Poor Man's Axi: DPL-like proofs in Type Theory

Wojciech Kołowski

Layers in Poor Man's Axi

In our original proposal of Poor Man's Axi, the language was split into two layers:

- A programming layer which consists of a strongly-typed functional programming language based on the Simply Typed Lambda Calculus.
- A logical layer which consists of second-order classical logic with equality and some primitives for reasoning by cases and induction (note that it's not full second-order logic, because it only allows quantification over functions, but not over propositions, predicates or relations).

Logic without proofterms

The logic was presented with a bunch of judgements, the most important being the true proposition judgement $\Gamma \mid \Delta \vdash P$. While this presentation does a good job of explaining what the logic is like, it does not address the problem of writing proofs from the perspective of the programmer.

Logic with proofterms

To deal with this problem, we introduce proofterms (also known as proof certificates), which are expressions that will serve as the "proofs" that the user will write. The manipulation of judgements which establish their correctness (which we will describe soon) is left to the language's proof checker.

Proofterms

Proofterms (here P are propositions, t are terms, x are variables):

$e ::=$

$P \mid$ **true** $\mid$ **exfalso** e

assume P **in** $e \mid$ **modus-ponens** $e_1\ e_2 \mid$

suppose-absurd P **in** $e \mid$ **absurd** $e_1\ e_2 \mid$

both $e_1\ e_2 \mid$ **left-and** $e \mid$ **right-and** $e \mid$

equivalence $e_1\ e_2 \mid$ **left-iff** $e \mid$ **right-iff** $e \mid$

left-either $P\ e \mid$ **right-either** $P\ e \mid$

constructive-dilemma $e_1\ e_2\ e_3 \mid$

double-negation $e \mid$

$e_1; e_2 \mid$

pick-any x **in** $e \mid$ **specialize** e **with** $t \mid$

exists t **such that** $e \mid$ **pick-witness** x **for** e_1 **in** $e_2 \mid$

refl $t \mid$ **rewrite** e_1 **in** $e_2 \mid$

case t **of** $(\texttt{inl}\ a \to e_1, \texttt{inr}\ b \to e_2) \mid$

Proofterms – overview

A quick glance at the above grammar:

- First, propositions are included in the syntactic category of proofterms, because we will refer to an assumption P with P itself.
- Then we have some proofterms for dealing with propositional logic, divided as usual into introduction and elimination forms.
- **double-negation** is our way of making the logic classical.
- $e_1; e_2$ is a cut (or, in programming language terms, a let-expression, although without any variable binding).
- Then we have some proofterms to deal with quantifiers.
- **refl** and **rewrite** are primitives for dealing with equality.
- Finally, **case** will be used for reasoning by cases on terms of type $A + B$.

Examples

Before we jump right into the rules, let's see some examples. The point of the first two is to show that our proofterms look very similar to proofs from DPLs. The third example shows how simple reasoning about programs might look like.

Example – propositional logic

Theorem: $(P \Rightarrow Q) \Rightarrow (Q \Rightarrow R) \Rightarrow P \Rightarrow R$.

Proof:
assume $P \Rightarrow Q$ **in**
 assume $Q \Rightarrow R$ **in**
  assume P **in**
   modus-ponens $(P \Rightarrow Q)$ P;
   modus-ponens $(Q \Rightarrow R)$ Q

The proof looks the same as the DPL one (page 71 in the DPL thesis), except that we don't have **begin** and **end**.

Example – first-order logic

Theorem: $(\forall x : A.\, P\ x \wedge Q\ x) \Rightarrow (\forall x : A.\, P\ x) \wedge (\forall x : A.\, Q\ x)$

Proof:

assume $\forall x : A.\, P\ x \wedge Q\ x$ **in**
  pick-any y **in**
    specialize $\forall x : A.\, P\ x \wedge Q\ x$ **with** y;
    left-and $P\ y \wedge Q\ y$;
  pick-any y **in**
    specialize $\forall x : A.\, P\ x \wedge Q\ x$ **with** y;
    right-and $P\ y \wedge Q\ y$;
  both $(\forall y : A.\, P\ y)$ $(\forall y : A.\, Q\ y)$

Again, the proof looks the same as the DPL one (page 156 in the DPL thesis), except we use indentation instead of **begin** and **end**.

Example – proof about a program

Consider the following program:
$\texttt{swap} : A + B \to B + A := \lambda x.\texttt{case}\ x\ \texttt{of}\ (\lambda a.\texttt{inr}\ a, \lambda b.\texttt{inl}\ b)$

It's pretty clear that this function is an involution.
Theorem: $\forall x : A + B.\, \texttt{swap}\ (\texttt{swap}\ x) = x$

Proof:
pick-any x **in**
  case x **of** $(\texttt{inl}\ a \to$ **refl** $a, \texttt{inr}\ b \to$ **refl** $b)$

The proof has the same structure as the proofterm you would write in Coq, except for the syntactic differences.

Correct proof judgement

We now proceed to describe proofterms in much greater detail. will modify the language in the following way: we throw away the true proposition judgement $\Gamma \mid \Delta \vdash P$ and replace it with the correct proof judgement $\Gamma \mid \Delta \vdash e : P$ which should be read: in typing context Γ and assumption context Δ, e is a proof of P.

The other judgements we need in our logic, the valid assumption context judgement $\Gamma \vdash \Delta \ \texttt{valid}$ and the well-formed proposition judgement $\Gamma \vdash P \ \texttt{prop}$, are unchanged. Note that we will set up the system so that $\Gamma \mid \Delta \vdash e : P$ entails $\Gamma \vdash \Delta \ \texttt{valid}$ and $\Gamma \vdash P \ \texttt{prop}$ as sanity checks.

Assumptions

$$\frac{\Gamma \vdash \Delta \ \texttt{valid} \quad P \in \Delta}{\Gamma \mid \Delta \vdash P : P}\,\textsc{Ass}$$

The basic rule of our logic is that we can use assumptions from the assumption context. Note that here P denotes both the proposition and its proof.

True and false

$$\frac{\Gamma \vdash \Delta \ \texttt{valid}}{\Gamma \mid \Delta \vdash \textbf{true} : \top}\ \textsc{True-Intro}$$

$$\frac{\Gamma \mid \Delta \vdash e : \bot}{\Gamma \mid \Delta \vdash \textbf{exfalso}\ e : P}\ \textsc{False-Elim}$$

true establishes the truth of the true proposition $\top$, whereas **exfalso** e proves any proposition P whatsoever, provided that e proves $\bot$.

Implication

$$\frac{\Gamma \mid \Delta, P \vdash e : Q}{\Gamma \mid \Delta \vdash \textbf{assume } P \textbf{ in } e : P \Rightarrow Q}\textsc{Impl-Intro}$$

$$\frac{\Gamma \mid \Delta \vdash e_1 : P \Rightarrow Q \quad \Gamma \mid \Delta \vdash e_2 : P}{\Gamma \mid \Delta \vdash \textbf{modus-ponens } e_1\ e_2 : Q}\textsc{Impl-Elim}$$

Conjunction

$$\frac{\Gamma \mid \Delta \vdash e_1 : P \quad \Gamma \mid \Delta \vdash e_2 : Q}{\Gamma \mid \Delta \vdash \textbf{both}\ e_1\ e_2 : P \wedge Q}\ \textsc{And-Intro}$$

$$\frac{\Gamma \mid \Delta \vdash e : P \wedge Q}{\Gamma \mid \Delta \vdash \textbf{left-and}\ e : P}\ \textsc{And-Elim-L}$$

$$\frac{\Gamma \mid \Delta \vdash e : P \wedge Q}{\Gamma \mid \Delta \vdash \textbf{right-and}\ e : Q}\ \textsc{And-Elim-R}$$

Disjunction

$$\frac{\Gamma \mid \Delta \vdash e : P}{\Gamma \mid \Delta \vdash \textbf{left-either}\ Q\ e : P \vee Q}\ \textsc{Or-Intro-L}$$

$$\frac{\Gamma \mid \Delta \vdash e : Q}{\Gamma \mid \Delta \vdash \textbf{right-either}\ P\ e : P \vee Q}\ \textsc{Or-Intro-R}$$

$$\frac{\Gamma \mid \Delta \vdash e_1 : P \vee Q \quad \Gamma \mid \Delta \vdash e_2 : P \Rightarrow R \quad \Gamma \mid \Delta \vdash e_3 : Q \Rightarrow R}{\Gamma \mid \Delta \vdash \textbf{constructive-dilemma}\ e_1\ e_2\ e_3 : R}\ \textsc{Or-Elim}$$

Biconditional

$$\frac{\Gamma \mid \Delta \vdash e_1 : P \Rightarrow Q \quad \Gamma \mid \Delta \vdash e_2 : Q \Rightarrow P}{\Gamma \mid \Delta \vdash \textbf{equivalence}\ e_1\ e_2 : P \Leftrightarrow Q}\ \textsc{Iff-Intro}$$

$$\frac{\Gamma \mid \Delta \vdash e : P \Leftrightarrow Q}{\Gamma \mid \Delta \vdash \textbf{left-iff}\ e : P \Rightarrow Q}\ \textsc{Iff-Elim-L}$$

$$\frac{\Gamma \mid \Delta \vdash e : P \Leftrightarrow Q}{\Gamma \mid \Delta \vdash \textbf{right-iff}\ e : Q \Rightarrow P}\ \textsc{Iff-Elim-R}$$

Negation

$$\frac{\Gamma \mid \Delta, P \vdash e : \bot}{\Gamma \mid \Delta \vdash \textbf{suppose-absurd}\ P\ \textbf{in}\ e : \neg P}\ \textsc{Not-Intro}$$

$$\frac{\Gamma \mid \Delta \vdash e_1 : \neg P \quad \Gamma \mid \Delta \vdash e_2 : P}{\Gamma \mid \Delta \vdash \textbf{absurd}\ e_1\ e_2 : \bot}\ \textsc{Not-Elim}$$

Classical logic

$$\frac{\Gamma \mid \Delta \vdash e : \neg\neg P}{\Gamma \mid \Delta \vdash \textbf{double-negation}\ e : P}\ \textsc{Classic}$$

Proof composition (or let binding, really)

$$\frac{\Gamma \mid \Delta \vdash e_1 : P \quad \Gamma \mid \Delta, P \vdash e_2 : Q}{\Gamma \mid \Delta \vdash e_1; e_2 : Q}\textsc{Cut}$$

Universal quantifier

$$\frac{\Gamma, y : A \mid \Delta \vdash e : P\,[x := y]}{\Gamma \mid \Delta \vdash \textbf{pick-any}\ y\ \textbf{in}\ e : \forall x : A.\, P}\ \textsc{Forall-Intro}$$

$$\frac{\Gamma \mid \Delta \vdash e : \forall x : A.\, P \quad \Gamma \vdash t : A}{\Gamma \mid \Delta \vdash \textbf{specialize}\ e\ \textbf{with}\ t : P\,[x := t]}\ \textsc{Forall-Elim}$$

Existential quantifier

$$\frac{\Gamma \vdash t : A \quad \Gamma \mid \Delta \vdash e : P\,[x := t]}{\Gamma \mid \Delta \vdash \textbf{exists } t \textbf{ such that } e : \exists x : A.\, P}\textsc{Exists-Intro}$$

$$\frac{\Gamma \vdash R\ \texttt{prop} \quad \Gamma \mid \Delta \vdash e_1 : \exists x : A.\, P \quad \Gamma, y : A \mid \Delta, P\,[x := y] \vdash e_2 : R}{\Gamma \mid \Delta \vdash \textbf{pick-witness } y \textbf{ for } e_1 \textbf{ in } e_2 : R}\textsc{Exis}$$

Reasoning by cases on terms (for sums)

$$\frac{\Gamma, x : A \vdash P\ \mathtt{prop} \quad \Gamma \vdash t : A + B \qquad \begin{array}{c} \Gamma, a : A \mid \Delta \vdash e_1 : P\,[x := \mathtt{inl}\ a] \\ \Gamma, b : B \mid \Delta \vdash e_2 : P\,[x := \mathtt{inr}\ b] \end{array}}{\Gamma \mid \Delta \vdash \mathbf{case}\ t\ \mathbf{of}\ (\mathtt{inl}\ a \to e_1, \mathtt{inr}\ b \to e_2) : P\,[x := t]}$$

Equality

$$\frac{\Gamma \vdash \Delta \ \texttt{valid} \quad \Gamma \vdash t : A}{\Gamma \mid \Delta \vdash \textbf{refl}\ t : t =_A t}\ \textsc{Eq-Intro}$$

$$\frac{\Gamma \mid \Delta \vdash e : t_1 =_A t_2 \quad \Gamma, x : A \vdash P\ \texttt{prop} \quad \Gamma \mid \Delta \vdash e' : P\,[x := t_1]}{\Gamma \mid \Delta \vdash \textbf{rewrite}\ e\ \textbf{in}\ e' : P\,[x := t_2]}\ \textsc{Eq-Elim}$$

Equality of functions

$$\frac{\Gamma \mid \Delta \vdash e : \forall x : A.\, f\ x =_B g\ x}{\Gamma \mid \Delta \vdash \textbf{funext}\ e : f =_{A \to B} g}\ \textsc{Funext}$$

Conversion rule

$$\frac{\Gamma, x : A \vdash P \ \texttt{prop} \quad \Gamma \mid \Delta \vdash e : P\,[x := t_1] \quad \Gamma \vdash t_1 \equiv t_2 : A}{\Gamma \mid \Delta \vdash e : P\,[x := t_2]}\textsc{Conv}$$