Assignment 9
Data Abstraction and Programs as Proofs

15-814: Types and Programming Languages
Frank Pfenning & David M Kahn

Due Thursday, November 11, 2021
70 points

This assignment is due by 11:59pm on the above date and it must be submitted electronically on Gradescope. You should hand in three files:

- `hw09.pdf` with your written solutions to the questions. You may exclude those questions whose answer is just code, but please leave a pointer to the code.
- `hw09.cbv` with the code for Task 1–3, where the solutions to the questions are clearly marked and auxiliary code (either from lecture or your own) is included so it passes the LAMBDA implementation.
- `hw09.prf` with the proofs (written as programs) for Tasks 5–7. It should pass the LAMBDA implementation. ***Note you will need version 1.1 (from November 5, 2021) or greater.***

This assignment has 10 bonus points in the sense that your total homework score is computed out of 600 points, but the total value of all assignments after this one is 610 points.

1 Representation Independence

In this problem you are asked to provide two different implementations of integers $\ldots, -2, -1, 0, 1, 2, \ldots$. 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 $\mathbf{pos} \cdot a$ and an integer $a \leq 0$ as $\mathbf{neg} \cdot -a$. Note that neither of these representations is unique.

Task 1 (5 pts) 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 if simplicity, we choose a unary representation for the natural numbers.

$$\begin{array}{lcl} \mathit{bool} & = & (\mathbf{true} : 1) + (\mathbf{false} : 1) \\ \mathit{nat} & = & \mu\alpha.\, (\mathbf{zero} : 1) + (\mathbf{succ} : \alpha) \\ \mathit{diff} & = & \mathit{nat} \times \mathit{nat} \end{array}$$

Provide implementations of each of the following functions. Your answers to each of the following should be included in the file `hw09.cbv`. If you need auxiliary functions on natural numbers, you should define them.

(i) Define a constant *d_zero* : *diff* representing the integer 0.

(ii) Define the function *d_inc* : *diff* $\to$ *diff* representing incrementing integers.

(iii) Define the function *d_dec* : *diff* $\to$ *diff* representing decrementing integers.

(iv) Define the function *d_is0* : *diff* $\to$ *bool* that tests whether the state of the counter represents 0.

Task 2 (5 pts) We consider an alternative *signed representation* of integers where

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

where $\mathbf{pos} \cdot x$ represents the integer x and $\mathbf{neg} \cdot x$ 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* : *sign*

(ii) *s_inc* : *sign* $\to$ *sign*

(iii) *s_dec* : *sign* $\to$ *sign*

(iv) *s_is0* : *sign* $\to$ *bool*

Task 3 (5 pts) 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* $\to$ *ictr*
  dec : *ictr* $\to$ *ictr*
  is0 : *ictr* $\to$ *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 4 (25 pts) 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* $= \overline{0}$ and *succ* $\overline{n} = \overline{n+1}$. 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* $\in [\![\mathit{INTCTR}]\!]$.

(ii) Prove that *d_zero* $\approx$ *s_zero* $\in [\![R]\!]$. If this is not straightforward, you may want to rethink your definition of R.

(iii) Prove that *d_inc* $\approx$ *s_inc* $\in [\![R \to R]\!]$.

(You should also convince yourself that *d_dec* $\approx$ *s_dec* $\in [\![R \to R]\!]$, but you do not need to prove it.)

(iv) Prove that *d_is0* $\approx$ *s_is0* $\in [\![R \to \mathit{bool}]\!]$.

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

2 Programs as Proofs

In some tasks below you are asked to provide proofs in the form of expressions in LAMBDA. Proof checking imposes some additional restrictions on the types and expressions in our language, namely, the absence of recursion and enforcing that all cases are covered in case expressions. Files with extension `.prf` (or, equivalently, option `--lang=prf`) enforce these restrictions. LAMBDA implements variadic records of type $\&_{i \in I}(i : \tau_i)$, where $\tau \,\&\, \sigma$ is short-hand for $(\mathbf{l} : \tau) \,\&\, (\mathbf{r} : \sigma)$ (or `('l : tau) & ('r : sigma)` in concrete syntax). The syntax of lazy pairs and projection should therefore use the right-hand sides of the definitions below:

$$\begin{array}{lcl} \langle e_1, e_2 \rangle & \triangleq & \texttt{(| 'l => e1 | 'r => e2 |)} \\ \mathsf{fst}\ e & \triangleq & \texttt{e.'l} \\ \mathsf{snd}\ e & \triangleq & \texttt{e.'r} \end{array}$$

For example, the proof from lecture that $(A \wedge B) \supset (B \wedge A)$ (which is $(A \,\&\, B) \to (B \,\&\, A)$ under the propositions-as-types interpretation) becomes the following:

```
1 decl sym_and : !a. !b. a & b -> b & a
2 defn sym_and = /\a. /\b. \p. (| 'l => p.'r | 'r => p.'l |)
```

Note that the quantification over propositions, which usually just made in our mathematical metalanguage (*the implication should hold for all propositions A and B*), is instead explicit in our proof language.

Your solutions should be in the file `hw09.prf`.

Task 5 (L15.1, 10 points) One proposition is *more general* than another if we can instantiate the propositional variables in the first to obtain the second. For example, $A \supset (B \supset A)$ is more general than $A \supset (\bot \supset A)$ (with $[\bot/B]$), $(C \wedge D) \supset (B \supset (C \wedge D))$ (with $[C \wedge D/A]$, but not more general than $C \supset (D \supset E)$.

For each of the following proof terms, give the most general proposition proved by it. (We are justified in saying "*THE most general*" because the most general proposition is unique up to the names of the propositional variables.)

(i) $\lambda u.\, \lambda w.\, \lambda k.\, w\,(u\,k)$

(ii) $\lambda w.\, \langle\!|(\lambda u.\, w\,(\mathbf{l}\cdot u)), (\lambda k.\, w\,(\mathbf{r}\cdot k))|\!\rangle$

(iii) $\lambda x.\, (\mathsf{fst}\, x)\,(\mathsf{snd}\, x)\,(\mathsf{snd}\, x)$

(iv) $\lambda x.\, \lambda y.\, \lambda z.\, (x\, z)\,(y\, z)$

Check the correctness of the typing in LAMBDA and include four definitions in `hw09.prf`. LAMBDA *can not* guarantee that your proposition is most general, so it only partially verifies your answer.

Task 6 (10 pts) We only briefly mentioned in lecture that $\neg A \triangleq A \supset \bot$ where $\bot$ represents falsehood. Falsehood has no introduction rule and one elimination rule $\bot E$ which is the nullary version $\vee E$. You can find these in the notes to Lecture 17. Under the Curry-Howard isomorphism, $\bot$ is related to the empty type 0, so $\neg A$ is interpreted as $A \to 0$.

While propositional intuitionistic logic does not allow the law of excluded middle $A \vee \neg A$ for arbitrary A, it is certainly true for *some* A (like $A = (B \supset B)$ or $A = \bot$). So it is *not* the case that $\neg(A \vee \neg A)$. Perhaps surprisingly, we can actually establish this fact in intuitionistic logic by proving $\neg\neg(A \vee \neg A)$. This is an instance of Glivenko's theorem which states that A is true in *classical propositional logic* if and only if $\neg\neg A$ is true in *intuitionistic propositional logic*.[1]

> Provide a proof term for $\neg\neg(A \vee \neg A)$, that is, provide an expression of type $((A + (A \to 0)) \to 0) \to 0$.

Your proof term should be included in `hw09.prf`. By the Curry-Howard isomorphism, there will be a corresponding natural deduction, but you do not have to typeset it. Depending on your background and experience, you may want to try to prove $\neg\neg(A \vee \neg A)$ first in natural deduction and then read off the proof term, or just write the program.

Task 7 (L17.3, 10 points) In lecture we mentioned how lazy and eager pairs induce different logical rules for conjunction (logical "and"). In particular, we found the elimination rule

$$\dfrac{\begin{array}{c}\overline{A\ \mathit{true}}\ x \quad \overline{B\ \mathit{true}}\ y \\ \vdots \\ A \mathbin{\overline{\wedge}} B\ \mathit{true} \qquad C\ \mathit{true}\end{array}}{C\ \mathit{true}}\ \overline{\wedge}E^{x,y}$$

where $A \mathbin{\overline{\wedge}} B$ corresponds to the eager pairs $A \times B$. This rule is different from the elimination rules for $\wedge$ which correspond to the projections from lazy pairs $A \mathbin{\&} B$. In this problem you will prove that the two forms of conjunction are equivalent, that is, imply each other in intuitionistic logic.

(i) Give the introduction rule for $\overline{\wedge}$, i.e., the rule that allows you to conclude $A \mathbin{\overline{\wedge}} B\ \mathit{true}$.

(ii) Prove that $(A \wedge B) \supset (A \mathbin{\overline{\wedge}} B)\ \mathit{true}$.

(iii) Prove that $(A \mathbin{\overline{\wedge}} B) \supset (A \wedge B)\ \mathit{true}$.

Give your answers to (ii) and (iii) both in the form of natural deduction and proof terms. The proof terms should be included in your file `hw09.prf`.

[1] "Propositional" here means that there are no quantifiers in A. With quantifiers, matters become more complicated.