Types and Programming Languages (15-814), Fall 2018
Assignment 4: Data Representation

Contact: 15-814 Course Staff

Due Tuesday, October 16, 2018, 10:30am

This assignment is due by 10:30am on the above date and it must be submitted electronically as a PDF file on Canvas. Please use the attached template to typeset your assignment and make sure to include your full name and Andrew ID. As before, problems marked "WB" are subject to the whiteboard policy; all other problems must be done individually.

We have again provided you the syntax, statics, and dynamics for our simple language from class. Please ensure that your terms are syntactically correct and that they have the right type!

General hint. Consider whether the results from a given task can be used to solve subsequent tasks!

Task 1 (0 points)**.** How long did you spend on this assignment? Please list the questions that you discussed with classmates using the whiteboard policy.

1 Natural Numbers in Binary Form

Recall the definition of natural numbers in binary representation in the first line below and the isomorphism satisfied by this type.

$$\begin{array}{lcl} \mathit{bin} & = & \rho(\alpha.\,(\mathsf{b0} : \alpha) + (\mathsf{b1} : \alpha) + (\epsilon : 1)) \\ \mathit{bin} & \cong & (\mathsf{b0} : \mathit{bin}) + (\mathsf{b1} : \mathit{bin}) + (\epsilon : 1) \end{array}$$

We also defined the constants and functions *bzero*, *dbl0*, and *dbl1* encapsulating the constructors for binary numbers.

$$\begin{array}{lcll} \mathit{bzero} & : & \mathit{bin} & = \mathbf{fold}\,(\epsilon \cdot \langle\rangle) \\ \mathit{dbl0} & : & \mathit{bin} \to \mathit{bin} & = \lambda x.\,\mathbf{fold}\,(\mathsf{b0} \cdot x) \\ \mathit{dbl1} & : & \mathit{bin} \to \mathit{bin} & = \lambda x.\,\mathbf{fold}\,(\mathsf{b1} \cdot x) \end{array}$$

Also recall the increment function from lecture:

$$
\begin{array}{l}
\mathit{inc} : \mathit{bin} \to \mathit{bin} = \\
\quad \mathbf{fix}(i.\, \lambda x.\, \mathbf{case}\ (\mathbf{unfold}\ x) \\
\qquad\qquad \{\ \mathtt{b0} \cdot y \Rightarrow \mathbf{fold}\ (\mathtt{b1} \cdot y) \\
\qquad\qquad \mid \mathtt{b1} \cdot y \Rightarrow \mathbf{fold}\ (\mathtt{b0} \cdot (i\ y)) \\
\qquad\qquad \mid \epsilon \cdot \_ \Rightarrow \mathbf{fold}\ (\mathtt{b1} \cdot (\mathbf{fold}\ (\epsilon \cdot \langle\!|\, |\!\rangle)))\ \}
\end{array}
$$

In order to simplify the typesetting task and make the code easier to write and read, we encourage you to use "verbatim" code[1] that exploits pattern matching, recursive definitions of functions by name, and constructor functions with implicit fold and unfold operations. For example, the definitions above might be written as

```
bzero = eps ()
dbl0 x = b0 x
dlb1 x = b1 x

inc (b0 y) = b1 y
inc (b1 y) = b0 (inc y)
inc (eps _) = b1 (eps ())
```

Task 2 (5 points, WB)**.** Define a function $\mathit{plus} : \mathit{bin} \to \mathit{bin} \to \mathit{bin}$ that adds two natural numbers in binary form. You may use the functions and constants defined above.

Task 3 (5 points, WB)**.** Define a function $\mathit{eqbits} : \mathit{bin} \to \mathit{bin} \to \mathit{bool}$ that returns *true* if the two arguments are the exact same sequence of bits and *false* otherwise.

Task 4 (5 points, WB)**.** Demonstrate that *eqbits* is **not** a correct implementation of equality of natural numbers in binary representation. Summarize what you see as the source of the issue.

Task 5 (10 points, WB)**.** The problem exhibited in previous task can be addressed in several ways. Explain **at least two approaches** of how the problem in the previous section may be addressed, and provide the details for two different solutions. Provide definitions and code in each case as appropriate, including a function $\mathit{eq} : \mathit{bin} \to \mathit{bin} \to \mathit{bool}$ that correctly implements equality of natural numbers in binary form.

[1] Use the `verbatim` environment.

2 Representing Integers

We can represent integers a as pairs $\langle x, y\rangle$ of natural numbers x and y in binary form where $a = x - y$. We'll use this as an example of a recursive type that is not actually recursive.

$$\begin{array}{lcl} \mathit{int} & = & \rho\,\alpha.\,\mathit{bin} \otimes \mathit{bin} \\ \mathit{int} & \cong & \mathit{bin} \otimes \mathit{bin} \end{array}$$

In the tasks below, you should look for opportunities to use the functions defined in the previous problem on natural numbers in binary form.

Task 6 (5 points, WB)**.** Define a function *binToInt* : *bin* $\to$ *int* that, when given a representation of a the natural number n, returns an integer representing n.

Task 7 (5 points, WB)**.** Define a constant *izero* : *int* representing the integer 0 as well as functions *iplus* and *iminus* of type *int* $\to$ *int* $\to$ *int* representing addition and subtraction on integers.

Task 8 (5 points, WB)**.** Propose an alternative representation of integers using sums instead of products and define how mathematical integers are represented. You do not need to implement any functions on this representation, but please explain in a few sentences which one you would prefer and why.

3 Reasoning Inductively with Data

The values of so-called *purely positive types* only contain other values but not arbitrary expression. They follow this grammar:

$$\text{Positive types} \quad \tau^+ \quad ::= \quad \tau_1^+ \otimes \tau_2^+ \mid \mathbf{1} \mid \textstyle\sum_{i\in I}(i : \tau_i^+) \mid \rho(\alpha^+.\,\tau^+) \mid \alpha^+$$

We can now specialize the typing rules to *values of purely positive type*, giving us both an easy way to reason about canonical forms and inductively prove properties of programs. The rules for the judgment $\vdash v :: \tau^+$ establish simultaneously that v is a value and that it has type τ^+.

$$\frac{\vdash v :: \tau_i^+}{\vdash i \cdot v :: \textstyle\sum_{i\in I}(i :: \tau_i^+)}\ (\text{I-}+)$$

$$\frac{\vdash v_1 :: \tau_1^+ \quad \vdash v_2 :: \tau_2^+}{\vdash \langle v_1, v_2\rangle :: \tau_1^+ \otimes \tau_2^+}\ (\text{I-}\otimes) \qquad \frac{}{\vdash \langle\,\rangle :: \mathbf{1}}\ (\text{I-}1)$$

$$\frac{\vdash v :: [\rho(\alpha^+.\,\tau^+)/\alpha^+]\tau^+}{\vdash \mathbf{fold}\, v :: \rho(\alpha^+.\,\tau^+)}\ (\text{I-}\rho)$$

Observe that there are no elimination rules (since they don't form values), and that there are no variables or contexts.

Theorem. For any expression v and purely positive type τ^+, we have that $\vdash v :: \tau^+$ iff $\cdot \vdash v : \tau^+$ and v *val*.

Proof. In each direction, by rule induction on the given derivation. From right to left we also use inversion on v *val* to conclude that the given rules are the only applicable ones. □

Task 9 (5 points, WB)**.** Give a detailed proof showing that there are no closed values of type $\rho(\alpha^+.\, \alpha^+ \otimes \alpha^+)$ by assuming $\vdash v :: \rho(\alpha^+.\, \alpha^+ \otimes \alpha^+)$ and deriving a contradiction.

We can further specialize the rules to particular types, such as the type of *bin* of bit strings from Problem 1:

$$\begin{array}{rcl} \mathit{bin} & = & \rho\,\alpha.\, (\mathsf{b0} : \alpha) + (\mathsf{b1} : \alpha) + (\epsilon : 1) \\ \mathit{bin} & \cong & (\mathsf{b0} : \mathit{bin}) + (\mathsf{b1} : \mathit{bin}) + (\epsilon : 1) \end{array}$$

$$\frac{}{\vdash_{\mathit{bin}} \mathbf{fold}\,(\epsilon \cdot \langle\!|\,|\!\rangle) :: \mathit{bin}} \qquad \frac{\vdash_{\mathit{bin}} v :: \mathit{bin}}{\vdash_{\mathit{bin}} \mathbf{fold}\,(\mathsf{b0} \cdot v) :: \mathit{bin}} \qquad \frac{\vdash_{\mathit{bin}} v :: \mathit{bin}}{\vdash_{\mathit{bin}} \mathbf{fold}\,(\mathsf{b1} \cdot v) :: \mathit{bin}}$$

These rules are complete for closed values of type *bin* (which we do not prove, but is straightforward). They are interesting because they give rise to an *induction principle* for bit strings which is just a rule induction over this new judgment.

Task 10 (5 points, WB)**.** Prove that for all closed values v of type *bin*,

$$\mathit{inc}\, v \mapsto^* w \qquad \text{for some value } w$$

We now define a decrement function on numbers in binary representation.

$$\begin{array}{l} \mathit{dec} : \mathit{bin} \to \mathit{bin} = \mathbf{fix}\, d.\, \lambda x. \\ \quad \mathbf{case\ unfold}\, x\, \{\, \mathsf{b0} \cdot y \Rightarrow \mathsf{b1} \cdot (d\, y) \\ \qquad\qquad\quad \mid \mathsf{b1} \cdot y \Rightarrow \mathsf{b0} \cdot y \\ \qquad\qquad\quad \mid \epsilon \cdot u \Rightarrow \epsilon \cdot u\, \} \end{array}$$

In surface syntax, using data constructors, pattern matching and recursive definitions:

```
dec (b0 y) = b1 (dec y)
dec (b1 y) = b0 y
dec (eps _) = eps ()
```

Task 11 (10 points, WB)**.** Attempt to prove that for all closed values v of type *bin*

$$dec\,(inc\,v) \mapsto^* v$$

by using rule induction on the value typing $\vdash_{bin} v :: bin$.

This proof will fail in one case. Indicate how this failure might be addressed in light of Problem 1. You do not need to write new code or carry out a revised proof.

A Syntax

Types τ and terms e are given by the following grammars, where I ranges over finite index sets:

$$\tau ::= \alpha \mid \tau_1 \to \tau_2 \mid \tau_1 \otimes \tau_2 \mid 1 \mid \textstyle\sum_{i \in I}(i : \tau_i) \mid \mathbin{\&}_{i \in I}(i : \tau_i) \mid \rho(\alpha.\, \tau)$$

$$\begin{aligned} e ::= {} & x \\ & \mid \lambda x.\, e \mid e_1\, e_2 \\ & \mid i \cdot e \mid \textbf{case}\ e\ \{i \cdot x_i \Rightarrow e_i\}_{i \in I} \\ & \mid \langle e_1, e_2 \rangle \mid \textbf{case}\ e_0\ \{\langle x_1, x_2 \rangle \Rightarrow e'\} \\ & \mid \langle\,\rangle \mid \textbf{case}\ e_0\ \{\langle\,\rangle \Rightarrow e'\} \\ & \mid \langle\!|\, i \hookrightarrow e_i \,|\!\rangle_{i \in I} \mid e \cdot i \\ & \mid \textbf{fold}(e) \mid \textbf{unfold}(e) \\ & \mid \textbf{fix}(x.e) \end{aligned}$$

As discussed in class, the $\mathbf{0}$ is a special case of sums with $I = \emptyset$.

B Statics

For any type constructor $\%$, we use the name (I-%) for a rule that introduces (constructs) an expression of type $\%$ and (E-%) for a rule that eliminates (destructs) an expression of type $\%$. Variables and fixed points are special, since they are not tied to any particular type.

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ (\textsc{Var}) \qquad \frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda x.e : \tau \to \tau'}\ (\text{I-}\to)$$

$$\frac{\Gamma \vdash e_1 : \tau \to \tau' \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\, e_2 : \tau'}\ (\text{E-}\to)$$

$$\frac{\Gamma \vdash e : \tau_j \quad (j \in I)}{\Gamma \vdash j \cdot e : \sum_{i \in I}(i : \tau_i)}\ (\text{I-}+)$$

$$\frac{\Gamma \vdash e : \sum_{i \in I}(i : \tau_i) \quad \Gamma, x_i : \tau_i \vdash e_i : \tau \quad (\forall i \in I)}{\Gamma \vdash \textbf{case}\ e\ \{i \cdot x_i \Rightarrow e_i\}_{i \in I} : \tau}\ (\text{E-}+)$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \otimes \tau_2}\ (\text{I-}\otimes) \qquad \frac{\Gamma \vdash e_0 : \tau_1 \otimes \tau_2 \quad \Gamma, x_1 : \tau_1, x_2 : \tau_2 \vdash e' : \tau}{\Gamma \vdash \textbf{case}\ e_0\ \{\langle x_1, x_2 \rangle \Rightarrow e'\} : \tau}\ (\text{E-}\otimes)$$

$$\frac{}{\Gamma \vdash \langle\,\rangle : 1}\ (\text{I-}1) \qquad \frac{\Gamma \vdash e_0 : 1 \quad \Gamma \vdash e' : \tau}{\Gamma \vdash \textbf{case}\ e_0\ \{\langle\,\rangle \Rightarrow e'\} : \tau}\ (\text{E-}1)$$

$$\frac{\Gamma \vdash e_i : \tau_i \quad (\forall i \in I)}{\Gamma \vdash \langle\!|\, i \hookrightarrow e_i \,|\!\rangle_{i \in I} : \mathbin{\&}_{i \in I}(i : \tau_i)}\ (\text{I-}\&) \qquad \frac{\Gamma \vdash e : \mathbin{\&}_{i \in I}(i : \tau_i) \quad (j \in I)}{\Gamma \vdash e \cdot j : \tau_j}\ (\text{E-}\&)$$

$$\frac{\Gamma \vdash e : [\rho(\alpha.\,\tau)/\alpha]\tau}{\Gamma \vdash \textbf{fold}(e) : \rho(\alpha.\,\tau)}\ (\text{I-}\rho) \qquad \frac{\Gamma \vdash e : \rho(\alpha.\,\tau)}{\Gamma \vdash \textbf{unfold}(e) : [\rho(\alpha.\,\tau)/\alpha]\tau}\ (\text{E-}\rho)$$

$$\frac{\Gamma, x : \tau \vdash e : \tau}{\Gamma \vdash \textbf{fix}(x.e) : \tau}\ (\textsc{Fix})$$

C Dynamics

For any type constructor $\%$ we write (V-%) for defining a value of type $\%$, (R-%) for a reduction where a destructor meets a constructor, and (CI-%) and (CE-%) for the congruence rules for constructors and destructors for the type $\%$.

$$\frac{}{\lambda x.e\ \mathit{val}}\ (\text{V-}\to) \qquad \frac{v_2\ \mathit{val}}{(\lambda x.\, e_1)\, v_2 \mapsto [v_2/x]e_1}\ (\text{R-}\to)$$

$$\frac{e_1 \mapsto e_1'}{e_1\, e_2 \mapsto e_1'\, e_2}\ (\text{CE-}\to_1) \qquad \frac{v_1\ \mathit{val} \quad e_2 \mapsto e_2'}{v_1\, e_2 \mapsto e_1\, e_2'}\ (\text{CE-}\to_2)$$

$$\frac{v\ \mathit{val}}{i \cdot v\ \mathit{val}}\ (\text{V-}+) \quad \frac{e \mapsto e'}{i \cdot e \mapsto i \cdot e'}\ (\text{CI-}+) \quad \frac{e \mapsto e'}{\mathbf{case}\ e\ \{i \cdot x_i \Rightarrow e_i\}_{i\in I} \mapsto \mathbf{case}\ e'\ \{i \cdot x_i \Rightarrow e_i\}_{i\in I}}\ (\text{CE-}+)$$

$$\frac{v_j\ \mathit{val}}{\mathbf{case}\ (j \cdot v_j)\ \{i \cdot x_i \Rightarrow e_i\}_{i\in I} \mapsto [v_j/x_j]e_j}\ (\text{R-}+)$$

$$\frac{v_1\ \mathit{val} \quad v_2\ \mathit{val}}{\langle v_1, v_2\rangle\ \mathit{val}}\ (\text{V-}\otimes) \qquad \frac{e_1 \mapsto e_1'}{\langle e_1, e_2\rangle \mapsto \langle e_1', e_2\rangle}\ (\text{CI-}\otimes_1) \qquad \frac{v_1\ \mathit{val} \quad e_2 \mapsto e_2'}{\langle v_1, e_2\rangle \mapsto \langle v_1, e_2'\rangle}\ (\text{CI-}\otimes_2)$$

$$\frac{e_0 \mapsto e_0'}{\mathbf{case}\ e_0\ \{\langle x_1, x_2\rangle \Rightarrow e'\} \mapsto \mathbf{case}\ e_0'\ \{\langle x_1, x_2\rangle \Rightarrow e'\}}\ (\text{CE-}\otimes)$$

$$\frac{v_1\ \mathit{val} \quad v_2\ \mathit{val}}{\mathbf{case}\ \langle v_1, v_2\rangle\ \{\langle x_1, x_2\rangle \Rightarrow e'\} \mapsto [v_1/x_1, v_2/x_2]e'}\ (\text{R-}\otimes)$$

$$\frac{}{\langle\,\rangle\ \mathit{val}}\ (\text{V-1}) \qquad \frac{e_0 \mapsto e_0'}{\mathbf{case}\ e_0\ \{\langle\,\rangle \Rightarrow e'\} \mapsto \mathbf{case}\ e_0'\ \{\langle\,\rangle \Rightarrow e'\}}\ (\text{CE-1}) \qquad \frac{}{\mathbf{case}\ \langle\,\rangle\ \{\langle\,\rangle \Rightarrow e'\} \mapsto e'}\ (\text{R-1})$$

$$\frac{}{\langle\!|i \hookrightarrow e_i|\!\rangle_{i\in I}\ \mathit{val}}\ (\text{V-}\&) \qquad \frac{e \mapsto e'}{e \cdot j \mapsto e' \cdot j}\ (\text{CI-}\&) \qquad \frac{}{\langle\!|i \hookrightarrow e_i|\!\rangle_{i\in I} \cdot j \mapsto e_j}\ (\text{R-}\&)$$

$$\frac{v\ \mathit{val}}{\mathbf{fold}(v)\ \mathit{val}}\ (\text{V-}\rho) \qquad \frac{e \mapsto e'}{\mathbf{fold}(e) \mapsto \mathbf{fold}(e')}\ (\text{CI-}\rho)$$

$$\frac{e \mapsto e'}{\mathbf{unfold}(e) \mapsto \mathbf{unfold}(e')}\ (\text{CE-}\rho) \qquad \frac{v\ \mathit{val}}{\mathbf{unfold}(\mathbf{fold}(v)) \mapsto v}\ (\text{R-}\rho)$$

$$\frac{}{\mathbf{fix}(x.e) \mapsto [\mathbf{fix}(x.e)/x]e}\ (\text{R-FIX})$$