hint: Remember that ">" is not a primitive connective in **Assn**. (The grammar of **Assn** is in Appendix B.)

Problem 5 [15 points]. In Problem Set 5, problems 3 and 4, it was shown how to construct an **Assn**, $POW(i, k, n)$ which meant "$k, n \geq 1$ and $i = k^n$." The solutions to those problems are included in Appendix E. A small modification of one formula, *CONSISTENT*, used in the construction will change meanings so that $POW(i, 1, n)$ means "$n \geq 1$ and $i = n!$," where $n! = n \times (n-1) \times \cdots \times 1$. Describe that modification.

Problem 6 [10 points]. Exhibit **Assn**'s A and B such that the partial correctness assertion $\{A\}c\{B\}$ means "c diverges when X is even." (The semantics of partial correctness assertions are included in Appendix D.)

Problem 7 [10 points]. Exhibit a simple $A \in$ **Assn** such that $A \Rightarrow \forall j.A$ is not valid. (No proof or explanation required.) The semantics of **Assn** are included in Appendix C.

Problem 8 [10 points]. Prove that for any $A \in$ **Assn**, if A is valid, then so is $\forall j.A$.

A Denotational Semantics of IMP

$$
\begin{aligned}
&\mathcal{A}[\![n]\!]\sigma = n\\
&\mathcal{A}[\![X]\!]\sigma = \sigma(X)\\
&\mathcal{A}[\![a_0 \ \mathbf{op}\ a_1]\!]\sigma = (\mathcal{A}[\![a_0]\!]\sigma)\ op\ (\mathcal{A}[\![a_1]\!]\sigma)\\
&\mathcal{B}[\![\mathbf{true}]\!]\sigma = \mathbf{true}\\
&\mathcal{B}[\![\mathbf{false}]\!]\sigma = \mathbf{false}\\
&\mathcal{B}[\![\neg b]\!]\sigma = \neg(\mathcal{B}[\![b]\!]\sigma)\\
&\mathcal{B}[\![a_0 \ \mathbf{op}\ a_1]\!]\sigma = (\mathcal{A}[\![a_0]\!]\sigma)\ op\ (\mathcal{A}[\![a_1]\!]\sigma)\\
&\mathcal{B}[\![b_0 \ \mathbf{op}\ b_1]\!]\sigma = (\mathcal{B}[\![b_0]\!]\sigma)\ op\ (\mathcal{B}[\![b_1]\!]\sigma)\\
&\mathcal{C}[\![\mathbf{skip}]\!]\sigma = \sigma\\
&\mathcal{C}[\![X := a]\!]\sigma = \sigma[n/X] \text{ where } n = \mathcal{A}[\![a]\!]\sigma\\
&\mathcal{C}[\![c_0; c_1]\!] = \mathcal{C}[\![c_1]\!] \circ \mathcal{C}[\![c_0]\!]\\
&\mathcal{C}[\![\mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1]\!]\sigma = \begin{cases} \mathcal{C}[\![c_0]\!]\sigma & \text{if } \mathcal{B}[\![b]\!]\sigma = \mathbf{true}\\ \mathcal{C}[\![c_1]\!]\sigma & \text{if } \mathcal{B}[\![b]\!]\sigma = \mathbf{false}\end{cases}\\
&\mathcal{C}[\![\mathbf{while}\, b\, \mathbf{do}\, c]\!] = \mathit{fix}(\Gamma_{bc}) \text{ where}\\
&\qquad \Gamma_{bc}(\varphi)(\sigma) = \begin{cases} (\varphi \circ \mathcal{C}[\![c]\!])\sigma & \text{if } \mathcal{B}[\![b]\!]\sigma = \mathbf{true}\\ \sigma & \text{if } \mathcal{B}[\![b]\!]\sigma = \mathbf{false}\end{cases}
\end{aligned}
$$

B Grammar of Assn

Let **Aexpv** be defined by

$$a ::= n \mid X \mid i \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$$

Then **Assn** is the set of expressions given by

$$
\begin{aligned}
A ::= &\ \mathbf{true} \mid \mathbf{false} \mid a_0 = a_1 \mid a_0 \le a_1\\
&\mid A_0 \wedge A_1 \mid A_0 \vee A_1 \mid \neg A \mid A_0 \Rightarrow A_1\\
&\mid \forall i.A \mid \exists i.A
\end{aligned}
$$

where a_0 and a_1 range over expressions in **Aexpv**.

C Semantics of Aexpv and Assn

Semantics of **Aexpv**:

$$\begin{aligned}
&\mathcal{A}v[\![n]\!]I\sigma = n\\
&\mathcal{A}v[\![X]\!]I\sigma = \sigma(X)\\
&\mathcal{A}v[\![i]\!]I\sigma = I(i)\\
&\mathcal{A}v[\![a_0 \textbf{ op } a_1]\!]\sigma = (\mathcal{A}v[\![a_0]\!]I\sigma)\ op\ (\mathcal{A}v[\![a_1]\!]I\sigma)
\end{aligned}$$

The satisfaction relation, $\models$, is defined as follows for **Assn**. For any $\sigma \in \Sigma_\perp$, and interpretation I:

$$\begin{aligned}
&\sigma \models^I \textbf{true}\\
&\sigma \models^I (a_0 = a_1) \text{ if } \mathcal{A}v[\![a_0]\!]I\sigma = \mathcal{A}v[\![a_1]\!]I\sigma,\\
&\sigma \models^I (a_0 \leq a_1) \text{ if } \mathcal{A}v[\![a_0]\!]I\sigma \leq \mathcal{A}v[\![a_1]\!]I\sigma,\\
&\sigma \models^I a_0 \wedge a_1 \text{ if } \sigma \models^I A \text{ and } \sigma \models^I B,\\
&\sigma \models^I a_0 \vee a_1 \text{ if } \sigma \models^I A \text{ or } \sigma \models^I B,\\
&\sigma \models^I \neg A \text{ if } \sigma \not\models^I A,\\
&\sigma \models^I a_0 \Rightarrow a_1 \text{ if } \sigma \not\models^I A \text{ or } \sigma \models^I B,\\
&\sigma \models^I \forall i.A \text{ if } \sigma \models^{I[n/i]} A \text{ for all } n \in \mathbf{N},\\
&\sigma \models^I \forall i.A \text{ if } \sigma \models^{I[n/i]} A \text{ for some } n \in \mathbf{N},\\
&\perp \models^I A
\end{aligned}$$

D Partial Correctness Assertions

The satisfaction relation for partial correctness assertions is defined by

$$\sigma \models^I \{A\}c\{B\} \quad \text{iff} \quad \sigma \models^I A \Rightarrow \mathcal{C}[\![c]\!]\sigma \models^I B.$$

An partial correctness assertion $\{A\}c\{B\}$ is valid ($\models \{A\}c\{B\}$) if

$$\forall I.\forall\sigma \in \Sigma_\perp.\ \sigma \models^I \{A\}c\{B\}$$

E Solution to PS 5, questions 3 and 4

For $n \geq 0, p \geq 2$ we write $(n)_p$ to denote the string of digits (between 0 and $p-1$) representing n in base p notation, and we use $\cdot$ to denote the concatenation of strings. For example:

$$\begin{aligned}
(5)_3 &= \text{"12"} \\
(21)_3 &= \text{"210"} \\
(5)_3 \cdot (21)_3 &= \text{"12210"} \\
&= (1 \times 3^4 + 2 \times 3^3 + 2 \times 3^2 + 1 \times 3^1 + 0 \times 3^0)_3 \\
&= (156)_3
\end{aligned}$$

Problem 3. In the answers to the this and the next problem, we will introduce the following convention for referring to predicates. Say that, as in class, we define a predicate

$$PRIME(p) \equiv (\neg(p \leq 1) \wedge \forall i_0.\forall j_0.((i_0 \times j_0 = p \wedge \neg(i_0 \leq 1)) \Rightarrow j_0 = 1))$$

If we then, in the course of defining another predicate, refer to $PRIME(k)$, this will be taken to mean the above predicate, ***with k substituted for p***. (To avoid scoping problems, we make sure to choose k to be an integer variable different from i and j.) We will also allow k to be an integer, *e.g.* $PRIME(5)$.

3(a) Write a formula $POWP \in$ Assn with free variables p, i which means "p is prime and i is a power of p".

Given $PRIME(p)$ as above, we can define $POWP(p,i)$ to be true when p is prime, $i \geq 1$, and every $j \neq 1$ which divides i is in turn a multiple of p.

$$\begin{aligned}
POWP(p,i) \equiv\ & (PRIME(p) \wedge (1 \leq i) \wedge \\
& \forall j_1.(\neg(j_1 = 1) \wedge \exists k_1.j_1 \times k_1 = i) \Rightarrow (\exists k_1.p \times k_1 = j_1))
\end{aligned}$$

3(b) Write a formula $LEN \in$ Assn with free variables i, j, p, such that LEN means "p is prime and $j = p^l$ where l is the length of the base p representation of i."

If $i \geq 1$, then $j = p^l$ should be the least power of p greater than i. If $i = 0$, then $l = 1$, so $j = p$.

$$\begin{aligned}
LEN(p,i,j) \equiv\ & ((i = 0 \Rightarrow j = p) \wedge \\
& (1 \leq i \Rightarrow (POWP(p,j) \wedge (i+1 \leq j) \wedge \\
& \qquad \forall k_2.(POWP(p,k_2) \wedge (i+1 \leq k_2) \Rightarrow (j \leq k_2)))))
\end{aligned}$$

3(c) Write a formula $CONCAT \in$ Assn with free variables p, i, j, k which means: "p is prime and $(i)_p \cdot (j)_p = (k)_p$."

We want $k = p^l \times i + j$ where $l = \text{length}((j)_p)$.

$$CONCAT(p, i, j, k) \equiv (\exists \ell_0.(LEN(p, j, \ell_0) \wedge k = i \times \ell_0 + j))$$

Problem 4.

4(a) For $k, n \geq 1$, let $s(k, n)$ be the string of numbers

$$0\,1\,k\,0\,2\,k^2 \ldots 0\,n\,k^n.$$

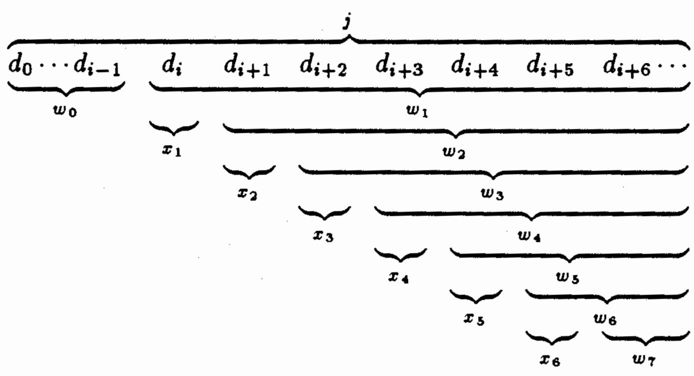
Describe a formula $KNS \in$ Assn with free variables j, k, n which means "$(j)_p = s(k, n)$ for some p."

This problem was made more complicated than necessary by the presence of the leading 0 in $s(k, n)$. Since the eventual goal is simply to represent the sequence $k^1 \ldots k^n$, we will here slightly change the definition of the problem, placing the first 0 at the end. From now on, let $s(k, n)$ be the string of numbers

$$1\,k\,0\,2\,k^2\,0 \ldots n\,k^n\,0$$

Now, an essential idea is that there must be some prime p large enough that every number in the string $s(k, n)$ is less than p, and thus can be represented as a single base-p digit.

We will first define some other useful predicates. We will start by defining $WINDOW(p, j, x_1, x_2, x_3, x_4, x_5, x_6)$ to be true if x_1 through x_6 are consecutive digits in the base-p representation of j. This can be done by using $CONCAT$ to say that j can be split into an initial segment w_0 (which we will then ignore) and a remainder w_1, and then assert that each w_i can be split into x_i and w_{i+1}. Visually, if d_i are the base-p digits of j from left to right, and all the x_i are less than p, then we're splitting j as follows:



$$
\begin{aligned}
WINDOW(p,j,x_1,x_2,x_3,x_4,x_5,x_6) \equiv \; & (\exists w_0.\exists w_1.\exists w_2.\exists w_3.\exists w_4.\exists w_5.\exists w_6.\exists w_7. \\
& \quad CONCAT(p,w_0,w_1,j)\wedge \\
& \quad CONCAT(p,x_1,w_2,w_1)\wedge \\
& \quad CONCAT(p,x_2,w_3,w_2)\wedge \\
& \quad CONCAT(p,x_3,w_4,w_3)\wedge \\
& \quad CONCAT(p,x_4,w_5,w_4)\wedge \\
& \quad CONCAT(p,x_5,w_6,w_5)\wedge \\
& \quad CONCAT(p,x_6,w_7,w_6))
\end{aligned}
$$

Now, using *WINDOW*, we can write an intermediate predicate *CONSISTENT*, which checks that, if the string of base-p digits $ia0$ appears in the base-p representation of j, then it is followed by $(i+1)(a \times k)0$

$$
\begin{aligned}
& CONSISTENT(p,j,i,k) \equiv \\
& \quad (\forall y_0.\forall y_1.\forall y_2.\forall y_3.(\,WINDOW(p,j,i,y_0,0,y_1,y_2,y_3)\wedge \\
& \qquad (0 \le y_0 \wedge y_0 \le p-1) \wedge (0 \le y_1 \wedge y_1 \le p-1)\wedge \\
& \qquad (0 \le y_2 \wedge y_2 \le p-1) \wedge (0 \le y_3 \wedge y_3 \le p-1)) \\
& \qquad \Rightarrow (y_1 = i+1 \wedge y_2 = y_0 \times k \wedge y_3 = 0))
\end{aligned}
$$

Predicates (with free variables p,j,k and p,j,k,n,w) to check that k will fit in a single base-p digit and the beginning and end of j are in the form $(0)_p \cdot (1)_p \cdot (k)_p$ and $(0)_p \cdot (n)_p \cdot (w)_p$ is

$$
\begin{aligned}
START(p,j,k) \equiv (k \le p-1) \wedge \; & (\exists z_0.\exists z_1.\exists z_2.CONCAT(p,1,z_0,j)\wedge \\
& \quad CONCAT(p,k,z_1,z_0)\wedge \\
& \quad CONCAT(p,0,z_2,z_1)) \\
FINISH(p,j,n,w) \equiv \; & (\exists z_0.\exists z_1.\exists z_2.CONCAT(p,z_0,z_1,j)\wedge \\
& \quad CONCAT(p,n,z_2,z_1)\wedge \\
& \quad CONCAT(p,w,0,z_2))
\end{aligned}
$$

A final intermediate predicate, which we'll need for the next subproblem, is one that, given p, checks the beginning, end, and each intermediate step of $(j)_p$.

$$
\begin{aligned}
KNS'(p,j,k,n) \equiv \; & (\exists w.(1 \le w \wedge w \le p-1 \wedge \\
& \quad START(p,j,k) \wedge FINISH(p,j,n,w))\wedge \\
& \forall i_1.(1 \le i_1 \wedge i_1 \le n-1) \Rightarrow CONSISTENT(p,j,i_1,k))
\end{aligned}
$$

Then,

$$KNS(j,k,n) \equiv \exists p.KNS'(p,j,k,n).$$

4(b) Describe a formula $POW \in$ Assn with free variables i, k, n which means "$i = k^n$."

We need only say that there are a p and a j such that $(j)_p$ is equal to $s(k, n)$ and the second-to-last digit of $(j)_p$ is i.

$$POW(i, k, n) \equiv (\exists p.\exists j.KNS'(p, j, k, n) \wedge FINISH(p, j, n, i))$$

Problem Set 6

Due: 4 November 1992

Reading assignment. Winskel §7.1–7.3

Problem 1. Prove that for $A, B \in \mathbf{Assn}$, if A is equivalent to B, then

1(a) $A \wedge C$ is equivalent to $B \wedge C$ for any $C \in \mathbf{Assn}$.

1(b) $\exists j.A$ is equivalent to $\exists j.B$ for any $j \in \mathbf{Intvar}$.

1(c) $A[a/j]$ is equivalent to $B[a/j]$ for any $a \in \mathbf{Aexp}$, $j \in \mathbf{Intvar}$.

Problem 2.

2(a) Prove $\exists j.(A \vee B)$ is equivalent to $(\exists j.A) \vee (\exists j.B)$.

2(b) Prove $\exists j.(A \Rightarrow B)$ is equivalent to $(\forall j.A) \Rightarrow (\exists j.B)$.

2(c) Prove $\exists j.(A \wedge B)$ is equivalent to $A \vee (\exists j.B)$ whenever $j \notin \mathrm{FV}(A)$.

Problem 3. Winskel Exer. 6.13.

Problem 4. Winskel Exer. 6.14.

Problem 5. Winskel Exer. 6.17.

Grade Statistics for Quiz 2

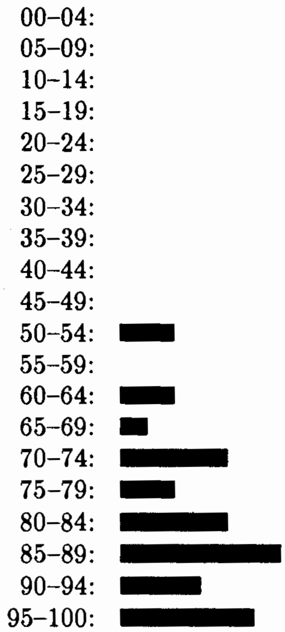
Number of quizzes taken: 29

Grade range: 51-100

Mean: 81

Median: 79

Histogram:



Quiz 2 Solutions

Instructions. This was a **closed book** exam; no notes either. Following the quiz, there were several appendices containing selected definitions cited in the problems.

Problem 1 [20 points]. **Explain why it follows directly from the definition of denotational semantics of IMP, and the equivalence of natural evaluation and denotational semantics, that**

1(a) [10 points].

$$\textbf{if } c_1 \sim c_1' \textbf{ and } c_2 \sim c_2', \textbf{ then } (c_1; c_2) \sim (c_1'; c_2'),$$

According to the definitions,

$$\mathcal{C}[\![c_1; c_2]\!] = \mathcal{C}[\![c_2]\!] \circ \mathcal{C}[\![c_1]\!]$$

and

$$\mathcal{C}[\![c_1'; c_2']\!] = \mathcal{C}[\![c_2']\!] \circ \mathcal{C}[\![c_1']\!].$$

Because the natural evaluation and denotational semantics are equivalent

$$c_1 \sim c_1' \Rightarrow \mathcal{C}[\![c_1]\!] = \mathcal{C}[\![c_1']\!]$$

and

$$c_2 \sim c_2' \Rightarrow \mathcal{C}[\![c_2]\!] = \mathcal{C}[\![c_2']\!],$$

so

$$\mathcal{C}[\![c_2]\!] \circ \mathcal{C}[\![c_1]\!] = \mathcal{C}[\![c_2']\!] \circ \mathcal{C}[\![c_1']\!].$$

Thus,

$$\mathcal{C}[\![c_1; c_2]\!] = \mathcal{C}[\![c_1'; c_2']\!]$$

and

$$c_1; c_2 \sim c_1'; c_2'$$

1(b) [10 points]. **and that**

$$\textbf{if } c \sim c', \textbf{ then } \mathbf{while}\, b\, \mathbf{do}\, c \sim \mathbf{while}\, b\, \mathbf{do}\, c'.$$

As above, $c \sim c' \Rightarrow \mathcal{C}[\![c]\!] = \mathcal{C}[\![c']\!]$. Thus, since

$$\Gamma_{bc}(\varphi)(\sigma) = \begin{cases} (\varphi \circ \mathcal{C}[\![c]\!])\sigma & \text{if } \mathcal{B}[\![b]\!]\sigma = \mathbf{true} \\ \sigma & \text{if } \mathcal{B}[\![b]\!]\sigma = \mathbf{false} \end{cases}$$

we know that $\Gamma_{bc} = \Gamma_{bc'}$, so

$$\mathit{fix}(\Gamma_{bc}) = \mathit{fix}(\Gamma_{bc'}),$$

so

$$\mathcal{C}[\![\mathbf{while}\, b\, \mathbf{do}\, c]\!] = \mathcal{C}[\![\mathbf{while}\, b\, \mathbf{do}\, c']\!]$$

by definition, so

$$\mathbf{while}\, b\, \mathbf{do}\, c \sim \mathbf{while}\, b\, \mathbf{do}\, c'.$$

Problem 2 [15 points]. **Let A, B, C be cpo's, and $f : B \to C$, $g : A \to B$ be continuous total functions. Prove that $f \circ g : A \to C$ is continuous.**

First, since g and f are monotonic, if $a \leq a'$ (for $a, a' \in A$) then $g(a) \leq g(a')$, so $f(g(a)) \leq f(g(a'))$, so $f \circ g$ is monotonic.

Now, take any nondecreasing ω-chain $a_0, a_1, \ldots$ in A. Since g is continuous,

$$\bigsqcup_i g(a_i) = g(\bigsqcup_i a_i).$$

But, since g is monotonic, $\{g(a_i)\}$ is also a nondecreasing ω-chain in B, so, since f is continuous,

$$\bigsqcup_i f(g(a_i)) = f(\bigsqcup_i g(a_i))$$

so, putting these together,

$$\bigsqcup_i f(g(a_i)) = f(g(\bigsqcup_i a_i)).$$

so $f \circ g$ is continuous.

Problem 3 [10 points]. **We define an extension $\mathbf{IMP}_{\text{val}}$ of the language IMP by adding a new Aexp construct "c valis a" which is a side-effect free version of the construct c resultis a considered previously. The natural evaluation semantics for valis is given by**

$$\frac{\langle c, \sigma\rangle \rightarrow \sigma', \langle a, \sigma'\rangle \rightarrow n}{\langle c \,\mathbf{valis}\, a, \sigma\rangle \rightarrow n}$$

The clauses defining the denotational semantics of the commands and expressions other than valis are the same for $\mathbf{IMP}_{\text{val}}$ as for IMP (except that, because expressions no longer always terminate, the set A of Aexp meanings becomes the *partial* functions $\Sigma \rightharpoonup \mathrm{Num}$ instead of the total functions $\Sigma \rightarrow \mathrm{Num}$; likewise for $B = \Sigma \rightharpoonup \mathbf{T}$).

Write the remaining denotational semantics clause for $\mathcal{A}[\![c \,\mathbf{valis}\, a]\!]$.

$$\mathcal{A}[\![c \,\mathbf{valis}\, a]\!] = \mathcal{A}[\![a]\!] \circ \mathcal{C}[\![c]\!].$$

Problem 4 [10 points]. **Write an Assn with free integer variables i and j which means "j is greater than twice the absolute value of i."**

The simplest answer took advantage of the fact that $i^2 = |i|^2$:

$$0 \leq j \wedge \neg(j \times j \leq 4 \times i \times i)$$

Another common solution was to present something along the lines of

$$(0 \leq i \wedge \neg(j \leq 2 \times i)) \vee (i \leq 0 \wedge \neg(j \leq -2 \times i))$$

or

$$(0 \leq i \Rightarrow \neg(j \leq 2 \times i)) \wedge (i \leq 0 \Rightarrow \neg(j \leq -2 \times i)).$$

Problem 5 [15 points]. **In Problem Set 5, problems 3 and 4, it was shown how to construct an Assn, $POW(i, k, n)$ which meant "$k, n \geq 1$ and $i = k^n$." The solutions to those problems are included in Appendix E. A small modification of one formula, $CONSISTENT$, used in the construction will change meanings so that $POW(i, 1, n)$ means "$n \geq 1$ and $i = n!$," where $n! = n \times (n-1) \times \cdots \times 1$. Describe that modification.**

The modification corresponds to changing the sequence $s(k, n)$ (where $k = 1$) from

$$1\, k\, 0\, 2\, k^2 \ldots n\, k^n\, 0$$

to

$$1\, k\, 0\, 2\, (k \times 2) \ldots n\, (k \times 2 \times \cdots n)\, 0.$$

A "consistent window" now looks like

$$i\, y_0\, 0\, (i+1)\, (y_0 \times (i+1))\, 0,$$

so *CONSISTENT* becomes

$$\begin{aligned}
&CONSISTENT(p, j, i, k) \equiv \\
&\quad (\forall y_0.\forall y_1.\forall y_2.\forall y_3.(\,WINDOW(p, j, i, y_0, 0, y_1, y_2, y_3) \wedge \\
&\qquad\qquad (0 \leq y_0 \wedge y_0 \leq p-1) \wedge (0 \leq y_1 \wedge y_1 \leq p-1) \wedge \\
&\qquad\qquad (0 \leq y_2 \wedge y_2 \leq p-1) \wedge (0 \leq y_3 \wedge y_3 \leq p-1)) \\
&\qquad\qquad \Rightarrow (y_1 = i+1 \wedge y_2 = y_0 \times y_1 \wedge y_3 = 0))
\end{aligned}$$

Problem 6 [10 points]. **Exhibit Assn's A and B such that the partial correctness assertion $\{A\}c\{B\}$ means "c diverges when X is even."**

This statement is equivalent to, "If c is executed in a state σ such that $\sigma(X)$ is even, then there is no state σ' such that $\mathcal{C}[\![c]\!]\sigma = \sigma'$." Thus, a correct solution is

$$A = \exists i.X = 2 \times i$$
$$B = \textbf{false}$$

Problem 7 [10 points]. **Exhibit a simple $A \in$ Assn such that $A \Rightarrow \forall j.A$ is not valid. (No proof or explanation required.)**

$$A \equiv \ (j = 0)$$

Explanation: for $A \Rightarrow \forall j.A$ not to be valid, there must be some state σ and interpretation I such that $\sigma \not\models^I A \Rightarrow \forall j.A$, which means that $\sigma \models^I A$ and $\sigma \not\models^I \forall j.A$. A simple example of an appropriate A is given above, since, while there are σ and I such that $\sigma \models^I A$, none of them give us $\sigma \models^I \forall i.j = 0$.

Problem 8 [10 points]. **Prove that for any $A \in$ Assn, if A is valid, then so is $\forall j.A$.**

Assume that A is valid. Then, for all states σ and interpretations I, $\sigma \models^I A$. In particular, for all interpretations of the form $I[n/j]$, with $n \in \textbf{Num}$, $\sigma \models^{I[n/j]} A$. Put another way, for all states σ and interpretations I, we have

$$\sigma \models^{I[n/j]} A \qquad \text{for all } n \in \textbf{Num}$$

so, by the definition of $\models$, for all σ and I,

$$\sigma \models^I \forall j.A.$$

The combination of this question and the preceding one seems, at first glance, to be a contradiction. Note, however, that the solution to problem 7 presented an assertion A that was *not valid*. The A above was true in some interpretations and false in others. Thus, the existence of such an A does not contradict the lemma here, which only deals with valid assertions.

E Solution to PS 5, questions 3 and 4

For $n \geq 0, p \geq 2$ we write $(n)_p$ to denote the string of digits (between 0 and $p-1$) representing n in base p notation, and we use $\cdot$ to denote the concatenation of strings. For example:

$$\begin{aligned}
(5)_3 &= \text{“12”} \\
(21)_3 &= \text{“210”} \\
(5)_3 \cdot (21)_3 &= \text{“12210”} \\
&= (1 \times 3^4 + 2 \times 3^3 + 2 \times 3^2 + 1 \times 3^1 + 0 \times 3^0)_3 \\
&= (156)_3
\end{aligned}$$

Problem 3. In the answers to the this and the next problem, we will introduce the following convention for referring to predicates. Say that, as in class, we define a predicate

$$PRIME(p) \equiv (\neg(p \leq 1) \wedge \forall i_0.\forall j_0.((i_0 \times j_0 = p \wedge \neg(i_0 \leq 1)) \Rightarrow j_0 = 1))$$

If we then, in the course of defining another predicate, refer to $PRIME(k)$, this will be taken to mean the above predicate, *with k substituted for p*. (To avoid scoping problems, we make sure to choose k to be an integer variable different from i and j.) We will also allow k to be an integer, *e.g.* $PRIME(5)$.

3(a) Write a formula $POWP \in$ Assn with free variables p, i which means "p is prime and i is a power of p".

Given $PRIME(p)$ as above, we can define $POWP(p,i)$ to be true when p is prime, $i \geq 1$, and every $j \neq 1$ which divides i is in turn a multiple of p.

$$\begin{aligned}
POWP(p,i) \equiv \;& (PRIME(p) \wedge (1 \leq i) \wedge \\
& \forall j_1.(\neg(j_1 = 1) \wedge \exists k_1.j_1 \times k_1 = i) \Rightarrow (\exists k_1.p \times k_1 = j_1))
\end{aligned}$$

3(b) Write a formula $LEN \in$ Assn with free variables i, j, p, such that LEN means "p is prime and $j = p^l$ where l is the length of the base p representation of i."

If $i \geq 1$, then $j = p^l$ should be the least power of p greater than i. If $i = 0$, then $l = 1$, so $j = p$.

$$\begin{aligned}
LEN(p,i,j) \equiv \;& ((i = 0 \Rightarrow j = p) \wedge \\
& (1 \leq i \Rightarrow (POWP(p,j) \wedge (i + 1 \leq j) \wedge \\
& \quad \forall k_2.(POWP(p,k_2) \wedge (i + 1 \leq k_2) \Rightarrow (j \leq k_2)))))
\end{aligned}$$

3(c) Write a formula $CONCAT \in$ Assn with free variables p, i, j, k which means: "p is prime and $(i)_p \cdot (j)_p = (k)_p$."

We want $k = p^l \times i + j$ where $l = \text{length}((j)_p)$.

$$CONCAT(p, i, j, k) \equiv (\exists \ell_0 . (LEN(p, j, \ell_0) \wedge k = i \times \ell_0 + j))$$

Problem 4.

4(a) For $k, n \geq 1$, let $s(k, n)$ be the string of numbers

$$0\,1\,k\,0\,2\,k^2 \ldots 0\,n\,k^n.$$

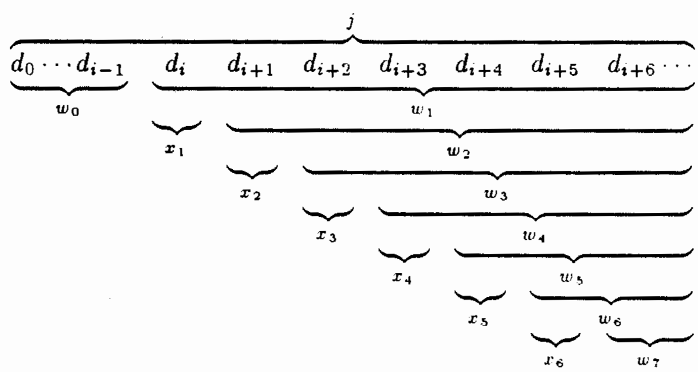
Describe a formula $KNS \in$ Assn with free variables j, k, n which means "$(j)_p = s(k, n)$ for some p."

This problem was made more complicated than necessary by the presence of the leading 0 in $s(k, n)$. Since the eventual goal is simply to represent the sequence $k^1 \ldots k^n$, we will here slightly change the definition of the problem, placing the first 0 at the end. From now on, let $s(k, n)$ be the string of numbers

$$1\,k\,0\,2\,k^2\,0 \ldots n\,k^n\,0$$

Now, an essential idea is that there must be some prime p large enough that every number in the string $s(k, n)$ is less than p, and thus can be represented as a single base-p digit.

We will first define some other useful predicates. We will start by defining $WINDOW(p, j, x_1, x_2, x_3, x_4, x_5, x_6)$ to be true if x_1 through x_6 are consecutive digits in the base-p representation of j. This can be done by using $CONCAT$ to say that j can be split into an initial segment w_0 (which we will then ignore) and a remainder w_1, and then assert that each w_i can be split into x_i and w_{i+1}. Visually, if d_i are the base-p digits of j from left to right, and all the x_i are less than p, then we're splitting j as follows:



$$
\begin{aligned}
WINDOW(p,j,x_1,x_2,x_3,x_4,x_5,x_6) \equiv\ & (\exists w_0.\exists w_1.\exists w_2.\exists w_3.\exists w_4.\exists w_5.\exists w_6.\exists w_7.\\
& CONCAT(p,w_0,w_1,j)\wedge\\
& CONCAT(p,x_1,w_2,w_1)\wedge\\
& CONCAT(p,x_2,w_3,w_2)\wedge\\
& CONCAT(p,x_3,w_4,w_3)\wedge\\
& CONCAT(p,x_4,w_5,w_4)\wedge\\
& CONCAT(p,x_5,w_6,w_5)\wedge\\
& CONCAT(p,x_6,w_7,w_6))
\end{aligned}
$$

Now, using $WINDOW$, we can write an intermediate predicate $CONSISTENT$, which checks that, if the string of base-p digits $ia0$ appears in the base-p representation of j, then it is followed by $(i+1)(a \times k)0$

$$
\begin{aligned}
& CONSISTENT(p,j,i,k) \equiv\\
& \quad (\forall y_0.\forall y_1.\forall y_2.\forall y_3.(\ WINDOW(p,j,i,y_0,0,y_1,y_2,y_3)\wedge\\
& \qquad (0 \le y_0 \wedge y_0 \le p-1) \wedge (0 \le y_1 \wedge y_1 \le p-1)\wedge\\
& \qquad (0 \le y_2 \wedge y_2 \le p-1) \wedge (0 \le y_3 \wedge y_3 \le p-1))\\
& \qquad \Rightarrow (y_1 = i+1 \wedge y_2 = y_0 \times k \wedge y_3 = 0))
\end{aligned}
$$

Predicates (with free variables p, j, k and p, j, k, n, w) to check that k will fit in a single base-p digit and the beginning and end of j are in the form $(0)_p \cdot (1)_p \cdot (k)_p$ and $(0)_p \cdot (n)_p \cdot (w)_p$ is

$$
\begin{aligned}
START(p,j,k) \equiv (k \le p-1) \wedge\ & (\exists z_0.\exists z_1.\exists z_2.CONCAT(p,1,z_0,j)\wedge\\
& CONCAT(p,k,z_1,z_0)\wedge\\
& CONCAT(p,0,z_2,z_1))\\
FINISH(p,j,n,w) \equiv\ & (\exists z_0.\exists z_1.\exists z_2.CONCAT(p,z_0,z_1,j)\wedge\\
& CONCAT(p,n,z_2,z_1)\wedge\\
& CONCAT(p,w,0,z_2))
\end{aligned}
$$

A final intermediate predicate, which we'll need for the next subproblem, is one that, given p, checks the beginning, end, and each intermediate step of $(j)_p$.

$$
\begin{aligned}
KNS'(p,j,k,n) \equiv\ & (\exists w.(1 \le w \wedge w \le p-1\wedge\\
& \quad START(p,j,k) \wedge FINISH(p,j,n,w))\wedge\\
& \forall i_1.(1 \le i_1 \wedge i_1 \le n-1) \Rightarrow CONSISTENT(p,j,i_1,k))
\end{aligned}
$$

Then,

$$KNS(j,k,n) \equiv \exists p.KNS'(p,j,k,n).$$

4(b) Describe a formula $POW \in$ Assn with free variables i, k, n which means "$i = k^n$."

We need only say that there are a p and a j such that $(j)_p$ is equal to $s(k, n)$ and the second-to-last digit of $(j)_p$ is i.

$$POW(i, k, n) \equiv (\exists p.\exists j. KNS'(p, j, k, n) \wedge FINISH(p, j, n, i))$$

Problem Set 7

Due: 11 November 1992

Reading assignment. Winskel, Appendix on Computability.

A *sublogic* of **Assn** is a set of **Assn**'s that is closed under Boolean combination, quantification, and substituting numbers or integer variables for integer variables or locations. For certain sublogics (an example will appear soon), there is a computational "quantifier elimination" procedure that will transform any **Assn**, A, in the sublogic into an equivalent *quantifier-free* **Assn**, $\widehat{A}$, in the sublogic.

Problem 1.

1(a) If a sublogic has a quantifier-elimination procedure, it can be used to develop procedures to decide

(i) given a formula A of the sublogic, and the values of $\sigma(X)$ for $X \in \mathrm{loc}(A)$ and $I(j)$ for $j \in \mathrm{FV}(A)$, whether $\sigma \models^I A$,

(ii) whether or not a formula of the sublogic is valid, and

(iii) whether or not two formulas of the sublogic are equivalent.

Describe procedures for (i), (ii) and (iii) assuming quantifier-elimination.

Hint: Use the fact that the quantifier-free **Assn**'s without free integer variables, are exactly the **Bexp**'s, so natural and/or one-step operational semantics provide procedures for calculating their truth value in any state.

1(b) Suppose for some sublogic that there is an "innermost existential-elimination" procedure that, for any integer variable j and *quantifier-free* **Assn**, A, in the sublogic, constructs another quantifier-free formula, $\mathcal{E}(j, A)$, in the sublogic, such that $\mathcal{E}(j, A)$ is equivalent to $\exists j.A$. Using innermost existential-elimination as a subprocedure, there is a straightforward recursive definition of a quantifier-elimination procedure for the sublogic. Describe it.

Hint: $\forall j.A$ is equivalent to $\neg\exists j.\neg A$.

Define *incremental arithmetic expressions*, **IncAexpv**, to be **Aexpv**'s without the multiplication or subtraction operations and with addition restricted to incrementing by an integer. Namely, the grammar for $a \in$ **IncAexpv** is

$$A ::= n \mid i \mid X \mid a + n \mid n + a$$

Let **IncAssn** be the set of **Assn**'s all of whose arithmetic subexpressions are **IncAexpv**'s. Note that **IncAssn** is a sublogic of **Assn**.

Define a *simple* incremental assertion to be a finite conjunction of assertions of one of the three forms

$$n \leq v, \; v \leq n, \; v_1 + n \leq v_2$$

where $n \in \mathbf{Num}$, $v, v_1, v_2 \in \mathbf{Loc} \cup \mathbf{Intvar}$, and v_1 and v_2 are not the same.

Problem 2.

2(a) Describe a special case of an innermost existential-elimination procedure which works just for simple assertions. That is, if A is a simple assertion, then so will be $\mathcal{E}(j, A)$.

Hint:

$$\exists j.(k + 1 \leq j \; \wedge \; i + (-3) \leq j \; \wedge \; j + 2 \leq i \; \wedge j \leq -4)$$

is equivalent to

$$\exists j.(k + 1 \leq j \; \wedge \; i - 3 \leq j \; \wedge \; j \leq i - 2 \; \wedge j \leq -4)$$

which is in turn equivalent to

$$k + 1 \leq i - 2 \; \wedge \; k + 1 \leq -4 \; \wedge \; i - 3 \leq i - 2 \; \wedge \; i - 3 \leq -4,$$

which is equivalent to the simple assertion

$$k + 3 \leq i \; \wedge \; k \leq -5 \; \wedge \; i \leq -1.$$

2(b) Describe a procedure that will transform any quantifier-free assertion $A \in$ **IncAssn** into an equivalent formula $\mathcal{S}(A)$ that is a finite disjunction of simple assertions.

Hint: $a_1 = a_2$ is equivalent to $(a_1 \leq a_2 \; \wedge \; a_2 \leq a_1)$. $\neg(a_1 \leq a_2)$ is equivalent to $(a_2 + (-1) \leq a_1)$. Any propositional expression is equivalent to a sum of products of propositional variables and their negations.

2(c) Explain how to combine the procedures of problems 2(a) and 2(b) to obtain an innermost existential-elimination procedure for every quantifier-free **IncAssn**, not just simple ones.

Hint: $\exists j.(A \vee B)$ is equivalent to $(\exists j.A) \vee (\exists j.B)$.

Problem 3. The preceding problems provide a computational procedure for determining validity and equivalence of **IncAssn**'s. The procedure is total—it is guaranteed to return the correct answer on every **IncAssn**—though there are several stages where huge computations may be needed. Describe briefly how this procedure could be programmed as a procedure definition F, in, say, Scheme, so *e.g.*, $(F\,\tilde{A})$ evaluates to `t` or `nil` according to whether or not $A \in$ **IncAssn** is valid, where $\tilde{A}$ is some straightforward representation of A as an S-expression. Indicate which stages or subprocedures of the computation may be the source of time-consuming (quadratic, exponential, ..., growth) subcomputations.

Optional Problem. Describe a complete axiom system for **IncAssn**'s. There should be a finite set of simple axiom schemes and rules whose soundness is obvious, and in which the only kind of assertions used are **IncAssn**'s.

Problem 4. Show that there is no **IncAssn** which means "j is even." Conclude that **IncAssn**'s are ***not*** expressive for the set of **IMP** commands whose arithmetic subexpressions are restricted to be **IncAexp**'s.

Hint: By the preceding problems, it is enough to show that no disjunction of simple assertions can mean "j is even."

Problem 5. Define **ResetCom** to be the set of **IMP** commands whose **Aexp**'s are restricted to be of the form $n \in$ **Num** or $X \in$ **Loc**; define **ResetAssn**'s likewise.

5(a) Show that every **IncAssn** is equivalent to a **ResetAssn**.

Let **diverge** be **while true do skip**. An **IMP** command which has no occurrences of whileloops other than **diverge** is said to be *while-free*. We state the following

Lemma. If $c \in$ **ResetCom**, then $c \sim c'$, for some while-free $c' \in$ **ResetCom**.

5(b) Assuming the Lemma, show that **ResetAssn** is expressive for **ResetCom**.

5(c) Conclude that the restriction of Hoare logic to **ResetAssn**'s and **ResetCom**'s is a complete proof system *whose proofs are computationally checkable.*

5(d) Optional. Prove the Lemma.

Hint: : Say $\sigma \asymp_L \sigma'$ iff σ and σ' satisfy exactly the same **ResetBexp**'s whose locations and numbers are in the set L. If $\sigma \asymp_L \sigma'$ and $X, Y, n \in L$, then $\mathcal{C}[\![X := Y]\!]\sigma \asymp_L \mathcal{C}[\![X := Y]\!]\sigma'$ and $\mathcal{C}[\![X := n]\!]\sigma \asymp_L \mathcal{C}[\![X := n]\!]\sigma'$. Thus, if $c \in$ **ResetCom**, L is the set of locations and numbers occuring in c, and $\sigma \asymp_L \sigma'$, then c will "execute in the same way" on σ and σ'. Namely, for any $c \in$ **ResetCom** and $\sigma \in \Sigma$, there is a $c'' \in$ **ResetCom** consisting solely of a finite sequence of assignment statements such that $\mathcal{C}[\![c]\!]\sigma' = \mathcal{C}[\![c'']\!]\sigma'$ for all $\sigma' \asymp_{\mathsf{loc}(c)\cup\mathsf{num}(c)} \sigma$. Now use the observation that if L is finite, then there are only finitely many $\asymp_L$ equivalence classes of states.

Notes on Expressiveness

The set, **DynAssn**, of "dynamic assertions" generalize both **Assn**'s and partial correctness assertions. The grammar for $D \in \mathbf{DynAssn}$ is

$$D ::= \quad a_1 = a_2 \mid a_1 \leq a_2 \mid \neg D \mid D_1 \wedge D_2 \mid D_1 \vee D_2 \mid D_1 \Rightarrow D_2 \mid$$

$$\mid \exists j.D \mid \forall j.D \mid \{D_1\}c\{D_2\}$$

Definition of $\sigma \models^I D$ is as for **Assn** and partial correctness assertions. We then have

$$\{D_1\}c\{D_2\} \quad \text{is equivalent to} \quad D_1 \Rightarrow \{\mathbf{true}\}c\{D_2\},$$

so

$$\models \{\ \{\mathbf{true}\}c\{D_2\}\ \}c\{D_2\}.$$

Let W be $\{\mathbf{true}\}c\{D\}$. Informally, W means "after doing c, the property D will hold." Thus, $\{W\}c\{D\}$ is valid, and if $\{D_1\}c\{D\}$ is valid, then D_1 implies W. For this reason, any formula equivalent to W is called a *weakest precondition of D under c*.

Some other useful equivalences:

$$\{\mathbf{true}\}\mathbf{if}\, b\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2\{D\} \ \text{ equiv } \ (b \Rightarrow \{\mathbf{true}\}c_1\{D\}) \wedge (\neg b \Rightarrow \{\mathbf{true}\}c_2\{D\}),$$

$$\{\mathbf{true}\}(c_1; c_2)\{D\} \ \text{ equiv } \ \{\mathbf{true}\}c_1\{\ \{\mathbf{true}\}c_2\{D\}\ \},$$

$$\{\mathbf{true}\}X := a\{B\} \ \text{ equiv } \ B[a/X] \qquad \text{for } B \in \mathbf{Assn}.$$

Note that B above must not be a **DynAssn** containing commands, since we have not defined substitution into such formulas (which is hard to do properly, because a location on the lefthand side of an assignment statement behaves more like a bound, than a free, identifier).

We will prove

Theorem 1 (Expressiveness). For all $c \in \mathbf{Com}$, $A \in \mathbf{Assn}$, there is a formula $W(c, A) \in \mathbf{Assn}$ such that $W(c, A)$ is a weakest precondition of A under c.

Corollary 1. There is a translation mapping any $D \in \mathbf{DynAssn}$ into an equivalent $\widehat{D} \in \mathbf{Assn}$.

Proof of corollary.

$$\begin{array}{rcl} \widehat{a_1 = a_2} & \text{is} & a_1 = a_2 \\ \widehat{D_1 \vee D_2} & \text{is} & \widehat{D_1} \wedge \widehat{D_2} \\ \widehat{\exists j.D} & \text{is} & \exists j.\widehat{D} \\ \widehat{\{D_1\}c\{D_2\}} & \text{is} & \widehat{D_1} \Rightarrow W(c, \widehat{D_2}) \end{array}$$

The other cases are similar. ■

Proof of Expressiveness Theorem. By induction on c:

$$\begin{aligned} W(\mathbf{skip}, A) &::= A, \\ W(X := a, A) &::= A[a/X], \\ W(\mathbf{if}\, b\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2, A) &::= (b \Rightarrow W(c_1, A)) \wedge (\neg b \Rightarrow W(c_2, A)), \\ W((c_1; c_2), A) &::= W(c_1, W(c_2, A)). \end{aligned}$$

That $W(c, A)$ is equivalent to $\{\mathbf{true}\}c\{A\}$ in each case above follows from the equivalences between **DynAssn**'s which we have already noted.

The remaining case $W(w, A)$ where w is $\mathbf{while}\, b\, \mathbf{do}\, c$ is fairly elaborate. It will actually be a bit more convenient, and without loss of generality, to express the input-output relation on states corresponding to w instead of its weakest precondition, as we now explain.

Let c' be an arbitrary command, and for notational convenience, say that $\text{loc}(c') = \{X_1, X_2\}$. We know that c' depends only on the values of X_1, X_2 in a state, so in this setting we can identify a state, σ, with the pair of numbers $\langle\sigma(X_1), \sigma(X_2)\rangle$. Let $\text{IO}_{c'}(i_1, i_2, j_1, j_2)$ be a formula with free variables i_1, i_2, j_1, j_2 which means $\mathcal{C}[\![c']\!]\langle i_1, i_2\rangle = \langle j_1, j_2\rangle$.

If $\text{IO}_{c'}(i_1, i_2, j_1, j_2) \in \mathbf{Assn}$, then we can define $W(c', A) \in \mathbf{Assn}$ to be

$$\forall i_1, i_2, j_1, j_2.(X_1 = i_1 \wedge X_2 = i_2 \wedge \text{IO}_{c'}(i_1, i_2, j_1, j_2)) \Rightarrow A[j_1/X_1, j_2/X_2]$$

Conversely, from **Assn**'s which are weakest preconditions for c' we can define $\text{IO}_{c'}(i_1, i_2, j_1, j_2) \in \mathbf{Assn}$ to be

$$(\ W(c', j_1 = X_1 \wedge j_2 = X_2) \wedge \neg W(c', \mathbf{false})\ )[i_1/X_1, i_2/X_2]$$

Note that $W(c', j_1 = X_1 \wedge j_2 = X_2)$ just means that *if* c' terminates, it does so in state $\langle j_1, j_2\rangle$; we need the other conjunct $\neg W(c', \mathbf{false})$ to assert that c' does terminate.

It will also be convenient to have a one-to-one coding of pairs of numbers into positive numbers. One way to do this is to define the one-to-one function $\text{mkpair} : (\mathbf{Num} \times \mathbf{Num}) \to \omega^+$ by

$$\text{mkpair}(n, m) ::= 2^{|n|} \cdot 3^{\text{sg}(n)} \cdot 5^{|m|} \cdot 7^{\text{sg}(n)}$$

where $|n|$ is the absolute value of n, $\mathrm{sg}(n) = 1$ if $n > 0$, and $\mathrm{sg}(n) = 0$ if $n \leq 0$.

We have seen in Problem Set 5 that $i = k^l$ is the meaning of an **Assn**, so it is easy to see that there is an **Assn** which means "$k = \mathrm{mkpair}(i, j)$." So we may assume for convenience that in addition to the basic arithmetic operators, the binary function symbol mkpair can occur in **Aexpv**'s.

Optional Exercise: Explain how to translate any extended **Assn** using the function symbol mkpair into an equivalent **Assn** without mkpair.

Finally, we can define IO_w using the same "window" idea from Problem Set 5 used to define exponentiation and factorial. Let $(k)_p$ denote the base p representation of k as in Problem Set 5. We say that there is a sequence, $(k)_p$, whose digits are codes of states, *viz.*, pairs of integers. The sequence begins with the code of state $\langle i_1, i_2 \rangle$, ends with $\langle j_1, j_2 \rangle$, and every two consecutive states are in the input-output relation of the body, c, of w. Also, every state but the final one satisfies the guard, b, of w.

Thus we can describe $\mathrm{IO}_w(i_1, i_2, j_1, j_2) \in$ **Assn** as follows:

$\exists k \exists p.$ "$(k)_p$ starts with digit $\mathrm{mkpair}(i_1, i_2)$" $\wedge$
  "$(k)_p$ ends with digit $\mathrm{mkpair}(j_1, j_2)$" $\wedge$
  $(\forall k_1, k_2, l_1, l_2.$
    "$\mathrm{mkpair}(k_1, k_2)$ and $\mathrm{mkpair}(l_1, l_2)$ are consecutive digits of $(k)_p$"
      $\Rightarrow (b[k_1/X_1][k_2/X_2] \wedge \mathrm{IO}_c(k_1, k_1, l_1, l_2)) \wedge$
  $\neg b[j_1/X_1][j_2/X_2]$

■

Review Material for Quiz 3

This handout contains selected problems and solutions from last year that we think will be helpful to you in reviewing for Quiz 3 (which takes place next Monday, November 16).

In addition to the enclosed material, we recommend that you take another look at last year's Quiz 3 (handed out this year as handout 25).

1 From Last Year's Quiz 4

Problem 4 [35 points]. We consider axioms for symmetries (rigid, "in place" transformations) of an equilateral triangle. For example, given the triangle with vertices labeled as in Figure 1, we can apply

Transformation "r": $r$otate the triangle 120° clockwise, obtaining the triangle in Figure 2;

Transformation "f": $f$lip the triangle about the vertical axis, obtaining the triangle in Figure 3;

Transformation "l": $l$eave the triangle unchanged, obtaining the triangle in Figure 3 again.

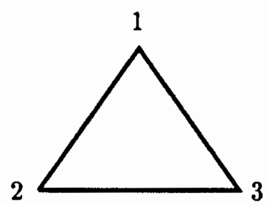


Figure 1: The original triangle.

Let W be the set of finite sequences (of length at least 1) of the letters r, f, and l. Elements of W are called *words* over the alphabet $\{r, f, l\}$.

By interpreting concatenation of letters as composition of permutations, we can associate with any word, w, a permutation, $[\![w]\!]$, of $\{1, 2, 3\}$ indicating the movement of vertices of a triangle. So the basic permutations defined by r and f are:

$$\begin{array}{lll} [\![r]\!](1) = 2, & [\![r]\!](2) = 3, & [\![r]\!](3) = 1. \\ [\![f]\!](1) = 1, & [\![f]\!](2) = 3, & [\![f]\!](3) = 2. \end{array}$$

Note that $[\![l]\!]$ is simply the identity function. Inductively, let $[\![aw]\!] = [\![a]\!] \circ [\![w]\!]$ for $a \in \{f, r, l\}$. For example, $[\![rfrl]\!](x) = r(f(r(l(x))))$, so

$$[\![rfrl]\!](1) = 1, \quad [\![rfrl]\!](2) = 3, \quad [\![rfrl]\!](3) = 2.$$

Define "truth", $\models$, of a "triangle" word equation as follows:

$$\models (w_1 = w_2) \quad \text{iff} \quad [\![w_1]\!] = [\![w_2]\!].$$

For example, $\models rfrl = f$.

4a [3 points]. Exhibit w_1 and w_2, such that

$$\not\models w_1 w_2 = w_2 w_1.$$

Solution: For example, $w_1 = r$ and $w_2 = f$.

4b [7 points]. The "standard" rules for equality are reflexivity, symmetry, transitivity, and congruence. State these rules for the case of word equations.

Solution:

$$\vdash w = w \qquad \text{(reflexivity)}$$

$$\frac{\vdash w_1 = w_2}{w_2 = w_1} \qquad \text{(symmetry)}$$

$$\frac{\vdash w_1 = w_2 \quad \vdash w_2 = w_3}{\vdash w_1 = w_3} \qquad \text{(transitivity)}$$

$$\frac{\vdash w_1 = w_2}{\vdash a w_1 = a w_2} \qquad \text{(left congruence)}$$

where $a \in \{r, f, l\}$

$$\frac{\vdash w_1 = w_2}{\vdash w_1 a = w_2 a} \qquad \text{(right congruence)}$$

where $a \in \{r, f, l\}$

4c [10 points]. Show that if a sound axiom system is strong enough to prove any word equal to one of the six "canonical" forms below, then we can obtain a sound and complete axiom system by adding the standard rules for equality. The six canonical forms are:

$$l, \quad r, \quad rr, \quad f, \quad rf, \quad rrf.$$

Solution: Suppose $\models w_1 = w_2$, *i.e.*, $[\![w_1]\!] = [\![w_2]\!]$. By the presumption, there are canonical forms $\hat{w}_1$ and $\hat{w}_2$ such that $\vdash w_1 = \hat{w}_1$, and $\vdash w_2 = \hat{w}_2$. Since the system is sound, $\models w_i = \hat{w}_i$. $[\![\hat{w}_1]\!] = [\![w_1]\!] = [\![w_2]\!] = [\![\hat{w}_2]\!]$.

In addition, each of the six "canonical" forms have different meanings. So, we have

$$\vdash w_1 = \hat{w}_1 \quad \text{and} \quad \vdash w_2 = \hat{w}_2$$

and by symmetry and transitivity, we conclude $\vdash w_1 = w_2$.

4d [15 points]. Consider the complete proof system for triangle word equations whose rules are just the standard rules for equality plus the axioms:

$$rrr = ff = ll = l \qquad \text{(unit)}$$

$$rl = lr = r, \quad fl = lf = f \qquad \text{(identity)}$$

$$fr = rrf \qquad \text{(swap)}$$

Briefly explain why this proof system is sound and complete. ***Hint***: Show how to prove that an arbitrary word equals one of the six canonical forms of problem 1.

Solution Assuming the result of Problem 1, it should be clear that all we need to do us show, using the above axioms and the rules for equality, that it is possible to prove that any triangle world is equal to one of the six canonical forms.

The following process will halt and reduce an arbitrary word to a canonical form.

Step 1 Erase all l's (unless $w \equiv l$, in which case we are done) This follows from the identity axioms, plus the rules for equality.

Step 2 Move all f's to the right. This is possible from the rules for equality and the swap axiom. So now we have a word containing only r's and f's with all f's on the right.

Step 3 Replace rrr (if it occurs) by l. This is possible from the rules for equality and the unit axiom.

Step 4 Erase all l's (unless $w \equiv l$, in which case we are done)

Step 5 If there is still rrr left in w go to Step 2.

Step 6 Replace ff (if it occurs) by l. This is possible from the rules for equality and the unit axiom.

Step 7 Erase all l's (unless $w \equiv l$, in which case we are done)

Step 8 If there is still ff left in w go to Step 5.

Clearly this will halt, as we are always making the word shorter.

Clearly if it halts it will have f's to the right of r's, if there are any l's left then the result is l. If there are r's left, they all must be adjacent on the left, thus by Steps 3 to 5, there can be no more than two r's. If there are f's left they all must be adjacent on the right, thus by steps 6-8, there can be no more than one f. This paragraph now precisely characterizes the canonical forms.

2 From Last Year's Problem Set 7:

Problem 2. In this problem, you will give a syntactic proof of the result of the preceding problem. Specifically, we want you to show that:

$$\text{if } \mathrm{loc}(B) \cap \mathrm{loc}(c) = \emptyset \text{ then } \vdash_{\text{Hoare}} \{B\}c\{B\}$$

Prove this by structural induction on c. Do so directly from the definition of the Hoare rules and axioms. (Do not appeal to the Completeness Theorem or the result of Problem 1).

Solution. We have one important Lemma:

Lemma 1. If $X \notin \mathrm{loc}(B)$, then $B \equiv B[n/X]$ (note: here we are using $\equiv$ to denote syntactic equality).

(This is proven by first proving (by structural induction) the analogous result for the extended **Aexp** language, and then by an induction on the structure of B.)

We now prove $\vdash_{\text{Hoare}} \{B\}c\{B\}$, by induction on the structure of c, taking cases on the structure of c.

[$c \equiv$ **skip**] Trivial. By the rule for **skip**, $\vdash_{\text{Hoare}} \{B\}\,\textbf{skip}\,\{B\}$.

[$c \equiv X := a$] By the rule for assignments, $\vdash_{\text{Hoare}} \{B[a/X]\}X := a\{B\}$. Since $\text{loc}(B) \cap \text{loc}_L(c) = \emptyset$, $X \notin \text{loc}(B)$. By the Lemma, $B[a/X] \equiv B$, thus

$$\vdash_{\text{Hoare}} \{B\}X := a\{B\}$$

is precisely the same statement as

$$\vdash_{\text{Hoare}} \{B[a/X]\}X := a\{B\},$$

and so we are done.

[$c \equiv c_0; c_1$] Since $\text{loc}_L(c_0) \subseteq \text{loc}_L(c)$, $\text{loc}(B) \cap \text{loc}_L(c_0) = \emptyset$. Thus, we may use induction to say

$$\vdash_{\text{Hoare}} \{B\}c_0\{B\}.$$

A similar argument gives us

$$\vdash_{\text{Hoare}} \{B\}c_1\{B\}.$$

Finally, we may apply the rule for sequencing, to obtain

$$\vdash_{\text{Hoare}} \{B\}c_0; c_1\{B\}.$$

[$c \equiv \textbf{if}\, b\, \textbf{then}\, c_0\, \textbf{else}\, c_1$] Similar uses of induction will give us: $\vdash_{\text{Hoare}} \{B\}c_0\{B\}$, and $\vdash_{\text{Hoare}} \{B\}c_1\{B\}$. Since $B \wedge b \Rightarrow B$ (by the definition of $\wedge$), we may use the rule of consequence to obtain: $\vdash_{\text{Hoare}} \{B \wedge b\}c_0\{B\}$. Similarly we can get $\vdash_{\text{Hoare}} \{B \wedge \neg b\}c_1\{B\}$. Finally, we may apply the rule for conditionals to obtain:

$$\vdash_{\text{Hoare}} \{B\}\textbf{if}\, b\, \textbf{then}\, c_0\, \textbf{else}\, c_1\{B\}.$$

[$c \equiv \textbf{while}\, b\, \textbf{do}\, c'$] Another use of induction will give us $\vdash_{\text{Hoare}} \{B\}c'\{B\}$. Since $B \wedge b \Rightarrow B$, we can use the rule of consequence to obtain

$$\vdash_{\text{Hoare}} \{B \wedge b\}c'\{B\}.$$

We can then apply the rule for while-loops to obtain $\vdash_{\text{Hoare}} \{B\}c\{B \wedge \neg b\}$. Finally, since $B \wedge \neg b \Rightarrow B$, we may use the rule of consequence to obtain:

$$\vdash_{\text{Hoare}} \{B\}c\{B\}.$$

Problem 3. In class we gave the following definition of the **DynAssn** abbreviation for the weakest precondition under c for $d \in$ **DynAssn**:

$$w(c, D) ::= \{\textbf{true}\}c\{D\}$$

Note we used a small "w" in this definition. This is not to be confused with $W(c, B) \in$ **Assn**, which we defined in class for $B \in$ **Assn** (although $w(c, B)$ is equivalent to $W(c, B)$).

Prove from the definition of validity for **DynAssn**:

$$\models w((c_1; c_2), D) \Rightarrow w(c_1, w(c_2, D))$$

(Of course the converse ($\Leftarrow$) also holds, but we thought that it was enough of an exercise to prove the equivalence in one direction)

Solution. So, we must show that:

$$\models \{\mathbf{true}\}c_1; c_2\{D\} \Rightarrow \{\mathbf{true}\}c_1\{\{\mathbf{true}\}c_2\{D\}\}$$

In other words, suppose $\sigma \models \{\mathbf{true}\}c_1; c_2\{D\}$, then we must show that

$$\sigma \models^I \{\mathbf{true}\}c_1\{\{\mathbf{true}\}c_2\{D\}\}.$$

Since $\sigma \models^I \mathbf{true}$, we must show that

$$\mathcal{C}[\![c_1]\!]\sigma \models^I \{\mathbf{true}\}c_2\{D\}.$$

We now have, two cases, either $\mathcal{C}[\![c_1]\!]\sigma$ is undefined ($\perp$), in which case we are done as $\perp \models^I$ *Anything*, or there is a $\sigma'' \in \Sigma$ such that $\mathcal{C}[\![c_1]\!]\sigma = \sigma''$, in which case, since $\sigma'' \models^I \mathbf{true}$, we must show that $\mathcal{C}[\![c_2]\!]\sigma'' \models^I D$. We now again have two cases: $\mathcal{C}[\![c_2]\!]\sigma''$ is undefined (in which case we are done), or $\mathcal{C}[\![c_2]\!]\sigma'' = \sigma' \in \Sigma$, and we must show $\sigma' \models^I D$.

We now use our other premise, $\sigma \models^I \{\mathbf{true}\}c_1; c_2\{D\}$, to show that $\sigma' \models^I D$. Well, since $\sigma \models^I \mathbf{true}$, $\mathcal{C}[\![c_1; c_2]\!]\sigma \models^I D$. But, by the definition of $\mathcal{C}[\![c_1; c_2]\!]$,

$$\mathcal{C}[\![c_1; c_2]\!]\sigma = \mathcal{C}[\![c_2]\!](\mathcal{C}[\![c_1]\!]\sigma) = \mathcal{C}[\![c_2]\!](\sigma'') = \sigma',$$

and so $\sigma' \models^I D$, and we're done!!

3 From last year's PS 8:

Problem 1.

Instructions. In this problem set we used the function $\#(x)$ to be the function which gives us the "Gödel-number" of x. Note, we are not assuming that there is some universal scheme for Gödel-numbering all things which we might wish to Gödel-number. Rather, we will use # in a variety of contexts, each of which might be Gödel-numbering different things. In any case, the intended meaning for # will either be more clearly spelled out, or will not be relevant.

In this problem we consider a new kind of formula, which we call **Bform** (to stand for *boolean formula*). The set of **Bform**'s is built up inductively out of a collection of boolean variables, and the boolean connectives: $\neg, \wedge, \vee$. We will let $P, P_1, P_2, Q, \ldots$ range over **Bform**'s and we let $p, p_1, p_2, q, \ldots$ range over boolean variables. Since **Bform**'s don't have **Loc**'s or **IntVar**'s contained within them, states and our old notion of I's will not be relevant to the semantics of **Bform**'s. Instead, we will use *Boolean Interpretations*, J : boolean variables $\rightarrow \{\mathbf{true}, \mathbf{false}\}$.

We wish to have a collection of equations between **Bform**'s. But first, we define: $\mathcal{B}f[\![\cdot]\!]$: boolean interpretations $\rightarrow \{\mathbf{true}, \mathbf{false}\}$, we do so by a structural induction. Specifically:

$$\mathcal{B}f[\![p]\!]J = J(p)$$

$$\mathcal{B}f[\![\neg p]\!]J = \begin{cases} \mathbf{true} & \text{if } \mathcal{B}f[\![p]\!]J = \mathbf{false} \\ \mathbf{false} & \text{if } \mathcal{B}f[\![p]\!]J = \mathbf{true} \end{cases}$$

$$\mathcal{B}f[\![p_1 \wedge p_2]\!]J = \begin{cases} \mathbf{true} & \text{if } \mathcal{B}f[\![p]\!]J = \mathbf{true} \text{ and } \mathcal{B}f[\![p]\!]J = \mathbf{true} \\ \mathbf{false} & \text{otherwise} \end{cases}$$

$$\mathcal{B}f[\![p_1 \vee p_2]\!]J = \begin{cases} \mathbf{true} & \text{if } \mathcal{B}f[\![p]\!]J = \mathbf{true} \text{ or } \mathcal{B}f[\![p]\!]J = \mathbf{true} \\ \mathbf{false} & \text{otherwise} \end{cases}$$

Our goal is to talk about equalities of the form $P_1 = P_2$, where $P_1, P_2 \in$ **Bform**. Our semantics for such equations is given by:

$$J \models P_1 = P_2 \quad \text{iff} \quad \mathcal{B}f[\![P_1]\!]J = \mathcal{B}f[\![P_2]\!]J$$

We say the equation $P_1 = P_2$ is valid, written $\models P_1 = P_2$, iff $J \models P_1 = P_2$ for all J.

We now have a semantics for equations between **Bform**'s, but we would also like to develop a logic ($\vdash$) for syntactically proving equalities between **Bform**'s. $\vdash$ will have the axiom of reflexivity, and the usual rules for equality:

$$P = P \qquad \text{(reflexivity)}$$

$$\frac{P_1 = P_2}{P_2 = P_1} \qquad \text{(symmetry)}$$

$$\frac{P_1 = P_2 \quad P_2 = P_3}{P_1 = P_3} \qquad \text{(transitivity)}$$

$$\left.\begin{array}{c} \dfrac{P_1 = P_2}{\neg P_1 = \neg P_2} \\ \dfrac{P_1 = P_2}{P_1 \,\mathbf{op}\, P = P_2 \,\mathbf{op}\, P} \\ \dfrac{P_1 = P_2}{P \,\mathbf{op}\, P_1 = P \,\mathbf{op}\, P_2} \end{array}\right\} \text{(congruence)}$$

for **op** $\in \{\vee, \wedge\}$.

We define the substitution of the **Bform** Q in for all occurrences of the boolean variable p, in the **Bform** R (written $R[Q/p]$) by an induction on the structure of R:

$$\begin{aligned}
p[Q/p] &= Q \\
p'[Q/p] &= p' \text{ if } p' \not\equiv p \\
(\neg R)[Q/p] &= \neg(R[Q/p]) \\
(R_1 \wedge R_2)[Q/p] &= (R_1[Q/p]) \wedge (R_2[Q/p]) \\
(R_1 \vee R_2)[Q/p] &= (R_1[Q/p]) \vee (R_2[Q/p])
\end{aligned}$$

1a Show that $\vdash Q_1 = Q_2$ implies $\vdash R[Q_1/p] = R[Q_2/p]$ by structural induction on R and the definition of substitution.

1b Every **Bform** is equal to a formula in full disjunctive normal form, *i.e.* a sum ($\vee$) of products ($\wedge$), with all products being products of the same set of variables or their negation. By a suitable ordering of variables and terms, one can define a *canonical form* for **Bform**'s, such that every P is equal to a unique P' in canonical form. State very clearly such a definition of canonical form for **Bform**'s.

1c Write down a set of axioms which, when combined with the usual axioms and rules for equality (written at the beginning of the problem), will have the property that

$$\vdash P = Q \quad \text{iff} \quad \models P = Q$$

In addition, briefly explain why your axioms have this property.

solution:

Following the comments is a sample solution from a member of last-year's class.

Problem 1 comments. The definition of **Bform** did not include the propositional constants **true** or **false**. No points were deducted, however, if solutions included the use of these constants. (Note that $P \wedge \neg P$ behaves exactly like **false** and $P \vee \neg P$ behaves exactly like **true**).

Many suggestions for canonical forms did not observe the difficulty that arises because p_2 and $(p_1 \wedge p_2) \vee (\neg p_1 \wedge p_2)$ are logically equivalent, and so must have the same canonical form. The hint suggested that if we are using variables p_1 and p_2 then the canonical form of p_2 should, in fact be $(p_1 \wedge p_2) \vee (\neg p_1 \wedge p_2)$.

There is a way to make p_2 the canonical form, but it is much, much harder to get right.

An alternative collection of axioms to make $\vdash$ complete could be:

The distributive laws:

$$\begin{aligned} P \wedge (Q \vee R) &= (P \wedge Q) \vee (P \wedge R) \\ P \vee (Q \wedge R) &= (P \vee Q) \wedge (P \vee R) \end{aligned}$$

The associativity laws:

$$\begin{aligned} (P \wedge (Q \wedge R)) &= ((P \wedge Q) \wedge R) \\ (P \vee (Q \vee R)) &= ((P \vee Q) \vee R) \end{aligned}$$

The commutativity laws:

$$\begin{aligned} P \vee Q &= Q \vee P \\ P \wedge Q &= Q \wedge P \end{aligned}$$

De Morgan's laws:

$$\begin{aligned} \neg(P \wedge Q) &= (\neg P) \vee (\neg Q) \\ \neg(P \vee Q) &= (\neg P) \wedge (\neg Q) \end{aligned}$$

The Idempotence laws:

$$\begin{aligned} P \wedge P &= P \\ P \vee P &= P \end{aligned}$$

Behavior of "**true**" and "**false**":

$$\begin{aligned} P \vee \neg P &= Q \vee (P \vee \neg P) \\ Q &= Q \wedge (P \vee \neg P) \\ P \wedge \neg P &= Q \wedge (P \wedge \neg P) \\ Q &= Q \vee (P \wedge \neg P) \end{aligned}$$

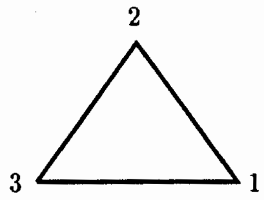


Figure 2: The original triangle after performing transformation r, a 120° clockwise rotation.

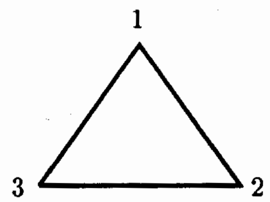


Figure 3: The original triangle after performing transformation f, a flip about the vertical axis.

1a) Proof of $(\vdash Q_1 = Q_2) \Rightarrow (\vdash R[Q_1/p] = R[Q_2/p])$ by structural induction on

BASE: case $R \equiv t$, $t \in \{true, false\}$

$$\begin{array}{lll} & \vdash t[Q_1/p] = t[Q_2/p] & \\ \Leftrightarrow & \vdash t = t & \text{by } \cancel{t[Q_1/p] = t} \text{ substitution} \\ \Leftrightarrow & true & \text{by reflexivity rule} \end{array}$$

case $R \equiv p$

$$\begin{array}{lll} & \vdash p[Q_1/p] = t[Q_2/p] & \\ \Leftrightarrow & \vdash Q_1 = Q_2 & \text{by substitution} \\ \Leftrightarrow & true & \text{by assumption } \vdash Q_1 = Q_2 \end{array}$$

case $R \equiv p'$

$$\begin{array}{lll} & \vdash p'[Q_1/p] = p'[Q_2/p] & \\ \Leftrightarrow & \vdash p' = p' & \text{by substitution} \\ \Leftrightarrow & true & \text{by reflexivity} \end{array}$$

INDUCTION case $R \equiv \neg R$

$$\begin{array}{lll} & \vdash (\neg R)[Q_1/p] = (\neg R)[Q_2/p] & \\ \Leftrightarrow & \vdash \neg(R[Q_1/p]) = \neg(R[Q_2/p]) & \text{by substitution} \\ \Leftarrow & \vdash R[Q_1/p] = R[Q_2/p] & \text{by congruence} \\ \Leftrightarrow & true & \text{by induction} \end{array}$$

case $R \equiv R_1 \; op \; R_2$, $op \in \wedge, \vee$

$$\begin{array}{lll} & \vdash (R_1 \; op \; R_2)[Q_1/p] = (R_1 \; op \; R_2)[Q_2/p] & \\ \Leftrightarrow & \vdash R_1[Q_1/p] \; op \; R_2[Q_1/p] = R_1[Q_2/p] \; op \; R_2[Q_2/p] & \text{by substitution} \end{array}$$

so:

$$\dfrac{\dfrac{\dfrac{\text{induction}}{\vdash R_1[Q_1/p] = R_1[Q_2/p]}}{\vdash R_1[Q_1/p] \; op \; R_2[Q_1/p] = R_1[Q_2/p] \; op \; R_2[Q_1/p]} \quad \dfrac{\dfrac{\text{induction}}{\vdash R_2[Q_1/p] = R_2[Q_2/p]}}{\vdash R_1[Q_2/p] \; op \; R_2[Q_1/p] = R_1[Q_2/p] \; op \; R_2[Q_2/p]} \; \text{congruence}}{\vdash R_1[Q_1/p] \; op \; R_2[Q_1/p] = R_1[Q_2/p] \; op \; R_2[Q_1/p]} \; \text{transitivity}$$

OK ✓

(b) If the Boolean variables range from $P_1 \ldots P_N$ order the variables "alphabetically" i.e. $P_1 < \overline{P_1} < P_2 < \overline{P_2} \ldots P_N < \overline{P_N}$. Within each product term arrange the variables alphabetically smallest first. The terms, which will be of equal length containing a variable or its complement, (not both) should be arranged in "dictionary" order, i.e. ordered by the first variable followed by the second, etc. ✓

(c) $\vdash P = Q$ iff $\models P = Q$

Pf ($\Rightarrow$) because all axioms are $\models$ and rules will preserve validity

($\Leftarrow$) the following rules and axioms are suffecient so that for every

P there is a unique "canonical" form Q s.t. $\vdash P = Q$

these axioms are ✓

commutative ✓ $A \text{ op } B = B \text{ op } A$ $\boxed{op \in \{\wedge, \vee\}}$

associative ✓ $A \text{ op} (B \text{ op } C) = (A \text{ op } B) \text{ op } C$

distributive ✓ $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$

✓ $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$

✓ $A \vee 1 = 1$ $\quad A \wedge 0 = 0$

✓ $A \vee 0 = A$ $\quad A \wedge 1 = A$

✓ $A \vee \overline{A} = 1$ $\quad A \wedge \overline{A} = 0$

✓ $A \vee A = A$ $\quad A \wedge A = A$

$A \vee (A \wedge B) = A$ $\quad A \wedge (A \vee B) = A$

$A \vee (\overline{A} \wedge B) = A \vee B$ $\quad A \wedge (\overline{A} \vee B) = A \wedge B$

$\overline{A \wedge B \wedge C} = \overline{A} \vee \overline{B} \vee \overline{C}$ $\quad \overline{A \vee B \vee C} = \overline{A} \wedge \overline{B} \wedge \overline{C}$

binary version are enough

these rules are sufficient to rewrite any Bform P in canonical form P'. Thus if $\models P = Q$ then $P = P'$ and $Q = P'$ since canonical form is unique. Thus by reflexivity $\models P = Q \Rightarrow \vdash P = Q$.

Quiz 3

Instructions. This is a **closed book** quiz; the Hoare Logic Rules appear in an appendix. There are four (4) problems of equal weight. Write your solutions for all problems on this quiz sheet in the spaces provided, including your ***name on each sheet.*** Ask for further blank sheets if you need them.

GOOD LUCK!

NAME

problem	*points*	*score*
1	25	
2	25	
3	25	
4	25	
Total	100	

GOOD LUCK!

Problem 1.

1(a) Exhibit a while-loop invariant suitable for a Hoare logic proof of

$$\{X = i \wedge Y = j\}\,\textbf{while}\, X \neq Y \textbf{ do } Y := Y + 1\,\{i \leq j\}\,.$$

1(b) Give a formal proof in Hoare logic of

$$\{X = 0\}\textbf{while true do}\,(Y := Y + 1;\ X := X - 1)\{X = 3\}.$$

Hint: **false** is a loop invariant.

Problem 2. Show that **Bexp** is expressive for **while**-free commands. That is, if $c \in \mathbf{Com}$ contains no **while**-loops and $b \in \mathbf{Bexp}$, then the weakest precondition $\{\mathbf{true}\}c\{b\}$ is equivalent to some $b' \in \mathbf{Bexp}$.

Hint: Induction on c.

Problem 3. Sketch how to transform any **Assn** into an equivalent **Assn** of the form

$$(Q_1 i_1)\ldots(Q_n i_n)[a = 0]$$

where each Q_i is either $\forall$ or $\exists$ and $a \in \mathbf{Aexpv}$.

Problem 4. The grammar for *exponential constant expressions*, **Ecexp**, is

$$e ::= 1 \mid e + e \mid e \times e \mid e^e$$

Meaning is defined as for arithmetic expressions, with superscript denoting exponentiation, *e.g.*, the meaning of $(1+(1+1))^{(1+1)\times((1+1)+(1+1))}$ is $3^{2\cdot 4}$, namely, 6,561. An expression **Ecexp** is said to be a *canonical form* if it is a sum of 1's (parenthesized to the left).

4(a) Write down a simple set of sound axioms for equations between **Ecexp**'s, which, together with the usual inference rules for equations (reflexivity, symmetry, transitivity, congruence), allow one to prove that any $e \in$ **Ecexp** equals a canonical form canon(e). Also, briefly explain how to use your axioms to prove that $e =$ canon(e).

4(b) Prove that your axiom system is complete.

A Hoare Logic

Axiom for **skip**:

$$\{A\}\,\mathbf{skip}\,\{A\}$$

Axiom for assignments:

$$\{B[a/X]\}X := a\{B\}$$

Rule for sequencing:

$$\frac{\{A\}c_0\{C\},\ \{C\}c_1\{B\}}{\{A\}(c_0;c_1)\{B\}}$$

Rule for conditionals:

$$\frac{\{A \wedge b\}c_0\{B\},\ \{A \wedge \neg b\}c_1\{B\}}{\{A\}\mathbf{if}\,b\,\mathbf{then}\,c_0\,\mathbf{else}\,c_1\{B\}}$$

Rule for while loops:

$$\frac{\{A \wedge b\}c\{A\}}{\{A\}\mathbf{while}\,b\,\mathbf{do}\,c\{A \wedge \neg b\}}$$

Rule of consequence:

$$\frac{\{A'\}c\{B'\}}{\{A\}c\{B\}} \qquad \text{(providing} \models (A \Rightarrow A') \wedge (B' \Rightarrow B)\ \text{)}$$

Problem Set 6 Solutions

Problem 1. Prove that for $A, B \in$ Assn, if A is equivalent to B, then

1(a) $A \wedge C$ is equivalent to $B \wedge C$ for any $C \in$ Assn.

$$\begin{aligned} \sigma \models^I A \wedge C & \text{ iff } \sigma \models^I A \text{ and } \sigma \models^I C \\ & \text{ iff } \sigma \models^I B \text{ and } \sigma \models^I C \qquad \text{since } A \text{ is equivalent to } B \\ & \text{ iff } \sigma \models^I B \wedge C \end{aligned}$$

1(b) $\exists j.A$ is equivalent to $\exists j.B$ for any $j \in$ Intvar.

$$\begin{aligned} \sigma \models^I \exists j.A & \text{ iff } \sigma \models^{I[n/j]} A \text{ for some } n \in \mathbf{N} \\ & \text{ iff } \sigma \models^{I[n/j]} B \text{ for some } n \in \mathbf{N} \\ & \text{ iff } \sigma \models^I \exists j.B \end{aligned}$$

1(c) $A[a/j]$ is equivalent to $B[a/j]$ for any $a \in$ Aexp, $j \in$ Intvar.

$$\begin{aligned} \sigma \models^I A[a/j] & \text{ iff } \sigma \models^{I[n/j]} A \text{ for } n = \mathcal{A}[\![a]\!]\sigma \ (1) \\ & \text{ iff } \sigma \models^{I[n/j]} B \text{ for } n = \mathcal{A}[\![a]\!]\sigma \\ & \text{ iff } \sigma \models^I A[a/j] \end{aligned}$$

Step (1) is true by a variant on Lemma 6.9.

Problem 2.

2(a) Prove $\exists j.(A \vee B)$ is equivalent to $(\exists j.A) \vee (\exists j.B)$.

$$\begin{aligned} \sigma \models^I \exists j.(A \vee B) & \text{ iff } \sigma \models^{I[n/j]} (A \vee B) \text{ for some } n \in \mathbf{N} \\ & \text{ iff } (\sigma \models^{I[n/j]} A) \text{ or } (\sigma \models^{I[n/j]} B) \text{ for some } n \in \mathbf{N} \\ & \text{ iff } (\sigma \models^{I[n/j]} A \text{ for some } n \in \mathbf{N}) \text{ or} \\ & \qquad (\sigma \models^{I[n/j]} B \text{ for some } n \in \mathbf{N}) \\ & \text{ iff } (\sigma \models^I \exists j.A) \text{ or } (\sigma \models^I \exists j.B) \end{aligned}$$

2(b) Prove $\exists j.(A \Rightarrow B)$ is equivalent to $(\forall j.A) \Rightarrow (\exists j.B)$.

$$
\begin{aligned}
\sigma \models^{I} \exists j.(A \Rightarrow B) \text{ iff } & \sigma \models^{I[n/j]} (A \Rightarrow B) \text{ for some } n \in \mathbf{N} \\
\text{iff } & (\sigma \not\models^{I[n/j]} A) \text{ or } (\sigma \models^{I[n/j]} B) \text{ for some } n \in \mathbf{N} \\
\text{iff } & (\sigma \not\models^{I[n/j]} A \text{ for some } n \in \mathbf{N}) \text{ or} \\
& \quad (\sigma \models^{I[n/j]} B \text{ for some } n \in \mathbf{N}) \\
\text{iff } & (\sigma \not\models^{I} \forall j.A) \text{ or } (\sigma \models^{I[n/j]} B \text{ for some } n \in \mathbf{N}) \\
\text{iff } & \sigma \models^{I} (\forall j.A) \Rightarrow (\exists j.B)
\end{aligned}
$$

2(c) Prove $\exists j.(A \wedge B)$ is equivalent to $A \wedge (\exists j.B)$ whenever $j \notin \mathrm{FV}(A)$.

$$
\begin{aligned}
\sigma \models^{I} \exists j.(A \wedge B) \text{ iff } & \sigma \models^{I[n/j]} (A \wedge B) \text{ for some } n \in \mathbf{N} \\
\text{iff } & \sigma \models^{I} (A \wedge B)[n/j] \text{ by the same variant on Lemma 6.9} \\
\text{iff } & \sigma \models^{I} A[n/j] \wedge B[n/j] \text{ by definition of substitution}
\end{aligned}
$$

However, a simple induction on the definition of substitution shows that if $j \notin \mathrm{FV}(A)$ then $A[n/j] \equiv A$, so

$$
\begin{aligned}
\sigma \models^{I} \exists j.(A \wedge B) \text{ iff } & \sigma \models^{I} A \wedge B[n/j] \\
\text{iff } & (\sigma \models^{I} A) \text{ and } (\sigma \models^{I} B[n/j]) \\
\text{iff } & (\sigma \models^{I} A) \text{ and } (\sigma \models^{I[n/j]} B) \\
\text{iff } & (\sigma \models^{I} A) \text{ and } (\sigma \models^{I} \exists j.B) \\
\text{iff } & \sigma \models^{I} A \wedge \exists j.B
\end{aligned}
$$

Problem 3. Prove, using the Hoare rules, the correctness of the partial correctness assertion:

$$
\begin{aligned}
& \{1 \leq N\} \\
& P := 0; C := 1; (\mathbf{while}\, C \leq N \,\mathbf{do}\, P := P + M; C := C + 1) \\
& \{P = M \times N\}
\end{aligned}
$$

By the rules for assignment and sequencing, we can get

$$
\frac{\begin{array}{c}\{1 \leq N \wedge 0 = 0 \wedge 1 = 1\} P := 0 \{1 \leq N \wedge P = 0 \wedge 1 = 1\} \\ \{1 \leq N \wedge P = 0 \wedge 1 = 1\} C := 1 \{1 \leq N \wedge P = 0 \wedge C = 1\}\end{array}}{\{1 \leq N \wedge 0 = 0 \wedge 1 = 1\} P := 0; C := 1 \{1 \leq N \wedge P = 0 \wedge C = 1\}}
$$

which the rule of consequence can simplify to

$$\{1 \leq N\} P := 0; C := 1 \{1 \leq N \wedge P = 0 \wedge C = 1\}.$$

For any assertion I, the assignment and sequencing rules give

$$\frac{\{I[C+1/C][P+M/P]\}P := P+M\{I[C+1/C]\} \quad \{I[C+1/C]\}C := 1\{I\}}{\{I[C+1/C][P+M/P]\}P := P+M; C := C+1\{I\}}$$

Now, let I be $(P = M \times (C-1)) \wedge (C \leq N+1)$. Then,

$$\begin{aligned} I[C+1/C][P+M/P] &\equiv (P+M = M \times (C+1-1)) \wedge (C+1 \leq N+1) \\ &\equiv (P = M \times (C-1)) \wedge (C \leq N) \\ &\equiv (P = M \times (C-1)) \wedge (C \leq N+1) \wedge (C \leq N) \\ &\equiv I \wedge (C \leq N) \end{aligned}$$

And thus, we have

$$\{I \wedge (C \leq N)\}P := P+M; C := C+1\{I\},$$

so, by the rule for while-loops

$$\frac{\{I \wedge (C \leq N)\}P := P+M; C := C+1\{I\}}{\{I\}\mathbf{while}\, C \leq N\, \mathbf{do}\, P := P+M; C := C+1\{I \wedge \neg(C \leq N)\}}$$

Finally, since

$$(1 \leq N \wedge P = 0 \wedge C = 1) \Rightarrow I$$

and

$$(I \wedge \neg(C \leq N)) \Rightarrow (P = N \times M)$$

we can use the rule of consequence to get

$$\begin{aligned} &\{1 \leq N \wedge P = 0 \wedge C = 1\} \\ &\mathbf{while}\, C \leq N\, \mathbf{do}\, P := P+M; C := C+1 \\ &\{P = N \times M\} \end{aligned}$$

and the rule for sequencing to get

$$\begin{aligned} &\{1 \leq N\} \\ &P := 0; C := 1; (\mathbf{while}\, C \leq N\, \mathbf{do}\, P := P+M; C := C+1) \\ &\{P = M \times N\}. \end{aligned}$$

Problem 4. Find an appropriate invariant to use in the while-rule for proving the following partial correctness assertion:

$$\{i = Y\}\mathbf{while}\, \neg(Y = 0)\, \mathbf{do}\, Y := Y-1; X := 2 \times X\{X = 2^i\}$$

The invariant we want is

$$I \equiv (X \times 2^Y = 2^i).$$

Given an assertion $POW(i, k, n)$ as described in Problem Set 20 which is true iff $i = k^n$, we can represent this assertion as

$$\forall j.\forall k.((POW(j, 2, Y) \wedge k = j \times X) \Rightarrow POW(2, i, k))$$

(One student pointed out, correctly, that this routine only works if X starts out with the value 1. Thus, the precondition really should be $\{i = Y \wedge X = 1\}$.)

Problem 5. Provide a Hoare rule for the repeat construct and prove it sound. (*cf.* Winskel's Exercise 5.9.)

Assuming a **repeat** construct with rules like

$$\frac{\langle c, \sigma\rangle \to \sigma'' \quad \langle b, \sigma''\rangle \to \mathbf{false} \quad \langle \mathbf{repeat}\, c\, \mathbf{until}\, b, \sigma''\rangle \to \sigma'}{\langle \mathbf{repeat}\, c\, \mathbf{until}\, b, \sigma\rangle \to \sigma'}$$

and

$$\frac{\langle c, \sigma\rangle \to \sigma' \quad \langle b, \sigma'\rangle \to \mathbf{true}}{\langle \mathbf{repeat}\, c\, \mathbf{until}\, b, \sigma\rangle \to \sigma'}$$

then a reasonable Hoare rule is

$$\frac{\{A\}c\{A\}}{\{A\}\mathbf{repeat}\, c\, \mathbf{until}\, b\{A \wedge b\}}$$

To prove soundness, assume that $\{A\}c\{A\}$ is valid. We can now show, by induction on derivations, that for any states σ and σ' such that $\sigma \models A$ and $\langle \mathbf{repeat}\, c\, \mathbf{until}\, b, \sigma\rangle \to \sigma'$, that $\sigma' \models A \wedge b$, and thus

$$\{A\}\mathbf{repeat}\, c\, \mathbf{until}\, b\{A \wedge b\}$$

is valid.

Let r be the loop $\mathbf{repeat}\, c\, \mathbf{until}\, b$. For the base case, assume that $\langle c, \sigma\rangle \to \sigma'$ and $\langle b, \sigma'\rangle \to \mathbf{true}$. By Proposition 6.4 we have $\sigma' \models b$, and, since $\{A\}c\{A\}$ is valid, we have $\sigma' \models A$. Thus, $\sigma' \models A \wedge b$.

For the sake of induction, assume that $\sigma'' \models A$ implies $\sigma' \models A \wedge b$ whenever $\langle r, \sigma''\rangle \to \sigma'$ follows from a subderivation of the derivation of $\langle r, \sigma\rangle \to \sigma'$. Then, if we have a derivation for $\langle r, \sigma\rangle \to \sigma'$, we have subderivations for $\langle c, \sigma\rangle \to \sigma''$ and $\langle r, \sigma''\rangle \to \sigma'$. Then, if $\sigma \models A$, it follows from the validity of $\{A\}c\{A\}$ that $\sigma'' \models A$, and then from the inductive assumption that $\sigma' \models A \wedge b$.

Thus, we have shown by induction that $\{A\}r\{A \wedge b\}$ is valid, if $\{A\}c\{A\}$ is, so the rule is sound.

The Four Squares Theorem (optional material)

Note that this is **optional** material. You will not be held responsible for it in this class. However, a few people expressed curiosity (or disbelief) in this theorem, so some might find the following proof interesting.

Review

We'll need to recall the following definition:

Definition 1. For any positive integer a and integers b_1 and b_2, we say that $b_1 \equiv b_2 \pmod{a}$ (pronounced "b_1 is congruent to b_2 modulo a") if $b_1 - b_2$ is divisible by a.

This definition has the following important properties, which are all easy to verify. (Think about them a minute to make sure, since they'll be important to the proof.)

- Congruence modulo a is reflexive, transitive, and symmetric:

  $b \equiv b \pmod{a}$;

  if $b_1 \equiv b_2 \pmod{a}$ and $b_2 \equiv b_3 \pmod{a}$ then $b_1 \equiv b_3 \pmod{a}$;

  if $b_1 \equiv b_2 \pmod{a}$ then $b_2 \equiv b_1 \pmod{a}$.

  In other words, congruence modulo a is an equivalence relation.

- If $b_1 \equiv b_2 \pmod{a}$ and $b'_1 \equiv b'_2 \pmod{a}$ then

  $$b_1 + b'_1 \equiv b_2 + b'_2 \pmod{a},$$
  $$b_1 \times b'_1 \equiv b_2 \times b'_2 \pmod{a}.$$

- For every integer b, there is some integer c with $0 \leq c < a$ such that $c \equiv b \pmod{a}$. An important consequence of this is that there are only a equivalence classes modulo a. In other words, in any collection of $a+1$ integers, at least two must be congruent modulo a.

Given these facts, we can proceed with the proof.

The four squares theorem

Theorem 1. For any nonnegative integer a, there are four integers x_0, x_1, x_2, x_3 such that

$$x_1^2 + x_2^2 + x_3^2 + x_4^2 = a.$$

Further, there are nonnegative integers a that cannot be represented as the sum of fewer than four squares.

Proof: The proof that follows is based on that in [1].

The first useful fact to note is that if we have two integers, each represented as the sum of four squares, then their product is also representable as the sum of four squares:

$$\begin{aligned}
&(x_1^2 + x_2^2 + x_3^2 + x_4^2)(y_1^2 + y_2^2 + y_3^2 + y_4^2) \\
&= (x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4)^2 + (x_1y_2 - x_2y_1 + x_3y_4 - x_4y_3)^2 \\
&\quad + (x_1y_3 + x_3y_1 + x_2y_4 + x_4y_2)^2 + (x_1y_4 - x_4y_1 + x_2y_3 - x_3y_2)^2
\end{aligned}$$

(This equation, due to Euler, can be "easily" verified by working out all the products. It's enough of a mess, though, that it's not worth showing the details here.) The important aspect of this equality is that it simplifies our problem as follows: since every integer greater than 1 can be represented as the product of some collection of prime numbers, if we know how to represent 0, 1 and 2 as sums of four squares, and we know how to represent any odd prime as the sum of four squares, then we can represent any nonnegative integer as the sum of four squares.

0, 1 and 2 are easy:

$$\begin{aligned}
0 &= 0^2 + 0^2 + 0^2 + 0^2 \\
1 &= 1^2 + 0^2 + 0^2 + 0^2 \\
2 &= 1^2 + 1^2 + 0^2 + 0^2
\end{aligned}$$

so we now have to find out how to represent any odd prime. To do this, we will proceed in two steps:

Lemma 1. For any odd prime p, there is a positive integer $b < p$, and integers x_1, x_2, x_3, x_4 such that

$$x_1^2 + x_2^2 + x_3^2 + x_4^2 = pb$$

Lemma 2. For any odd prime p, if there are integers b and x_1, x_2, x_3, x_4 as in Lemma 1 and $b > 1$, then there is a positive integer $b' < b$ and integers y_1, y_2, y_3, y_4 such that

$$y_1^2 + y_2^2 + y_3^2 + y_4^2 = pb'$$

With these two lemmas, it is clear (by a simple induction) that for any odd prime p, there must be a set of integers z_1, z_2, z_3, z_4 such that

$$z_1^2 + z_2^2 + z_3^2 + z_4^2 = p \times 1 = p.$$

Together with the Euler equality given above, it follows that any nonnegative integer can be represented as the sum of four squares. To see that four squares are necessary in general, note that

$$7 = 2^2 + 1^2 + 1^2 + 1^2,$$

but an enumeration of the possibilities will show that there is no way to represent 7 with fewer than four squares. ■

Proof of Lemma 1

Here's where we start to use the notion of "congruence modulo p." Consider the sequence of integers

$$0^2, 1^2, \ldots, \left(\frac{p-1}{2}\right)^2$$

(Remember that p is odd, so $(p-1)/2$ is an integer.) If any two distinct numbers c_1^2 and c_2^2 in this sequence were congruent modulo p, then, by definition, we would have $c_1^2 - c_2^2$ divisible by p, which means that $(c_1 + c_2)(c_1 - c_2)$ would be divisible by p. Since p is prime, that would mean that either $c_1 + c_2$ or $c_1 - c_2$ would be divisible by p. Since both $c_1 + c_2$ and $c_1 - c_2$ must be less than p, greater than $-p$, and non-zero, this is impossible, so no two elements of this sequence are congruent modulo p.

By a similar argument, no two elements of the sequence

$$-1 - 0^2, -1 - 1^2, \ldots, -1 - \left(\frac{p-1}{2}\right)^2$$

are congruent modulo p.

Now, consider the sequence

$$-1 - \left(\frac{p-1}{2}\right)^2, \ldots, -1 - 0^2, 0^2, 1^2, \ldots, \left(\frac{p-1}{2}\right)^2.$$

There are $p+1$ elements in this sequence, so there must be two that are congruent modulo p. Since we've already established that neither the first half nor the second half contains any pairs of congruent elements, any pair of congruent elements must contain one element from each half. Thus, there must be numbers

$-1 - x_1^2$ from the first (negative) half and x_2^2 from the second (positive) half such that

$$-1 - x_1^2 \equiv x_2^2 \pmod{p},$$

and thus

$$x_1^2 + x_2^2 + 1 \equiv 0 \pmod{p}.$$

Since $x_1^2 + x_2^2 + 1$ is positive and divisible by p, there must be some positive integer b such that

$$x_1^2 + x_2^2 + 1 = pb.$$

Finally, by the definition of the sequences,

$$\begin{aligned} x_1^2 + x_2^2 + 1 &\le \left(\frac{p-1}{2}\right)^2 + \left(\frac{p-1}{2}\right)^2 + 1 \\ &= \frac{(p-1)^2 + 2}{2} \\ &= \frac{p^2 - 2p + 4}{2} \\ &< p^2 \qquad \text{when } p \ge 2, \end{aligned}$$

so $b < p$. ■

Proof of Lemma 2

Assume that we have some p, b and x_1, x_2, x_3, x_4 as in Lemma 1, with $b > 1$. Then, either b is even or b is odd.

Case 1: b is even

If b is even, then pb is even, so $x_1^2 + x_2^2 + x_3^2 + x_4^2$ is even. This means that either none, two, or all four of the x_i's are even. If any of them are even, then assume without loss of generality that x_1 and x_2 are even.

Now, it follows from this that $x_1 - x_2$, $x_1 + x_2$, $x_3 - x_4$ and $x_3 + x_4$ are all even. Since

$$2(x_i^2 + x_j^2) = (x_i - x_j)^2 + (x_i + x_j)^2,$$

it follows that

$$2pb = (x_1 - x_2)^2 + (x_1 + x_2)^2 + (x_3 - x_4)^2 + (x_3 + x_4)^2,$$

where each of these squares is divisible by 4. Thus, we have

$$p \times \frac{b}{2} = \left(\frac{x_1 - x_2}{2}\right)^2 + \left(\frac{x_1 + x_2}{2}\right)^2 + \left(\frac{x_3 - x_4}{2}\right)^2 + \left(\frac{x_3 + x_4}{2}\right)^2,$$

so, letting $b' = b/2$, we are done.

Case 2: b is odd

A consequence of the properties we gave for congruence modulo a is:

For any integer x, there is an integer y such that

$$x \equiv y \pmod{b} \qquad \text{and} \qquad -\frac{b}{2} < y \leq \frac{b}{2}.$$

If b is odd, then an integer y can never equal $b/2$, so this statement can be strengthened to require

$$-\frac{b-1}{2} \leq y \leq \frac{b-1}{2}.$$

From this, it follows that there are integers y_1, y_2, y_3, y_4 such that

$$y_i \equiv x_i \pmod{b} \qquad \text{and} \qquad -\frac{b-1}{2} \leq y_i \leq \frac{b-1}{2}$$

for each $0 < i \leq 4$. Since $x_1^2 + x_2^2 + x_3^2 + x_4^2$ is divisible by b, we have

$$x_1^2 + x_2^2 + x_3^2 + x_4^2 \equiv 0 \pmod{b},$$

which means that

$$y_1^2 + y_2^2 + y_3^2 + y_4^2 \equiv 0 \pmod{b}.$$

This, in turn, means that $y_1^2 + y_2^2 + y_3^2 + y_4^2$ (which must be nonnegative) is divisible by b, so there is some nonnegative b' such that

$$y_1^2 + y_2^2 + y_3^2 + y_4^2 = bb'.$$

Since $-(b-1)/2 \leq y_i \leq (b-1)/2$ for each y_i, we know that

$$0 \leq y_1^2 + y_2^2 + y_3^2 + y_4^2 \leq (b-1)^2 < b^2,$$

so $b' < b$.

Now, we know that $0 \leq b' < b$. To see that, in fact, $0 < b'$, note that if $b' = 0$ then it must be true that $y_1 = y_2 = y_3 = y_4 = 0$. It follows that $x_i \equiv 0 \pmod{b}$ for all x_i, which means that each x_i is divisible by b, which in turn means that each x_i^2 is divisible by b^2, and thus so is $x_1^2 + x_2^2 + x_3^2 + x_4^2$. But this means that pb is divisible by b^2, so p is divisible by b, which contradicts the fact that p is prime. Thus, $0 < b' < b$.

To summarize so far, we have

$$\begin{aligned} x_1^2 + x_2^2 + x_3^2 + x_4^2 &= pb \\ y_1^2 + y_2^2 + y_3^2 + y_4^2 &= bb' \end{aligned}$$

with $0 < b' < b < p$. This gives us

$$pb^2b' = (x_1^2 + x_2^2 + x_3^2 + x_4^2)(y_1^2 + y_2^2 + y_3^2 + y_4^2).$$

By the Euler identity we used in Theorem 1, this means that

$$\begin{aligned} pb^2b' &= (x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4)^2 + (x_1y_2 - x_2y_1 + x_3y_4 - x_4y_3)^2 \\ &\quad + (x_1y_3 + x_3y_1 + x_2y_4 + x_4y_2)^2 + (x_1y_4 - x_4y_1 + x_2y_3 - x_3y_2)^2. \end{aligned}$$

Now, we can verify that each of these four squares is divisible by b^2 as follows:

- For each $1 \leq i \leq 4$, $x_i \equiv y_i \pmod b$ by definition. Thus
$$x_iy_i \equiv x_i^2 \pmod b,$$
so, since
$$x_1^2 + x_2^2 + x_3^2 + x_4^2 \equiv 0 \pmod b,$$
we have
$$x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4 \equiv 0 \pmod b,$$
so $x_1y_1+x_2y_2+x_3y_3+x_4y_4$ is divisible by b, and $(x_1y_1+x_2y_2+x_3y_3+x_4y_4)^2$ is divisible by b^2.

- Again, since $x_i \equiv y_i \pmod b$, it follows that $x_iy_j \equiv x_jy_i \pmod b$ for all x_i, y_i, x_j, y_j. Thus, we have
$$\begin{aligned} x_1y_2 - x_2y_1 + x_3y_4 - x_4y_3 &\equiv 0 \pmod b \\ x_1y_3 - x_3y_1 + x_2y_4 - x_4y_2 &\equiv 0 \pmod b \\ x_1y_4 - x_4y_1 + x_2y_3 - x_3y_2 &\equiv 0 \pmod b \end{aligned}$$

  So each of these expressions is divisible by b, so their squares are all divisible by b^2.

Thus, we have

$$\begin{aligned} pb' &= \left(\frac{x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4}{b}\right)^2 + \left(\frac{x_1y_2 - x_2y_1 + x_3y_4 - x_4y_3}{b}\right)^2 \\ &\quad + \left(\frac{x_1y_3 + x_3y_1 + x_2y_4 + x_4y_2}{b}\right)^2 + \left(\frac{x_1y_4 - x_4y_1 + x_2y_3 - x_3y_2}{b}\right)^2, \end{aligned}$$

which gives us $pb' = z_1^2 + z_2^2 + z_3^2 + z_4^2$ for some $0 < b' < b$ and integers z_1, z_2, z_3, z_4. ■

References

[1] Ivan Niven and Herbert S. Zuckerman. *An Introduction to the Theory of Numbers.* John Wiley & Sons, Inc., New York, fourth edition, 1980.

Quiz 3 Solutions and Grading Statistics

Instructions. This was a **closed book** quiz; the Hoare Logic Rules appeared in an appendix. There were four (4) problems, each worth 25 points. Gradings statistics are given below, and sample solutions follow.

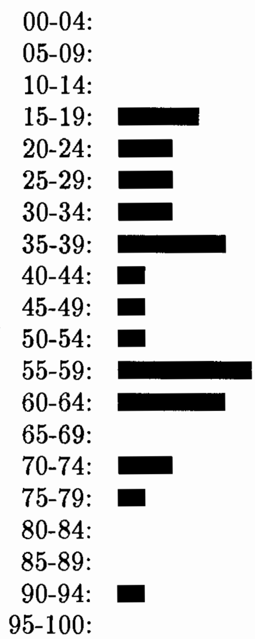
Number of quizzes taken: 29

Grade range: 15-92

Mean: 46

Median: 46

Histogram:



Problem 1.

1(a) Exhibit a while-loop invariant suitable for a Hoare logic proof of

$$\{X = i \wedge Y = j\}\, \textbf{while}\, X \neq Y\, \textbf{do}\, Y := Y + 1\, \{i \leq j\}\,.$$

The simplest we found was $(j \leq Y \wedge X = i)$. Clearly, this is implied by the precondition, and the conjunction of it and $\neg(X \neq Y)$ implies the postcondition.

1(b) Give a formal proof in Hoare logic of

$$\{X = 0\}\textbf{while true do}\,(Y := Y + 1;\, X := X - 1)\{X = 3\}.$$

Hint: **false** is a loop invariant.

Since $true[X-1/X] \equiv true$ and $true[Y+1/Y] \equiv true$

we have

$$\begin{array}{ll}
\{true\}\ Y := Y+1\ \{true\} \qquad \{true\}\ X := X-1\ \{true\} & \text{by assignment} \\ \hline
\{true\}\ Y := Y+1;\ X := X+1\ \{true\} & \text{by sequencing} \\ \hline
\{true\}\ \text{while } true \text{ do } Y := Y+1;\ X := X-1\ \{true \wedge \neg true\} & \text{by consequence } (\{true \wedge true\} \Rightarrow \{true\}) \text{ and while-loop} \\ \hline
\{true\}\ \text{while } - - - - -\ \{false\} & \text{by consequence} \\ \hline
\{X = 0\}\ \text{while } - - - -\ \{X = 3\} & \text{by consequence } (\{X=0\} \Rightarrow \{true\} \wedge \{false\} \Rightarrow \{X=3\})
\end{array}$$

Problem 2. Show that **Bexp** is expressive for **while**-free commands. That is, if $c \in$ **Com** contains no **while**-loops and $b \in$ **Bexp**, then the weakest precondition $\{\mathbf{true}\}c\{b\}$ is equivalent to some $b' \in$ **Bexp**.

Hint: Induction on c.

As shown in the expressiveness proof:

$$W(\text{skip}, b) \equiv b$$

$$W(X := a, b) \equiv b[a/X]$$ (which is a Bexp since Bexps are closed under substitution)

Then, by induction,
if $W(c_0, b)$ is representable as a Bexp for all b, and $W(c_1, b')$ similarly, then by induction

$$W(c_0; c_1, b) \equiv W(c_0, W(c_1, b)) \equiv W(c_0, b')$$

for some Bexp b', which is then representable as a Bexp.

and similarly

$$W(\text{if } b' \text{ then } c_0 \text{ else } c_1, b)$$
$$\equiv (b' \Rightarrow W(c_0, b)) \wedge (\neg b' \Rightarrow W(c_1, b))$$
$$\equiv (\neg b' \vee W(c_0, b)) \wedge (b' \vee W(c_1, b))$$

is representable as a Bexp by induction.

Problem 3. Sketch how to transform any **Assn** into an equivalent **Assn** of the form

$$(Q_1 i_1)\ldots(Q_n i_n)[a = 0]$$

where each Q_i is either $\forall$ or $\exists$ and $a \in$ **Aexpv**.

0) Convert $A \Rightarrow B$ to $\neg A \vee B$ everywhere.

1) Push all $\neg$ inward as far as possible using

$\neg(A \wedge B) \equiv \neg A \vee \neg B \qquad \neg(A \vee B) \equiv \neg A \wedge \neg B$

$\neg \exists j . A \equiv \forall j . \neg A \qquad \neg \forall j . A \equiv \exists j . \neg A$

$\neg\neg A = A$

2) Replace $\neg(a_0 \leq a_1)$ with $(a_1 \leq a_0 \wedge \neg(a_1 = a_0))$

3) Replace $\neg(a_0 = a_1)$ with

$\exists i, j, k, l .\ (a_0 - a_1)^2 = i^2 + j^2 + k^2 + l^2 + 1$

4) Replace $a_0 \leq a_1$ with

$\exists i, j, k, l .\ a_0 + i^2 + j^2 + k^2 + l^2 = a_1$

5) Replace $a_0 = a_1$ with $a_0 - a_1 = 0$

6) Move all quantifiers out, renaming variables as necessary

e.g. $A \wedge \exists j . B \equiv \exists k . A \wedge B[k/j]$

where k is fresh.

7) Eliminate $\wedge$ and $\vee$ with

$a_0 = 0 \wedge a_1 = 0 \equiv a_0^2 + a_1^2 = 0$

$a_0 = 0 \vee a_1 = 0 \equiv a_0 \times a_1 = 0$

Problem 3. Sketch how to transform any **Assn** into an equivalent **Assn** of the form

$$(Q_1 i_1)\dots(Q_n i_n)[a = 0]$$

where each Q_i is either $\forall$ or $\exists$ and $a \in$ **Aexpv**.

0) Convert $A \Rightarrow B$ to $\neg A \vee B$

1) Push all $\neg$ inward as far as possible using

$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$

$$\neg(A \vee B) \equiv \neg A \wedge \neg B$$

$$\neg \exists j.\, A \equiv \forall j.\, \neg A$$

$$\neg \forall j.\, A \equiv \exists j.\, \neg A$$

$$\neg\neg A \equiv A$$

2) replace $\neg(a_0 \leq a_1)$ with $(a_1 \leq a_0 \wedge \neg(a_1 = a_0))$

3) replace all $\neg(a_0 = a_1)$ with

$$\exists i, j, k, l.\ (a_0 - a_1) \times (a_0 - a_1) + i^2 - j^2 - k^2 - l^2 - 1 = 0$$

4) replace all $a_0 \leq a_1$ with

$$\exists i, j, k, l.\ a_1 - a_0 - i^2 - j^2 - k^2 - l^2 = 0$$

5) replace all $a_0 = a_1$ with $a_0 - a_1 = 0$

6) Move all quantifiers out, renaming variables as necessary.

e.g. $A \wedge \exists j.\, B \equiv \exists k.\, A \wedge B[k/j]$ where k is fresh.

7) Eliminate $\wedge$ and $\vee$ with

$$a_0 = 0 \wedge a_1 = 0 \equiv a_0^2 + a_1^2 = 0$$

$$a_0 = 0 \vee a_1 = 0 \equiv a_0 \times a_1 = 0$$

Problem 4. The grammar for *exponential constant expressions*, **Ecexp**, is

$$e ::= 1 \mid e + e \mid e \times e \mid e^e$$

Meaning is defined as for arithmetic expressions, with superscript denoting exponentiation, *e.g.*, the meaning of $(1+(1+1))^{(1+1)\times((1+1)+(1+1))}$ is $3^{2\cdot4}$, namely, 6,561. An expression **Ecexp** is said to be a *canonical form* if it is a sum of 1's (parenthesized to the left).

4(a) Write down a simple set of sound axioms for equations between **Ecexp**'s, which, together with the usual inference rules for equations (reflexivity, symmetry, transitivity, congruence), allow one to prove that any $e \in$ **Ecexp** equals a canonical form canon(e). Also, briefly explain how to use your axioms to prove that $e =$ canon(e).

The simplest set of rules we found was

1) $e_0^{e_1+1} = e_0 \times e_0^{e_1}$
2) $e^1 = e$
3) $e_0 \times (e_1 + 1) = (e_0 \times e_1) + e_0$
4) $e \times 1 = e$
5) $e_0 + (e_1 + 1) = (e_0 + e_1) + 1$

~~converting expressions that match the rule~~
As an algorithm: starting at the innermost, uppermost expressions, apply applicable rules from left-hand-side to right-hand-side. This will eventually convert any Ecexp to canonical form.

4(b) Prove that your axiom system is complete.

Given that the rules are sound and sufficient to prove $\vdash e = \text{canon}(e)$ for any e, we have:

if $\models e_0 = e_1$ then ~~[illegible]~~ $\vdash e_0 = \text{canon}(e_0)$ and (*)
$\vdash e_1 = \text{canon}(e_1)$ (**)

by soundness, then, $\models e_0 = \text{canon}(e_0)$ and
$\models e_1 = \text{canon}(e_1)$,

so by ~~[illegible]~~ symmetry & transitivity
$\models \text{canon}(e_0) = \text{canon}(e_1)$

Since canonical forms can only have the same value if they are identical (simple proof by induction on the length), canon(e_0) and canon(e_1) are the same expression. Then, by symmetry from (**)

$\vdash \text{canon}(e_1) = e_1$ (***)

and by transitivity from (*) and (***)

$\vdash e_0 = e_1$

A Hoare Logic

Axiom for **skip***:*

$$\{A\}\,\mathbf{skip}\,\{A\}$$

Axiom for assignments:

$$\{B[a/X]\}X := a\{B\}$$

Rule for sequencing:

$$\frac{\{A\}c_0\{C\},\ \{C\}c_1\{B\}}{\{A\}(c_0;c_1)\{B\}}$$

Rule for conditionals:

$$\frac{\{A \wedge b\}c_0\{B\},\ \{A \wedge \neg b\}c_1\{B\}}{\{A\}\mathbf{if}\,b\,\mathbf{then}\,c_0\,\mathbf{else}\,c_1\{B\}}$$

Rule for while loops:

$$\frac{\{A \wedge b\}c\{A\}}{\{A\}\mathbf{while}\,b\,\mathbf{do}\,c\{A \wedge \neg b\}}$$

Rule of consequence:

$$\frac{\{A'\}c\{B'\}}{\{A\}c\{B\}} \qquad (\text{providing} \models (A \Rightarrow A') \wedge (B' \Rightarrow B)\ )$$

Problem Set 8

Due: 25 November 1992

Reading assignment. Winskel, Appendix on Computability.

Problem 1. Winskel, Exercise A.6.

Let **Funcvar** be a set whose elements, F, are called *function variables.* Associated with each $F \in$ **Funcvar** is a nonnegative integer called arity(F).

Let $\mathbf{Aexp}_f$ to be **Aexp** with one more clause in its grammar, namely,

$$a ::= F(a_1, \ldots, a_{\text{arity}(F)}).$$

A *function environment* is a mapping R, from function variables to partial functions on numbers which "respects arity", that is,

$$\mathrm{R}(F) : \mathbf{N}^{\text{arity}(F)} \rightharpoonup \mathbf{N}$$

for all $F \in$ **Funcvar**. Let R be the set of function environments.

Now the meaning of an $\mathbf{Aexp}_f$ will depend not only on a state, σ, but also on a function environment, R. It will be helpful technically to use the bijection mapping any partial function $f : \mathbf{N}^n \rightharpoonup \mathbf{N}$ to the strict total function $f_\perp : (\mathbf{N}_\perp)^n \to \mathbf{N}_\perp$ where f and $f_\perp$ agree when all arguments are nonbottom, and $f_\perp(m_1, \ldots, m_n) = \perp$ whenever $m_i = \perp$ for some m_i. Now given a R and σ, the meaning of an $\mathbf{Aexp}_f$ will be a strict total function from $(\mathbf{N}_\perp)^n$ to $\mathbf{N}_\perp$. To define meaning of $\mathbf{Aexp}_f$'s we now have the obvious clause

$$\mathcal{A}[\![F(a_1, \ldots, a_n)]\!]\mathrm{R}\sigma = \mathrm{R}(F)(\mathcal{A}[\![a_1]\!]\mathrm{R}\sigma, \ldots, \mathcal{A}[\![a_n]\!]\mathrm{R}\sigma)$$

where $n = \text{arity}(F)$. Note that (as, for example, in $\mathbf{IMP}_r$), the value of an expression may be "undefined", *i.e.*, equal to $\perp \in \mathbf{N}_\perp$.

Let $\mathbf{IMP}_f$ be the extension of **IMP** which uses $\mathbf{Aexp}_f$'s instead of just **Aexp**'s. Note that the domain C of command meanings is now $\mathrm{R} \to (\Sigma_\perp \to \Sigma_\perp)$. It will also be convenient to identify the "bottom state", $\perp_\Sigma$ with the function mapping every location to the "bottom number" $\perp_{\mathbf{N}}$.

For any $\rho \in$ R, and $c \in \mathbf{Com}_f$, and sequence $Y_1, \ldots, Y_{n+1}$ of locations, where the first $n \geq 0$ locations are distinct, we let

$$\{c\}_{\mathrm{R}, Y_1, \ldots, Y_{n+1}} : \mathbf{N}_\perp^n \to \mathbf{N}_\perp$$

be the *strict* function on numbers computed by c when function variables are interpreted according to R, number arguments are placed in locations $Y_1, \ldots, Y_n$, and the answer is left in location Y_{n+1}. To be precise, we define

$$\{c\}_{\mathrm{R},Y_1,\ldots,Y_{n+1}}(m_1,\ldots,m_n) = \mathcal{C}[\![c]\!]\mathrm{R}(\sigma_0[m_1/Y_1]\ldots[m_n/Y_n])(Y_{n+1})$$

where σ_0 is the state mapping all locations to zero. We say a partial function f is R-*computable* iff $f_\perp$ is the function $\{c\}_{\mathrm{R},Y_1,\ldots,Y_{n+1}}$ for some $c \in \mathbf{IMP}_f$ and locations $Y_1, \ldots, Y_{n+1}$.

Note that a partial function f is computable as defined in Winskel,§A.1 iff $f_\perp = \{c\}_{\mathrm{R},Y_1,\ldots,Y_{n+1}}$ for some R, Y's, and a $\mathbf{Com}_f$ which contains no occurrences of function variables, that is c is an ordinary **Com**.

Problem 2. Show that if $g : \mathbf{N}_\perp^3 \to \mathbf{N}_\perp$ and $f_i : \mathbf{N}_\perp^2 \to \mathbf{N}_\perp$ are R-computable functions for $i = 1, 2, 3$, then so is $h : \mathbf{N}_\perp^2 \to \mathbf{N}_\perp$ where

$$h(m_1, m_2) = g(f_1(m_1, m_2), f_2(m_1, m_2), f_3(m_1, m_2))$$

Problem 3.

3(a) If the value of a function variable, F, is already computable from the other function variables, then there is no need for it commands. Namely, suppose that $\mathrm{R}(F) = \{c'\}_{\mathrm{R},Y_1,\ldots,Y_{n+1}}$ for some $c' \in \mathbf{Com}_f$ which contains no occurrences of F. Show that then *every rho*-computable function is computable by a $\mathbf{Com}_f$ which contains no occurrences of F by explaining how to transform any $c'' \in \mathbf{Com}_f$ into a c' without F such that $\{c''\}_{\mathrm{R},Y'_1,\ldots,Y'_{n+1}} = \{c'\}_{\mathrm{R},Y_1,\ldots,Y_{n+1}}$.

Hint: There is a bit of "expression compiling" required here which you may find awkward to explain. We will give nearly full credit to solutions which assume that F only occurs as an outermost symbol of an $\mathbf{Aexp}_f$—never as the operator of a subexpression—and that F does not occur at all in Boolean expressions.

3(b) An **IMP**-environment R is *computable* if $\mathrm{R}_0(F)$ is computable (*i.e.*, by an ordinary **IMP** command) for all $F \in \mathbf{Funcvar}$. Conclude that if R is *computable*, then every R-computable function is computable.

Problem 4. Suppose $f(n, m) = g(m)$ for $n \leq 0$, and $f(n, m) = h(n - 1, m, f(n - 1, m))$ for $n > 0$. Show that if $g_\perp$ and $h_\perp$ are R-computable, then so is $f_\perp$.

Problem 5.

5(a) Show that if a set of numbers is R-checkable, then it is the range of a R-computable function.

5(b) Show that if a set of numbers is the range of R-computable function, f, of one argument, then it is R-checkable. *Hint*: Search for n such that $f(\text{left}(n))$ converges in at most $\text{right}(n)$ steps to the input being checked.

5(c) Show that a set is the range of a R-computable function iff it is the range of R-computable function of one argument. *Hint*: mkpair.

Semantics by Translation

In addition to operational, denotational, and axiomatic semantics, one can assign semantics to a source language by explaining how to translate it into another target language with known semantics. If the target language is low-level enough, this corresponds to compilation.

As an illustration, we describe a translation from $\mathbf{IMP}_r$, the language **IMP** extended with **resultis**, to a quite restricted subset, $\mathbf{IMP}_{ue}$, of **IMP** in which expressions are "unnested," *viz.*, assignments are of the form $X := Y$ op Z or $X := 0$, and **Bexp**'s are of the form $0 \le X$.

Because $\mathbf{IMP}_r$ expressions have side-effects, we translate both expressions and commands into $\mathbf{IMP}_{ue}$ commands, using designated "output" locations to save the expression values. The idea is to translate $a \in \mathbf{Aexp}_r$ into an $\mathbf{IMP}_{ue}$ command equivalent to $X := a$, where X is some fresh "output" location. We assume a function fresh : $\mathcal{P}ow_f(L) \rightarrow \mathbf{Loc}$ such that $\mathrm{fresh}(L) \notin L$ for every finite subset, L, of locations. Since the translated code needs "temporary" locations, we actually only ask that the translation of a be equivalent to $X := a$ in its side-effects on $\mathrm{loc}(a)$. To avoid interference among output locations, the translation takes a finite set of locations as a second parameter. These are the locations not to be interfered with.

Definition 1. For states σ_1, σ_2 and $L \subseteq \mathbf{Loc}$,

$$\sigma_1 =_L \sigma_2 \text{ iff } \sigma_1(X) = \sigma_2(X) \text{ for all } X \in L,$$

and for $c, c' \in \mathbf{Com}_r$,

$$c \sim_L c' \quad \text{iff} \quad \sigma_1 =_L \sigma_2 \text{ implies } \mathcal{C}[\![c]\!]\sigma_1 =_L \mathcal{C}[\![c]\!]\sigma_2.$$

For $a \in \mathbf{Aexp}_r$, $b \in \mathbf{Bexp}_r$, $c \in \mathbf{Com}_r$, and $L \in \mathcal{P}ow_f(\mathbf{Loc})$, we will now define $\mathrm{Trans}(\cdot, L) \in \mathbf{IMP}_{ue}$ so that

$$\mathrm{Trans}(a, L) \quad \sim_{L'} \quad X := a,$$

where $X = \mathrm{fresh}(\mathrm{loc}(a) \cup L)$ and $L' = \mathrm{loc}(a) \cup L \cup \{X\}$, and similarly

$$\mathrm{Trans}(b, L) \quad \sim_{L'} \quad \mathbf{if}\, b\, \mathbf{then}\, X := 1\, \mathbf{else}\, X := 0,$$

$$\mathrm{Trans}(c, L) \quad \sim_{\mathrm{loc}(c) \cup L} \quad c\,.$$

For a equal to $a_0 + a_1$ we have:

$$\mathrm{Trans}(a, L) ::= \mathrm{Trans}(a_0, L \cup \mathrm{loc}(a));\ \mathrm{Trans}(a_1, L');\ X := X + Y$$

where $X = \text{fresh}(\text{loc}(a) \cup L)$, $L' = \text{loc}(a) \cup L \cup \{X\}$, and $Y = \text{fresh}(L')$. Note that X serves as the output location for the subtranslation of a_0 as well as for a.

For a equal to $c\,\textbf{resultis}\,a_0$:

$$\text{Trans}(a, L) ::= \text{Trans}(c, L \cup \text{loc}(a)); \text{Trans}(a_0, L \cup \text{loc}(a))$$

For b equal to $a_0 \leq a_1$:

$\text{Trans}(b, L) ::=$
$\quad \text{Trans}(a_0, L \cup \text{loc}(a))$; $\text{Trans}(a_1, L')$;
$\quad Y := Y - X$;
$\quad \textbf{if}\, 0 \leq Y\, \textbf{then}\, (X := 0;\ X := X + 1)\, \textbf{else}\, X := 0$

where $X = \text{fresh}(\text{loc}(b) \cup L)$, $L' = \text{loc}(b) \cup L \cup \{X\}$, and $Y = \text{fresh}(L')$.

And so forth. Here are three optional exercises to help understand how and why the translation works.

Exercise 1. Describe the translation for $X := a$.

Exercise 2. Describe translation for $\textbf{while}\, b\, \textbf{do}\, c$.

Exercise 3. Write out the result of translating of the following command into $\textbf{IMP}_{ue}$:

$$\textbf{if}\, 5 \leq (\text{Euclid}\ \textbf{resultis}\ M)\, \textbf{then}\, N := 0\, \textbf{else}\, M := (M := (X + M)\, \textbf{resultis}\, 2 \times M)$$

Even $\textbf{IMP}_{ue}$ is complicated by theoretical standards, and it is an important theoretical exercise to see how simple a language one can further translate into. Some of what we do in the following optional exercises resembles compiling **IMP** down to RISC register-transfer, but mostly the translated code is too inefficient to be of any practical interest. It makes a good story though:

Exercise 4. Explain further how to translate $\textbf{IMP}_{ue}$ into a subset of $\textbf{IMP}_{ue}$ in which the only assignments are of the form $X := X + 1$, $X := X - 1$, or $X := 0$. *Hint*: $X := 0$ is used to initialize temporary locations. First eliminate $\times$ using repeated addition. Then eliminate addition using repeated incrementing or decrementing and copying assignments, *i.e.*, $X := Y$. Then simulate copying assignments with repeated incrementing or decrementing.

Exercise 5. Explain further how to translate $\mathbf{IMP}_{ue}$ into a subset of $\mathbf{IMP}_{ue}$ in which the only assignments are of the form $X := X + 1$, $X := X - 1$, or $X := 0$, and the only **Bexp**'s are of the form $X = 0$. Such commands are called *counter machines. Hint*: Simulate the test $0 \leq X$ by incrementing or decrementing X until it equals zero. Since there is no way offhand to tell whether to increment or decrement, do them alternately a growing number of times.

Exercise 6. Explain how to translate any counter machine c into a counter machine c' with at most *one* while-loop and no nested conditionals. *Hint*: Translate c into a sequence of labelled, guarded assignment instructions and goto's. Then simulate the labels and goto's by setting and testing "flags," *i.e.*, fresh zero-one valued locations, within the body of a single while-loop.

Exercise 7. (a) Explain how to translate any counter machine c into a counter machine c' which simulates it using only *two* locations. Note that c' doesn't have enough locations to be $=_{\mathrm{loc}(c)}$ to c, so we ask only that c' halts in the zero state σ_0 iff c does too. *Hint*: Use a representation like that of the pairing function mkpair of Handout 32, Notes on Expressiveness, so the contents $\vec{m}_n$ of n locations are coded into a single location as

$$2^{m_1}3^{\mathrm{sg}(m_1)}5^{m_2}7^{\mathrm{sg}(m_2)}11^{m_3}13^{\mathrm{sg}(m_3)}\cdots(p_{2n-1})^{m_n}(p_{2n})^{\mathrm{sg}(m_n)}$$

where p_n is the n^{th} prime. Simulating an increment of the third location corresponds to multiplying the code number by 11.

(b) Conclude that the zero-state halting problem for "two-counter" machines is undecidable.

Exercise 8. Explain how to decide the zero-state halting problem for one-counter machines.

Problem Set 9

Due: 7 December 1992

Reading assignment. Notes on Expressibility, Checkability, Decidability—Handout 40.

Problem 1. $S_1 \operatorname{join} S_2$ is defined to be $\{n \mid \operatorname{left}(n) \in S_1 \text{ and } \operatorname{right}(n) \in S_2\}$.

1(a) Show that if S is expressible, then so is $S \operatorname{join} \bar{S}$.

1(b) Show that $S_1 \operatorname{join} S_2$ is an lub of S_1 and S_2 under $\leq_m$.

1(c) Show that if S is not decidable, then neither $S \operatorname{join} \bar{S}$ nor it's complement is checkable.

1(d) Give an example of an expressible set S such that neither S nor $\bar{S}$ is checkable.

Problem 2.

2(a) A set of numbers is ***nontrivial*** iff neither it nor its complement is empty. Prove that all nontrivial decidable sets are $\leq_m$ to each other.

2(b) Prove that a nonempty checkable set is undecidable iff the set and its complement are $\leq_m$-incomparable (that is, neither is $\leq_m$ the other).

2(c) Prove that the checkable sets are the closure of the decidable sets under "right projection", namely, the operation mapping a set $S \subseteq \mathbf{N}$ to $\operatorname{right}(S)$.

2(d) Prove that the expressible sets are the closure of the decidable sets under right projection and complement. ***Hint***: Every **Assn** is equivalent to an assertion of the form $(\neg)\exists i_1.(\neg)\exists i_2.\ldots(\neg)\exists i_n.a = 0$, where the parenthesized negations are optional.

Problem 3. For any set $S \subseteq \mathbf{N}$, let ρ_S be the function environment (*cf.* Problem Set 8) such that $\rho(F_0) = \mathrm{char}_S$ and $\rho(F_n)$ is a function with empty domain for $n > 1$ ($\mathrm{char}_S : \mathbf{N} \to \{0,1\}$ is the characteristic function of S, where $\mathrm{char}_S(n) = 1$ iff $n \in S$). A set S_1 is said to be *Turing reducible* to a set S_2, in symbols $S_1 \leq_T S_2$ iff char_{S_1} is ρ_{S_2}-computable. It is also suggestive to say that "S_1 is S_2-decidable" iff $S_1 \leq_T S_2$.

3(a) Show that $\bar{S} \leq_T S$ for any set S.

3(b) Explain why $\leq_T$ is transitive.

3(c) The self-halting problem *relative to* S is

$$H^{(S)} ::= \left\{ c \in \underset{fv}{\mathbf{Com}} \mid \{c\}_{\rho_S, X_1, X_1}(\#c) \downarrow \right\}.$$

Prove that $S <_T H^{(S)}$. *Hint*: Repeat the proof that the halting problem is undecidable, but for $\mathbf{Com}_{fv}$'s in function environment ρ_S.

Problem Set 9—Revised

Due: 7 December 1992

Reading assignment. Notes on Expressibility, Checkability, Decidability—Handout 40.

Two corrections are incorporated below: The definition of join in the earlier version of this problem set did not yield the lub claimed in problem 1(b). Also in problem 2(b), "nonempty" needed to replaced with "nontrivial."

Definition. A set of numbers is ***nontrivial*** iff neither it nor its complement is empty.

Problem 1. $S_1 \operatorname{join} S_2$ is defined to be

$$\{\text{mkpair}(1, n) \mid n \in S_1\} \cup \{\text{mkpair}(2, n) \mid n \in S_2\}.$$

1(a) Show that if S is expressible, then so is $S \operatorname{join} \bar{S}$.

1(b) Show that $S_1 \operatorname{join} S_2$ is an upper bound of S_1 and S_2 under $\leq_m$. Moreover, if S_1 and S_2 are nontrivial, then show it is a *least* upper bound of S_1 and S_2 under $\leq_m$.

1(c) Show that if S is not decidable, then neither $S \operatorname{join} \bar{S}$ nor it's complement is checkable.

1(d) Give an example of an expressible set S such that neither S nor $\bar{S}$ is checkable.

Problem 2.

2(a) Prove that all nontrivial decidable sets are $\leq_m$ to each other.

2(b) Prove that a nontrivial checkable set is undecidable iff the set and its complement are $\leq_m$-incomparable (that is, neither is $\leq_m$ the other).

2(c) Prove that the checkable sets are the closure of the decidable sets under "right projection", namely, the operation mapping a set $S \subseteq \mathbf{N}$ to right(S).

2(d) Prove that the expressible sets are the closure of the decidable sets under right projection and complement. ***Hint***: Every **Assn** is equivalent to an assertion of the form $(\neg)\exists i_1.(\neg)\exists i_2.\ldots(\neg)\exists i_n.a = 0$, where the parenthesized negations are optional.

Problem 3. For any set $S \subseteq \mathbf{N}$, let ρ_S be the function environment (*cf.* Problem Set 8) such that $\rho(F_0) = \text{char}_S$ and $\rho(F_n)$ is a function with empty domain for $n > 1$ ($\text{char}_S : \mathbf{N} \rightarrow \{0, 1\}$ is the characteristic function of S, where $\text{char}_S(n) = 1$ iff $n \in S$). A set S_1 is said to be ***Turing reducible*** to a set S_2, in symbols $S_1 \leq_T S_2$ iff char_{S_1} is ρ_{S_2}-computable. It is also suggestive to say that "S_1 is S_2-decidable" iff $S_1 \leq_T S_2$.

3(a) Show that $\bar{S} \leq_T S$ for any set S.

3(b) Explain why $\leq_T$ is transitive.

3(c) The self-halting problem *relative to* S is

$$H^{(S)} ::= \left\{ c \in \mathbf{Com}_{fv} \mid \{c\}_{\rho_S, X_1, X_1}(\#c) \downarrow \right\}.$$

Prove that $S <_T H^{(S)}$. ***Hint***: Repeat the proof that the halting problem is undecidable, but for $\mathbf{Com}_{fv}$'s in function environment ρ_S.

Outline: Lectures 1–36

1. (Fri, 9/11) Administrivia. Sample **IMP** while-program, Euclid, p.34; brief sketch of partial correctness and termination.

2. (Mon, 9/14) Syntax of **IMP** and "natural" evaluation semantics of for **Aexp**. Derivation tree for $\langle (M+N)\times N, \sigma[10/N][6/M]\rangle \to 160$.

3. (Wed, 9/16) Natural eval rules for **Com**. Derivation tree for $\langle$Euclid, $\sigma[10/\mathrm{N}][6/\mathrm{M}]\rangle \to \sigma[2/M][2/N]$. Uniqueness of derivation tree for each configuration; exists for **Aexp**, **Bexp**, and *while-free* **Com**, but $\langle \textbf{while true do}\, c, \sigma\rangle \not\to$ for all c, σ. No proofs.

4. (Fri, 9/18) One-step rules. Example $\langle \mathit{Euclid}, \sigma[10/N][6/M]\rangle \to_1^* \sigma[2/M][2/N]$. Remark: $\to_1$ is total, functional, computable relation. Inductive def of transitive closure. Statement of"equivalence" of one-step and natural rules: $\gamma \to_1^* \delta$ iff $\gamma \to \delta$ for all configurations γ and values $\delta \in \mathbf{N} \cup \mathbf{T} \cup \Sigma$.

5. (Mon, 9/21) Proof of equiv of natural and one-step semantics.

6. (Wed, 9/23) Proof by induction on deriv. of functionality of command evaluation (Winsk, 3.11). Proof by minimum principle that $\langle \textbf{while true do}\, c, \sigma\rangle \not\to$ (Winsk. 3.12).

7. (Fri, 9/25) Formal def of derivations, and induction on them (§3.4). Set R of rule instances determines a monotone, continuous, operator $\hat{R}$ on sets (§4.4) with derivable elements = $\mathit{fix}(\hat{R})$.

8. (Mon, 9/28) (Winskel §5.4) Def and examples of cpo's, monotone and continuous functions. Contrast with usual (epsilon-delta) continuity.

9. (Wed, 9/30) Examples of $\hat{R}_0(A)$ for $R_0 = \{\emptyset/3, \emptyset/4, \{n, n+1\}/n+2\}$. Proof that $\hat{R}$ is continuous. Proof of fixed points of continuous functions on cpo's.

10. (Fri, 10/2) QUIZ 1, IN CLASS, on lectures 1–8

11. (Mon, 10/5) Comments on Quiz 1. Discussion of wellformed and non-wellformed recursive function def's, eg, $e(x) = e(x+1)$, $f(x) = f(x+1)+1$, for g, h functions on ω^+: $g(1) = 1$; $g(x+y) = g(x)+g(y)$, $h(1) = 1$; $h(x+y) = h(x) + 2h(y)$. Function def by structural induction, eg, length and depth of a derivation, def of loc_L (§3.5) and statement w/o proof: c only effects $\mathrm{loc}_L(c)$ (Winskel 4.7). Brief mention of capturing computational behavior of recursive def's by choosing *least* partial functions satisfying constraints.

12. (Wed, 10/7) Motivation for fixed points as explanation of recursion: While-loops as fixed points of mappings on command meanings. Command meanings, C, will be partial functions $\in \Sigma \rightharpoonup \Sigma$ (meanings of expressions will be total functions from states to **Num** or **T**). Statement of equivalence of denotational and natural semantics: $\mathrm{Eval}(c) = \mathcal{C}[\![c]\!]$. Then define denotational semantics by structural induction assuming $\Gamma_{\mathbf{while}}$ (Winskel p.62) has a least fixed point.

13. (Fri, 10/9) Motivate $\Gamma_{\mathbf{while}}$ by considering $G : \mathbf{Com} \to C$ where $G(\mathbf{while})$ = unwind-once-**while**. Outline proof that Eval(**while**) is fixed point of Γ; observe that there may be other fixed points: every comand is fixed point of $\Gamma_{\mathbf{while\,true\,do\,skip}}$. State that Eval(**while**) is *least* fixed point and $\Gamma_{\mathbf{while}}$ has least fixed point because it is continuous on cpo C.

    (Mon, 10/12) COLUMBUS DAY

14. (Wed, 10/14) Properties like $\mathrm{Eval}((c_1; c_2)) = \mathrm{Eval}(c_2) \circ \mathrm{Eval}(c_1)$. Comments on proof of equivalence of natural and denotational semantics (Winskel Thm. 5.7): by structural induction, with subinduction for **while** case.

15. (Fri, 10/16) First-order arithmetic: **Assn**'s and their meaning. Assn's for "is prime," "divides," "lcm." Inductive def of free variables.

16. (Mon, 10/19) Formal def of $\sigma \models^I A$ (Winskel§6.3). Validity, satisfiability, invalidity.

17. (Wed, 10/21) Semantic of partial correctness; $\sigma \models \{\mathbf{true}\}c\{\mathbf{false}\}$ iff c diverges in state σ; A equiv $\{\mathbf{true}\}\,\mathbf{skip}\,\{A\}$. Sample axioms and rules of Hoare logic, mention soundness, hint about completeness and incompleteness.

18. (Fri, 10/23) Def of equivalent **Assn**'s. Lemma: A equiv B iff $\models (A \Rightarrow B)\ \&\ (B \Rightarrow A)$. Lemma: $\neg\forall j.A$ equiv $\exists j.\neg A$. Substitution Lemma: for expressions (WinskelLemma 6.8).

19. (Mon, 10/26) Substitution Lemma: for **Assn**'s (WinskelLemma 6.9). Prove validity of Hoare Assignment axiom. Lemma: A only depends on $FV(A)$ and loc(A). Lemma: If $j \notin FV(A)$, then $\forall j.(A \vee B)$ equiv $(\forall j.A) \vee (\forall j.B)$ equiv $A \vee (\forall j.B)$. Proofs omitted.

    EVENING QUIZ 2, Mon, 10/26, on lectures 9, 11–18

20. (Wed, 10/28) Soundness of inference versus antecedents implying consequent. Mention optional exercise: which Hoare rules are valid as implications. Informal soundness of Hoare rules and proof example: Euclid.

21. (Fri, 10/30) Soundness of Hoare loop invariant rule. Weakest preconditions and Dynamic Assertions. Translate dynamic assertions into assertions, assuming expressiveness.

22. (Mon, 11/2) Prove expressiveness of **Assn** for **IMP**. Corollary: Relative completeness of Hoare logic.

23. (Wed, 11/4) Thm: Every **Assn** equiv to prenex.[polynomial = 0]. Intro to rules for **Aexp** equations and sum-of-products polynomial representation of **Aexp**'s.

24. (Fri, 11/6) Notion of canonical form. Deriving enough equational axioms to put **Aexp**'s into sum-of-monomials form.

25. (Mon, 11/9) Canonical forms for **Aexp**'s as polynomials-with-multivariate-polynomial-coefficients. Proof that distinct canonical forms have distinct meanings by induction on number of variables. Completeness and decidability for polynomial equations.

    (Wed, 11/11) VETERAN'S DAY

26. (Fri, 11/13) Gödel numbers of **Assn**'s. $A_{p(m,n)} \equiv A_n[m/i_0]$ for expressible p. Nonexpressibility of **Truth** for **Assn**'s.

27. (Mon, 11/16) QUIZ 3, IN CLASS, on lectures 19–25

28. (Wed, 11/18) Complete proof of nonexpressibility of **Truth**. Define **IMP** checkable and state Lemma: Checkable implies expressible. Mention incompleteness.

29. (Fri, 11/20) DROP DATE & Underground Guide Survey. Prove Checkable implies Expressible using expressiveness. Define **IMP**-decidable proof system as having **IMP**-decidable proof relation. State Lemma: **IMP**-decidable proof system has **IMP**-checkable set of provable Assn's, so **Provable** $\neq$ **Truth**.

30. (Mon, 11/23) Def of computable, decidable. Remark: **IMP**-computable same as $\mathbf{IMP}_r$-computable by Handout—but not obvious. Decidable implies checkable. Thm: D decidable and f total computable implies f(D) checkable. Cor: Provable assertions are checkable.

31. (Wed, 11/25) Vocabulary: Checkable = r.e., decidable = recursive, computable = (partial) recursive. Decidable closed under intersection, complement. Mention dovetailing, f(r.e.) is r.e. Discussion of thesis that Effectively decidable = **IMP**-decidable. The set of (Gödel numbers of) sentences is a decidable set; likewise, the set of commands. Incompleteness Theorem: In a sound proof system, there is a true sentence which is not provable.

    (Fri, 11/27) THANKSGIVING HOLIDAY

32. (Mon, 11/30) Uncheckability of $\bar{H}$ where H is the self-halting problem, so undecidability of H. Venn diagram of decidable, checkable, co-checkable, expressible. Decidable iff checkable and co-checkable. Universal **IMP** command, u. Checkability of halting problem.

33. (Wed, 12/2) Uncheckability of zero-halting problem (by reduction). $\leq_m$ and Rice's Theorem.

34. (Fri, 12/4) Incompleteness of substitution instances of the single **Assn** $W(\mathbf{false}, c_H)[n/X_1]$. Hilbert's 10th. Mention other undecidable problem: semigroup word problem; tiling problem (no time to mention zero matrix product problem; CFG equivalence and ambiguity problem, CSG emptiness).

35. (Mon, 12/7) $\mathbf{IMP}_{\mathrm{par}}$ala Brookes. Noncompositionality of state transition semantics. Def of observational congruence. X:=X not cong skip; some other identities do hold.

36. (Wed, 12/9) Compositional "interrupt sequence" semantics "interrupt sequences" fully abstract when n-ary-test-and-set is added to $\mathbf{IMP}_{\mathrm{par}}$. Comments on what was not covered: higher-order-IMP. Follow-up courses: 6.821 (programming linguistics and semantics), 6.830 (research in logic and semantics of programs), 6.840 (computability and complexity), 6.826 (Systems modelling and specification), Math and Philosophy courses in Logic.

    (Thu, 12/17) (Exam Period) QUIZ 4, 1:30–3:30 in du Pont, on lectures 26, 28–36

Problem Set 7 Solutions

A *sublogic* of Assn is a set of Assn's that is closed under Boolean combination, quantification, and substituting numbers or integer variables for integer variables or locations. For certain sublogics (an example will appear soon), there is a computational "quantifier elimination" procedure that will transform any Assn, A, in the sublogic into an equivalent *quantifier-free* Assn, $\widehat{A}$, in the sublogic.

Problem 1.

1(a) If a sublogic has a quantifier-elimination procedure. it can be used to develop procedures to decide

- **(i) given a formula A of the sublogic. and the values of $\sigma(X)$ for $X \in \mathrm{loc}(A)$ and $I(j)$ for $j \in \mathrm{FV}(A)$, whether $\sigma \models^I A$.**
- **(ii) whether or not a formula of the sublogic is valid, and**
- **(iii) whether or not two formulas of the sublogic are equivalent.**

Describe procedures for (i), (ii) and (iii) assuming quantifier-elimination.

(i) An **Assn** without quantifiers is (equivalent to) a **Bexp**. (The only differences are expressions of the form $A \Rightarrow B$, which can be replaced with the equivalent $\neg A \vee B$.) Thus, given a quantifier-elimination procedure, we can apply it to remove all the quantifiers in an expression, replace all variables and locations with their values in σ, and apply the one-step evaluation rules to evaluate the resulting (variable-free) **Bexp**.

(A subtlety that had some people worried was the possibility that the quantifier-elimination procedure would introduce new free variables i for which we wouldn't have the appropriate values $I(i)$. Note, however, that a simple induction on the structure of an **Assn** shows that the truth of the resulting **Assn** can't depend on the values of any new variables. Thus, after eliminating all quantifiers, we can replace all new variables with 0. This gives us a quantifier-elimination procedure that doesn't introduce any new variables, so, without loss of generality, we can assume that our given QE procedure doesn't either.)

(ii) Three (easily proven) facts will aid us here:

1. As noted in Quiz 2 (handout 30), an **Assn**, A, is valid iff the $\forall j.A$ is valid.
2. An **Assn** A with a location X is valid iff the **Assn** $A[i/X]$ is valid, where $i \notin \mathrm{FV}(A)$.
3. An **Assn** A with no free variables is valid iff it is true in some state and interpretation (since its truth value must be the same in any state and interpretation).

Thus, to determine the validity of an assertion A in a sublogic with quantifier elimination, first replace all its locations with fresh variables, then universally quantify over all free variables in the new term, then pick any state and interpretation and apply the procedure given in step (i) to determine the truth of the assertion in the chosen state and interpretation.

(iii) For two assertions A and B, use the procedure in step (ii) to determine the validity of $(A \Rightarrow B) \wedge (B \Rightarrow A)$.

1(b) Suppose for some sublogic that there is an "innermost existential-elimination" procedure that, for any integer variable j and *quantifier-free* Assn, A, in the sublogic, constructs another quantifier-free formula, $\mathcal{E}(j, A)$, in the sublogic, such that $\mathcal{E}(j, A)$ is equivalent to $\exists j.A$. Using innermost existential-elimination as a subprocedure, there is a straightforward recursive definition of a quantifier-elimination procedure for the sublogic. Describe it.

Replace each universally quantified subexpression $\forall j.A$ with the equivalent negated existentially quantified expression $\neg\exists j.\neg A$. Then, working from the innermost expressions outward, replace each existentially quantified expression $\exists j.A$ with $\mathcal{E}(j, A)$.

Define *incremental arithmetic expressions*, IncAexpv, to be Aexpv's without the multiplication or subtraction operations and with addition restricted to incrementing by an integer. Namely, the grammar for $a \in$ IncAexpv is

$$a ::= n \mid i \mid X \mid a + n \mid n + a$$

Let IncAssn be the set of Assn's all of whose arithmetic subexpressions are IncAexpv's. Note that IncAssn is a sublogic of Assn.

Define a *simple* incremental assertion to be a finite conjunction of assertions of one of the three forms

$$n \leq v, \quad v \leq n, \quad v_1 + n \leq v_2$$

where $n \in$ Num, $v, v_1, v_2 \in$ Loc $\cup$ Intvar, and v_1 and v_2 are not the same.

Problem 2.

2(a) Describe a special case of an innermost existential-elimination procedure which works just for simple assertions. That is, if A is a simple assertion, then so will be $\mathcal{E}(j, A)$.

Let A be a simple assertion. Adjust every inequality in A so that j appears (positive and) alone on one or the other side of its $\leq$-sign. Let $\mathcal{E}(j, A)$ be the conjunction of all inequalities $a_0 \leq a_1$ such that:

- $a_0 \leq a_1$ appears in A and $j \notin \mathrm{FV}(a_0 \leq a_1)$, or
- $a_0 \leq j$ and $j \leq a_1$ appear in the adjusted A.

(Note that it is possible for $\mathcal{E}(j, A)$ to be an empty conjunction. We will treat this as **true**.)

2(b) Describe a procedure that will transform any quantifier-free assertion $A \in$ IncAssn into an equivalent formula $\mathcal{S}(A)$ that is a finite disjunction of simple assertions.

Given such an assertion A, perform the following modifications:

1. Change all subassertions $A_0 \Rightarrow A_1$ to $\neg A_0 \vee A_1$.
2. Iterate DeMorgan's law as often as necessary to change all $\neg(A_0 \wedge A_1)$ to $\neg A_0 \vee \neg A_1$ and all $\neg(A_0 \vee A_1)$ to $\neg A_0 \wedge \neg A_1$.
3. Change all $\neg\neg A_0$ to A_0..
4. Change all $\neg(a_0 = a_1)$ to $((a_0 \leq a_1) \wedge \neg(a_1 \leq a_0)) \vee ((a_1 \leq a_0) \wedge \neg(a_0 \geq a_1))$.
5. Change all $a_0 = a_1$ to $(a_0 \leq a_1) \wedge (a_1 \leq a_0)$.
6. Change all $\neg(a_0 \leq a_1)$ to $a_1 + 1 \leq a_0$.

These steps will reduce an assertion to conjunctions, disjunctions and inequalities. Note that, by the form of an **IncAssn**, the arithmetic expression on either side of an inequality can consist at most of a sum of integers and, possibly, a single variable or location, so we can easily reduce any inequality to the proper form for simple assertions. (To reduce an inequality of the form $n_0 \leq n_1$, replace it with $(X \leq 0) \vee (0 \leq X)$ for any location X if the inequality is true, and $(X \leq 0) \wedge (1 \leq X)$ if the inequality is false.)

Finally, distribute all $\wedge$'s according to the law $A \wedge (B \vee C) \equiv (A \wedge B) \vee (B \wedge C)$, and we are left with a finite disjunction of simple assertions.

2(c) Explain how to combine the procedures of problems 2(a) and 2(b) to obtain an innermost existential-elimination procedure for every quantifier-free IncAssn, not just simple ones.

To eliminate an inner quantifier $\exists j.A$, apply the procedure in 2(b) to convert A to a disjunction of simple assertions, then distribute the quantifier down to the simple assertions using the rule $\exists j.(A \vee B) \equiv (\exists j.A) \vee (\exists j.B)$, then apply the existential-elimination procedure for simple assertions to each of the quantified subexpressions in the resulting disjunction.

Problem 3. The preceding problems provide a computational procedure for determining validity and equivalence of IncAssn's. The procedure is total—it is guaranteed to return the correct answer on every IncAssn—though there are several stages where huge computations may be needed. Describe briefly how this procedure could be programmed as a procedure definition F, in, say, Scheme, so *e.g.*, $(F\ \tilde{A})$ evaluates to `t` or `nil` according to whether or not $A \in$ IncAssn is valid, where $\tilde{A}$ is some straightforward representation of A as an S-expression. Indicate which stages or subprocedures of the computation may be the source of time-consuming (quadratic, exponential, ..., growth) subcomputations.

Let n be the length of the expression A, according to some reasonable measure. To determine validity,

1. Add universal quantifiers to A (replacing locations with variables) as described in 1(a), part ii. This will take time proportional to n.
2. Apply the procedure in 2(c) as described in 1(b) to eliminate the quantifiers. For each quantifier, this can take time exponential in n. (If you have trouble seeing this, consider that the task of distributing the $\wedge$'s over the $\vee$'s can approximately double the size of the expression at each step.)
3. Given the resulting quantifier- (and location- and variable-) free assertion, evaluate it. This will take time approximately linear in the size of the new, possibly exponentially larger expression.

To determine equivalence of two expressions, A and B, apply the same procedure to $(A \Rightarrow B) \wedge (B \Rightarrow A)$ as described in 1(a), part iii.

Problem 4. Show that there is no IncAssn which means "j is even." Conclude that IncAssn's are *not* expressive for the set of IMP commands whose arithmetic subexpressions are restricted to be IncAexp's.

By the preceding problems, it is enough to show that no disjunction of simple assertions with only free variable j can mean "j is even."

Assume, for the sake of contradiction, that there is such an expression, A. Say that A is a disjunction of n conjunctions, A_1 through A_n. Now, for any even x it must be the case that all of the A_i are false for $j = x - 1$ and at least one A_i is true for $j = x$. Since there is an infinite set of even numbers, and only finite A_i, there must be at least one A_i that is false at $j = x - 1$ and true at $j = x$ for an infinite collection of x's. Consider that A_i, and its subterms B_1 through B_m for some m.

By a similar argument, since A_i is a finite conjunction of assertions B_k, there must be some B_k that is false for $j = x - 1$ and true for $j = x$ for an infinite number of integers x. But each B_i is either $j \le k$ or $k \le j$ for some integer k, so this is impossible. Thus, there is no **IncAssn** expressing "j is even."

However, even if we restrict our commands to use **IncAexp**'s, we can generate the command

$$c ::= \; Y := X; \mathbf{while}\, \neg(X = 1) \wedge \neg(Y = 1)\, \mathbf{do}\, X := X + 2; Y := Y - 2$$

But, since c terminates in σ exactly when $\sigma(X)$ is odd, the weakest precondition $W(c, \mathbf{false})$ is exactly "X is even," which is not expressible in **IncAssn**. Thus **IncAssn**is not expressive for the set of **IMP** commands whose arithmetic expressions are restricted to be **IncAexp**'s.

Problem 5. Define ResetCom to be the set of IMP commands whose Aexp's are restricted to be of the form $n \in$ Num or $X \in$ Loc: define ResetAssn's likewise.

5(a) Show that every IncAssn is equivalent to a ResetAssn.

(Note that "defining **ResetAssn**'s likewise" must include allowing integer variables as well as locations in the assertions.)

By the previous problems, for any **IncAssn**, A, there is an equivalent assertion A' which a finite disjunction of simple assertions. Since $n \le X$ and $X \le n$ are already **ResetAssn**'s, it remains only to show that there is a **ResetAssn** equivalent to any expression of the form $X + n \le Y$.

$X + 0 \le Y$ is equivalent to $X \le Y$, which is a **ResetAssn**.

$X + 1 \le Y$ is equivalent to $(a_0 < a_1) \wedge \neg(a_0 = a_1)$, which is a **ResetAssn**. In general, then, if n is positive, then $X + n \le Y$ is equivalent to

$$(X + 1 \le i_1) \wedge (i_1 + 1 \le i_2) \wedge \ldots \wedge (i_{n-1} + 1 \le Y)$$

which is thus equivalent to a **ResetAssn**.

Finally, $X + (-1) \leq Y$ is equivalent to $\forall j.((j+1 \leq X) \Rightarrow (j \leq Y))$, which is equivalent to a **ResetAssn**. In general, then, if n is negative, then $X + n \leq Y$ is equivalent to

$$(X + (-1) \leq i_1) \wedge (i_1 + (-1) \leq i_2) \wedge \ldots \wedge (i_{n-1} + (-1) \leq Y)$$

which is thus equivalent to a **ResetAssn**.

Let diverge be while true do skip. An IMP command which has no occurrences of while-loops other than diverge is said to be *while-free*. We state the following

Lemma. If $c \in$ ResetCom, then $c \sim c'$, for some while-free $c' \in$ ResetCom.

5(b) Assuming the Lemma, show that ResetAssn is expressive for ResetCom.

As in Handout 32, we can define the weakest precondition $W(c, A)$ inductively as follows:

$$\begin{aligned}
W(\mathbf{skip}, A) &::= A, \\
W(X := a, A) &::= A[a/X], \\
W((c_1; c_2), A) &::= W(c_1, W(c_2, A)), \\
W(\mathbf{if}\, b\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2, A) &::= (b \Rightarrow W(c_1, A)) \wedge (\neg b \Rightarrow W(c_2, A)), \\
W(\mathbf{diverge}, A) &::= \mathbf{true}
\end{aligned}$$

Thanks to the lemma, these rules are sufficient to define $W(c, A)$ for (a command equivalent to) any command c. That $W(c, A)$ is the weakest precondition follows from the proofs in Handout 32. That each is a **ResetAssn** (assuming A is) follows immediately from the definition of **ResetAssn**'s. The only interesting case is that of $W(X := a, A)$, which relies on the fact that a must be a location or integer; thus, $A[a/X]$ must still be a **ResetAssn**.

5(c) Conclude that the restriction of Hoare logic to ResetAssn's and ResetCom's is a complete proof system *whose proofs are computationally checkable.*

(The following is an outline of the proof. More detail wouldn't be amiss.)

Consider the relative completeness proof in Handout 32. There we showed that, given proofs of any given valid implication, the Hoare rules would give us a proof of any valid partial correctness assertion.

First, we can verify the relative completeness of the restricted system by inspection of the more general relative completeness result; we just verify that a proof of a **ResetAssn**-based partial correctness assertion about **ResetCom**'s relies only on the proofs of valid implications $A \Rightarrow B$ in **ResetAssn** and the (shorter) proofs of other **ResetAssn**-based p.c.a.'s.

Given relative completeness, we can get completeness by simply adding all valid **ResetAssn** implications to the proof system as axioms. Since, by 5(b), the validity of any **ResetAssn** can be computationally checked, and each step of a Hoare-logic proof is easily checkable, we have our complete, checkable proof system.

Problem Set 8 Solutions

Problem 1. Describe how to transform a command c_1 into one which meets the description "do c_1 for S steps or until c_1 halts (whichever happens first)" (Winskel, Exercise A.6). ***Hint*: Any convenient notion of step will serve for the purposes of this problem—*not* necessarily the $\rightarrow_1$version of step, which will probably be inconvenient here. The only properties "steps" must have is that (1) for any command, c, it is IMP-decidable, given n and the values in a state of $\mathrm{loc}(c)$, whether c halts in $\leq n$ steps in that state, and (2) c halts in a state iff it halts in a finite number of steps. A convenient notion of step satisfying these criteria is the number of executions of while-loop bodies, which can be counted by inserting assignments $S := S - 1$ at the beginning of every while-loop body and suitably modifying while-loop guards.**

Using the notion "step" from the hint (though with a slightly different implementation method), we can define a transformation on commands by structural induction, as follows. Assuming that S is unused in c_1, define the transformed program $\widehat{c_1}$ by:

$$\begin{aligned}
\widehat{\mathbf{skip}} &\stackrel{\text{def}}{=} \mathbf{skip} \\
\widehat{X := a} &\stackrel{\text{def}}{=} \mathbf{if}\, 1 \leq S\, \mathbf{then}\, X := a\, \mathbf{else}\, \mathbf{skip} \\
\widehat{c_0; c_1} &\stackrel{\text{def}}{=} \widehat{c_0}; \widehat{c_1} \\
\widehat{\mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1} &\stackrel{\text{def}}{=} \mathbf{if}\, b\, \mathbf{then}\, \widehat{c_0}\, \mathbf{else}\, \widehat{c_1} \\
\widehat{\mathbf{while}\, b\, \mathbf{do}\, c} &\stackrel{\text{def}}{=} \mathbf{while} (b \wedge 1 \leq S)\, \mathbf{do}\, \widehat{c}; S := S - 1
\end{aligned}$$

Problem 2. Show that if $g : \mathrm{N}^3 \rightharpoonup \mathrm{N}$ and $f_i : \mathrm{N}^2 \rightharpoonup \mathrm{N}$ are IMP-computable partial functions for $i = 1, 2, 3$, then so is $h : \mathrm{N}^2 \rightharpoonup \mathrm{N}$ where

$$h(m_1, m_2) = g(f_1(m_1, m_2), f_2(m_1, m_2), f_3(m_1, m_2)).$$

For convenience let functions be computed with tidy commands as in the appendix. Thus, let g be $\{c_g\}$ and let each f_i be $\{c_i\}$ for tidy commands c_g, c_1, c_2, c_3. Further, let X'_1, X'_2, Y_1, Y_2 be locations untouched by c_g or any c_i. Then h is $\{c_h\}$, where c_h is the tidy command

$$\begin{aligned}
&X'_1 := X_1; X'_2 := X_2; c_1; Y_1 := X_1; \\
&X_1 := X'_1; X_2 := X'_2; c_2; Y_2 := X_1; \\
&X_1 := X'_1; X_2 := X'_2; c_3; \\
&X_3 := X_1; X_2 := Y_2; X_1 := Y_1; c_g; \\
&X'_1 := 0; X'_2 := 0; Y_1 := 0; Y_2 := 0;
\end{aligned}$$

In words, the input values are saved, then each c_i command in turn is executed on those input values and the return values are saved; then the return values from the c_i's are all passed to c_g, and then the command tidies up after itself.

Problem 3. Show that if g and h are IMP-computable total functions, then so is f where

$$f(n,m) = \begin{cases} g(m) & \text{for } n \leq 0, \\ h(n-1, m, f(n-1,m)) & \text{for } n > 0. \end{cases}$$

Again, given tidy commands c_g and c_h and fresh variables C, N and M, we can produce c_f as follows:

$$\begin{array}{l} C := 1; N := X_1; M := X_2; \\ X_1 := M; X_2 := 0; c_g; \\ \textbf{while}(C \leq N)\,\textbf{do} \\ \quad X_3 := X_1; X_1 := C; X_2 := M; c_h; \\ \quad C := C + 1; \\ C := 0; N := 0; M := 0 \end{array}$$

In the next problem we formalize the idea that if a function is computable by an IMP command, then it can be used freely in Aexp's to define further IMP-computable functions. Though it won't come up on this problem set, this same formal machinery will later be useful for defining computability relative to arbitrary functions on numbers—even if these functions are not computable—and it will also be useful if and when we add function declarations to IMP.

Let Funcvar be a set whose elements, F, are called *function variables.* Associated with each $F \in$ Funcvar is a nonnegative integer called arity(F). Let Aexp$_{fv}$ to be Aexp with one more clause in its grammar, namely,

$$a ::= F(a_1, \ldots, a_{\text{arity}(F)}).$$

The meaning of an Aexp$_{fv}$ depends on the values of its function variables. It will be helpful technically to use the bijective coorespondence between partial functions $f : \mathbf{N}^n \to \mathbf{N}$ and the strict total functions $f_\perp : (\mathbf{N}_\perp)^n \to \mathbf{N}_\perp$, where f and $f_\perp$ agree on all n-tuples of arguments in N for which f is defined, and $f_\perp$ is $\perp$ otherwise. A *function environment* will be a mapping, ρ, from function variables to functions on $\mathbf{N}_\perp$ which "respects arity", that is,

$$\rho(F) : (\mathbf{N}_\perp)^{\text{arity}(F)} \to \mathbf{N}_\perp$$

for all $F \in$ Funcvar. Let R be the set of function environments. The domain, A, of $\mathbf{Aexp}_{fv}$ meanings will be $\mathrm{R} \to \Sigma_\perp \to \mathrm{N}_\perp$, with the bottom value reflecting the fact that functions computed by commands are partial because of command divergence.

To define meaning of $\mathbf{Aexp}_{fv}$'s we insert ρ's into the defining clauses for Aexp's and add one more straightforward clause:

$$\mathcal{A}[\![F(a_1,\ldots,a_n)]\!]\rho\sigma = \rho(F)(\mathcal{A}[\![a_1]\!]\rho\sigma,\ldots,\mathcal{A}[\![a_n]\!]\rho\sigma)$$

where $n = \mathrm{arity}(F)$.

Let $\mathbf{IMP}_{fv}$ be the extension of IMP which uses $\mathbf{Aexp}_{fv}$'s instead of Aexp's. Note that the domain C of command meanings is now $\mathrm{R} \to (\Sigma_\perp \to \Sigma_\perp)$.

For any $\rho \in \mathrm{R}$, $c \in \mathbf{Com}_{fv}$, and sequence $\vec{Y}_{n+1} = Y_1,\ldots,Y_{n+1} \in$ Loc, where the first n Y's are distinct, we let

$$\{c\}_{\rho,\vec{Y}_{n+1}} : (\mathrm{N}_\perp)^n \to \mathrm{N}_\perp$$

be the *strict* function on numbers computed by c when function variables are interpreted according to ρ, arguments $\vec{m}_n \in \mathrm{N}_\perp$ are placed successively in locations $\vec{Y}_n$, and the answer is left in location Y_{n+1}. That is, let σ_0 be the state mapping all locations to zero; then

$$\{c\}_{\rho,\vec{Y}_{n+1}}(\vec{m}_n) = \begin{cases} \sigma(Y_{n+1}), & \text{if } \vec{m}_n \in \mathrm{N} \text{ and } \mathcal{C}[\![c]\!]\rho(\sigma_0[\vec{m}_n/\vec{Y}_n]) = \sigma \neq \perp_\Sigma, \\ \perp_\mathrm{N} & \text{otherwise.} \end{cases}$$

We say a function $f : (\mathrm{N}_\perp)^n \to \mathrm{N}_\perp$ is *ρ-computable* iff $f = \{c\}_{\rho,\vec{Y}_{n+1}}$ for some $c \in \mathbf{IMP}_{fv}$ and locations $\vec{Y}_{n+1}$. We also say a partial function $f : \mathrm{N}^n \rightharpoonup \mathrm{N}$ is ρ-computable iff $f_\perp$ is.

It is not hard to see, and you may assume, that a partial function f is IMP-computable as defined in Winskel, §A.1 iff f is ρ-computable by some $c \in \mathbf{Com}_{fv}$ which contains no occurrences of function variables, that is, c is an ordinary Com.

Problem 4.

4(a) If the value of a function variable, F, is already computable from the other function variables, then there is no need for it commands. Namely, suppose that $\rho(F) = \{c_F\}_{\rho,\vec{Y}_{n+1}}$ for some $c_F \in \mathbf{Com}_{fv}$ which contains no occurrences of F. Show that then *every* ρ-computable function is computable by a $\mathbf{Com}_{fv}$ which contains no occurrences

of F by explaining how to transform any $c \in \mathbf{Com}_{fv}$ into a c' without F such that $\{c'\}_{\rho,\vec{Y'}_{n+1}} = \{c\}_{\rho,\vec{Y}_{n+1}}$.

Given any $c \in \mathbf{Com}_{fv}$, the following inductive definition will provide a (very inefficient) translation $\widehat{c}$ such that F does not occur in $\widehat{c}$.

First, let c'_F be c_F with all locations renamed to be distinct from any locations in c. In particular, let values be passed to and from the command via new variables $Y_{F,1}$ through $Y_{F,n+1}$.

Next, to each subexpression a_0, b_0 or c_0 of c, associate a distinct variable X_{a_0}, X_{b_0}, etc. such that

1. all X_{a_0}, X_{b_0}, etc. are distinct from every variable from in c or c'_F;
2. every occurence of an **Aexp**, **Bexp** or **Com** in c has its own, distinct variable.

In particular, note that two distinct occurences of, say, "$X := 3$" within c will be associated with distinct variables. (You can think of it as assigning a fresh variable to each node on the parse tree of c.) The structure of the definitions should protect us from any ambiguity.

Now, define the translation of an $\mathbf{Aexp}_{fv}$, a, as follows:

- If $a \equiv n$ then $\widehat{a} \stackrel{\text{def}}{=} (X_a := n)$
- If $a \equiv X$ then $\widehat{a} \stackrel{\text{def}}{=} (X_a := X)$
- If $a \equiv a_0 \text{ op } a_1$ then $\widehat{a} \stackrel{\text{def}}{=} (\widehat{a_0}; \widehat{a_1}; X_a := X_{a_0} \text{ op } X_{a_1})$
- If $a \equiv F'(a_1, a_2, \ldots, a_n)$ for some function variable F' distinct from F, then
$$\widehat{a} \stackrel{\text{def}}{=} (\widehat{a_1}; \cdots; \widehat{a_n}; X_a := F'(X_{a_1}, \ldots, X_{a_n}))$$
- If $a \equiv F(a_1, a_2, \ldots, a_n)$ then
$$\widehat{a} \stackrel{\text{def}}{=} (\widehat{a_1}; Y_{F,1} := X_{a_1}; \cdots; \widehat{a_n}; Y_{F,n} := X_{a_n}; c'_F; X_a := Y_{F,n+1})$$

In English, this translation gives, for every sub-**Aexp** a in c, a corresponding command that will leave the value of a in X_a.

A similar translation exists for **Bexp**'s, leaving a value 0 in X_b if b would evaluate to **false** and 1 otherwise:

- If $b \equiv \textbf{true}$ then $\widehat{b} \stackrel{\text{def}}{=} (X_b := 1)$

- If $b \equiv \textbf{false}$ then $\widehat{b} \stackrel{\text{def}}{=} (X_b := 0)$
- If $b \equiv b_0 \ \textbf{op} \ b_1$ then

$$\widehat{b} \stackrel{\text{def}}{=} (\widehat{b_0}; \widehat{b_1}; \textbf{if}(X_{b_0} = 1 \ \textbf{op} \ X_{b_1} = 1) \ \textbf{then} \ X_b := 1 \ \textbf{else} \ X_b := 0)$$

- If $b \equiv a_0 \ \textbf{op} \ a_1$ then

$$\widehat{b} \stackrel{\text{def}}{=} (\widehat{a_0}; \widehat{a_1}; \textbf{if}(X_{a_0} \ \textbf{op} \ X_{a_1}) \ \textbf{then} \ X_b := 1 \ \textbf{else} \ X_b := 0)$$

Finally, we can translate commands:

- If $c \equiv \textbf{skip}$ then $\widehat{c} \stackrel{\text{def}}{=} \textbf{skip}$
- If $c \equiv X := a$ then $\widehat{c} \stackrel{\text{def}}{=} (\widehat{a}; X := X_a)$
- If $c \equiv c_0; c_1$ then $\widehat{c} \stackrel{\text{def}}{=} (\widehat{c_0}; \widehat{c_1})$
- If $c \equiv \textbf{if} \ b \ \textbf{then} \ c_0 \ \textbf{else} \ c_1$ then $\widehat{c} \stackrel{\text{def}}{=} (\widehat{b}; \textbf{if} \ X_b = 1 \ \textbf{then} \ \widehat{c_0} \ \textbf{else} \ \widehat{c_1})$
- If $c \equiv \textbf{while} \ b \ \textbf{do} \ c$ then $\widehat{c} \stackrel{\text{def}}{=} (\widehat{b}; \textbf{while} \ X_b = 1 \ \textbf{do}(\widehat{c}; \widehat{b}))$

4(b) A function environment, ρ, is *computable* if $\rho(F)$ is IMP-computable for all $F \in$ Funcvar. Conclude that if ρ is *computable*, then every ρ-computable function is IMP-computable.

By part (a), if every function that an $\textbf{IMP}_{fv}$-command refers to is computable, then all function variables can be "compiled out," leaving plain **IMP** code which computes the same function. Thus, any computable function is **IMP**-computable.

Problem Set 9 Solutions

(Note that a couple of corrections were made in the problem set after it was initially handed out. These solutions are based on the revised problem set.)

Problem 1. $S_1\,\mathrm{join}\,S_2$ **is defined to be**

$$\{\mathrm{mkpair}(1,n) \mid n \in S_1\} \cup \{\mathrm{mkpair}(2,n) \mid n \in S_2\}.$$

1(a) Show that if S is expressible, then so is $S\,\mathrm{join}\,\bar{S}$.

From earlier problem sets and class discussion we know that we can construct an **Assn** E_{left} which is true iff $i_1 = \mathrm{left}(i_0)$ and similarly for E_{right}. Now, if S is expressible, then there is some **Assn** E_S which is true iff $i_0 \in S$. Then $S\,\mathrm{join}\,\bar{S}$ is expressible as

$$\begin{array}{l}((\exists i_1.E_{\mathrm{left}} \wedge i_1 = 1) \wedge (\exists i_1.E_{\mathrm{right}} \wedge E_S[i_1/i_0]))\vee \\ ((\exists i_1.E_{\mathrm{left}} \wedge i_1 = 2) \wedge (\exists i_1.E_{\mathrm{right}} \wedge \neg E_S[i_1/i_0]))\end{array}$$

1(b) Show that $S_1\,\mathrm{join}\,S_2$ is an upper bound of S_1 and S_2 under $\leq_m$. Moreover, if S_1 and S_2 are nontrivial, then show it is a *least* upper bound of S_1 and S_2 under $\leq_m$.

The functions $\mathrm{in}_\ell(m) \stackrel{\mathrm{def}}{=} \mathrm{mkpair}(1,m)$ and $\mathrm{in}_r(m) \stackrel{\mathrm{def}}{=} \mathrm{mkpair}(2,m)$ are clearly computable and total, and have the property that $m \in S_1$ iff $\mathrm{in}_\ell(m) \in S_1\,\mathrm{join}\,S_2$, and similarly for S_2 and in_r. Thus $S_1 \leq_m S_1\,\mathrm{join}\,S_2$ and $S_2 \leq_m S_1\,\mathrm{join}\,S_2$.

Now, assume S_1 and S_2 are nontrivial. To see that $S_1\,\mathrm{join}\,S_2$ is a *least* upper bound, consider any S such that $S_1 \leq_m S$ and $S_2 \leq_m S$. S must be nontrivial (why?) so there is some integer $m \notin S$. Now, by definition, there are a total computable f_1 and f_2 such that $n \in S_1$ iff $f_1(n) \in S$ and similarly for f_2. But then, we have the function

$$f(n) = \begin{cases} f_1 \circ \mathrm{right}(n) & \text{if } \mathrm{left}(n) = 1 \\ f_2 \circ \mathrm{right}(n) & \text{if } \mathrm{left}(n) = 2 \\ m & \text{otherwise} \end{cases}$$

which is clearly total and computable, with the property that $n \in S_1\,\mathrm{join}\,S_2$ iff $f(n) \in S$. Thus $S_1\,\mathrm{join}\,S_2$ is a least upper bound.

1(c) Show that if S is not decidable, then neither $S \operatorname{join} \bar{S}$ nor it's complement is checkable.

Assume S undecidable. Then if $S \operatorname{join} \bar{S}$ is checkable then, since checkability inherits downwards, so are both S and $\bar{S}$. But if S and $\bar{S}$ are checkable, S is decidable, which is a contradiction, so $S \operatorname{join} \bar{S}$ is not checkable. Further, by Lemma 4 on Handout 40, $\overline{S \operatorname{join} \bar{S}}$ is also an upper bound on S and $\bar{S}$, so the same argument applies.

1(d) Give an example of an expressible set S such that neither S nor $\bar{S}$ is checkable.

Let K be the "halting problem," as defined in class. We know the set K is undecidable but expressible, so, by the steps above, the set $S = K \operatorname{join} \bar{K}$ is expressible but neither S nor $\bar{S}$ is checkable.

Problem 2. A set of numbers is *nontrivial* iff neither it nor its complement is empty.

2(a) Prove that all nontrivial decidable sets are $\leq_m$ to each other.

If S is decidable then char_S is computable. Take any nontrivial set S', and let n and m be such that $n \in S'$ and $m \notin S'$. Then, since char_S is computable, so is

$$f(x) = \begin{cases} n & \text{if } \mathrm{char}_S(x) = 1 \\ m & \text{if } \mathrm{char}_S(x) = 0 \end{cases}$$

which is a many-to-one reduction from S to S'.

2(b) Prove that a nontrivial checkable set is undecidable iff the set and its complement are $\leq_m$-incomparable (that is, neither is $\leq_m$ the other).

By Lemma 4 on Handout 40, $S \leq_m \bar{S}$ iff $\bar{S} \leq_m S$. If S is decidable, then $\bar{S}$ is also decidable, so, by the previous problem, $S \leq_m \bar{S}$. If S is undecidable, then, since S is checkable, $\bar{S}$ is not. Since checkability inherits downward, $\bar{S} \not\leq_m S$, so $S \not\leq_m \bar{S}$.

2(c) Prove that the checkable sets are the closure of the decidable sets under "right projection", namely, the operation mapping a set $S \subseteq \mathbf{N}$ to right(S).

This follows almost immediately as a corollary to Theorem 2 of Handout 40. The proof of that theorem shows how to construct any checkable set as the left projection of a decidable set. Reversing left() and right() in that construction shows that any checkable set is also the right projection of some decidable set. Conversely, since right() is total and computable, the theorem states that any such right projection of a decidable set is checkable.

2(d) Prove that the expressible sets are the closure of the decidable sets under right projection and complement.)

Let $\mathcal{C}$ be the closure of the decidable sets under complement and right projection, and let $\mathcal{E}$ be the expressible sets.

First, $\mathcal{C} \subseteq \mathcal{E}$ because every decidable set is checkable and therefore expressible, and because expressible sets are obviously closed under complement and right projection. (In particular, if A with free variable i_0 expresses some set S, then

$$\exists i_1.\text{“}i_0 = \text{right}(i_1)\text{”} \wedge A[i_1/i_0]$$

expresses right(S).)

Conversely, we may suppose S is expressible by an **Assn** of the given form. For $k \in \mathrm{N}, 0 < i$, define $(k)_1 = \text{left}(k)$ and $(k)_{i+1} = (\text{right}(k))_i$. So we have for $1 \leq i \leq m$

$$(\ \text{mkpair}(k_1, \text{mkpair}(k_2, \ldots, \text{mkpair}(k_{m-1}, \text{mkpair}(k_m, j)) \ldots))\ )_i = k_i.$$

Let

$$D_n ::= \{k \in \mathrm{N} \mid \models (a = 0)[(k)_n/i_1, (k)_{n-1}/i_2, \ldots, (k)_1/i_n]\}.$$

D_n is clearly **IMP**-decidable. Now

$$D_{n-1} ::= \{k \in \mathrm{N} \mid \models (\exists i_n.a = 0)[(k)_{n-1}/i_1, (k)_{n-2}/i_2, \ldots, (k)_1/i_{n-1}]\}$$

is precisely right(D_n) and

$$\bar{D}_{n-1} ::= \{k \in \mathrm{N} \mid \models (\neg\exists i_n.a = 0)[(k)_{n-1}/i_1, (k)_{n-2}/i_2, \ldots, (k)_1/i_{n-1}]\}.$$

Continuing in this way, we obtain $S = D_1$ from D_n by a series of applications of right and complement. Hence $\mathcal{E} \subseteq \mathcal{C}$.

Problem 3. For any set $S \subseteq \mathrm{N}$, let ρ_S be the function environment (*cf.* Problem Set 8) such that $\rho(F_0) = \text{char}_S$ and $\rho(F_n)$ is a function with empty domain for $n > 1$ ($\text{char}_S : \mathrm{N} \to \{0,1\}$ is the characteristic function of S, where $\text{char}_S(n) = 1$ iff $n \in S$). A set S_1 is said to be *Turing reducible* to a set S_2, in symbols $S_1 \leq_T S_2$ iff char_{S_1} is ρ_{S_2}-computable. It is also suggestive to say that “S_1 is S_2-decidable” iff $S_1 \leq_T S_2$.

3(a) Show that $\bar{S} \leq_T S$ for any set S.

Given ρ_S as described, the characteristic function for $\bar{S}$ can be computed by

$$X_1 := 1 - F_0(X_1)$$

thus $\text{char}_{\bar{S}}$ is S-decidable.

3(b) Explain why $\leq_T$ is transitive.

It follows from the constructions on the last problem set that if char_{S_1} is ρ_{S_2}-computable and char_{S_2} is ρ_{S_3} computable, then char_{S_1} is ρ_{S_3} computable, by inserting a (properly renamed) instance of the program for char_{S_2} into the program for char_{S_1} whenever the value of $\text{char}_{S_2}(X)$ is needed for some X.

3(c) The self-halting problem *relative to* S is

$$H^{(S)} ::= \left\{c \in \mathbf{Com}_{fv} \mid \{c\}_{\rho_S, X_1, X_1}(\#c) \downarrow\right\}.$$

Prove that $S <_T H^{(S)}$. *Hint*: Repeat the proof that the halting problem is undecidable, but for $\mathbf{Com}_{fv}$'s in function environment ρ_S.

We first prove that $S \leq_T H^{(S)}$. That is, we have to show that char_S is $\rho_{H^{(S)}}$-computable.

As in the proof for the undecidability of the zero-halting problem given in class, for $n \in \mathbf{N}$ define $d_n \in \mathbf{Com}_{fv}$ to be

$$\textbf{if } F_0(n) = 1 \textbf{ then skip else diverge}.$$

If $n \in S$, then in function environment $\rho_{H^{(S)}}$, command d_n halts in all states, so $d_n \in H^{(S)}$. On the other hand, if $n \notin S$, then under this environment d_n never halts, so $d_n \notin H^{(S)}$. Letting $f(n) = \#d_n$ we have

$$n \in S \text{ iff } f(n) \in H^{(S)}.$$

As usual, f is a composition of mkpairs and constants, so is clearly computable. Thus we have that $S \leq_m H^{(S)}$. Now the result follows immediately from the following useful little

Lemma.

$$S_1 \leq_m S_2 \text{ implies } S_1 \leq_T S_2.$$

To prove the lemma, suppose $f : \mathbf{N} \to \mathbf{N}$ is a many-one reduction from S_1 to S_2. In function environment $\rho_{S_2}[f/F_1]$, the following $\mathbf{Com}_{fv}$ trivially computes char_{S_1}:

$$X_1 := F_0(F_1(X_1)).$$

Since f is computable and therefore certainly ρ_{S_2}-computable, we conclude from Problem Set 8 that char_{S_1} is ρ_{S_2}-computable, that is, $S_1 \leq_T S_2$.

For the second part of the solution, we must show that $H^{(S)} \not\leq_T S$. Suppose it were. Then by 3(a) and 3(b), also $\overline{H^{(S)}} \leq_T S$. We now essentially repeat, in the function environment ρ_S, the proof that the complement of the self-halting problem is not checkable.

If the interpretation of F_1 is $\text{char}_{\overline{H^{(S)}}}$, then

$$\textbf{if } F_1(X_1) = 1 \textbf{ then skip else diverge}$$

is a checker for $\overline{H^{(S)}}$. Since $\text{char}_{\overline{H^{(S)}}}$ is ρ_S-computable, we have by Problem Set 8, that there is also a checker for $\overline{H^{(S)}}$ under environment ρ_S. Let $c_0 \in \mathbf{Com}_{fv}$ be such a checker for $\overline{H^{(S)}}$ in environment ρ_S. Namely, we have for all $c \in \mathbf{Com}_{fv}$ in environment ρ_S,

$$c \text{ does not halt on input } \#c \text{ iff}$$

$$c \in \overline{H^{(S)}} \quad \text{iff} \quad c_0 \text{ halts on input } \#c$$

. Let c be c_0 to obtain a contradiction.

Quiz 4

Instructions. This is a **closed book** quiz. There are five (5) problems of roughly equal weight. Write your solutions for all problems on this quiz sheet in the spaces provided, including your *name on each sheet.* Ask for further blank sheets if you need them. You have two hours.

GOOD LUCK!

NAME

problem	*points*	*score*
1	20	
2	20	
3	15	
4	20	
5	25	
Total	100	

GOOD LUCK!

Problem 1 [20 points]. For each of the following problems indicate which properties it has ($\surd$) or does not have ($\times$):

	is decidable	is checkable	has checkable complement	is expressible
the self-halting problem $H = \{\#c \mid c \in \mathbf{Com} \text{ and } c \text{ halts on input } \#c\}$				
the valid equations between **Aexp**'s (arithmetic expressions)				
the valid **Bexp**'s (Boolean expressions)				
the unsatisfiable **Assn**'s (first-order arithmetic formulas)				
$\{\#c \mid c \in \mathbf{Com} \text{ and } c \text{ halts on some input}\}$				
$\{\#c \mid c \in \mathbf{Com} \text{ and } c \text{ is } \mathbf{while}\text{-free}\}$				
$\{\#c \mid c \in \mathbf{Com} \text{ and } c \text{ halts on input 0 in at most 1000 steps}\}$				
$\{n \in \mathbf{N} \mid n \geq \text{ some element of the self-halting problem } H\}$				

Problem 2 [20 points]. Outline a proof that if a set D and its complement $\overline{D}$ are checkable, then D is decidable.

Problem 3 [15 points]. For each of the following classes of sets indicate which closure properties it has ($\surd$) or does not have ($\times$):

	intersection	complement	mapping a set S to $f(S)$	mapping a set S to $H^{(S)}$
decidable sets				
checkable sets				
expressible sets				
finite sets				

where f is any total computable function and $H^{(S)}$ is the self-halting problem relative to S (*i.e.*, $H^{(S)} = \left\{c \in \mathbf{Com}_{fv} \mid \{c\}_{\rho_S, X_1, X_1}(\#c) \text{ halts}\right\}$).

Problem 4 [20 points]. For $S \subseteq \mathbf{N}$, let $2S = \{2n \mid n \in S\}$ and likewise $S+1 = \{n+1 \mid n \in S\}$. For $S_1, S_2 \subseteq \mathbf{N}$, prove that $(2S_1) \cup (2S_2 + 1)$ is a least upper bound of S_1 and S_2 under many-one reducibility, $\leq_m$.

Problem 5 [25 points].

5(a) [5 points]. Explain why if a set S is many-one reducible to H (in symbols, $S \leq_m H$), then S is checkable. You may cite without proof any relevant properties of H and $\leq_m$ established in class or notes.

5(b) [20 points]. Prove conversely that every checkable set is $\leq_m H$. ***Hint***: Similar to the proof that $H \leq_m H_0$ or the proof of Rice's Theorem. But Rice's theorem does *not* apply.

Quiz 4 Solutions

This was a **closed book** quiz. There were five (5) problems of roughly equal weight.

Problem 1 [20 points]. **For each of the following problems indicate which properties it has ($\surd$) or does not have ($\times$):**

	is decidable	is checkable	has checkable complement	is expressible
the self-halting problem $H = \{\#c \mid c \in \mathbf{Com}$ and c halts on input $\#c\}$	$\times$	$\surd$	$\times$	$\surd$
the valid equations between **Aexp**'s (arithmetic expressions)	$\surd$	$\surd$	$\surd$	$\surd$
the valid **Bexp**'s (Boolean expressions)	$\times$	$\times$	$\surd$	$\surd$
the unsatisfiable **Assn**'s (first-order arithmetic formulas)	$\times$	$\times$	$\times$	$\times$
$\{\#c \mid c \in \mathbf{Com}$ and c halts on some input$\}$	$\times$	$\surd$	$\times$	$\surd$
$\{\#c \mid c \in \mathbf{Com}$ and c is **while**-free $\}$	$\surd$	$\surd$	$\surd$	$\surd$
$\{\#c \mid c \in \mathbf{Com}$ and c halts on input 0 in at most 1000 steps$\}$	$\surd$	$\surd$	$\surd$	$\surd$
$\{n \in \mathbf{N} \mid n \geq$ some element of the self-halting problem $H\}$	$\surd$	$\surd$	$\surd$	$\surd$

Explanation:

Recall that a set is decidable iff it is both checkable and co-checkable, and if a set is checkable or co-checkable then it is expressible.

- The self-halting problem was shown in class to be checkable but not decidable.
- The appendix to Winskel gives a proof that the valid equations between **Aexp**'s are a decidable set.
- If we could check the validity of **Bexp**'s, then we could check the validity of inequations of the form $\neg(a = 0)$ where $a \in \mathbf{Aexp}$, which was shown to

be uncheckable in the appendix. However, if a **Bexp**, b, is not valid, then there is a substitution of numbers for its locations such that b is false. So, a checker for non-validity of **Bexp**'s only needs to run through all possible substitutions of numbers for variables in loc(b) until it finds one for which b evaluates to **false**.

- If the unsatisfiable **Assn**'s were expressible then so would the satisfiable ones be. A closed **Assn** is true iff it is satisfiable, so this would allow us to construct an expression for Truth, which has been shown to be inexpressible.

- This can be checked by gradually checking each machine against each input to see if that machine halts in n steps, gradually increasing n. If this set were decidable, though, then we could construct a decider for the "halts on 0" problem, H_0.

- It requires only a bounded, syntactic check to detect the presence of **while**-statements in a command, so this is decidable.

- This can be decided by running c on 0 for 1000 steps, and checking if c is done. This procedure will always terminate.

- Since all codes of machines are nonnegative, all elements of H are as well. Thus, there is a least $\#c \in H$. This set can be decided by comparing any given n against $\#c$.

Problem 2 [20 points]. **Outline a proof that if a set D and its complement $\overline{D}$ are checkable, then D is decidable.**

If D and $\overline{D}$ are checkable, then they both have checkers, c_D and $c_{\overline{D}}$. We can construct a decider for D by generating a program which saves its input, picks with some positive value S, and then alternately runs c_D and on that input for S steps and $c_{\overline{D}}$ on the same input for S steps, repeating and gradually increasing S until one of c_D or $c_{\overline{D}}$ halts. Since any given $n \in \mathbf{N}$ is either in D or $\overline{D}$, exactly one of the checkers must eventually halt. If c_D halts then return the value 1, and if $c_{\overline{D}}$ halts then return the value 0.

Problem 3 [15 points]. **For each of the following classes of sets indicate which closure properties it has ($\surd$) or does not have ($\times$):**

	intersection	complement	mapping a set S to $f(S)$	mapping a set S to $H^{(S)}$
decidable sets	$\surd$	$\surd$	$\times$	$\times$
checkable sets	$\surd$	$\times$	$\surd$	$\times$
expressible sets	$\surd$	$\surd$	$\surd$	$\surd$
finite sets	$\surd$	$\times$	$\surd$	$\times$

where f is any total computable function and $H^{(S)}$ is the self-halting problem relative to S (*i.e.*, $H^{(S)} = \left\{c \in \mathbf{Com}_{fv} \mid \{c\}_{\rho_S, X_1, X_1}(\#c) \text{ halts}\right\}$).

Explanation:

- 1. The intersection of two decidable sets can be decided by running the two deciders in sequence and returning 1 if either returned 1, and 0 otherwise.
  2. The complement can be decided by running a decider and returning 1 if it returns 0, and 0 if it returns 1.
  3. The right projection of a decidable set can be undecidable, so the decidable sets are not closed under arbitrary computable functions.
  4. Even taking the trivially decidable set $\emptyset$, $H^{(\emptyset)} = H$, which is undecidable.
- 1. The intersection of two checkable sets can be checked by interleaving steps from the two checkers, and halting if either halts.
  2. The complement of a checkable set may not be checkable; *e.g.*, $\overline{H}$.
  3. Handout 40, Theorem 4 states that $f(C)$ is checkable if f is computable and C is checkable.
  4. In the solutions to Problem Set 9 we demonstrated that $S \leq_m S'$ implies $S \leq_T S'$ and that $H^{(S)} \not\leq_T S$ for any S. Thus, in particular, $H^{(H)} \not\leq_m H$; but all checkable sets are $\leq_m H$, so $H^{(H)}$ is not checkable, even though H is.
- We've already seen how to express most of these. The only tricky one is constructing $H^{(S)}$ for an expressible S, but we'll leave this one as an exercise for the reader.

- 1. The intersection of two finite sets is finite.
  2. The complement of a finite set is always infinite.
  3. If we map a finite set through any function, we'll get only a finite set of results.
  4. See above under "decidability."

Problem 4 [20 points]. **For $S \subseteq \mathbf{N}$, let $2S = \{2n \mid n \in S\}$ and likewise $S+1 = \{n+1 \mid n \in S\}$. For $S_1, S_2 \subseteq \mathbf{N}$, prove that $(2S_1) \cup (2S_2+1)$ is a least upper bound of S_1 and S_2 under many-one reducibility, $\leq_m$.**

The proof is virtually identical to that for the definition of "join" given in Problem Set 9. Here the reductions from S_1 and S_2 to $(2S_1) \cup (2S_2 + 1)$ are $f(n) = 2n$ and $g(n) = 2n + 1$ respectively. $(2S_1) \cup (2S_2 + 1)$ is a least upper bound, because if $S_1 \leq_m S$ and $S_2 \leq_m S$ with reductions f' and g', then $(2S_1) \cup (2S_2 + 1) \leq_m S$ with reduction

$$h(n) = \begin{cases} f'(n/2) & \text{if } n \text{ is even} \\ g'((n-1)/2) & \text{if } n \text{ is odd} \end{cases}$$

Problem 5 [25 points].

5(a) [5 points]. **Explain why if a set S is many-one reducible to H (in symbols, $S \leq_m H$), then S is checkable. You may cite without proof any relevant properties of H and $\leq_m$ established in class or notes.**

H is checkable, and checkability inherits downwards.

5(b) [20 points]. **Prove conversely that every checkable set is $\leq_m H$. *Hint*: Similar to the proof that $H \leq_m H_0$ or the proof of Rice's Theorem. But Rice's theorem does *not* apply.**

Any checkable set S with checker c_S can be reduced to H under the function

$$f(n) = \#(X_1 := n; c_S).$$

This program code returned for n represents a function that will halt on its own number (or any input) iff c_S halts on input n. Thus, $f(n) \in H$ iff $n \in S$, so $S \leq_m H$.