Type systems with first class polymorphisms using Attribute Grammars

Master thesis defense

Tamar Christina
T.Christina@students.uu.nl

University of Utrecht

April 21, 2011

Outline

Type inferencing

"Type inference refers to the ability to deduce automatically the type of an expression in a programming language."

Type inferencing

> *"Type inference refers to the ability to deduce automatically the type of an expression in a programming language."*

For example the *identity* function $\lambda x \rightarrow x : \alpha \rightarrow \alpha$

Hindley-Milner

- Damas-Milner

Hindley-Milner

- Damas-Milner
- Principle type

Hindley-Milner

- Damas-Milner
- Principle type
- Decidable inferencing

Higher-rank types

- Haskell '98 types are rank-1

Higher-rank types

- Haskell '98 types are rank-1
- $a \rightarrow \beta \rightarrow a$

Higher-rank types

- Haskell '98 types are rank-1
- $a \to \beta \to a$
- $\forall\ a\ \beta . a \to \beta \to a$

Higher-rank types

- Haskell '98 types are rank-1
- $a \rightarrow \beta \rightarrow a$
- $\forall\ a\ \beta . a \rightarrow \beta \rightarrow a$
- $\forall\ a . a \rightarrow (\forall\ \beta . \beta \rightarrow a)$

Higher-rank types

- Haskell '98 types are rank-1
- $a \to \beta \to a$
- $\forall\ a\ \beta . a \to \beta \to a$
- $\forall\ a . a \to (\forall\ \beta . \beta \to a)$
- $\forall\ \beta . (\forall\ a . a \to a) \to \beta \to \beta$

Higher-rank types

- Haskell '98 types are rank-1
- $a \rightarrow \beta \rightarrow a$
- $\forall\ a\ \beta . a \rightarrow \beta \rightarrow a$
- $\forall\ a . a \rightarrow (\forall\ \beta . \beta \rightarrow a)$
- $\forall\ \beta . (\forall\ a . a \rightarrow a) \rightarrow \beta \rightarrow \beta$

Refers to the the number of $\forall$s nested to the left of a $(\rightarrow)$

Higher-rank types

$$poly = \lambda f \rightarrow (f\ 1, f\ \text{'c'})$$

cannot be expressed without Higher-Rank types.

SystemF

Provides typing support for higher-rank functions

Terms

- Type abstraction ($\Lambda X.t$)
- Type application (t [T])

SystemF

Provides typing support for higher-rank functions

Terms

- Type abstraction ($\Lambda X.t$)
- Type application (t [T])

Types

- Type variables (X)
- Universal types ($\forall X.T$)

SystemF

Typing of the *id* function

- $id = \lambda x.x$

SystemF

Typing of the *id* function

- $id = \lambda x.x$
- $id = \Lambda X.\lambda x : X- > x : X$

SystemF

Typing of the *id* function

- $id = \lambda x.x$
- $id = \Lambda X.\lambda x : X- > x : X$
- Typing $id\ 3$

SystemF

Typing of the *id* function

- $id = \lambda x.x$
- $id = \Lambda X.\lambda x : X- > x : X$
- Typing $id\ 3$
- id [Int] = [X→Int]$(\lambda x : X.x)$

SystemF

- $poly = \lambda f \rightarrow (f\ 1, f\ \text{'c'})$ is now typeable

SystemF

- $poly = \lambda f \rightarrow (f\ 1, f\ \texttt{'c'})$ is now typeable
- Undecidable

SystemF

- $poly = \lambda f \rightarrow (f\ 1, f\ \text{'c'})$ is now typeable
- Undecidable
- Requires annotation

SystemF

- $poly = \lambda f \rightarrow (f\ 1, f\ \text{'c'})$ is now typeable
- Undecidable
- Requires annotation
- $poly = \lambda(f :: \forall\ a \rightarrow Int) \rightarrow (f\ 1, f\ \text{'c'})$

The problem

Type systems are specified using typing rules

The problem

Type systems are specified using typing rules

$$\text{Var:} \quad \frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$$

However..

The problem

Disconnect between typing rules and implementation

- Nondeterministic

The problem

Disconnect between typing rules and implementation

- Nondeterministic
- Implicit assumptions

The problem

Disconnect between typing rules and implementation

- Nondeterministic
- Implicit assumptions
- Implementation does not resemble typing rules

The problem

Disconnect between typing rules and implementation

- Nondeterministic
- Implicit assumptions
- Implementation does not resemble typing rules
- Complexity explodes with size of AST

The problem

Disconnect between typing rules and implementation

- Nondeterministic
- Implicit assumptions
- Implementation does not resemble typing rules
- Complexity explodes with size of AST
- A lot of it due to language used

Goal

Implement type system using attribute grammars

- Easier to understand

Goal

Implement type system using attribute grammars

- Easier to understand
- Easier to prove correct

Goal

Implement type system using attribute grammars

- Easier to understand
- Easier to prove correct
- Easier to document and scale

Contributions

- Implementation using attribute grammars

Contributions

- Implementation using attribute grammars
- Implementation & specification for the HML type system for EH

Outline

Context Free Grammars

- Can only describe syntax

Context Free Grammars

- Can only describe syntax
- Cannot specify any context-sensitive conditions
- $a^n b^n c^n$

Context Free Grammars

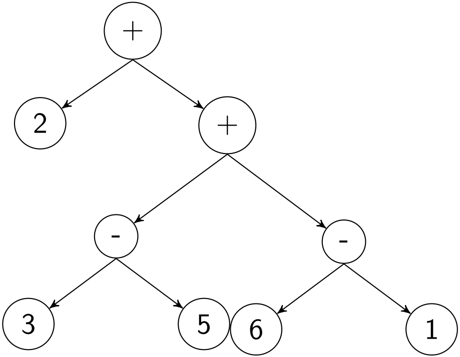
- Can only describe syntax
- Cannot specify any context-sensitive conditions
- $a^n b^n c^n$
- A way to define semantics/meaning

Context Free Grammars

$$
\begin{aligned}
\langle \mathit{Expr} \rangle &\rightarrow \langle \mathit{number} \rangle \mid \langle \mathit{Expr} \rangle \, \langle \mathit{operator} \rangle \, \langle \mathit{Expr} \rangle \\
\langle \mathit{number} \rangle &\rightarrow \langle \mathit{digit} \rangle \mid \langle \mathit{digit} \rangle \langle \mathit{number} \rangle \\
\langle \mathit{digit} \rangle &\rightarrow \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9} \\
\langle \mathit{relational\ operator} \rangle &\rightarrow - \mid +
\end{aligned}
$$

: BNF definition for Expressions

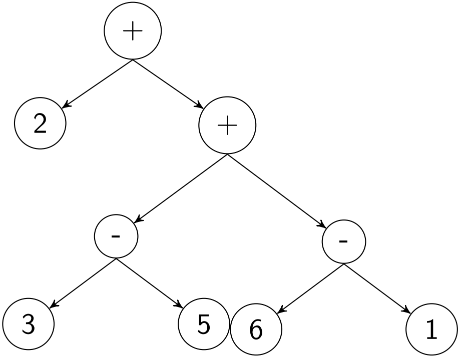
Parse Trees



: Parse tree example for "2 + (3 - 5) + (6 - 1)"

- AG's are additions to CFG

Parse Trees



: Parse tree example for "2 + (3 - 5) + (6 - 1)"

- AG's are additions to CFG
- Nodes are a production or a non-terminal

Outline

Outline

Outline

Outline

Outline