On the Cubic Bottleneck in Subtyping and Flow Analysis

Nevin Heintze* David McAllester†

February 18, 1997

Abstract

A variety of program analysis methods have worst case time complexity that grows cubicly in the length of the program being analyzed. Cubic complexity typically arises in control flow analyses and the inference of recursive types (including object types). It is often said that such cubic performance can not be improved because these analyses require "dynamic transitive closure". Here we prove linear time reductions from the problem of determining membership for languages defined by 2-way nondeterministic pushdown automata (2NPDA) to problems of flow analysis and typability in the Amadio-Cardelli type system. An $O(n^3)$ algorithm was given for 2NPDA acceptability in 1968 and is still the best known. The reductions are factored through the problem of "monotone closure" and we propose linear time reduction of the monotone closure as a method of establishing "monotone closure hardness" for program analysis problems. A sub-cubic procedure for a monotone closure hard problem would imply a sub-cubic procedure for membership in any 2NPDA language.

1 Introduction

The static analysis of computer programs is useful in a variety of ways. It allows bugs to be found at compile time and can be used to ensure that certain classes of errors can never occur at run time. Here we are concerned with establishing the inherent complexity of a family of program analyses problems.

When designing polynomial time program analysis algorithms one is often faced with the question of whether a given cubic time procedure can be improved — can the same problem be solved in sub-cubic time. Here we present a new tool for establishing hardness results for cubic time problems — conditions which make it unlikely that a sub-cubic algorithm can be found. Proving a cubic time hardness results avoids wasted effort in trying to find a sub-cubic procedure. Our cubic time hardness methods also seem

*Bell Labs, 600 Mountain Ave, Murray Hill, NJ 07974, nch@bell-labs.com.

†AT&T Labs, 600 Mountain Ave, Murray Hill, NJ 07974, dmac@research.att.com.

to provide insight into the true source of the time complexity of a variety of program analysis problems.

Cubic time complexity is common in program analyses based on methods similar to control flow analysis such as that developed by Shivers [15]. This includes algorithms for inferring recursive types such as the the Palsberg and O'Keefe method of determining typability in the Amadio-Cardelli type system [13, 1]. Cubic behavior also arises in set-based analyses [4, 9, 10].

Our approach to proving cubic time hardness results is derived from a result of Radford Neal's which states that the problem of determining membership in languages defined by 2-way nondeterministic pushdown automata (2NPDA) can be reduced in linear time to the problem of "monotone closure" [12]. A cubic time algorithm for membership in a language defined by a 2NPDA was given in 1968 [2] but no sub-cubic algorithm has been found. Languages defined by (one way) nondeterministic pushdown automata (NPDA) are precisely the context free languages. Membership in a context free language can be determined in sub-cubic time using sub-cubic procedures for matrix multiplication. But a 2NPDA can define languages such as $a^n b^n c^n$ which are not context free and no sub-cubic membership test is known. Neal's theorem implies that if a sub-cubic procedure could be found for monotone closure, then membership in languages defined by a 2NPDA could be done in sub-cubic time. We take this to be a kind of "cubic time hardness" result for the monotone closure problem.

The monotone closure problem has some advantages over 2NPDA as a tool for establishing hardness results. The monotone closure problem is simpler and hence easier to use in reductions. Also, it appears that monotone closure is closer in structure to the problems that arise in program analysis. Monotone closure can be viewed as a generalization of transitive closure. A problem instance consists of a set E of edges plus a target edge $n \rightarrow m$. Each edge in E is an arc of the form $n_1 \rightarrow n_2$, $n_1 \rightarrow f(n_2)$ or $f(n_1) \rightarrow n_2$. Here n and m range over symbolic constants called *basic nodes* and f ranges over a given set of monadic function symbols. The monotone closure problem is to determine whether the target edge $n \rightarrow m$ can be derived from the edges in E using the inference rules of transitivity and monotonicity where monotonicity states that if $n_1 \rightarrow n_2$ then $f(n_1) \rightarrow f(n_2)$. It is possible to show that one need only derive arcs of the form $s \rightarrow t$ where s and t are expressions of the form n or $f(n)$ that appear in arcs in E [11, 8]. So, as in transitive closure, the set of derivable arcs can be enumerated in a naive way in cubic time. But unlike transitive closure, no sub-cubic algorithm is known.

As examples of a monotone closure hardness results we given linear time reduction of monotone closure to control flow analysis and typability the Amadio-Cardelli type system. These results indicate that the known cubic time algorithms for these problems are unlikely to be easily improved. We believe that this cubic time hardness method can be applied to a variety of analysis problems that intuitively involve flow analysis or recursive types.

2 2NPDA and Monotone Closure

Here we review Neal's result in [12] that if the monotone closure problem can be solved in sub-cubic time then so can the membership problem for any language defined by a 2NPDA. The input string to a pushdown automata is written on a tape where the end of the input is marked by a special tape symbol. The input tape is read only but the automata has access to a push-down store of unbounded depth. A configuration of the machine consists of a machine state, the location in the input currently being read, and the string of symbols in the pushdown store. The transition table of the (nondeterministic) machine specifies a set of possible actions for each triple of a machine state, top stack symbol, and input symbol. An action can push or pop the pushdown store (where any stack symbol may be pushed) and move the read head left or right (in a two-way automata). In the initial configuration for a given input string the automata is in a specified initial state, is reading the first symbol in the string, and the push down store is empty. The matching accepts the input if there exists a sequence of allowed actions leading to an accepting machine state.

We now reduce the problem of whether a given 2NPDA accepts a given input string to the problem of monotone closure. We consider a particular machine running on a particular input. We represent a configuration of the 2NPDA computation by a term of the form $f_1(f_2(\ldots f_k(n_{i,s})))$ where $n_{i,s}$ is a basic node representing the position i of the head and the state s of the machine and f_1, f_2, …, f_k is a sequence of function symbols representing the state of the push-down store with f_k representing to top of the store (the readable symbol). The set of possible machine actions can now be encoded in "rewrite rules". For example, suppose the given symbol at position i allows the machine to pop the symbol g from the push-down store when the machine is in state s and reading position i. Then (for the given input string) we have the rewrite rule $g(n_{i,s}) \rightarrow n_{j,w}$ where j is the next head position and w is the next machine state. Since the input string is given, one can write down the set of actions allowed at each string position — each action is represented by a rewrite rule. Rewrite rules representing push actions have the form $g(n_{i,s}) \rightarrow g(f(n_{j,w}))$ which can be represented by the "standard form" rules $g(n_{i,s}) \rightarrow a$, $a \rightarrow g(b)$, and $b \rightarrow f(n_{j,w})$. The 2NPDA can always be modified so that if it accepts a given string then it reaches a standard accepting configuration with an empty push–down store and with the head positioned at the initial tape symbol. Now the given machine accepts the given input if and only if the term representing the initial configuration can be rewritten to the term representing the standard accepting configuration using these rewrite rules. The process of rewriting corresponds precisely to the set of arcs derivable by transitivity and monotonicity. So the given machine accepts the given input if and only if one can derive $n_{0,s} \rightarrow n_{0,w}$ where s is the start state and w is the canonical accept state.

Note that the number of rewrite rules generated is at most Tn where T is the size of the transition table of the machine and n is the length of the input string. For a fixed machine this reduction is linear in the length of the input string. Hence if there exists a sub-cubic procedure for monotone closure then there exists a sub-cubic procedure for determining membership in any language defined by a 2NPDA.

By increasing the number of machine states by a factor of $\log n$ we can convert any given machine to one that accepts the same language but uses only two push down symbols. When the above construction is applied to a machine with only two push down symbols the resulting monotone closure problem has only two function symbols. So if the monotone closure problem could be solved sub-cubicly for problems with only two function symbols then 2NPDA membership could also. Two function symbols suffice for "2NPDA hardness".

From the above proof it should also be clear that monotone closure remains 2NPDA hard if we add a reflexivity rule stating that $x \rightarrow x$ for any x.

3 Bimonotone Closure

Our objective now is show that control flow and typability in Amadio-Cardelli type system are both monotone closure hard. In these systems one is working with λ-terms that denote procedures. We are concerned with two semantically meaningful operations that can be applied to the (types) of these procedures — the operators *dom* and *ran* that take a procedure (or type) and return its domain or range respectively. There is a natural order on procedures where, intuitively, $f \rightarrow g$ provided that it is safe to use g anywhere that f can be used. The operation *ran* is monotone, i.e. $n_1 \rightarrow n_2$ implies $ran(n_1) \rightarrow ran(n_2)$. But unfortunately, *dom* is anti-monotone, i.e., if $n_1 \rightarrow n_2$ then $dom(n_2) \rightarrow dom(n_1)$. The presence of monotone and anti-monotone operations on function types motivates the study of a variant of monotone closure with one monotone and one anti-monotone operation. We call this bimonotone closure.

We now show that monotone closure can be linear time reduced to bimonotone closure and hence bimonotone closure is also monotone closure hard. Suppose we are given a monotone closure problem with two function symbols f and g. The problem consists of a finite set of edges E and a target edge $n \rightarrow m$. We want to determine whether $n \rightarrow m$ follows from E and the inference rules of transitivity and monotonicity. We now reduce this problem to bimonotone closure where f is monotone and g is anti-monotone. We first introduce a new basic node n' for each node n in E. Intuitively, the arcs have the reverse meaning when applied to these "negative" nodes, e.g., $m' \rightarrow n'$ is equivalent to $n \rightarrow m$. We then construct a new set of arcs E' where E' has two arcs for each arc in E defined by the condition that if E constrains $s \rightarrow t$ then E' contains $tran(s) \rightarrow tran(t)$ and $tran'(t) \rightarrow tran'(s)$ where the operations *tran* and *tran'* are defined as follows.

$$tran(n) = n, \;\; tran(f(n)) = f(n), \;\; tran(g(n)) = g(n')$$

$$tran'(n) = n', \;\; tran'(f(n)) = f(n'), \;\; tran'(g(n)) = g(n)$$

For example, the arc $n \rightarrow g(m)$ becomes $n \rightarrow g(m')$ and $g(m) \rightarrow n'$. We will argue that $n \rightarrow m$ is derivable from E using the monotone rules if and only if it is derivable from

E' using a monotone rule for f and an anti-monotone rule for g.

Like monotone closure, bimonotone closure also has the property that it suffices to derive only those arcs involving expressions that appear in E'. All expressions in E' have the form n, n', $f(n)$, $f(n')$, $g(n)$ or $g(n')$. Expressions of the form n, $f(n)$ or $g(n')$ will be called *positive* and expressions of the form n', $f(n')$, and $g(n)$ will be called *negative*. All arcs in E' either involve two positive expressions or two negative expressions (*trans* produces positive expressions and *trans'* produces negative expressions). Furthermore, this property is preserved by the inference rules — every derived arc either involves two positive expressions or two negative expressions. Arcs involving positive expressions will be called positive arcs and arcs involving negative expressions will be called negative arcs. The anti-monotonicity rule derives a negative arc from a positive arc (and vice-versa) while inverting the direction of the arrow. We define the "flip" $F(s)$ of an expression s to be the result of replacing n by n' and n' by n. The flip of a positive expression is negative and the flip of a negative expression is positive. The bimonotone inference rules applied to the arcs in E' have the property that if they derive an arc $s \rightarrow t$ then they also derive the arc $F(t) \rightarrow F(s)$ where F is the flip operation. The inference processes are related by the invariant that an arc $s \rightarrow t$ is derivable from E under the monotone rules if and only if both $\mathit{trans}(s) \rightarrow \mathit{trans}(t)$ and $\mathit{trans}'(t) \rightarrow \mathit{trans}'(s)$ are derivable from E' using the bimonotone rules. So the arc $n \rightarrow m$ is derivable from E using the monotone rules if and only if it is derivable from E' using the bimonotone rules.

As with monotone closure, bimonotone closure remains 2NPDA hard if we include the reflexivity rule that $x \rightarrow x$ for any x.

4 Control Flow Analysis

Control flow analysis has received considerable attention lately. For the case of the pure λ-calculus it seems safe to say that there is a "standard" cubic time analysis (sometimes called 0-CFA) [15, 14]. In this section we show that standard control flow analysis on the pure λ-calculus is monotone closure hard and hence is unlikely to be easily done in sub-cubic time.

The pure λ-calculus is defined by the following grammar.

$$e \ ::= \ x \ \mid \ \lambda x.e \ \mid \ (e_1 \ e_2)$$

Bound and free variables are defined in the usual way. We formulate standard control flow analysis in a somewhat nonstandard way. In our formulation control flow analysis takes as input a term e and derives arcs of the form $u \rightarrow w$ where u and w are subterms of e. The set of derived "flow arcs" between subterms of e is defined by the following three inference rules.

$$\frac{\texttt{INPUT}((f\ w)) \quad f \rightarrow \lambda x.u}{x \rightarrow w,\ \ (f\ w) \rightarrow u} \qquad \frac{\texttt{INPUT}(u)}{u \rightarrow u} \qquad \frac{e \rightarrow u \quad u \rightarrow w}{e \rightarrow w}$$

Antecedents of the form $\texttt{INPUT}(u)$ in the above rules signify that u is a subterm of the input term. We have assumed that the input term has been α-renamed so that semantically distinct variables have been given distinct names. We can think of the arcs in the flow analysis as defining a kind of abstract rewrite relation, i.e., $u \rightarrow w$ signifies that if we repeatedly reduce the input term using β-reductions then some some substitution instance of u may rewrite to a substitution instance of w. A formal relationship to environment evaluation can be found in [10].

All arcs derivable by the above rules are between subterms of the input term. The $O(n^2)$ derivable arcs can be enumerated by simply running the inference rules to completion in a forward chaining manner. A naive implementation of the transitivity rule takes $O(n^3)$ time so we get an $O(n^3)$ procedure.

Given an input term e, and a "target arc" of the form $u \rightarrow w$, we take the control flow problem to be that of determine if the given arc can be derived using the above rules applied to the given input term. Note that the problem of determining if a given arc can be derived using transitive closure can be decided in time linear in the number of input arcs. Since we are given the target arc one might think that a sub-cubic procedure was possible. But we show here that this problem is monotone closure hard and hence unlikely to have an easily found sub-cubic procedure.[1]

The proof of monotone closure hardness for control flow analysis is by reduction of bimonotone closure with reflexivity. Let E be the set of arcs and $n \rightarrow m$ the target arc of an instance of the bimonotone closure problem where the monotone operations is called *ran* and the anti-monotone operation is called *dom* (for range and domain respectively). We will think of the basic nodes in E as procedures each with a domain and range. The inference rules above can be viewed as constructing domains and ranges for procedure objects. e.g., if $x \rightarrow u$ then u is in the domain of $\lambda x.w$ and if $w \rightarrow v$ then v is in the range of $\lambda x.w$.

Unfortunately the reduction based on this intuition is fairly complex. The reduction builds a "machine" which uses the flow analysis rules to implement the bimonotone closure rules. We will use $\bot$ as an abbreviation for the λ-term $(\lambda h.(h\ h)\ \lambda h.(h\ h))$. Note that the inference rules derive $\bot \rightarrow (h\ h)$ but no other arcs from $\bot$. α-variants of the identity function $\lambda x.x$ can be used to "store" sets of terms. Consider a term of the form $((\lambda s.b)\ \lambda x.x)$. This will generate the arc $s \rightarrow \lambda x.x$. Inside the term b the variable s can

[1] For simply typed pure λ-calculus terms of bounded order and arity we have recently shown that this control flow problem can be solved in linear time.

be used as a set into which items can be inserted and out of which items can be extracted. To insert an item v into the set s we include in b the application $(s\ v)$. Elements can be extracted from the set represented by s using the expression $(s\ \bot)$. More precisely, the rules derive $(s\ \bot) \rightarrow v$ for each v that is inserted into the set s.

Note that the term $((\lambda s.b)\ \lambda x.x)$ can be viewed as a definition of the variable s where s can then be used in b. More generally we will use

$$\textbf{let} \quad x_1 = e_1; \ldots; x_k = e_k \ \textbf{in} \ e$$

as an abbreviation for $((\lambda x_1.((\lambda x_2.\ldots((\lambda x_k.e)\ e_k)\ \ldots\ )\ e_2))\ e_1)$. Often we want to construct a term containing a certain set of subterms without regard for how those subterms are included except that "side effects" of the inclusion should be avoided. For terms $e_1, \ldots, e_n$ we take the expression $\{e_1, \ldots, e_n\}$ to be an abbreviation for the *term*

$$\textbf{let} \quad y_1 = e_1; \ldots; y_n = e_n \ \textbf{in} \ \bot$$

where $y_1, \ldots, y_n$ are fresh variables.

The reduction can now be defined as follows. Given a bimonotone closure problem E with target arc $n \rightarrow m$ we construct the following term of the pure λ calculus.

$$\begin{array}{ll}
\textbf{let} & Id_n = \lambda x.x, \ n \text{ a basic node} \\
& Id_{dom(n)} = \lambda x.x, \ n \text{ a basic node} \\
& Id_{ran(n)} = \lambda x.x, \ n \text{ a basic node} \\
& \text{in } \{(Id_n\ \lambda y.\{(Id_{dom(n)}\ y)\}, \text{ for } n \text{ a basic node} \\
& \quad ((Id_n\ \bot)\ (Id_{dom(n)}\ \bot)), \text{ for } n \text{ a basic node} \\
& \quad (Id_n\ \lambda y.(Id_{ran(n)}\ \bot)), \text{ for } n \text{ a basic node} \\
& \quad (Id_{ran(n)}\ ((Id_n\ \bot)\ \bot))), \text{ for } n \text{ a basic node} \\
& \quad (Id_s\ (Id_t\ \bot)), \text{ for } s \rightarrow t \in E\}
\end{array}$$

We assume that distinct bound variables in the above term, such as the formal parameters to the various identity functions, are α-renamed so as to be distinct. The control flow problem is to determine whether control flow applied to the above term derives the arc $x_n \rightarrow x_m$ where $n \rightarrow m$ is the original target arc and x_n and x_m are the bound variables of Id_n and Id_m respectively.

We now prove the correctness of the reduction. For each expression s of the form n, $dom(n)$ or $ran(n)$ in E we define the variable x_s to be the bound variable of the identity function Id_s. The last line in the above term ensures that if $s \rightarrow t \in E$ then $x_s \rightarrow x_t$ is derivable by flow analysis. We now show that the set of arcs of this form which are derivable by flow analysis is closed under the inference rules of bimonotone closure with reflexivity. Clearly such arcs are closed under transitivity and reflexivity. We need to show that they are also closed under montonicity of the function *ran* and anti-monotonicity of the function *dom*. First we consider *ran*. Suppose that control flow analysis derives $x_n \rightarrow x_m$. We want to show that in this case it also derives

$x_{ran(n)} \rightarrow x_{ran(m)}$. Since the above term contains $(Id_m\ \lambda y.(ran(m)\ \bot))$ we have $x_m \rightarrow \lambda y.(ran(m)\ \bot)$. So by transitivity we have $x_n \rightarrow \lambda y.(ran(m)\ \bot))$. But the above term contains $(Id_{ran(n)}\ ((Id_n\ \bot)\ \bot))$ and we now have that $(Id_n\ \bot) \rightarrow \lambda y.(Id_{ran(m)}\ \bot)$. So we now have that $x_{ran(n)} \rightarrow x_{ran(m)}$. Now we prove that the derived arcs are closed under the antimonotone rule for dom. Suppose that control flow derives $x_n \rightarrow x_m$. We want to show that in this case it also derives $x_{dom(m)} \rightarrow x_{dom(n)}$. Because the above term includes $(Id_m\ \lambda y.\{(Id_{dom(m)}\ y)\})$ and we have $x_n \rightarrow x_m$, we also have $x_n \rightarrow \lambda y.\{(Id_{dom(m)}\ y)\}$. Because the term includes $((Id_n\ \bot)\ (Id_{dom(n)}\ \bot))$, and $(Id_n\ \bot) \rightarrow \lambda y.\{(Id_{dom(m)}\ y)\}$, we get $y \rightarrow x_{dom(n)}$ and hence $x_{dom(m)} \rightarrow x_{dom(n)}$. So we have now proved that flow analysis generates all arcs of the form $x_s \rightarrow x_t$ where bimonotone closure generates $s \rightarrow t$. We omit the proof that no extraneous arcs are generated.

5 Amadio-Cardelli Typability

Finally we show that typability in the Amadio-Cardelli system of recursive types is monotone closure hard and a sub-cubic procedure is therefore unlikely to be easily found. In the Amadio-Cardelli system all pure λ-terms are typable. We therefore now consider a modified term grammar which includes the numerical constant 0.

$$e ::= x \mid 0 \mid (e_1,\ e_2) \mid \lambda x.e$$

By the recently established equivalence of flow analysis and recursive types [13] a term e is typable in the Amadio-Cardelli system if and only if there is no application $(f\ w)$ in e such that the flow rules of the previous section derive $f \rightarrow 0$, i.e., there is no attempt to apply the numerical constant 0 as a procedure.

In order to make terms potentially untypable we need to include the non-procedure 0 in a variant of the reduction used in the previous section. This is done by usnig pairs of procedure part and a "flag" part where the flag part is either the harmless term $\bot$ or the dangerous term 0. We use the notation $\langle u,\ w\rangle$, representing the pair of u and w, as an abbreviation for the term $\lambda z.((z\ u)\ w)$. We let $\mathit{first}(p)$ be an alternate notation for $(p\ \lambda xy.x)$ and we let $\mathit{second}(p)$ notate $(p\ \lambda xy.y)$.

Now given an instance E of the bimonotone closure problem with target arc $n \rightarrow m$ we construct the following term.

$$
\begin{array}{ll}
\textbf{let} & Id_n = \lambda x.x,\ n \text{ a basic node} \\
& Id_{dom(n)} = \lambda x.x,\ n \text{ a basic node} \\
& Id_{ran(n)} = \lambda x.x,\ n \text{ a basic node} \\
& \text{in } \{(Id_n\ \langle \lambda y.\{(Id_{dom(n)}\ y)\},\ \bot\rangle),\ \text{for } n \text{ a basic node} \\
& \quad (\mathit{first}((Id_n\ \bot))\ (Id_{dom(n)}\ \bot)),\ \text{for } n \text{ a basic node} \\
& \quad (Id_n\ \langle \lambda y.(Id_{ran(n)}\ \bot),\ \bot\rangle),\ \text{for } n \text{ a basic node} \\
& \quad (Id_{ran(n)}\ (\mathit{first}((Id_n\ \bot))\ \bot))),\ \text{for } n \text{ a basic node} \\
& \quad (Id_s\ (Id_t\ \bot)),\ \text{for } s \to t \in E \\
& \quad (Id_m\ \langle \bot,\ 0\rangle), (\mathit{second}((Id_n\ \bot))\ \bot),\ \text{for target } n \to m\}
\end{array}
$$

This term is similar to the one used in the previous section except that pairs are used here where functions were used before. The first component of each pair is a function that plays the same role as the corresponding function in the previous term. However, using pairs allows the second component of the pair to carry a "bomb" which can explode if it reaches the appropriate place. Note that the only occurrence of *second* is in the very last line of the above term. The second of most terms is $\bot$. The only way to get 0 as a value, i.e., to generate arcs of the for $u \to 0$, is if second is applied to the pair $\langle \bot,\ 0\rangle$ which also appears in the last line. Now if the bimonotone inference procedure fails to derive the target arc then the flow analysis will fail to generate any arcs of the form $u \to 0$. So the term is typable. On the other hand, if the bimonotone closure generates the target arc then the flow analysis generates $\mathit{second}((Id_n\ \bot)) \to 0$ and the term is untypable.

6 Conclusions

We have shown that it is unlikely that one can easily find a sub-cubic procedure for monotone closure — a sub-cubic procedure for monotone closure would imply a sub-cubic procedure for membership in languages defined by 2NPDA and this has not been done in the almost 30 years since a cubic procedure was found. We have also shown that monotone closure can be be reduced in linear time to various other problems which are then, by definition, monotone closure hard. In particular, we have shown that standard flow analysis for the pure lambda calculus and typability in the Amadio-Cardelli system of recursive types are both monotone closure hard. We expect that similar proofs can be constructed for a variety of program analysis problems.

References

[1] R. Amadio and L. Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, 1993. Also in Proc. POPL91.

[2] A. Aho, J. Hopcroft, and J. Ullmann, "Time and tape complexity of pushdown automaton languages", *Information and Control*, vol. 13, no. 3, pp. 186-206, 1968.

[3] A. Bondorf and J. Jorgensen, "Efficient analysis for realistic off-line partial evaluation", *Journal of Functional Programming*, Vol. 3, No. 3, July 1993.

[4] N. Heintze, "Set-Based Analysis of ML Programs", *ACM Conference on Lisp and Functional Programming*, pp 306–317, 1994.

[5] N. Heintze, "Control-Flow Analysis and Type Systems", *Static Analysis Symposium*, 1995, pp 189–206.

[6] N. Jones, "Flow Analysis of Lambda Expressions", *Symp. on Functional Languages and Computer Architecture*, pp. 66-74, 1981.

[7] N. Jones, "Flow Analysis of Lazy Higher-Order Functional Programs", in *Abstract Interpretation of Declarative Languages*, S. Abramsky and C. Hankin (Eds.), Ellis Horwood, 1987.

[8] Robert Givan and David McAllester. New results on local inference relations. In *Principles of Knolwedge Representation and Reasoning: Proceedings of the Third International Conference*, pages 403–412. Morgan Kaufman Press, October 1992. internet file ftp.ai.mit.edu:/pub/users/dam/kr92.ps.

[9] David McAllester. Inferring recursive data types. http://www.ai.mit.edu/people/dam/rectypes.ps.

[10] David McAllester and Nevin Heintze. On the compexity of set-based analysis. http://www.ai.mit.edu/people/dam/setbased.ps.

[11] D. McAllester and R. Givan. Taxonomic syntax for first order inference. *JACM*, 40(2):246–283, April 1993. internet file ftp.ai.mit.edu:/pub/users/dam/jacm1.ps.

[12] R. Neal, "The computational complexity of taxonomic inference", unpublished manuscript, 18 pages, 1989, (ftp://ftp.cs.utoronto.ca/pub/radford/taxc.ps.Z).

[13] J. Palsberg and P. O'Keefe, "A Type System Equivalent to Flow Analysis", *POPL-95*, pp. 367–378, 1995.

[14] J. Palsberg and M. Schwartzbach, "Safety Analysis versus Type Inference" *Information and Computation*, Vol. 118, No. 1, pp. 128-141, April 1995.

[15] O. Shivers, "Control Flow Analysis in Scheme", *Proc. 1988 ACM Conf. on Programming Language Design and Implementation*, Atlanta, pp. 164–174, June 1988.