15–312: Principles of Programming Languages

Final Examination

May 6, 2010

- There are 19 pages in this examination, comprising 6 questions worth a total of 140 points.
- You may refer to your personal notes and to the textbook (*Practical Foundations of Programming Languages—PFPL*), but to no other person or source, during the examination.
- You have 180 minutes to complete this examination.
- Please read each question completely before attempting to solve any part.
- Please answer all questions in the space provided with the question.
- There are two scratch sheets at the end for your use.

Full Name: _____

Andrew ID: _____

Question:	Conversions	System F	Monads	Dynamic	Parallelism	Concurrency	Total
Points:	25	5	30	30	25	25	140
Score:							

Question 1 [25]: Conversions

You may have used *Google calculator*. Like any calculator, it will do arithmetic calculations for you. Unlike your typical calculator, it gives you a convenient way to do unit conversions. For example, you can ask it to compute "12 feet 2 inches + 3.6 meters" and it will return 7.3084 meters. You can also ask fancier questions like "12 feet 2 inches + 3.6 meters in fathoms" (the result is 3.996 fathoms — the fathom is a British unit of water depth), or to convert mass values, and even currencies. One thing it will not do is mix incompatible quantities, for example "2 inches + 3 grams". In this exercise, we will study the programming language concepts that underlie this tool, and then extend it.

We start with a simple language that supports a fixed choice of *quantities* (here length, mass and currency), a fixed selection of units for those quantities, and a few useful operations to form expressions (a measurement in a unit, the sum of two expressions, and the explicit conversion into a compatible unit). Such expressions are classified by the type $\mathsf{float}(q)$ for an appropriate quantity q; the type scalar classifies the scalar coefficients of units (e.g., "3.6" in "3.6 meter").

Quantities	q	::=	$\mathsf{length} \mid \mathsf{mass} \mid \mathsf{currency}$
Types	τ	::=	$\mathsf{scalar} \mid \mathsf{float}(q)$
Units	u	::=	$\mathsf{meter} \mid \mathsf{inch} \mid \mathsf{fathom} \mid \mathsf{oz} \mid \mathsf{gram} \mid \mathsf{USD} \mid \mathsf{QAR} \mid \mathsf{EUR}$
Expressions	e	::=	$\overline{n} \mid eu \mid e_1 + e_2 \mid e\ \mathsf{into}\ u$

The straightforward static semantics commits all measurements in an expressions to a single quantity. These rules rely on the judgment $u\ \mathsf{is\_a}\ q$ which states that unit u refers to the quantity q, for example we would have $\mathsf{meter}\ \mathsf{is\_a}\ \mathsf{length}$.

$$\frac{}{\overline{n} : \mathsf{scalar}}\ \mathsf{tp\_sc}$$

$$\frac{e : \mathsf{scalar} \quad u\ \mathsf{is\_a}\ q}{eu : \mathsf{float}(q)}\ \mathsf{tp\_n} \qquad \frac{e_1 : \mathsf{float}(q) \quad e_2 : \mathsf{float}(q)}{e_1 + e_2 : \mathsf{float}(q)}\ \mathsf{tp\_pl} \qquad \frac{e : \mathsf{float}(q) \quad u\ \mathsf{is\_a}\ q}{e\ \mathsf{into}\ u : \mathsf{float}(q)}\ \mathsf{tp\_in}$$

The dynamic semantics extends the standard step evaluation judgment $e \mapsto e'$ with a *conversion table* X, obtaining $e \mapsto_X e'$. Actual conversions are performed by the auxiliary conversion judgment $X \models \overline{1}\, u = \overline{k}\, u'$ which uses the conversion table X to determine that $\overline{k}$ units of u' correspond to one unit of u — for example we have $X \models \overline{1}\,\mathsf{fathom} = \overline{1.8288}\,\mathsf{meter}$. The value judgment is similarly written $e\ \mathsf{val}_X$ and holds for expressions of the form $\overline{n}\, u$.

$$\frac{}{\overline{n}\, u\ \mathsf{val}_X}\ \mathsf{val\_n}$$

$$\frac{e_1 \mapsto_X e_1'}{e_1 + e_2 \mapsto_X e_1' + e_2}\ \mathsf{ev\_pl1} \qquad \frac{e_1\ \mathsf{val}_X \quad e_2 \mapsto_X e_2'}{e_1 + e_2 \mapsto_X e_1 + e_2'}\ \mathsf{ev\_pl2}$$

$$\frac{X \models \overline{1}\, u_1 = \overline{k}\, u_2}{\overline{n_1}\, u_1 + \overline{n_2}\, u_2 \mapsto_X (\overline{kn_1 + n_2})u_2}\ \mathsf{ev\_pl3}$$

$$\frac{e \mapsto_X e'}{e\ \mathsf{into}\ u \mapsto_X e'\ \mathsf{into}\ u}\ \mathsf{ev\_in1} \qquad \frac{X \models \overline{1}\, u' = \overline{k}\, u}{\overline{n}\, u'\ \mathsf{into}\ u \mapsto_X (\overline{kn})u}\ \mathsf{ev\_in2}$$

Given this setup, the language so obtained admits the obvious safety properties, which are omitted.

You will now extend this language with a mechanism to add units and conversion factors (e.g., the Swedish "aln", which is equal to 0.5937 meter), with another mechanism to add new quantities (e.g., time or temperature), and with scalar functions. The resulting language extension is as follows:

Quantities	q	::=	$\ldots \mid \rho$
Types	τ	::=	$\ldots \mid \mathsf{some}(\rho.\tau) \mid \tau_1 \to \tau_2$
Units	u	::=	$\ldots \mid \upsilon$
Expressions	e	::=	$\ldots \mid \mathsf{letq}(\rho.\upsilon.e) \mid \mathsf{letu}\ \overline{1}\upsilon = e_1\ \mathsf{in}\ e_2 \mid x \mid \lambda x : \mathsf{scalar}.\, e \mid e_1\, e_2$

Here, ρ is a quantity variable and υ is a unit variable. These operators have the following meaning: $\mathsf{letq}(\rho.\upsilon.e)$ creates a new quantity ρ and a measurement unit υ for ρ for use inside expression e; $\mathsf{letu}\ \overline{1}\upsilon = e_1\ \mathsf{in}\ e_2$ creates a new measurement unit υ equal to the value of e_1 for use inside expression e_2; functions have the usual behavior but their argument is restricted to scalar values. The creation of a new quantity ρ is witnessed by the type $\mathsf{some}(\rho.\tau)$. (If this works better for you, feel free to do the later parts of this question to gain practice.)

(a) (8 points) Define the static semantics of this language extension by giving typing rules for letq and letu. Your typing judgment is now $\Delta; \Gamma \vdash e : \tau$, where the context Δ has the form $\upsilon_1\ \mathsf{is\_a}\ q_1, \ldots, \upsilon_n\ \mathsf{is\_a}\ q_n$ for distinct unit variables υ_i (note that some q_j may be a quantity variable) and the context Γ contains standard typing assumptions of the form $x : \mathsf{scalar}$.

Solution:

$$\frac{(\Delta, \upsilon\ \mathsf{is\_a}\ \rho); \Gamma \vdash e : \tau}{\Delta; \Gamma \vdash \mathsf{letq}(\rho.\upsilon.e) : \mathsf{some}(\rho.\tau)}\ \mathsf{tp\_letq}$$

$$\frac{\Delta; \Gamma \vdash e_1 : \mathsf{float}(q) \quad (\Delta, \upsilon\ \mathsf{is\_a}\ q); \Gamma \vdash e_2 : \tau}{\Delta; \Gamma \vdash \mathsf{letu}\ \overline{1}\upsilon = e_1\ \mathsf{in}\ e_2 : \tau}\ \mathsf{tp\_letu}$$

(b) (9 points) Define the dynamic semantics of this language extension by giving evaluation rules for letq and letu. The evaluation judgment is upgraded to $\langle X, e\rangle \mapsto \langle X', e'\rangle$, where X and X' are the conversion tables before and after the evaluation step, respectively. You may extend a conversion table with entries such as $\overline{1}aln = \overline{0.5937}\,\mathsf{meter}$. The rule for values shall remain unchanged.

Solution:

$$\frac{(\upsilon \# X)}{\langle X, \mathsf{letq}(\rho.\upsilon.e)\rangle \mapsto \langle (X, \overline{1}\upsilon = \overline{1}\upsilon), e\rangle}\ \mathsf{ev\_letq}$$

$$\frac{\langle X, e_1\rangle \mapsto \langle X', e_1'\rangle}{\langle X, \mathsf{letu}\ \overline{1}\upsilon = e_1\ \mathsf{in}\ e_2\rangle \mapsto \langle X', \mathsf{letu}\ \overline{1}\upsilon = e_1'\ \mathsf{in}\ e_2\rangle}\ \mathsf{ev\_letu1}$$

$$\frac{(\upsilon \# X)}{\langle X, \mathsf{letu}\ \overline{1}\upsilon = \overline{n}\,u\ \mathsf{in}\ e_2\rangle \mapsto \langle (X, \overline{1}\upsilon = \overline{n}\,u), e_2\rangle}\ \mathsf{ev\_letu2}$$

(c) (4 points) If you were to implement the language so obtained and you wanted to know how many Swedish alns correspond to 2.3 fathoms using the expression

$$\mathsf{letu}\ \overline{1}\,aln = \overline{0.5937}\,\mathsf{meter}\ \mathsf{in}\ \overline{2.3}\,\mathsf{fathom}\ \mathsf{into}\ aln$$

you may get the following answer: $\overline{7.085}\,\mathsf{var7062}$. The scalar part is correct, but what does the reported unit, var7062, has to do with alns?? Explain what has happened. Recall that units are stored in the conversion tables at run-time. (If you have time, suggest a way to fix the problem.)

Solution: Because *aln* is a bound variable in this expression, it can be α-renamed as needed. The implementation has simply "freshened" it into some internal identifier, here var7062.

Optional answer: There is no perfect solution to this problem. One way is to extend letu with a third argument, a string, that specifies how the unit should be printed. This brings up a new problem: what if the user mistakenly redefines the string corresponding to an existing unit (and possibly get some really weird-looking results)? How do deal with this? Simply turning a blind eye is a recipe for disaster given that to this day not everybody agrees on how much a *gallon* holds, and that there are over a dozen interpretations of what a *league* is. Given this confusing reality, the safest solution may be for an implementation to raise a warning (or an error) if unit names get redefined.

(d) (4 points) Define an expression of type $\mathsf{some}(\rho.\mathsf{float}(\rho) \to \mathsf{float}(\rho))$ that converts any Fahrenheit temperature into Celsius. The conversion formula is $\mathrm{C} = 5/9\,(\mathrm{F} - 32)$. Note that it is not as simple as it looks and that you are likely to use every construct you have just introduced in the grammar. (Feel free to write scalars as fractions, e.g., $\overline{5/9}$ for the scalar corresponding to 5/9.)

Solution:

$$\mathsf{letq}(temp.F.\mathsf{letu}\ \overline{1}C = \overline{5/9}\,F\ \mathsf{in}\ \lambda x : \mathsf{scalar}.\, xF + \overline{-32}\,C)$$

Question 2 [5]: System F

Answer the following questions in the space provided.

(a) (3 points) Please give a definition in System F of binary search trees with numbers as keys, which would be described in ML as follows:

```
datatype bst = Empty | Node of bst * nat * bst
```

Solution:

$$\forall t.t \to (t \to \texttt{nat} \to t \to t) \to t.$$

(b) (2 points) Please give a *fully simplified* term of the foregoing System F type corresponding to the binary search tree containing two elements, 2 and 3:

Solution:

$$\Lambda t.\lambda m : t.\lambda n : t \to \texttt{nat} \to t \to t.n\,(n\,m\,\texttt{2}\,m)\,\texttt{3}\,m$$

Your solution should not contain any terms that can be reduced to simpler form by applying an elimination form to an introduction form.

Question 3 [30]: Confidentiality

Modernized Algol is based on a *monadic* type system, which distinguishes *expressions* from *commands*. The meaning of a command depends on the contents of the active assignables, whereas the meaning of an expression is independent of the assignables. More generally, a monadic type system may be seen as isolating a notion of a *sensitive* computation from a *robust* computation in such a way that (a) any robust computation may be seen as *pro forma* sensitive, and (b) any computation that depends on a sensitive computation is itself deemed sensitive. In the context of Modernized Algol a sensitive computation is a command, which may use assignables, and a robust computation is an expression, which does not.

In this question we explore another application of a monadic type system to the problem of ensuring *confidentiality*. We postulate that a program has two sorts of input ports and two sorts of output ports, the *public* and the *private*, and we wish to protect the confidentiality of data arising from the private port. This amounts to ensuring that *any data dependent on input from the private port may not be written to the public port*. This can be guaranteed using a monadic type system that generalizes the one used in Modernized Algol.

We first need two *security levels*, `pub` and `priv`, representing the sensitivity of an operation. The typing judgement for expressions, $\Gamma \vdash e : \tau$ is as usual, but the typing judgement for commands now has the form

$$\Gamma \vdash m \sim \tau \mathbin{@} (r, w)$$

where r represents the *read level*, and w represents the *write level*, of the command m. The read level is to be thought of as an *upper bound* on the security level of the reads within m, and the write level is to be thought of as a *lower bound* on the security level of the writes within m. Therefore a public read can be at read level `pub` or `priv`, but a private read can only be at read level `priv`. Similarly, a private write can occur at write level `pub` or `priv`, but a private read can occur only at write level `priv`.

- $(\texttt{pub}, \texttt{priv})$ represents the base level of access - only reading from the public channel and only writing to the private channel.
- $(\texttt{pub}, \texttt{pub})$ is the security level of a program that reads only from the public channel, but can write to either the public or the private channel.
- $(\texttt{priv}, \texttt{priv})$ is the security level of a program that reads from either the public or the private channel, but can only write to the private channel.
- $(\texttt{priv}, \texttt{pub})$ would be the security of a program that reads from and writes to any channel it wants. *Such a command may violate confidentiality* because it admits commands that do private reads and public writes.

We introduce an ordering among security levels in which we regard `priv` as being "more sensitive" than `pub`:

$$\frac{}{\texttt{pub} \sqsubseteq \texttt{pub}} \qquad \frac{}{\texttt{priv} \sqsubseteq \texttt{priv}} \qquad \frac{}{\texttt{pub} \sqsubseteq \texttt{priv}}$$

This relation is reflexive ($l \sqsubseteq l$), transitive (if $l \sqsubseteq l'$ and $l' \sqsubseteq l''$, then $l \sqsubseteq l''$), and anti-symmetric (if $l \sqsubseteq l'$ and $l' \sqsubseteq l$, then $l = l'$).

It does not violate confidentiality to regard a command that reads the public port as being one that (vacuously) reads the private port. Similarly, it does not violate confidentiality to regard

a command that writes the private port as being one that (vacuously) writes the public port. This leads to the following *inclusion rule* for commands:

$$\frac{\Gamma \vdash m \sim \tau \mathbin{@} (r, w) \quad r \sqsubseteq r' \quad w' \sqsubseteq w}{\Gamma \vdash m \sim \tau \mathbin{@} (r', w')}$$

That is, a command that reads at a level may be regarded as reading at any higher level, and a command that writes at a level may be regarded as writing at any lower level.

Corresponding to the assignment of security levels to commands, we must enrich the type of suspended commands to record the security level of the encapsulated command. That is, the type $\mathsf{cmd}[(r, w)](\tau)$ is the type of suspended commands that yield a value of type τ and that have security level (r, w).

$$\frac{\Gamma \vdash m \sim \tau \mathbin{@} (r, w)}{\Gamma \vdash \mathsf{cmd}(m) : \mathsf{cmd}[(r, w)](\tau)}$$

The static semantics is organized so that each command has a *principal* security level in which the read level is as small as possible and the write level is as large as possible.

(a) (5 points) State the *principal* typing rule for the "return" command, $\mathsf{ret}(e)$.

Solution: $\dfrac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{ret}(e) \sim \tau \mathbin{@} (\mathtt{pub}, \mathtt{priv})}$

(b) (5 points) State the *most general possible* typing rule for the "bind" command, $\mathsf{seq}(e; x.m)$. *Hint*: the security level of e may *differ* from the security level of m. You will need *two* premises relating the read and write levels of e to the read and write levels of m.

Solution: $\dfrac{\Gamma \vdash e : \mathsf{cmd}[(r, w)](\tau) \quad \Gamma, x{:}\tau \vdash m \sim \tau' \mathbin{@} (r', w') \quad r \sqsubseteq r' \quad w' \sqsubseteq w}{\Gamma \vdash \mathsf{seq}(e; x.m) \sim \tau' \mathbin{@} (r', w')}$

There are *four* commands for performing input and output: `rdPriv`, `rdPub`, `wrPub(`e`)`, and `wrPriv(`e`)`. The read commands yield a string, and the write commands take a string as argument and yield type `unit`.

Here are the principal typing rules for two of these commands:

$$\frac{}{\Gamma \vdash \mathtt{rdPub} \sim \mathtt{string} \mathbin{@} (\mathtt{pub}, \mathtt{priv})} \qquad \frac{\Gamma \vdash e : \mathtt{string}}{\Gamma \vdash \mathtt{wrPriv(}e\mathtt{)} \sim \mathtt{unit} \mathbin{@} (\mathtt{pub}, \mathtt{priv})}$$

By the inclusion rule given above these commands may also be given *any* security level (r, w)! This corresponds to the idea that there are no confidentiality restrictions on public reads or private writes.

(c) (5 points) Give the principal typing rule for private reads and public writes. This means to give the *smallest* read level and the *largest* write level possible consistent with the intended meaning of the security level of each command.

Solution:

$$\frac{}{\Gamma \vdash \mathtt{rdPriv} \sim \mathtt{string} \mathbin{@} (\mathtt{priv}, \mathtt{priv})} \qquad \frac{\Gamma \vdash e : \mathtt{string}}{\Gamma \vdash \mathtt{wrPub(}e\mathtt{)} \sim \mathtt{unit} \mathbin{@} (\mathtt{pub}, \mathtt{pub})}$$

The essential reason why this language guarantees confidentiality comes down to the following assertion:

The command $\texttt{seq}(\texttt{cmd}(\texttt{rdPriv}); x.\texttt{wrPub}(x))$ is ill-typed at level $(\texttt{pub}, \texttt{priv})$.

That is, we cannot perform a private read followed by a public write in a program that is well-typed at the most general security level (that is, with the least read level and greatest write level).

Prove the critical case of this assertion by assuming that $\Gamma \vdash \texttt{seq}(\texttt{cmd}(\texttt{rdPriv}); x.\texttt{wrPub}(x)) \sim \tau \mathbin{@} (\texttt{pub}, \texttt{priv})$ is derivable by the typing rule for sequencing, and deriving a contradiction.

(d) (5 points) By inversion of the typing rule for sequencing, these are *all* of the typing judgements that may be derived from the assumption.

Solution:

1. $\Gamma \vdash \texttt{cmd}(\texttt{rdPriv}) : \texttt{cmd}[(r, w)](\texttt{string})$;
2. $\Gamma \vdash \texttt{rdPriv} \sim \texttt{string} \mathbin{@} (r, w)$;
3. $\Gamma, x{:}\texttt{string} \vdash \texttt{wrPub}(x) \sim \tau \mathbin{@} (\texttt{pub}, \texttt{priv})$, where $\tau = \texttt{unit}$.

(e) (5 points) Give all of the ordering judgements among security levels that are derivable from these facts. You will need to invert the sequencing rule, and you will need to take account of the inclusion rule!

Solution:

1. $r \sqsubseteq \texttt{pub}$ and $\texttt{priv} \sqsubseteq w$ (from the sequencing rule).
2. $\texttt{priv} \sqsubseteq r$ (from the read private rule)

(f) (5 points) Draw a contradiction from the ordering judgements just derived.

Solution: If $\texttt{priv} \sqsubseteq r$, then $r = \texttt{priv}$, but then $r \not\sqsubseteq \texttt{pub}$.

Question 4 [30]: Dynamic Typing

Let us extend Dynamic PCF (PFPL Chapter 22) with the following additional constructs inspired by those found in dynamic languages such as Lisp or Scheme:

$$d \quad ::= \quad \ldots \mid \mathsf{cons}(d_1; d_2) \mid \mathsf{nil} \mid \mathsf{car}(d) \mid \mathsf{cdr}(d) \mid \mathsf{ifcons}(d; d_1; d_2)$$

The expression $\mathsf{cons}(d_1; d_2)$ forms a pair of dynamic values as a dynamic value, and the expression nil may be thought of as the "null pair". The operations car and cdr extract the first and second components, respectively, of a pair created with $\mathsf{cons}(d_1; d_2)$. The expression $\mathsf{ifcons}(d; d_1; d_2)$ evaluates d, and evaluates either d_1 or d_2 according to whether the value of d is nil.

It is common in dynamic languages with these constructs to represent lists as compositions of cons cells ending with nil, so that the list $[1, 2, 3]$ would be represented by the expression

$$\mathsf{cons}(1; \mathsf{cons}(2; \mathsf{cons}(3; \mathsf{nil}))).$$

Using this representation the app function to append lists may be written in Dynamic PCF as follows:

```
fix app is
  λ x.
    λ y.
      ifcons x then cons(car(x),app(cdr(x))(y)) else y
```

First, each call to the function incurs *two* dynamic checks (because it is curried). Second, the expressions $\mathsf{car}(x)$ and $\mathsf{cdr}(x)$ incur dynamic checks to ensure that x is a dynamic pair, but only one such check is really necessary. We will explore the sources of these redundancies, and how they may be remedied in a hybrid typed language.

(a) (10 points) Give the dynamic semantics rules for these constructs, *bearing in mind* that this is a dynamic language. You must define the judgement $d \mapsto d'$ and $d\ \mathsf{err}$. You will require four auxiliary judgements, $d\ \mathsf{is\_cons}\ (d_1, d_2)$, $d\ \mathsf{isnt\_cons}$, $d\ \mathsf{is\_nil}$, and $d\ \mathsf{isnt\_nil}$, which you are to define. To get you started, here are the congruence rules for transition and error propagation; you are to give the other rules.

$$\frac{d_1 \mapsto d_1'}{\mathsf{cons}(d_1; d_2) \mapsto \mathsf{cons}(d_1'; d_2)} \qquad \frac{d_1\ \mathsf{value} \quad d_2 \mapsto d_2'}{\mathsf{cons}(d_1; d_2) \mapsto \mathsf{cons}(d_1; d_2')}$$

$$\frac{d_1\ \mathsf{err}}{\mathsf{cons}(d_1; d_2)\ \mathsf{err}} \qquad \frac{d_1\ \mathsf{value} \quad d_2\ \mathsf{err}}{\mathsf{cons}(d_1; d_2)\ \mathsf{err}}$$

$$\frac{d \mapsto d'}{\mathsf{car}(d) \mapsto \mathsf{car}(d')} \qquad \frac{d\ \mathsf{err}}{\mathsf{car}(d)\ \mathsf{err}} \qquad \frac{d \mapsto d'}{\mathsf{cdr}(d) \mapsto \mathsf{cdr}(d')} \qquad \frac{d\ \mathsf{err}}{\mathsf{cdr}(d)\ \mathsf{err}}$$

$$\frac{d \mapsto d'}{\mathsf{ifcons}(d; d_1; d_2) \mapsto \mathsf{ifcons}(d'; d_1; d_2)} \qquad \frac{d\ \mathsf{err}}{\mathsf{ifcons}(d; d_1; d_2)\ \mathsf{err}}$$

Concentrate only on the extensions described above; do not bother with rules governing the other constructs of Dynamic PCF.

Solution:

$$\frac{d \text{ is\_cons } (d_1, d_2)}{\mathsf{car}(d) \mapsto d_1} \qquad \frac{d \text{ isnt\_cons}}{\mathsf{car}(d) \text{ err}} \qquad \frac{d \text{ is\_cons } (d_1, d_2)}{\mathsf{cdr}(d) \mapsto d_2} \qquad \frac{d \text{ isnt\_cons}}{\mathsf{cdr}(d) \text{ err}}$$

$$\frac{d \text{ value} \quad d \text{ is\_cons } (\_, \_)}{\mathsf{ifcons}(d; d_1; d_2) \mapsto d_1} \qquad \frac{d \text{ value} \quad d \text{ isnt\_cons}}{\mathsf{ifcons}(d; d_1; d_2) \mapsto d_2}$$

$$\frac{}{\mathsf{cons}(d_1; d_2) \text{ is\_cons } (d_1, d_2)} \qquad \frac{}{\mathsf{cons}(d_1; d_2) \text{ isnt\_nil}} \qquad \frac{}{\mathsf{nil} \text{ is\_nil}} \qquad \frac{}{\mathsf{nil} \text{ isnt\_cons}}$$

Recall that Hybrid PCF is an extension of statically typed PCF with a type dyn of dynamic values. For the purposes of this question, we will further assume that Hybrid PCF has product and option types. Now we extend Hybrid PCF with two classes, Cons and Nil, of values of type dyn. The new and cast operations for these classes have the following typing rules:

$$\frac{\Gamma \vdash e : \mathsf{unit}}{\Gamma \vdash \mathsf{new}[\mathsf{Nil}](e) : \mathsf{dyn}} \qquad \frac{\Gamma \vdash e : \mathsf{dyn}}{\Gamma \vdash \mathsf{cast}[\mathsf{Nil}](e) : \mathsf{unit}}$$

$$\frac{\Gamma \vdash e : \mathsf{dyn} \times \mathsf{dyn}}{\Gamma \vdash \mathsf{new}[\mathsf{Cons}](e) : \mathsf{dyn}} \qquad \frac{\Gamma \vdash e : \mathsf{dyn}}{\Gamma \vdash \mathsf{cast}[\mathsf{Cons}](e) : \mathsf{dyn} \times \mathsf{dyn}}$$

The dynamics for these constructs is as in Hybrid PCF:

$$\frac{e \text{ value}}{\mathsf{cast}[c](\mathsf{new}[c](e)) \mapsto e} \qquad \frac{e \text{ value} \quad c \neq c'}{\mathsf{cast}[c](\mathsf{new}[c'](e)) \text{ err}}$$

(b) (5 points) Give definitions of nil, $\mathsf{cons}(e_1; e_2)$, $\mathsf{car}(e)$ and $\mathsf{cdr}(e)$ in Hybrid PCF so that they implement the dynamics given above and they obey the following typing rules:

$$\frac{}{\Gamma \vdash \mathsf{nil} : \mathsf{dyn}} \qquad \frac{\Gamma \vdash e_1 : \mathsf{dyn} \quad \Gamma \vdash e_2 : \mathsf{dyn}}{\Gamma \vdash \mathsf{cons}(e_1; e_2) : \mathsf{dyn}} \qquad \frac{\Gamma \vdash e : \mathsf{dyn}}{\Gamma \vdash \mathsf{car}(e) : \mathsf{dyn}} \qquad \frac{\Gamma \vdash e : \mathsf{dyn}}{\Gamma \vdash \mathsf{cdr}(e) : \mathsf{dyn}}$$

Solution:

$$\begin{aligned} \mathsf{nil} &:= \mathsf{new}[\mathsf{Nil}](\langle\rangle) \\ \mathsf{cons}(e_1; e_2) &:= \mathsf{new}[\mathsf{Cons}](\langle e_1, e_2 \rangle) \\ \mathsf{car}(e) &:= \mathsf{fst}(\mathsf{cast}[\mathsf{Cons}](e)) \\ \mathsf{cdr}(e) &:= \mathsf{snd}(\mathsf{cast}[\mathsf{Cons}](e)) \end{aligned}$$

To define the conditional branch $\mathsf{ifcons}(e; e_1; e_2)$ in Hybrid PCF it is sufficient to enrich it with a new expression, $\mathsf{as}[c](e)$, that returns null if its argument is not of class c, and that returns $\mathsf{just}(e')$ if it is of class c and its instance data is e'.

$$\frac{\Gamma \vdash e : \mathsf{dyn}}{\Gamma \vdash \mathsf{as}[\mathsf{Nil}](e) : \mathsf{opt}(\mathsf{unit})} \qquad \frac{\Gamma \vdash e : \mathsf{dyn}}{\Gamma \vdash \mathsf{as}[\mathsf{Cons}](e) : \mathsf{opt}(\mathsf{dyn} \times \mathsf{dyn})}$$

The dynamics of $\mathsf{as}[c](e)$ is given by the following rules (omitting the obvious congruence rules):

$$\frac{e \text{ value}}{\mathsf{as}[c](\mathsf{new}[c](e)) \mapsto \mathsf{just}(e)} \qquad \frac{c \neq c' \quad e \text{ value}}{\mathsf{as}[c](\mathsf{new}[c'](e)) \mapsto \mathsf{null}}$$

(c) (5 points) Give a definition of $\mathsf{ifcons}(e; e_1; e_2)$, which evaluates to e_1 is the value of e is of class Cons and to e_2 otherwise.

Solution:

$$\mathsf{ifcons}(e; e_1; e_2) \quad ::= \quad \mathsf{ifnull}(\mathsf{as}[\mathsf{Cons}](e); e_2; \_.e_1)$$

(d) (5 points) Translate the function *app* given above into an expression of Hybrid PCF of type dyn → dyn → dyn by expanding the definitions of $\mathsf{car}(e)$, $\mathsf{cdr}(e)$, and $\mathsf{ifcons}(e; e_1; e_2)$ given above.

Solution:

```
fix app : dyn → dyn → dyn is
  λ x : dyn.
    λ y : dyn.
      ifnull(as[Cons](x); y;
        _.new[Cons](⟨fst(cast[Cons](x)),
             app(snd(cast[Cons](x)))(y)⟩))
```

(e) (5 points) The solution to the preceding question reveals two redundant dynamic class checks arising from the casts of values that are known to be of class Cons. Rewrite the answer to the preceding question to eliminate the redundant checks by taking advantage of the bound variable of the ifnull elimination form for option types.

Solution:

```
fix app : dyn → dyn → dyn is
  λ x : dyn.
    λ y : dyn.
      ifnull(as[Cons](x); y;
        u.new[Cons](⟨fst(u), app(snd(u))(y)⟩))
```

Question 5 [25]: Parallelism

In this problem, we will analyze the cost of a parallel algorithm to select the k^{th} smallest element from an unordered sequence of numbers. To do this, we will first extend the data-parallel language described in Chapter 43 of *PFPL* with a filter construct, which applies a given function to every element of the sequence, and selects those elements for which the function evaluates to true. The typing judgement and cost dynamics are given by the following rules.

$$\frac{\Gamma \vdash e_1 : \tau\ \mathsf{seq} \quad \Gamma, x : \tau \vdash e_2 : \mathsf{bool}}{\Gamma \vdash \mathsf{filter}(e_1; x.e_2) : \tau\ \mathsf{seq}}\ (TFilter)$$

$$\frac{e_1 \Downarrow^{c_1} \mathsf{seq}(v_0, ..., v_{n-1}) \quad \mathsf{map}(\mathsf{seq}(v_0, ..., v_{n-1}); x.e_2) \Downarrow^{c_2} \mathsf{seq}(b_0, ..., b_{n-1})}{\mathsf{filter}(e_1; x.e_2) \Downarrow^{c_1 \oplus c_2 \oplus (\otimes_{j=0}^{n-1} \mathbf{1})} \mathsf{seq}(v_i \mid b_i = \mathsf{true})}\ (EFilter)$$

The algorithm, written below, works as follows: we first pick a pivot element – in this case, we simply pick the first element in the sequence. In parallel, we then filter the sequence twice, selecting those elements less than the pivot into the sequence *less*, and selecting those elements greater than the pivot into the sequence *more*. We then compare k to the size of the two sequences. There are three cases to consider:

- the *less* sequence contains the element we are looking for. In this case we recurse on *less*.
- the pivot is the element we are looking for.
- the *more* sequence contains the element we are looking for. In this case, we recurse on *more*, adjusting k appropriately.

Below is the algorithm described above, with snippets of code missing for you to implement

```
fix selectk : nat seq → nat → nat is
 λ x : nat seq.λ k:nat.
  let pivot = x[0] in
   letpar less = ... and
          more = ... in
    if k < len(less) then
     selectk less k
    else
     if k < len(x) - len(more) then
      pivot
     else
      selectk more (k - (len(x) - len(more))
```

(a) (5 points) Give the first piece of missing code, that computes the value of less by selecting the elements of the sequence less than the pivot.

Solution: $filter(x; y.y < pivot)$

For this problem, assume that the cost of the arithmetic operations (subtraction, comparison) are computed in constant time. We are concerned with the size of the input as related

to the length of the sequence of numbers, not of the size of the numbers themselves. Also consider the input to be a value, that is, a fully computed sequence of numbers.

The running time of this algorithm is dependent on the input.

(b) (5 points) Give an input to *selectk* that exhibits the worst case running time. Assume that the filter operation returns the list of filtered elements in the same order as they appear in the input.

Solution: $selectk(\mathsf{seq}(0, 1, 2, ..., n-1), n-1)$

Let us now consider the expected running time for this algorithm. Let us now assume that the filter operation returns the list of filtered elements in random order. This, in effect, will allow us to assume that the expected size of *less* is equal to the expected size of *more*.

(c) (5 points) What is the asymptotic work of this algorithm? This is the time needed to run this algorithm on a single processor. Write, and solve, the recurrence function as a function of the length of the sequence.

Solution: For some constants k_0, k_1,
$W(0) = k_0$
$W(2n) = k_0 + 4nk_1 + W(n)$
$W(n) = O(n)$

(d) (5 points) What is the asymptotic depth of this algorithm? This is the time needed to run this algorithm on an unbounded number of processors. Write, and solve, the recurrence function as a function of the length of the sequence.

Solution: For some constants k_0, k_1,
$D(0) = k_0$
$D(2n) = k_0 + 4k_1 + D(n)$
$D(n) = O(lg(n))$

(e) (5 points) What is the parallelizability ratio of this algorithm? What does this imply about the ability to make use of multiple processors when executing this algorithm?

Solution: The parallelizability ratio is $n/lg(n)$. For a fixed number of processors, p, this means that we can make efficient use of them for sufficiently large problems, where $n/lg(n) >= p$. For a given input size n, this is the maximum number of processors that can be used to solve this problem efficiently; any extra processors would be forced to sit idle.

Question 6 [25]: The Spi-Calculus

Suppose Alice and Bob wish to securely communicate on a well-known channel. To do so, they use the following protocol. First, Alice and Bob must exchange their public keys.

Second, if Alice wants to contact Bob, she sends him a message consisting of a *nonce*, a freshly generated unguessable secret that identifies the message, encrypted by Bob's public key. When Bob receives this message, he decrypts it using his private key.

Third and finally, he responds to Alice with another (freshly generated) nonce encrypted by Alice's public key. When Alice receives this message, Alice and Bob now have the two nonces as a shared secret. This sort of protocol is useful in real-world situations when worrying about the threat of being compromised by a malicious third party who also knows about the open channel on which Alice and Bob are communicating.

In this question we formulate this protocol in an extension of the *synchronous* π-calculus with support for encryption, called the *spi-calculus*. It has the ability to generate *key pairs* consisting of a public and corresponding private key and is able to encrypt communication on a channel with a given key.

$$\begin{array}{rcl} P & ::= & \$(E) \mid 1 \mid \nu c.P \mid \nu(K, \overline{K}).P \mid P_1 \mid\mid P_2 \\ E & ::= & !_K c(a_1, ..., a_n); P \mid ?_{\overline{K}} c(a_1, ..., a_n.P) \mid 0 \mid E_1 + E_2 \end{array}$$

The operation $\nu(K, \overline{K}).P$ allocates a private and public key pair. The send event is parameterized by a public key, K and the receive event is parameterized by the corresponding private key, $\overline{K}$. This corresponds to the idea that messages sent on a channel are encrypted using the public key of the intended recipient, and are decrypted by the recipient using that principal's private key. (Each principal is responsible to protect its private key!) The send and receive events generate the obvious actions, which are synchronized as specified by the following rule:

$$\frac{P_1 \overset{!_K c(a_1,...,a_n)}{\longmapsto} P_1' \quad P_2 \overset{?_{\overline{K}} c(a_1,...,a_n)}{\longmapsto} P_2'}{P_1 \mid\mid P_2 \mapsto P_1' \mid\mid P_2'}$$

In other words synchronization is possible only when the receiver has the private key corresponding to the public key used by the sender.

Now that we have encryption in the spi-calculus, we can define Alice and Bob's handshake protocol as a process P of the form $\nu(K, \overline{K}).\nu c.(P_A \mid\mid P_B)$, where P_A represents Alice, P_B represents Bob. The key pair $K, \overline{K}$ and channel c are keys and channels well-known to Alice and Bob over which they may exchange their public keys. While not made explicit in the question, it is assumed that other principals also know about c and K, and also that after exchanging public keys, Alice and Bob have some way to verify that those keys actually belong to each other (such as meeting in person). You are to write the two process calculus expressions, P_A and P_B representing Alice and Bob in the protocol described above.

As shown in the protocol description, P_A and P_B can each be broken down into three phases; P_{Ai} represents what Alice does in phase i, and P_{Bi} represents what Bob does in phase i. In general, each phase for both Alice and Bob consist of a series of name generations (both keys and channels), synchronous sends, and synchronous receives, though not all of these have to occur in each phase. For nonces, they can use channel names, though only because channel names act as fresh, unguessable secrets.

(a) (11 points) Give the definitions of P_{A1} and P_{B1}.

Solution:

$$\begin{array}{rcl} P_{A1} & := & \nu(K_A, \overline{K_A}).\$(?_{\overline{K}}c(K_B.\$(!_{K_B}c(K_A); P_{A2}))) \\ P_{B1} & := & \nu(K_B, \overline{K_B}).\$(!_K c(K_B); \$(?_{\overline{K_B}}c(K_A.P_{B2}))) \end{array}$$

(b) (14 points) Give the definitions of P_{A2}, P_{B2}, P_{A3}, and P_{B3}.

Solution:

$$\begin{array}{rcl} P_{A2} & := & \nu a.\$(!_{K_B}c(a); P_{A3}) \\ P_{B2} & := & \$(?_{\overline{K_B}}c(a.P_{B3})) \\ \\ P_{A3} & := & \$(?_{\overline{K_A}}c(b.P'_A)) \\ P_{B3} & := & \nu b.\$(!_{K_A}c(b); P'_B) \end{array}$$

Scratch Work:

Scratch Work: