The proof assistant PAF!

Séverine Maingaud

Université Paris Diderot
Laboratoire Preuves, Programmes et Systèmes

19th February 2009

KEY SYMPOSIUM’09
Who and What?

PhD student at the PPS Lab under the supervision of Alexandre Miquel and Vincent Balat.

**Topic and aim:**
designing logic tools and developing a proof assistant (PA) to certify ML programs with imperative features (references)

**Starting point:**
PAF!: PA, developed by S.Baro and P.Manoury, devoted to the functional fragment of ML.
Outline

Functional ML, core language
  ML terms
  Typing

PAF!
  Specification and verification
  Examples
  Consistency

Adding imperative features
  ML with reference
  Aim and constraints
Functional fragment of ML
A polymorphic static-typed language.

Everything is an expression!
Terms include constructions to handle booleans, unary integers, lists and functions:

Syntax

\[
t \ ::= \ x \mid () \\
  \mid \text{fun } x \to t \mid t \ t \\
  \mid \text{let } x = t \text{ in } t \mid \text{let rec } x = t \text{ in } u \\
  \mid \text{true} \mid \text{false} \mid t = t \mid \text{if } t \text{ then } t \text{ else } t \\
  \mid 0 \mid S \ t \mid \text{(case}_n\ t \text{ of } 0 \to t \mid Sx \to t) \\
  \mid [ ] \mid t :: t \mid \text{(case}_l\ t \text{ of } [ ] \to t \mid x :: y \to t)
\]
Examples

```ml
let rec fact n = case n of
  0 → 1
| S m → n * (fact m)

let rec loop n = 42 + (loop (n+1))
```

Evaluation ⟷

- $\beta$-reduction for `fun` and `let`
- $\delta$-rules for each primitive (if, case ... )
Static typing of ML

Intuition

$t : \text{nat}$ If $t$ actually returns a value, then this is a natural number.

A term $t$ is of type $\tau$ if there is a typing derivation of the judgement $t : \tau$:

- fact : nat → nat
- loop : nat → nat

- Decidable and automatically inferred
- Prevents a lot of errors at run-time
- No warranty w.r.t. termination
Strong typing of PAF!

Intuition

$t$ of type $\text{nat} \rightarrow \text{nat}$:
If applied to a natural integer, it actually returns a natural integer.

Strong typing rejects non-terminating programs.

Strong-typing predicate of PAF! ($t \Downarrow \tau$)

- Proven by the user
- Semi-decidable (user directed)
- Warranty: type of the output and termination
PAF! in a few words

Usage: verify ML programs already written.

Proof engine: based on a second order logic à la AF2.

First order elements are terms of ML.

Characteristics:
- An injection from booleans to formulas
- Strong-typing predicate
- Symbolic evaluation
Specification formulas

Grammar:

\[ A ::= X(t_1, \ldots, t_n) \]
\[ \quad | \ \neg A \ | \ A \rightarrow A \ | \ A \land A \ | \ A \lor A \]
\[ \quad | \ \forall x \downarrow \tau. \ A \ | \ \exists x \downarrow \tau. \ A \ | \ \forall \alpha. \ A \ | \ \forall X^n. \ A \]
\[ \quad | \ I(t) \quad \text{injection of booleans} \]
\[ \quad | \ t \downarrow \tau \quad t \text{ terminates returning a value of type } \tau \]

Remark:

- \[ \forall x \downarrow \tau. \ A \equiv \forall x. (x \downarrow \tau \rightarrow A) \]
- The formula \( I(\text{true}) \) will be denoted by \( \top \)
  and \( I(\text{false}) \) by \( \bot \).
Deductive system (1/3)

- Introduction and elimination (connectives et quantifiers):
  \[ \Gamma \vdash A \quad \Gamma \vdash B \quad \Gamma \vdash A \land B \quad \Gamma \vdash A \quad \Gamma \vdash B \]

- Induction on data types

- Symbolic evaluation:
  \[ \Gamma \vdash A(t) \quad t \rightsquigarrow t' \quad \Gamma \vdash A(t') \]
  \[ \Gamma \vdash A(t) \quad t' \rightsquigarrow t \quad \Gamma \vdash A(t') \]

▷ Undecidable
Deductive system (2/3)

- Strong typing rules

base values

$$\Gamma \vdash () \downarrow \text{unit}$$

$$\Gamma \vdash \text{true} \downarrow \text{bool}$$

$$\Gamma \vdash \text{false} \downarrow \text{bool}$$

$$\Gamma \vdash 0 \downarrow \text{nat}$$

$$\Gamma \vdash \text{S}t \downarrow \text{nat}$$

$$\Gamma \vdash \text{t} \downarrow \tau$$

$$\Gamma \vdash \text{l} \downarrow \tau \text{ list}$$

$$\Gamma \vdash \text{t} :: \text{l} \downarrow \tau \text{ list}$$

functions and applications

$$\Gamma, x \downarrow \tau \vdash t \downarrow \tau'$$

$$\Gamma \vdash \text{fun } x \rightarrow t \downarrow \tau \rightarrow \tau'$$

$$\Gamma \vdash (t)u \downarrow \tau'$$

$$\Gamma \vdash \text{t} \downarrow \tau \rightarrow \tau'$$

$$\Gamma \vdash u \downarrow \tau$$

$$\Gamma \vdash \text{fun } x \rightarrow t \downarrow \tau \rightarrow \tau'$$

$$\Gamma \vdash u \downarrow \tau$$
### Deductive system (3/3)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma \vdash A \land B$</td>
<td>$\Gamma \vdash A$ and $\Gamma \vdash B$</td>
<td>$\Gamma \vdash A \land B$</td>
</tr>
<tr>
<td>$\Gamma \vdash A \rightarrow B$</td>
<td>$\Gamma \vdash A$</td>
<td>$\Gamma \vdash B$</td>
</tr>
<tr>
<td>$\Gamma \vdash \forall \alpha. \tau$</td>
<td>$\Gamma \vdash t \downarrow \tau$, $\Gamma \vdash \tau[\alpha \mapsto \tau']$</td>
<td>$\Gamma \vdash t \downarrow \forall \alpha. \tau$</td>
</tr>
<tr>
<td>$\Gamma \vdash 0 \downarrow \text{nat}$</td>
<td></td>
<td>$\Gamma \vdash [\ ] \downarrow \forall \alpha. (\alpha \ 	ext{list})$</td>
</tr>
<tr>
<td>$\Gamma \vdash t \downarrow \text{nat}$</td>
<td></td>
<td>$\Gamma \vdash t \downarrow \tau$, $\Gamma \vdash t \downarrow \tau \ 	ext{list}$</td>
</tr>
<tr>
<td>$\Gamma \vdash S t \downarrow \text{nat}$</td>
<td></td>
<td>$\Gamma \vdash t :: [ \ ] \downarrow \tau \ 	ext{list}$</td>
</tr>
<tr>
<td>$\Gamma \vdash (t) \downarrow \text{unit}$</td>
<td></td>
<td>$\Gamma \vdash \text{true} \downarrow \text{bool}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Gamma \vdash \text{false} \downarrow \text{bool}$</td>
</tr>
<tr>
<td>$\Gamma \vdash \exists x \downarrow \tau. A$</td>
<td>$\Gamma \vdash A$ and $\Gamma \vdash \exists x \downarrow \tau. A$</td>
<td>$\Gamma \vdash \forall \alpha. \tau$</td>
</tr>
<tr>
<td>$\Gamma \vdash A[x \mapsto t]$</td>
<td>$\Gamma \vdash t \downarrow \tau$</td>
<td>$\Gamma \vdash A[x \mapsto \text{true}]$ and $\Gamma \vdash A[x \mapsto \text{false}]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Gamma \vdash \forall x \downarrow \text{bool.} A$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Gamma \vdash A[x \mapsto 0]$, $\Gamma \vdash n \downarrow \text{nat}$, $\Gamma \vdash A[x \mapsto n] \rightarrow A[x \mapsto S n]$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Gamma \vdash \forall x \downarrow \text{nat.} A$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Gamma \vdash A[x \mapsto []]$, $\Gamma \vdash t \downarrow \tau$, $\Gamma \vdash t \downarrow \tau$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Gamma \vdash t :: [ \ ] \downarrow \tau \ 	ext{list}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Gamma \vdash \text{true} \downarrow \text{list.} A$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Gamma \vdash \text{false} \downarrow \text{list.} A$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Gamma \vdash \text{false} \downarrow \text{list.} A$</td>
</tr>
</tbody>
</table>

---

**Functional ML, core language**
Examples

\begin{align*}
\text{let rec } \text{fact } n &= \text{case } n \text{ of } \\
& \quad 0 \rightarrow 1 \\
& \quad S \, m \rightarrow n \times (\text{fact } m)
\end{align*}

We show the totality of fact: \( \forall n \downarrow \text{nat}. \ \text{fact } n \downarrow \text{nat} \)

\[
\begin{array}{c}
\frac{\vdash 0 \downarrow \text{nat}}{\vdash 1 \downarrow \text{nat}} \quad \text{XNat} \\
\frac{\vdash \text{fact } 0 \downarrow \text{nat}}{\vdash \text{fact } 0 \downarrow \text{nat}} \quad \text{evG} \\
\frac{\Pi \vdash \text{fact } (S \, n) \downarrow (S \, n) \times (\text{fact } n)}{\Pi \vdash \text{fact } (S \, n) \downarrow (S \, n) \times (\text{fact } n) \downarrow \text{nat}} \quad \text{evG} \\
\frac{\vdash (S \, n) \downarrow \text{nat}, \text{fact } n \downarrow \text{nat} \vdash \text{fact } (S \, n) \downarrow \text{nat}}{\Pi \vdash \forall n \downarrow \text{nat}. \ \text{fact } n \downarrow \text{nat}} \quad \text{evG} \\
\frac{\vdash * \downarrow \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}}{\vdash * \downarrow \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}} \quad \text{THM} \\
\frac{\vdash n \downarrow \text{nat}}{\vdash S \, n \downarrow \text{nat}} \quad \text{XNat} \\
\frac{\Pi \vdash n \downarrow \text{nat}, \text{fact } n \downarrow \text{nat} \vdash (S \, n) \times (\text{fact } n) \downarrow \text{nat}}{\Pi \vdash n \downarrow \text{nat}, \text{fact } n \downarrow \text{nat} \vdash (S \, n) \times (\text{fact } n) \downarrow \text{nat}} \quad \text{XApp}
\end{array}
\]
Reasoning on non-terminating programs

\[
\text{let rec div n = } \\
\quad \text{if } (n \mod 2) = 0 \text{ then } n/2 \text{ else } \text{div } n
\]

div is well-defined over even numbers.

\[
\frac{n \downarrow \text{nat}, x \downarrow \text{nat}, I(n=2x) \vdash x \downarrow \text{nat}}{\text{ID}} \quad \frac{\text{div } 2x \sim x \downarrow x}{\text{EVALG}}
\]

\[
\frac{n \downarrow \text{nat}, x \downarrow \text{nat}, I(n=2x) \vdash \text{div } 2x \downarrow \text{nat}}{\text{RW}} \quad \frac{\exists x \downarrow \text{nat. } I(n=2x) \vdash \exists x \downarrow \text{nat. } I(n=2x) \vdash \text{div } n \downarrow \text{nat}}{\exists^1 i}
\]

\[
\frac{n \downarrow \text{nat}, \exists x \downarrow \text{nat. } I(n=2x) \vdash \text{div } n \downarrow \text{nat}}{\text{RW}} \quad \frac{\forall n \downarrow \text{nat. } (\exists x \downarrow \text{nat. } I(n=2x)) \rightarrow \text{div } n \downarrow \text{nat}}{\forall^1 i + \rightarrow i}
\]
For real

This is a light presentation.

Actually, it is possible to define inductive data types.

The second order is used to automatically generate an inductive rule for each new type.
Consistency

The consistency of the proof engine has been proven constructing a very simple model of realisability.

Interpretation

- terms are interpreted by closed terms
- types by sets of closed terms (*candidats de réductibilité*)
- formulas are interpreted by 0 or 1 (e.g. $[\top] = 1$ and $[\bot] = 0$)
Theorems

Correctness
If a formula is derivable in PAF!, then it is true in the model.

\[ \vdash A \implies \llbracket A \rrbracket = 1 \]

In particular, if \( \vdash (f \; n) \downarrow \text{nat} \), then \( f \; n \) actually returns an integer.

Consistency
No contradiction can be proven in PAF!

\( \not\vdash \bot \)
Outline

Functional ML, core language
- ML terms
- Typing

PAF!
- Specification and verification
- Examples
- Consistency

Adding imperative features
- ML with reference
- Aim and constraints
ML with references

Everything is still an expression!
Handle mutable objects: references, which can be manipulated only using the following primitives to create, modify and read them.

Syntax
\[
t ::= \ldots \\
    \text{ref } t \mid t := t \mid !t
\]

Examples
\[
\text{let count = } \\
\text{let r = ref 0 in} \\
\text{fun () } \rightarrow \text{ r := !r + 1 ; !r}
\]
Adapt PAF!

The eval-rule is no longer available
Thus, if count has never been called, then eval-rule allowed to prove $I(\text{count()} = 1)$.

Equality is no longer symmetrical!: $\not\vdash I(\text{count()} = \text{count()}$)

Constraints

- keep the approach of PAF! and prove programs already written
- do not use a monadic translation (toward functional fragment)
Work in progress

A natural way is **Dynamic Logic**

**Grammar formulas**

\[ A ::= \ldots \]

\[ | \quad [t \text{ as } x]A \]

**Kind of formula we want to prove**

\[ [\text{count()} \text{ as } x] \; I(\text{count()} = Sx) \]

**Kind of rules we have**

\[ \Gamma \vdash A\{^v/x\} \quad \Gamma \vdash [t \text{ as } y][r := y \text{ as } x]A \]

\[ \Gamma \vdash [v \text{ as } x]A \quad \Gamma \vdash [r := t \text{ as } x]A \]
Thanks a lot!