

---

Universität des Saarlandes  
Programming Systems Lab

# Polymorphic Lambda Calculus with Dynamic Types

Bachelor Thesis  
Task Formulation

Matthias Berg

Tutors: Guido Tack, Gert Smolka

# Motivation

---

Open Programming System:

- Not all components available at compile time
- Components linked dynamically
- Dynamic type checking needed

Possible construct: *type case*

- Notation:  $\text{case } t_1 : T_1 \text{ of } x : T_2 \Rightarrow t_2 \text{ else } t_3$
- Provides branching dependent on type  $T_1$  of subterm  $t_1$
- Evaluates to  $t_2[x := t_1]$  iff  $T_1 = T_2$  dynamically otherwise to  $t_3$

# Type Case

---

Example:

$$\begin{aligned} \text{rep} = \lambda X. \lambda x : X. & \text{case } x : X \text{ of } x' : \text{bool} \Rightarrow \text{"bool"} \\ & \text{else case } x : X \text{ of } x' : \text{int} \Rightarrow \text{"int"} \\ & \text{else "unknown"} \end{aligned}$$

Given a type and a term of this type  $\text{rep}$  returns a string representation of the type.

Problem: Type case destroys parametricity of type abstraction.

$\text{abstype } \text{Number} = \text{int}$   
 $\text{implementation: } [\dots]$

$\text{rep } \text{Number } n \longrightarrow^* \text{"int"}$

# Dynamic Type Name Generation

---

Solution: Generate new type names dynamically.

- Notation:  $\text{new } X = T \text{ in } t$
- Type name  $X$  can be used in  $t$  in place of  $T$ .
- Use global state for dynamically generated type names instead of coercions (Rossberg's approach).

Example:

```
new  $X = \text{int}$  in  
abstype  $Number = X$   
implementation: [...]
```

```
 $\text{rep } Number\ n \longrightarrow^* \text{''unknown''}$ 
```

# Laziness

---

Call-by-need extension for simply typed  $\lambda$ -calculus:

- Notation:  $\text{lazy } x = t \text{ in } t'$
- Variable  $x$  can be used in  $t'$  in place of  $t$  (similar to `let` )
- Evaluate  $t$  as late as possible.
- Use global state for modelling relationship between  $x$  and  $t$ .

Similar construct can be used to express lazy linking:

$$\text{lazy } \langle X, x \rangle = \langle T, t \rangle \text{ in } t'$$

# Task Formulation

---

Develop a calculus,  $\lambda_F^N$ , an extension of system F with constructs for type case and a binder for new type names.

- Use global state instead of coercions.
- Use evaluation contexts.
- Prove the unique type, the progress and the preservation property.

Extend the simply typed  $\lambda$ -calculus with call-by-need evaluation and give proofs for the safety properties.

Use this construct to model lazy linking in system F or in  $\lambda_F^N$

Give a model implementation.

# References

---

- M. Abadi, L. Cardelli, B. Pierce, and D. Rémy. Dynamic typing in polymorphic languages. *Journal of Functional Programming*, 5(1):111-130, Jan. 1995.
- Catherine Dubois, François Rouaix, Pierre Weis. Extensional polymorphism. *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, p.118-129, January 23-25, 1995, San Francisco, California, United States.
- Robert Harper, Greg Morrisett. Compiling polymorphism using intensional type analysis. *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, p.130-141, January 23-25, 1995, San Francisco, California, United States.
- John C. Mitchell , Gordon D. Plotkin, Abstract types have existential type, *ACM Transactions on Programming Languages and Systems (TOPLAS)*, v.10 n.3, p.470-502, July 1988
- Joachim Niehren, Jan Schwinghammer, Gert Smolka. A Concurrent Lambda Calculus with Futures. Technical Report, Programming Systems Lab, 2004.
- Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, Feb, 2002.

# References

---

- Andreas Rossberg. Generativity and dynamic opacity for abstract types. *Proceedings of the 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, pages 241-252, 2003.
- Andreas Rossberg. What are Components. Slides, March 2004.
- Gert Smolka. Personal communication. <http://www.ps.uni-sb.de/~smolka/>, March 2004.
- Eijiro Sumii and Benjamin C. Pierce, A bisimulation for dynamic sealing. *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of programming languages*. ACM Press, January 2004.