Sebastian Zimmeck

An Introduction to Type Inference

Professor Alfred V. Aho - COMS E6998-2
Advanced Topics in Programming
Languages and Compilers

November 29, 2011

Presentation Overview

1. Introduction
2. Lambda Calculus
3. Hindley-Milner Type Inference
4. Object-oriented Type Inference
5. Concluding Thoughts

1. Introduction

- Type Inference / Type Reconstruction: Determining the Type of an Expression in a Programming Language
- Duck Typing: Determining the Type of an Expression from the Way it is used
- For Type-checked Languages that do not require Declaration of Names
- Object-oriented Languages: E.g., C# (version 3.0), C++ (C++11)
- Functional Languages: E.g., ML, OCaml, Haskell

2. Lambda Calculus

- Lambda Calculus: Small Turing complete Programming Language [1]
- x = Variable
  EE' = Application
  $\lambda x.E$ = Lambda Abstraction
- Grammar: $E ::= x \mid EE' \mid \lambda x.E$
- λ = Binding Operator
  (In $\lambda x.xy$, Variable x is bound, Variable y is free)

2. Lambda Calculus

- Axioms of Lambda Calculus
  - α-Equivalence: Change of bound Variable Name, e.g., $\lambda x.E = \lambda y.E[y/x]$
  - β-Equivalence: Application of Function to Arguments, e.g., $(\lambda x.E)y = E[y/x]$
  - η-Equivalence: Elimination of Redundant Lambda Abstractions, e.g., if x is bound in E, $\lambda x.(Ex) = E$
- Substitution
  - Process of replacing all free Occurrences of a Variable by an Expression

3. Hindley-Milner Type Inference

- Hindley-Milner Type System [2,3,4] is based on Lambda Calculus
- Extension: Let-clause (only syntactic sugar)
  let x = E in E'
  (let x = E) in E'
  $(\lambda x.E')E$
- Hindley-Milner Type Inference =
  Type Inference Rules + Unification

3. Hindley-Milner Type Inference

Hindley-Milner Type Inference Rules [5]

1. For Variables

$$\frac{\tau \rightarrow \Gamma(x)}{\Gamma \vdash_{HM} x : \tau}$$

2. For Applications

$$\frac{\Gamma \vdash_{HM} E_1 : \tau_1 \rightarrow \tau_2 \qquad \Gamma \vdash_{HM} E_2 : \tau_1}{\Gamma \vdash_{HM} E_1 E_2 : \tau_2}$$

3. Hindley-Milner Type Inference

Hindley-Milner Type Inference Rules [5]

3. For Lambda Abstractions

$$\frac{\Gamma / x \cup \{x : \tau_1\} \vdash_{HM} E : \tau_2}{\Gamma \vdash_{HM} \lambda x \rightarrow E : \tau_1 \rightarrow \tau_2}$$

4. For Let-Clauses

$$\frac{\Gamma \vdash_{HM} E_1 : \tau_1 \qquad \Gamma / x \cup \{x : generalize(\Gamma, \tau_1)\} \vdash_{HM} E_2 : \tau_2}{\Gamma \vdash_{HM} let x = E_1 in E_2 : \tau_2}$$

3. Hindley-Milner Type Inference

- Unification can be used for Equalizing Type Expressions (or find that they cannot be equalized) [2]
- Examples of Unification [6]
  - *U(Knows(John,x), Knows(John, Jane)) = {x/Jane}*
  - *U(Knows(John,x), Knows(y, Mother(y))) = {y/John, x/Mother(John)}*
  - *U(Knows(John,x), Knows(x, Elizabeth)) = fail*

4. Object-oriented Type Inference

- Polymorphism: Polymorphic code Fragments can be executed with Arguments of different Types
- Basic Type Inference Algorithm: Early Type Inference Algorithm for Object-oriented Languages [7]
- Cartesian Product Algorithm: Improves Precision and Efficiency over Previous Algorithms and deals directly with Inheritance [8]

4. Object-oriented Type Inference

Basic Type Inference Algorithm [7]

- Builds a directed Graph with Nodes representing Type Variables and Edges representing Constraints
- Whenever an Edge is added to the Graph, Type Information is propagated to the next higher Program Level

4. Object-oriented Type Inference

<u>Basic Type Inference Algorithm</u> [7]

- Example

$$x := y$$

- As it holds that type(y) $\subseteq$ type (x), it is ensured that only valid Types are inferred

4. Object-oriented Type Inference

Cartesian Product Algorithm [8]

- Example

rcvr max: arg

R = type(rcvr)
A = type(arg)

R = {r_1, r_2, …, r_s}
A = {a_1, a_2, …, a_t}

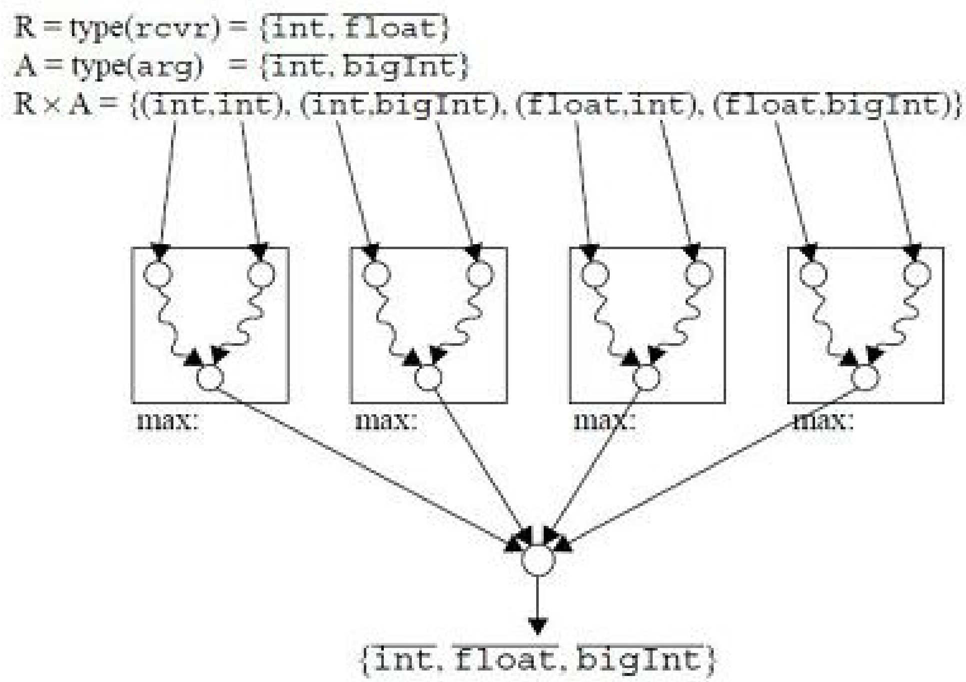
4. Object-oriented Type Inference

Cartesian Product Algorithm [8]

1. The Cartesian Product Algorithm computes the Cartesian Product of the Types
   R * A = {(r_1,a_1), …, (r_1,a_t), …,
   (r_s,a_1), …, (r_s,a_t)}
2. Then the Algorithm propagates each (r_i,a_j) into a separate max-template (if such already exists for a given pair, it is reused)
3. All Templates are unioned

4. Object-oriented Type Inference

Cartesian Product Algorithm [8]



5. Concluding Thoughts

- Lambda Calculus provides a fundamental Type Inference Environment, particularly, for Functional Languages
- Popularity of object-oriented Programming Languages gradually lead to new Type Inference Algorithms and Environments
- Type Inference is at the Core of Programming Language Design, making it desirable to fully work it out in the early Stages of Language Development
- Parallelism between Type Checking and Type Inference may provide further Insights

References

[1] A. Church. A set of postulates for the foundation of logic. Annals of Mathematics, 33:346-366, 1932.

[2] J. R. Hindley. The principal type-scheme of an object in combinatory logic. Transactions of the American Mathematical Society, 146:29-60, 1969.

[3] R. Milner. A theory of type polymorphism in programming. Journal of Computer and System Science, 17:348-374, 1978.

[4] L. Damas and R. Milner. Principal type-schemes for functional programs. In POPL '82: Proceedings of the 9th ACM SIGAPLAN-SIGACT Symposium on Principles of Programming Languages, pages 207-212. ACM, 1982.

References

[5] B. Heeren, J. Hage, and D. Swierstra. Generalizing hindley-milner type inference algorithms. Technical Report UU-CS-2001-031, Institute of Information and Computing Sciences, Utrecht University, 2002.

[6] S. J. Russell and P. Norvig. Artificial Intelligence: A modern Approach. Prentice Hall Series in Artificial Intelligence. Pearson Education, 2nd edition, 2003.

[7] J. Palsberg and M. I. Schwartzbach. Objected-oriented type inference. In Proceedings of OOPSLA'91, ACM SIGPLAN Sixth Annual Conference on Object-Oriented Programming Systems, Languages and Applications, pages 146-161, October 1991.

[8] O. Agesen. The cartesian product algorithm: Simple and precise type inference of parametric polymorphism. In ECOOP '95 Conference Proceedings, Aarhus, Denmark, August 1995.

Thank You Very Much!