PML and Strong Normalisation

Christophe Raffalli

LAMA

TYPES 2008

PML: a new ML with proofs and specifications

- Constraint checking versus type inference or constraint solving.
- “unification” of Cartesian products.
- Extensible records and variant types
- Implicit subtyping
- Exceptions (they are important)
- Programs as types.
- Programs as proofs.

PML: a new ML with proofs and specifications

- Constraint checking versus type inference or constraint solving.
- “unification” of Cartesian products.
- Extensible records and variant types
- Implicit subtyping
- Exceptions (they are important)
- Programs as types.
- Programs as proofs.

Not today’s topic:

- The current state of the implementation.
- What is a proof in PML.
- Some examples.

PML: a new ML with proofs and specifications

- Constraint checking versus type inference or constraint solving.
- “unification” of Cartesian products.
- Extensible records and variant types
- Implicit subtyping
- Exceptions (they are important)
- Programs as types.
- Programs as proofs.

Today’s topic:

- The type-checking in the pure lambda-calculus case.
- Proof of strong normalisation.
- GC on types.
- Generalisation.

Outline

PML architecture

Type-checking

A black box ensuring that

- Programs can not fail.
- Programs can not loop without `let rec`.

Proof-checking

Another black box that

- Detects inaccessible position
- Checks proofs

An example for proof checking

```
val rec eq_nat:(nat => nat => bool) x y =
  match x, y with
  | Z[], Z[] -> True[]
  | S[x], S[y] -> eq_nat x y
  | _ -> False[]

val rec eq_nat_symmetric x:nat y:nat
    |- bimply (eq_nat x y) (eq_nat y x)
  proof
    match x, y with
    | Z[], Z[] -> 8<
    | Z[], S[] -> 8<
    | S[], Z[] -> 8<
    | S[x'], S[y'] ->
       let hr = eq_nat_symmetric x' y' in 8<
```

Constraints from pure lambda-terms

- Annotate each subterm with a distinct type variable.
- Except for variables, annotated uniformly

$$\delta = (\lambda x^{\alpha}(x^{\alpha}x^{\alpha})^{\beta})^{\gamma}$$

- Extract constraints:

$$(u^{\alpha}v^{\beta})^{\gamma} \Rightarrow \alpha \subset \beta \to \gamma \qquad (\lambda x^{\beta}.t^{\gamma})^{\alpha} \Rightarrow \beta \to \gamma \subset \alpha$$

For δ the gives:

$$\alpha \subset \alpha \to \beta, \alpha \to \beta \subset \gamma$$

Constraint saturation

Saturation rule:

$$\frac{\beta \to \gamma \subset \alpha_1 \subset \ldots \subset \alpha_n \subset \beta' \to \gamma'}{\beta' \subset \beta \qquad \gamma \subset \gamma'}$$

Example: $(\delta\, \delta)$:

$$((\lambda x^{\alpha}(x^{\alpha} x^{\alpha})^{\beta})^{\gamma} \;\; (\lambda x^{\alpha'}(x^{\alpha'} x^{\alpha'})^{\beta'})^{\gamma'})^{\rho}$$

$$\begin{array}{l}
\alpha \subset \alpha \to \beta, \alpha \to \beta \subset \gamma \\
\alpha' \subset \alpha' \to \beta', \alpha' \to \beta' \subset \gamma' \\
\gamma \subset \gamma' \to \rho \\
\beta \subset \rho, \gamma' \subset \alpha \\
\beta' \subset \beta, \alpha \subset \alpha'
\end{array}$$

Checking that constraints are well founded (1)

Example: $(\delta\,\delta)$:

$$((\lambda x^{\alpha}(x^{\alpha}\,x^{\alpha})^{\beta})^{\gamma}\;\;(\lambda x^{\alpha'}(x^{\alpha'}\,x^{\alpha'})^{\beta'})^{\gamma'})^{\rho}$$

$$\begin{array}{l}\alpha \subset \alpha \to \beta, \alpha \to \beta \subset \gamma\\ \alpha' \subset \alpha' \to \beta', \alpha' \to \beta' \subset \gamma'\\ \gamma \subset \gamma' \to \rho\\ \beta \subset \rho, \gamma' \subset \alpha\\ \beta' \subset \beta, \alpha \subset \alpha'\end{array}$$

implies:

$$\alpha' \to \beta' \subset \gamma' \subset \alpha \subset \alpha'$$

This is bad !

Checking that constraints are well founded (1)

Introduce new constraints:

- $\alpha < \beta$: α "defined" before β
- $\alpha \leq \beta$: α "defined" before or at the same time than β

- $\alpha \subset \beta \Rightarrow \alpha \leq \beta$
- $\alpha \subset \beta \rightarrow \gamma \Rightarrow \emptyset$
- $\beta \rightarrow \gamma \subset \alpha \Rightarrow \beta < \alpha, \gamma < \alpha$

$$\alpha' \rightarrow \beta' \subset \gamma' \subset \alpha \subset \alpha' \text{ implies } \alpha' < \alpha'$$

$(\delta\, \delta)$ is not accepted by PML but δ is accepted.

Strong normalisation

Theorem

All accepted terms are strongly normalisable

Proof outline:

1. Order the type variables according to the $<$ constraints.
2. Associate "candidates" to the type variable in that order: If $T_1 \subset \beta, \ldots, T_n \subset \beta$ take $|\beta| = \bigcup_i T_i$
3. Other constraints are satisfied.
4. A subterm belongs to the interpretation of its annotation.

Constraint checking versus constraint solving

- Solving the constraints requires a syntax for types
- We have constraints like $\alpha_1 \subset \beta, \alpha_2 \subset \beta$
- Solution: $\beta = \alpha_1 \cup \alpha_2$
- We have constraints like $\beta \subset \alpha_1, \beta \subset \alpha_2$
- Solution: $\beta = \alpha_1 \cap \alpha_2$
- One adds union and intersection to types.
- Then, you may have constraints like

$$\alpha_1 \cap \alpha_2 \subset \beta_1 \cup \beta_2$$

- ???

PML's solution

Hide the constraints

Constraints can be used to generate nice error messages (similar to Joe Wells works)

Program as types

- Types in PML are translated to partial identity function.
- Typecast is just composition.

Accessibility

All constraints are not useful up to the end:

- $A^{\subset}(t^{\rho}, \alpha)$ (α is $\subset$-accessible) means that constraints outside t^{ρ} can create a new relation $\alpha \subset \beta$.

$$\frac{}{A^{\subset}(t^{\rho}, \rho)} \qquad \frac{x^{\alpha} \text{ free in}}{A^{=}(t^{\rho}, \alpha)}$$

$$\frac{A^{\subset}(t^{\rho}, \alpha) \quad \beta \subset \alpha}{A^{\subset}(t^{\rho}, \beta)} \qquad \frac{A^{\subset}(t^{\rho}, \alpha) \quad \beta \to \gamma \subset \alpha}{A^{\supset}(t^{\rho}, \beta) \quad A^{\subset}(t^{\rho}, \gamma)}$$

$\ldots$ $\ldots$

- PML:
  Incremental check + remove inaccessible type variable.

Generalization

The rule

$$\frac{\cap\rho \text{ s.t. } [\forall\alpha_1, \ldots, \alpha_n.(\alpha_i \to \beta \subset \rho \wedge \ldots)] \subset \varphi_1 \cdots \subset \varphi_n \subset \beta \to \gamma}{\rho' \subset \beta \to \gamma \quad \alpha'_i \to \beta \subset \rho' \quad \ldots \quad \rho' < \varphi_1}$$

- Where to generalize ? Everywhere !
- What to generalize ? All but the type of variables ?
- Inefficient and even non terminating !

Generalization

The rule

$$\frac{\cap\rho \text{ s.t. } [\forall\alpha_1, \ldots, \alpha_n.(\alpha_i \to \beta \subset \rho \wedge \ldots)] \subset \varphi_1 \cdots \subset \varphi_n \subset \beta \to \gamma}{\rho' \subset \beta \to \gamma \quad \alpha'_i \to \beta \subset \rho' \quad \ldots \quad \rho' < \varphi_1}$$

- Where to generalize ? Everywhere !
- What to generalize ? All but the type of variables ?
- Inefficient and even non terminating !

Generalization

The rule

$$\frac{\cap\rho \text{ s.t. } [\forall\alpha_1,\ldots,\alpha_n.(\alpha_i \to \beta \subset \rho \wedge \ldots)] \subset \varphi_1 \cdots \subset \varphi_n \subset \beta \to \gamma}{\rho' \subset \beta \to \gamma \quad \alpha'_i \to \beta \subset \rho' \quad \ldots \quad \rho' < \varphi_1}$$

- Where to generalize ? Everywhere !
- What to generalize ? All but the type of variables ?
- Inefficient and even non terminating !

Generalization

The rule

$$\frac{\cap\rho \text{ s.t. } [\forall\alpha_1, \ldots, \alpha_n.(\alpha_i \to \beta \subset \rho \wedge \ldots)] \subset \varphi_1 \cdots \subset \varphi_n \subset \beta \to \gamma}{\rho' \subset \beta \to \gamma \quad \alpha'_i \to \beta \subset \rho' \quad \ldots \quad \rho' < \varphi_1}$$

- Where to generalize ? Everywhere !
- What to generalize ? All but the type of variables ?
- Inefficient and even non terminating !

Generalization

The rule

$$\frac{\cap\rho \text{ s.t. } [\forall\alpha_1,\ldots,\alpha_n.(\alpha_i \to \beta \subset \rho \wedge \ldots)] \subset \varphi_1 \cdots \subset \varphi_n \subset \beta \to \gamma}{\rho' \subset \beta \to \gamma \quad \alpha'_i \to \beta \subset \rho' \quad \ldots \quad \rho' < \varphi_1}$$

- Where to generalize ? Everywhere !
- What to generalize ? All but the type of variables ?
- Inefficient and even non terminating !

Minimal Generalization

It suffices to generalize

- Variables that are $=$-accessible from the root.
- Variables that are "in between" the root and other generalized variables.

Idea: if α is $=$-accessible from the root of t, two distinct "uses" of t may create $\beta \subset \alpha$ and $\alpha \subset \beta'$. But β and β' should not be related.

Theorems

Terminating algorithm and strong normalization.

Example: $(\lambda x^{\alpha}(x^{\alpha}\ \ (y^{\beta}\ y^{\beta})^{\gamma})^{\varphi})^{\rho}$

$$\alpha \subset \gamma \to \varphi, \beta \subset \beta \to \gamma, \alpha \to \varphi \subset \rho$$

- ρ is $\subset$-accessible from the root.
- φ is $\subset$-accessible from the root.
- α is $\supset$-accessible from the root.
- γ is $\subset$-accessible from the root.
- φ is $\supset$-accessible from the root.
- φ is $=$-accessible from the root.
- α is between φ and ρ.

Generalized type:

$$\cap\rho[\forall\alpha, \varphi.(\alpha \subset \gamma \to \varphi \wedge \alpha \to \varphi \subset \rho)]$$

Conclusion/Further work

- Core language is now stable, proof checking working
- Termination check for `let rec`
- Effect, IO
- Arrays
- Equality support
- Extensible grammars
- Reflection
- ...