Universität des Saarlandes
Programming Systems Lab

Polymorphic Lambda Calculus with Dynamic Types

Bachelor's Thesis
Final Presentation

Matthias Berg

Advisors: 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: $\texttt{case}\ t_1 : T_1\ \texttt{of}\ x : T_2 \Rightarrow t_2\ \texttt{else}\ t_3$
- 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:

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

Given a type and a term of this type rep returns a string representation of the type.

Problem: Type case destroys parametricity of type abstraction.

$\texttt{abstype}\ Number = int$
$\texttt{implementation:}\ [\ldots]$

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

Dynamic Type Name Generation

Solution [Rossberg]: Generate new type names dynamically.

- Notation: `new` $X = T$ `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` $X = int$ `in`
`abstype` $Number = X$
`implementation:` $[\ldots]$

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

λ_F^N: Syntax

λ_F^N = System F + `case` + `new`

$x \in \mathit{Var}$			Variables
$X \in \mathit{TVar}$			Type Variables
$T \in \mathit{Typ}$	::=	$X \mid T \rightarrow T \mid \forall X.T$	Types
$v \in \mathit{Val}$	::=	$\lambda x : T.t \mid \lambda X.t \mid x$	Values
$t \in \mathit{Ter}$	::=	$x \mid \lambda x : T.t \mid t\ t \mid \lambda X.t \mid t\ T$	Terms
		$\mid$ `case` $v : T$ `of` $x : T \Rightarrow t$ `else` t	
		$\mid$ `new` $X = T$ `in` t	
$\kappa \in \mathit{Kind}$	::=	$*$	Kinds
$\Gamma \in \mathit{Env}$	::=	$\emptyset \mid \Gamma, x : T \mid \Gamma, X : \kappa$	Environments
$N \in \mathit{State}$	::=	$\phi \mid N, X = T$	States

λ_F^N: Reduction

$$E ::= \circ \mid E\ t \mid v\ E \mid E\ T$$

$$\begin{array}{l} \quad E((\lambda x : T.t)\ v) \mid N \\ \longrightarrow E(t[x := v]) \mid N \end{array}$$

$$\begin{array}{l} \quad E((\lambda X.t)\ T) \mid N \\ \longrightarrow E(t[X := T]) \mid N \end{array}$$

$$\begin{array}{l} \quad E(\texttt{case}\ \ v : T\ \ \texttt{of}\ \ x : T' \Rightarrow t\ \ \texttt{else}\ \ t') \mid N \quad \text{(if } T = T'\text{)} \\ \longrightarrow E(t[x := v]) \mid N \end{array}$$

$$\begin{array}{l} \quad E(\texttt{case}\ \ v : T\ \ \texttt{of}\ \ x : T' \Rightarrow t\ \ \texttt{else}\ \ t') \mid N \quad \text{(if } T \neq T'\text{)} \\ \longrightarrow E(t') \mid N \end{array}$$

$$\begin{array}{l} \quad E(\texttt{new}\ \ X = T\ \ \texttt{in}\ \ t) \mid N \quad (X \text{ fresh}) \\ \longrightarrow E(t) \mid N, X = T \end{array}$$

λ_F^N: Typing

Terms:

$$\frac{\Gamma \vdash v : T \qquad \Gamma, x : T' \vdash t : T'' \qquad \Gamma \vdash t' : T''}{\Gamma \vdash \texttt{case}\ v : T\ \texttt{of}\ x : T' \Rightarrow t\ \texttt{else}\ t' : T''}$$

$$\frac{\Gamma \vdash T : \kappa \quad \Gamma \vdash t[X := T] : T'}{\Gamma \vdash \texttt{new}\ X = T\ \texttt{in}\ t : T'}$$

Configurations:
Substitute all type names with their corresponding type.

$$\frac{\Gamma \vdash N \quad \Gamma \vdash Nt : T}{\Gamma \vdash t|N : T}$$

λ_F^N: Properties

Uniqueness:

$$\Gamma \vdash t|N : T \;\wedge\; \Gamma \vdash t|N : T' \quad \Longrightarrow \quad T = T'$$

Progress:

$$\vdash t|N : T \quad \Longrightarrow \quad t \in \mathit{Val} \;\vee\; \exists t', N' : t|N \longrightarrow t'|N'$$

Preservation:

$$\Gamma \vdash t|N : T \;\wedge\; t|N \longrightarrow t'|N' \quad \Longrightarrow \quad \Gamma \vdash t'|N' : T$$

Lemmas:

$$\Gamma \vdash E(t)|N : T \quad \Longrightarrow \quad \exists T' : \Gamma \vdash Nt : T'$$

$$\Gamma \vdash E(t)|N : T \wedge \Gamma \vdash Nt : T' \wedge \Gamma \vdash Ns : T' \Longrightarrow \Gamma \vdash E(s)|N : T$$

Laziness

Call-by-need extension for simply typed λ-calculus [Alice, Schwinghammer]:

- Notation: $\texttt{lazy}\ x\ \texttt{=}\ t\ \texttt{in}\ t'$
- Variable x can be used in t' in place of t (similar to `let`).
- Evaluate t as late as possible, i.e. when x occurs as left-hand side of an application.
- Use global state $\mu \in Var \xrightharpoonup{fin} Ter$ for modelling relationship between x and t.
- Use a stack S to memorise which terms need to be evaluated
- Define reduction over pairs of states and stacks: $\mu | S$

λ_s^L: Syntax

λ_s^L = λ_s + `lazy`

$x \in \mathit{Var}$			Variables
$X \in \mathit{TVar}$			Type Variables
$T \in \mathit{Typ}$	::=	$X \mid T \to T$	Types
$t \in \mathit{Ter}$	::=	$x \mid \lambda x : T.t \mid t\ t$	Terms
		$\mid$ `lazy` x = t `in` t	
$v \in \mathit{Val}$	::=	$\lambda x : T.t \mid x$	Values
$S \in \mathit{Stack}$	::=	$\phi \mid S, x$	Stacks
$\mu \in \mathit{State}$	=	$\mathit{Var} \overset{\mathit{fin}}{\rightharpoonup} \mathit{Ter}$	States
$\Gamma \in \mathit{Env}$	=	$\mathit{Var} \overset{\mathit{fin}}{\rightharpoonup} \mathit{Typ}$	Environments

λ_s^L: Reduction

$E ::= \circ \mid E\ t \mid v\ E$

$$\begin{array}{ll} & \mu, x = E((\lambda x' : T.t)\ v) \mid S, x \\ \longrightarrow & \mu, x = E(t[x' := v]) \mid S, x \end{array}$$

$$\begin{array}{ll} & \mu, x = E(\texttt{lazy}\ x_1\ \texttt{=}\ t_1\ \texttt{in}\ t_2) \mid S, x \qquad (x_1 \text{ fresh}) \\ \longrightarrow & \mu, x_1 = t_1, x = E(t_2) \mid S, x \end{array}$$

$$\begin{array}{ll} & \mu, x = E(x_1\ v) \mid S, x \qquad (\text{if } x_1 \in dom(\mu)) \\ \longrightarrow & \mu, x = E(x_1\ v) \mid S, x, x_1 \end{array}$$

$$\begin{array}{ll} & \mu, x = v \mid S, x \qquad (\text{if } S \neq \phi) \\ \longrightarrow & \mu[x := v] \mid S \end{array}$$

To evaluate some term t, start with configuration $\{x = t\}|\phi, x$

λ_s^L: Typing

Example: $x_1 = E(x_2) \in \mu$ and $x_2 = E(x_1) \in \mu$

- Types of x_1 and x_2 depend on each other
- Type inference not possible
- Typing rules must forbid such situations
- Condition: Dependencies must be acyclic
- Stack must respect dependencies

$$dep_\mu = \{(x_1, x_2) \mid \{x_1, x_2\} \subseteq dom(\mu) \ \wedge \ x_2 \in FV(\mu\, x_1)\}$$

$$\frac{\begin{array}{c} dom(\Gamma) \cap dom(\mu) = \emptyset \quad \mu \vdash S : x_0 \quad dep_\mu acyclic \\ \exists \Gamma' \supseteq \Gamma : dom(\Gamma') = dom(\Gamma) \cup dom(\mu) \wedge \\ \Gamma' x_0 = T \ \wedge \ \forall x = t \in \mu : \Gamma' \vdash t : \Gamma' x \end{array}}{\Gamma \vdash \mu \mid S : T}$$

λ_s^L: Properties

Uniqueness

$$\Gamma \vdash \mu | S : T \ \wedge \ \Gamma \vdash \mu | S : T' \quad \Longrightarrow \quad T = T'$$

Progress

$$\begin{aligned} \vdash \mu | S : T \quad \Longrightarrow \quad & (\exists \mu_1, x, v : \mu = (\mu_1, x = v) \ \wedge \ S = \phi, x) \\ & \vee \ \exists \mu', S' : \mu | S \longrightarrow \mu' | S' \end{aligned}$$

Preservation

$$\Gamma \vdash \mu | S : T \ \wedge \ \mu | S \longrightarrow \mu' | S' \quad \Longrightarrow \quad \Gamma \vdash \mu' | S' : T$$

Lazy Linking

Lazy linking can be expressed similarly:

- Notation: `lazy <`X, x`> = <`T, t`> in` t'
- Since abstract types consist of a type and a term, `lazy` introduces two binders.
- Analogically states map pairs of variables to terms.

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.
- Alice Team. *The Alice System*. Programming Systems Lab, Saarland University, `http://www.ps.uni-sb.de/alice/`, 2003
- 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.

References

- Benjamin C. Pierce. *Types and Programming Languages.* The MIT Press, Feb, 2002.
- 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. Programming Systems Lab, Saarland University. Slides, March 2004.
- Andreas Rossberg, Didier Le Botlan. Personal communication. Programming Systems Lab, Saarland University. March 2004.
- Gert Smolka. Personal communication. Programming Systems Lab, Saarland University. 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.