15–312: Principles of Programming Languages

Midterm Examination
(Sample Solutions)

March 9, 2017

- There are 11 pages in this examination, comprising 3 questions worth a total of 90 points.
- You may refer to your personal notes and to *Practical Foundations of Programming Languages*, but not to any other person or source.
- You may use your laptop or tablet as long as you only refer to the aforementioned sources and disable WiFi and other network connections at all times.
- You have 80 minutes to complete this examination.
- Please answer all questions in the space provided with the question.
- There are three scratch sheets at the end for your use.

Full Name: ______________________________

Andrew ID: ______________________________

Question:	Type Safety	Algebra	Binary	Total
Points:	30	30	30	90
Score:				

Question 1 [30]: System T++

After spending hours implementing the dynamics of **T**, Jan is frustrated with the performance of his interpreter. He notices that running the function mod2 defined below takes about $4n$ steps when applied to the numeral $\overline{n}$. The reason is that the recursor in mod2 is evaluated $n+1$ times.

$$\begin{array}{lcl} \texttt{flip} & \triangleq & \lambda\,(x:\texttt{nat})\,\texttt{rec}\,x\,\{\texttt{z}\hookrightarrow\texttt{s}(\texttt{z})\mid\texttt{s}(n)\,\texttt{with}\,y\hookrightarrow\texttt{z}\} \\ \texttt{mod2} & \triangleq & \lambda\,(x:\texttt{nat})\,\texttt{rec}\,x\,\{\texttt{z}\hookrightarrow\texttt{z}\mid\texttt{s}(n)\,\texttt{with}\,y\hookrightarrow\texttt{flip}(y)\} \end{array}$$

To improve the evaluation speed of **T**, Jan comes up with a new recursor that is supposed to be twice as fast as the standard recursor. The resulting language is System T++: **T** extended with Jan's recursor.

$$\begin{array}{llll} \textsf{Exp} & e & ::= & \ldots \\ & & \mid\ \texttt{jrec}\{e_0;x.y.e_1\}(e) & \texttt{jrec}\,e\,\{\texttt{z}\hookrightarrow e_0\mid\texttt{s}(\texttt{s}(x))\,\texttt{with}\,y\hookrightarrow e_1\} \quad \text{Jan's recursor} \end{array}$$

Jan's recursor is defined by the following extension of the statics and dynamics.

$$\frac{\Gamma\vdash e:\texttt{nat}\quad \Gamma\vdash e_0:\tau\quad \Gamma,x:\texttt{nat},y:\tau\vdash e_1:\tau}{\Gamma\vdash\texttt{jrec}\{e_0;x.y.e_1\}(e):\tau}\ \textit{(j-type)}$$

$$\frac{e\longmapsto e'}{\texttt{jrec}\{e_0;x.y.e_1\}(e)\longmapsto\texttt{jrec}\{e_0;x.y.e_1\}(e')}\ \textit{(j-step)} \qquad \frac{}{\texttt{jrec}\{e_0;x.y.e_1\}(\texttt{z})\longmapsto e_0}\ \textit{(j-zero)}$$

$$\frac{\texttt{s}(\texttt{s}(e))\ \textsf{val}}{\texttt{jrec}\{e_0;x.y.e_1\}(\texttt{s}(\texttt{s}(e)))\longmapsto[e,\texttt{jrec}\{e_0;x.y.e_1\}(e)/x,y]e_1}\ \textit{(j-succ)}$$

When Jan presents his idea at the weekly PLunch meeting, Bob is repelled by the name of the new language. He also points out that System T++ is not even type safe.

(a) [10 points] Show that System T++ is not type safe. *Hint*: Provide an example and show that it violates the progress theorem or the preservation theorem.

> **Solution:** Consider the following expression e_{stuck}.
>
> $$\texttt{jrec}\,\texttt{s}(\texttt{z})\,\{\texttt{z}\hookrightarrow\texttt{z}\mid\texttt{s}(\texttt{s}(n))\,\texttt{with}\,y\hookrightarrow\texttt{z}\}$$
>
> We have $\Gamma\vdash e_{\text{stuck}}:\texttt{nat}$ by applying the rule *j-type*. However, there is no expression e' such that $e_{\text{stuck}}\longmapsto e'$. This can be proved by rule induction. Similarly, we cannot derive e_{stuck} $\textsf{val}$. Consequently, the progress theorem does not hold for System T++.

(Question continues on the next page)

(b) Even though, Bob does not believe in the future of System T++, he proposes a change to Jan's recursor that will make the language type safe. Bob's version of System T++ is defined by adding the following recursor to **T**.

Exp	e	$::=$		$\dots$	
		$\mid$	$\texttt{brec}\{e_0; e_1; x.y.e_2\}(e)$	$\texttt{brec}\, e\, \{\texttt{z} \hookrightarrow e_0 \mid \texttt{s}(\texttt{z}) \hookrightarrow e_1 \mid \texttt{s}(\texttt{s}(x))\, \texttt{with}\, y \hookrightarrow e_2\}$	Bob's recursor

The dynamics of Bob's recursor is given by the following rules.

$$\frac{e \longmapsto e'}{\texttt{brec}\{e_0; e_1; x.y.e_2\}(e) \longmapsto \texttt{brec}\{e_0; e_1; x.y.e_2\}(e')}\ \textit{(b-step)}$$

$$\frac{}{\texttt{brec}\{e_0; e_1; x.y.e_2\}(\texttt{z}) \longmapsto e_0}\ \textit{(b-zero)} \qquad \frac{}{\texttt{brec}\{e_0; e_1; x.y.e_2\}(\texttt{s}(\texttt{z})) \longmapsto e_1}\ \textit{(b-one)}$$

$$\frac{\texttt{s}(\texttt{s}(e))\ \mathsf{val}}{\texttt{brec}\{e_0; e_1; x.y.e_2\}(\texttt{s}(\texttt{s}(e))) \longmapsto [e, \texttt{brec}\{e_0; e_1; x.y.e_2\}(e)/x, y]e_2}\ \textit{(b-succ)}$$

i. [5 points] Define the statics of Bob's recursor so that the new System T++ is type safe.

Solution:

$$\textit{(b-type)}\ \frac{\Gamma \vdash e : \texttt{nat} \quad \Gamma \vdash e_0 : \tau \quad \Gamma \vdash e_1 : \tau \quad \Gamma, x : \texttt{nat}, y : \tau \vdash e_1 : \tau}{\Gamma \vdash \texttt{brec}\{e_0; e_1; x.y.e_2\}(e) : \tau} \qquad (1)$$

ii. [5 points] Re-implement the function `mod2` exclusively using Bob's recursor. (You are not allowed to use the standard recursor of **T** in your code.)

Solution:

$$\texttt{mod2} \ = \ \lambda\,(x : \tau)\, \texttt{brec}\, x\, \{\texttt{z} \hookrightarrow \texttt{z} \mid \texttt{s}(\texttt{z}) \hookrightarrow \texttt{s}(\texttt{z}) \mid \texttt{s}(\texttt{s}(n))\, \texttt{with}\, y \hookrightarrow y\}$$

(Question continues on next page.)

iii. 10 points Prove the progress theorem for Bob's System T++. You only have to consider the case in which e is the recursor $\texttt{brec}\{e_0; e_1; x.y.e_2\}(e)$.

Theorem 1.1 (Progress)**.** *If $e : \tau$, then either e val or $e \longmapsto e'$ for some e'.*

Hint: Recall that the proof proceeds by induction on the type derivation. You have to consider the case in which $\Gamma \vdash \texttt{brec}\{e_0; e_1; x.y.e_2\}(e) : \tau$ is derived by the type rule you defined in Part i. You can use the following canonical forms lemma.

Lemma 1.2. *If $e : \texttt{nat}$ and e val then either*

- *e is $\texttt{z}$ or*
- *e is $\texttt{s}(e')$ for an e' such that $e' : \texttt{nat}$ and e' val.*

Hint 2: Do a case distinction on e.

Solution:

Assume $\Gamma \vdash \texttt{brec}\{e_0; e_1; x.y.e_2\}(e) : \tau$ has been derived by the rule *b-type*.
Then $\Gamma \vdash e : \texttt{nat}$.
By the induction hypothesis either e val or $e \longmapsto e'$ for some e'.
In the letter case, we complete the proof by using the rule *b-step* to derive $\texttt{brec}\{e_0; e_1; x.y.e_2\}(e) \longmapsto \texttt{brec}\{e_0; e_1; x.y.e_2\}(e')$.
If e val then it follows from the canonical-forms lemma that either $e = \texttt{z}$, $e = \texttt{s}(e')$ for some e' with e' val and $e' : \texttt{nat}$. We can apply the CFL again to e' to conclude that either $e' = \texttt{z}$ or $e' = \texttt{s}(e'')$ for some e''. Therefore $e = \texttt{z}$, $e = \texttt{s}(\texttt{z})$, or $e = \texttt{s}(\texttt{s}(e''))$.
In the first case we can apply *b-zero* to step to e_0. In the second case, we can use *b-one* to step to e_1. In the third case, we can apply the rule *b-succ* to make a step.

Question 2 [30]: Type Algebra

Two types σ and τ are *isomorphic*, written $\sigma \cong \tau$, iff there are two functions $f : \sigma \to \tau$ and $g : \tau \to \sigma$ that are mutually inverse. In this question you are to exhibit such pairs of functions as evidence that two types are isomorphic in **T** enriched with product and sum types.

Important: You must exhibit an inverse pair, but you do *not* have to prove that they are inverses.

(a) Exhibit the mutually inverse functions that witness each of the following isomorphisms:

i. [4 points] $\mathtt{1} \times \tau \cong \tau$:

- $f \triangleq$
- $g \triangleq$

Solution:

- $f \triangleq \lambda\,(x : \mathtt{1} \times \tau)\, x \cdot r$
- $g \triangleq \lambda\,(x : \tau)\, \langle\langle\rangle, x\rangle$

ii. [4 points] $\mathtt{0} + \tau \cong \tau$:

- $f \triangleq$
- $g \triangleq$

Solution: $\mathtt{0} + \tau \cong \tau$:

- $f \triangleq \lambda\,(x : \mathtt{0} + \tau)\, \mathtt{case}\, x\, \{\mathtt{l} \cdot x_1 \hookrightarrow \mathtt{case}\, x_1\, \{\,\} \mid \mathtt{r} \cdot x_2 \hookrightarrow x_2\}$
- $g \triangleq \lambda\,(x : \tau)\, \mathtt{r} \cdot x$

(Question continues on next page.)

iii. 4 points $\rho \times (\sigma + \tau) \cong (\rho \times \sigma) + (\rho \times \tau)$:

- $f \triangleq$
- $g \triangleq$

Solution:

- $f \triangleq \lambda\,(x : \rho \times \sigma + \tau)\,\texttt{case}\,x \cdot r\,\{\texttt{l} \cdot y_1 \hookrightarrow \texttt{l} \cdot \langle x \cdot l, y_1\rangle \mid \texttt{r} \cdot y_2 \hookrightarrow \texttt{r} \cdot \langle x \cdot l, y_2\rangle\}$.
- $g \triangleq \lambda\,(z : (\rho \times \sigma) + (\rho \times \tau))\,\texttt{case}\,z\,\{\texttt{l} \cdot \langle x, y\rangle \hookrightarrow \langle x, \texttt{l} \cdot y\rangle \mid \texttt{r} \cdot \langle x, z\rangle \hookrightarrow \langle x, \texttt{r} \cdot z\rangle\}$.

iv. 4 points $(\sigma_1 + \sigma_2) \to \tau \cong (\sigma_1 \to \tau) \times (\sigma_2 \to \tau)$:

- $f \triangleq$
- $g \triangleq$

Solution:

- $f \triangleq \lambda\,(x : (\sigma_1 + \sigma_2) \to \tau)\,\langle \lambda\,(x_1 : \sigma_1)\,x(\texttt{l} \cdot x_1), \lambda\,(x_2 : \sigma_2)\,x(\texttt{r} \cdot x_2)\rangle$
- $g \triangleq \lambda\,(z : (\sigma_1 \to \tau) \times (\sigma_2 \to \tau))$
  $\lambda\,(x : \sigma_1 + \sigma_2)\,\texttt{case}\,x\,\{\texttt{l} \cdot x_1 \hookrightarrow z \cdot \texttt{l}(x_1) \mid \texttt{r} \cdot x_2 \hookrightarrow z \cdot \texttt{r}(x_2)\}$

(Question continues on next page.)

(b) In the following questions, you are to use the preceding isomorphisms. In addition, you can use the following isomorphisms.

1. Commutativity and associativity of product and sum up to isomorphism. (For example, $\sigma \times \tau \cong \tau \times \sigma$.)
2. Each type constructor respects isomorphism in the sense that if $\sigma_1 \cong \sigma_2$, then $\sigma_1 \times \tau \cong \sigma_2 \times \tau$, and so on.
3. $\sigma \to (\tau_1 \times \tau_2) \cong (\sigma \to \tau_1) \times (\sigma \to \tau_2)$
4. $\mathtt{1} \to \tau \cong \tau$

If applicable, state clearly which isomorphisms you apply in your reasoning by citing the question number or the number of the previous enumeration.

Note: You are able to reason about these isomorphisms *without* defining new functions.

i. [4 points] Recall that the type $\mathtt{2}$ is defined to be $\mathtt{1} + \mathtt{1}$. Show that $\mathtt{2} \times \tau \cong \tau + \tau$:

ii. [4 points] Define τ^2 to be $\tau \times \tau$. Show that $\tau^2 \cong \mathtt{2} \to \tau$:

iii. [4 points] Recall that $\tau\ \mathtt{opt}$ is defined as $\mathtt{1} + \tau$. Show that $(\tau\ \mathtt{opt})^2 \cong \tau^2 + \mathtt{2} \times \tau + \mathtt{1}$. (In other words, show that $\tau\ \mathtt{opt} \cong \sqrt{\tau^2 + \mathtt{2} \times \tau + \mathtt{1}}$!)

(Question continues on next page.)

iv. 2 points Explain in one English sentence the meaning of the preceding isomorphism. Your answer should have the form "*A pair of optional values of type τ is ...*."

Solution:

1. Using the foregoing isomorphisms, show that $2 \times \tau \cong \tau + \tau$:

$$\begin{aligned} 2 \times \tau &\cong (1 + 1) \times \tau \\ &\cong \tau \times (1 + 1) \\ &\cong (\tau \times 1) + (\tau \times 1) \\ &\cong (1 \times \tau) + (1 \times \tau) \\ &\cong \tau + \tau. \end{aligned}$$

2. Using the foregoing isomorphisms, show that $\tau^2 \cong 2 \to \tau$:

$$\begin{aligned} 2 \to \tau &\cong (1 + 1) \to \tau \\ &\cong (1 \to \tau) \times (1 \to \tau) \\ &\cong \tau \times \tau \\ &= \tau^2. \end{aligned}$$

3. Using the foregoing isomorphisms, show that $\tau\ \mathtt{opt} \cong \sqrt{\tau^2 + 2 \times \tau + 1}$:

$$\begin{aligned} (\tau\ \mathtt{opt})^2 &= (\tau + 1)^2 \\ &\cong (\tau + 1) \times (\tau + 1) \\ &\cong ((\tau + 1) \times \tau) + ((\tau + 1) \times 1) \\ &\cong ((\tau \times \tau) + (1 \times \tau)) + ((1 \times \tau) + (1 \times 1)) \\ &\cong \tau^2 + (\tau + \tau) + 1 \\ &\cong \tau^2 + 2 \times \tau + 1 \end{aligned}$$

4. *A pair of optional values of type τ is either either a pair of values of type τ, or a single value of type τ two different ways, or nothing.*

Question 3 [30]: Binary Natural Numbers

The formulation of natural numbers in **T** uses a unary representation. In this question you are to explore a binary representation of natural numbers based on the following idea: a natural number is either 0, twice another natural number, $2 \times n$, or one more than twice a natural number, $2 \times n + 1$. (Quick quiz: how is 1 to be written using this formulation of natural numbers?)

Here is the statics of the type of natural numbers in binary.

$$\frac{}{\Gamma \vdash \mathtt{z} : \mathtt{bin}} \qquad \frac{\Gamma \vdash e : \mathtt{bin}}{\Gamma \vdash \mathtt{tw}(e) : \mathtt{bin}} \qquad \frac{\Gamma \vdash e : \mathtt{bin}}{\Gamma \vdash \mathtt{twpo}(e) : \mathtt{bin}}$$

$$\frac{\Gamma \vdash e : \mathtt{bin} \quad \Gamma \vdash e_0 : \tau \quad \Gamma, x : \tau \vdash e_1 : \tau \quad \Gamma, x : \tau \vdash e_2 : \tau}{\Gamma \vdash \mathtt{binrec}\{\tau\}(e; e_0; x.e_1; x.e_2) : \tau}$$

The introductory forms are for zero, twice a number, and one more than twice a number. The elimination form, correspondingly, has three cases.

Here is the (lazy) dynamics for the binary formulation of natural numbers:

$$\frac{}{\mathtt{z}\ \mathsf{val}} \qquad \frac{}{\mathtt{tw}(e)\ \mathsf{val}} \qquad \frac{}{\mathtt{twpo}(e)\ \mathsf{val}}$$

$$\frac{e \longmapsto e'}{\mathtt{binrec}\{\tau\}(e; e_0; x.e_1; x.e_2) \longmapsto \mathtt{binrec}\{\tau\}(e'; e_0; x.e_1; x.e_2)}$$

$$\frac{}{\mathtt{binrec}\{\tau\}(\mathtt{z}; e_0; x.e_1; x.e_2) \longmapsto e_0}$$

$$\frac{}{\mathtt{binrec}\{\tau\}(\mathtt{tw}(e); e_0; x.e_1; x.e_2) \longmapsto [\mathtt{binrec}\{\tau\}(e; e_0; x.e_1; x.e_2)/x]e_1}$$

$$\frac{}{\mathtt{binrec}\{\tau\}(\mathtt{twpo}(e); e_0; x.e_1; x.e_2) \longmapsto [\mathtt{binrec}\{\tau\}(e; e_0; x.e_1; x.e_2)/x]e_2}$$

Defining the successor function on the binary natural numbers requires an idea similar to that used to define the predecessor function on the unary natural numbers. The successor of $\overline{0}$ is easily defined outright. The successor of $\overline{2 \times n + 1}$ is is twice the successor of $\overline{n}$, which we are given by induction. But how can we compute the successor of $\overline{2 \times n}$ given only $\overline{n+1}$?

As with the predecessor in the unary case, the trick is to compute more: given $\overline{n}$, compute the pair $\langle \overline{n}, \overline{n+1} \rangle$. The desired successor is the second component, but having the first component is helpful exactly in the awkard case just mentioned.

(Question continues on next page.)

(a) i. [4 points] The *binary numeral* $\overline{n}$ of type $\texttt{bin}$ for the natural number n is given in terms of $\texttt{z}$, $\texttt{tw}(-)$ and $\texttt{twpo}(-)$ amounting to the binary expansion of n. Define the two numerals $\overline{10}$ and $\overline{14}$.

$\overline{10} \triangleq$

$\overline{14} \triangleq$

ii. [10 points] Define the function $\texttt{succ}' : \texttt{bin} \to (\texttt{bin} \times \texttt{bin})$ such that $\texttt{succ}'(\overline{n})$ evaluates to $\langle \overline{n}, \overline{n+1} \rangle$.

iii. [5 points] Define $\texttt{succ} : \texttt{bin} \to \texttt{bin}$ in terms of $\texttt{succ}'$ such that $\texttt{succ}(\overline{n})$ evaluates to $\overline{n+1}$.

Solution:

- The binary numeral for n is written as the reversed binary expansion of n. $\overline{10}$ is 1010 in binary, and hence is represented as $\texttt{tw}(\texttt{twpo}(\texttt{tw}(\texttt{twpo}(\texttt{z}))))$. $\overline{14}$ is 1110 in binary, and hence is represented as $\texttt{tw}(\texttt{twpo}(\texttt{twpo}(\texttt{twpo}(\texttt{z}))))$.
- Define $\texttt{succ}'$ to be the function

$$\begin{array}{ll} \lambda\,(n : \texttt{bin}) & \texttt{binrec}\{\texttt{bin} \times \texttt{bin}\}(\\ & n; \\ & \langle \texttt{z}, \texttt{twpo}(\texttt{z}) \rangle; \\ & m.\langle \texttt{tw}(m \cdot \texttt{l}), \texttt{twpo}(m \cdot \texttt{l}) \rangle; \\ & m.\langle \texttt{twpo}(m \cdot \texttt{l}), \texttt{tw}(m \cdot \texttt{r}) \rangle \\ &) \end{array}$$

- Define $\texttt{succ}$ as $\lambda\,(n : \texttt{bin})\,\texttt{succ}'(n) \cdot \texttt{r}$

(Question continues on next page.)

(b) The binary numerals admit a Church representation in System F similar to that for the unary numerals. Recall that the unary Church representation of `nat` is the polymorphic type $\forall(t.t \to (t \to t) \to t)$, which expresses the type of the iterator for each number n. The Church representation of `bin` is given by the polymorphic type

$$\forall(t.t \to (t \to t) \to (t \to t) \to t).$$

i. [6 points] Define the introductory forms $\mathtt{z}$, $\mathtt{tw}(e)$, and $\mathtt{twpo}(e)$ as elements of the preceding polymorphic type.

- $\mathtt{z}$:
- $\mathtt{tw}(e)$:
- $\mathtt{twpo}(e)$:

ii. [5 points] Define the recursor $\mathtt{binrec}\{\tau\}(e; e_0; x.e_1; x.e_2)$ in terms of the binary Church representation.

Solution:

- The introductory forms are defined as follows:
  - $\mathtt{z} \triangleq \Lambda(t)\,\lambda\,(z : t)\,\lambda\,(w : t \to t)\,\lambda\,(w' : t \to t)\,z$
  - $\mathtt{tw}(e) \triangleq \Lambda(t)\,\lambda\,(z : t)\,\lambda\,(w : t \to t)\,\lambda\,(w' : t \to t)\,w(e[t](z)(w)(w'))$
  - $\mathtt{twpo}(e) \triangleq \Lambda(t)\,\lambda\,(z : t)\,\lambda\,(w : t \to t)\,\lambda\,(w' : t \to t)\,w'(e[t](z)(w)(w'))$
- The eliminatory form is defined as follows:

$$\mathtt{binrec}\{\tau\}(e; e_0; x.e_1; x.e_2) \triangleq e[\tau](e_0)(\lambda\,(x : \tau)\,e_1)(\lambda\,(x : \tau)\,e_2)$$