

Assignment 8

Data Abstraction

15-814: Types and Programming Languages
Jan Hoffmann & Corinthia Beatrix Aberlé

Due Thursday, November 14, 2024
75 points

You should hand in two files:

- `hw08.pdf` with your written solutions to the questions.
- `hw08.cbv` with the code for the LAMBDA implementation tasks.

1 Definability of existential types

In this assignment, we study properties of *existential types* in System F. For background and further information on existential types, see Chapter 17 of PFPL.

Existential types may be encoded using universal types:

$$\begin{aligned}\exists t. \tau &:= \forall u. (\forall t. \tau \rightarrow u) \rightarrow u \\ \text{pack}\{t.\tau\}(\rho, e) &= \Lambda u. \lambda c : (\forall t. \tau \rightarrow u). c[\rho](e) \\ \text{open}\{t.\tau; \tau_2\}(e, t, x.e_2) &= e[\tau_2](\Lambda t. \lambda x : \tau. e_2)\end{aligned}$$

Note that we annotate `open` with the type τ_2 computed by the client.

We may check that the encoding preserves the static semantics of existential types.

Task 1 (5 points) Given Δ, t type $\Gamma \vdash \tau$ type, $\Delta \vdash \rho$ type, and $\Delta \Gamma \vdash e : [\rho/t]\tau$, show that $\Delta \Gamma \vdash \text{pack}\{t.\tau\}(\rho, e) : \exists t. \tau$ under the given encoding.

Task 2 (5 points) Given $\Delta \Gamma \vdash e : \exists t. \tau$, Δ, t type $\Gamma, x : \tau \vdash e_2 : \tau_2$, and $\Delta \vdash \tau_2$ type, show that $\Delta \Gamma \vdash \text{open}\{t.\tau; \tau_2\}(e, t, x.e_2) : \tau_2$ under the given encoding.

We may check that the encoding preserves the call-by-name dynamics of existential types:

Task 3 (5 points) Show that $\text{pack}\{t.\tau\}(\rho, e) \text{ val}$.

Task 4 (5 points) Prove that if $e \mapsto e'$, then $\text{open}\{t.\tau; \tau_2\}(e, t, x.e_2) \mapsto^* \text{open}\{t.\tau; \tau_2\}(e', t, x.e_2)$.

Task 5 (15 points) Show that $\text{open}\{t.\tau; \tau_2\}(\text{pack}\{t.\tau\}(\rho, e), t, x.e_2) \mapsto^* [\rho/t, e/x]e_2$.

2 Representation Independence

In this problem you are asked to provide an implementation of an integer counter. In the first one we represent an integer a as a pair $\langle x, y \rangle$ of two natural numbers x and y where $a = x - y$. In the second, we represent an integer $a \geq 0$ as $\text{pos} \cdot a$ and an integer $a \leq 0$ as $\text{neg} \cdot -a$. Note that neither of these representations is unique.

We say that a pair $\langle x, y \rangle$ of natural numbers represents the integer a if $a = x - y$. We call this the *difference representation* and call the representation type *diff*. For the sake of simplicity, we choose a unary representation for the natural numbers.

$$\begin{aligned} \text{bool} &= (\text{true} : 1) + (\text{false} : 1) \\ \text{nat} &= \mu\alpha. (\text{zero} : 1) + (\text{succ} : \alpha) \\ \text{diff} &= \text{nat} \times \text{nat} \end{aligned}$$

The implementations of each of the following counter operations are given in `hw09.cbv`:

- (i) A constant $d_zero : \text{diff}$ representing the integer 0.
- (ii) The function $d_inc : \text{diff} \rightarrow \text{diff}$ representing incrementing integers.
- (iii) The function $d_dec : \text{diff} \rightarrow \text{diff}$ representing decrementing integers.
- (iv) The function $d_is0 : \text{diff} \rightarrow \text{bool}$ that tests whether the state of the counter represents 0.

Task 6 (10 points) We consider an alternative *signed representation* of integers where

$$\text{sign} = (\text{pos} : \text{nat}) + (\text{neg} : \text{nat})$$

where $\text{pos} \cdot x$ represents the integer x and $\text{neg} \cdot x$ represents the integer $-x$.

Define the following functions in analogy with the previous set of functions and include them in the file `hw09.cbv`.

- (i) $s_zero : \text{sign}$
- (ii) $s_inc : \text{sign} \rightarrow \text{sign}$
- (iii) $s_dec : \text{sign} \rightarrow \text{sign}$
- (iv) $s_is0 : \text{sign} \rightarrow \text{bool}$

Task 7 (5 points) With the definitions from previous two tasks you should be able to implement the following signature for an integer counter:

```
INTCTR = {
  type ictr
  init : ictr
  inc : ictr → ictr
  dec : ictr → ictr
  is0 : ictr → bool
}
```

where *init*, *inc*, *dec* and *is0* have their obvious specification with respect to integers (with *init* representing a counter with initial value 0), generalizing the natural number counter defined in lecture. Provide the following definitions in the file `hw09.cbv`.

- (i) The type *INTCTR* as an existential type.
- (ii) *DiffCtr* : *INTCTR*, using the difference representation of integers.
- (iii) *SignCtr* : *INTCTR*, using the signed representation of integers.

Task 8 (25 points) In this task you are asked to show that the two implementations of integer counters from the previous subtask are logically equivalent. Make sure your previous implementations are correct or else this may be difficult!

In the proofs below you may freely use the correctness of functions on unary numbers, specifically $zero = \bar{0}$ and $succ\ \bar{n} = \bar{n} + \bar{1}$. If you need properties of other functions on (unary) natural numbers you should carefully state them and assume them as lemmas, but you do not need to prove them.

- (i) Define a relation *R* that allows you to prove $DiffCtr \approx SignCtr \in \llbracket INTCTR \rrbracket$.
- (ii) Prove that $d_zero \approx s_zero \in \llbracket R \rrbracket$. If this is not straightforward, you may want to rethink your definition of *R*.
- (iii) Prove that $d_inc \approx s_inc \in \llbracket R \rightarrow R \rrbracket$.
(You should also convince yourself that $d_dec \approx s_dec \in \llbracket R \rightarrow R \rrbracket$, but you do not need to prove it.)
- (iv) Prove that $d_is0 \approx s_is0 \in \llbracket R \rightarrow bool \rrbracket$.

If all of the above holds, then you know that no client of your *INTCTR* interface can distinguish the two implementations!