Subcubic Control Flow Analysis Algorithms

Jan Midtgaard
David Van Horn

MAY 2009 ROSKILDE UNIVERSITY COMPUTER SCIENCE RESEARCH REPORT #125

Jan Midtgaard and David Van Horn

Computer Science
Roskilde University
P. O. Box 260
DK–4000 Roskilde
Denmark

Telephone: +45 4674 3839
Telefax: +45 4674 3072
Internet: http://www.ruc.dk/dat_en/
E-mail: datalogi@ruc.dk

ISSN 0109–9779

Research reports are available electronically from:

http://www.ruc.dk/dat_en/research/reports/

Subcubic Control Flow Analysis Algorithms

Jan Midtgaard[1] and David Van Horn[2]

[1] Roskilde University
[2] Northeastern University

Abstract. We give the first direct subcubic algorithm for performing control flow analysis of higher-order functional programs. Despite the long held belief that inclusion-based flow analysis could not surpass the "cubic bottleneck," we apply known set compression techniques to obtain an algorithm that runs in time $O(n^3/\log n)$ on a unit cost random-access memory model machine. Moreover, we refine the initial flow analysis into two more precise analyses incorporating notions of reachability. We give subcubic algorithms for these more precise analyses and relate them to an existing analysis from the literature.

1 History and Motivation

Control flow analysis (CFA) is a fundamental static analysis of higher-order programming languages and forms the basis of a range of other analyses. It determines for each call site of a program a set of functions which may be applied when the program is run. Over a decade ago, Heintze and McAllester [1997b] proved deciding these problems to be at least as hard as 2NPDA, the class of problems decided by two-way nondeterministic push-down automata, and argued this provided evidence the "cubic bottleneck" of flow analysis could not be overcome. This and several other papers [Neal, 1989, Heintze and McAllester, 1997b,c, Melski and Reps, 2000, McAllester, 2002] state that the cubic algorithm for 2NPDA has not been improved since its formulation by Aho et al. [1968] — an oversight in the history of events; Rytter [1985] improved the cubic bound by a logarithmic factor. Since then, Rytter's technique has been used in various contexts: in diameter verification, in boolean matrix multiplication, and for the all pairs shortest paths problem [Basch et al., 1995, Zwick, 2006, Chan, 2007] as well as for reachability in recursive state machines [Chaudhuri, 2008], and for maximum node-weighted k-clique [Vassilevska, 2009] to name but a few. In particular, Chaudhuri [2008] recently used Rytter's techniques to formulate a subcubic algorithm for the related problem of context-free language (CFL) reachability. Perhaps unbeknownst to most, indirectly this constitutes the first subcubic CFA algorithm when combined with a reduction due to Melski and Reps [2000].

In this paper, we recall Rytter's improvement, investigate the implications, and formulate a simpler, direct subcubic control flow analysis algorithm. Using the initial analysis as an offset we formulate two refined, increasingly precise analyses incorporating *reachability*, both of which are also implementable in subcubic

time. Finally we relate all three analyses and prove the final refinement equivalent to a set-based analysis with reachability due to Heintze and McAllester [1997a], thereby giving the first subcubic algorithm thereof.

2 Definitions

We recall the definition of 2NPDA, the implementation of *fast sets* on which we base our algorithms, and the specification of CFA.

2.1 2NPDA

The class 2NPDA represents the languages recognizable by a two-way nondeterministic push-down automaton. The familiar deterministic and non-deterministic PDAs found in undergraduate textbooks [Martin, 1997] are one-way: consuming their input from left-to-right. In contrast, two-way NPDAs accept their input on a read-only input tape (marked with special begin and end markers), on which they can move the read-head forwards, backwards, or not at all.

2.2 Fast Sets

Rytter [1985] applies to the recognition of 2NPDA languages a known set compression technique, which we now recall.

Assume we are concerned with operations over a set $\{0, \ldots, n-1\}$ and that the RAM model has $\Theta(\log n)$ word size. A standard representation of a subset hereof is *a characteristic vector* or *bit vector* of length n, with one bit representing each element. Now break the vector into $O(n/\log n)$ words each of $\Theta(\log n)$ bits. Following Chaudhuri [2008], we refer to the resulting structure as a *fast set*. An element can be inserted into a word in $O(1)$ time by *or*'ing in a 1 at the proper position. Furthermore, the set-difference of two words can be computed in $O(1)$ time by *and*'ing the first word with the complement of the second.

As a consequence, an element can be inserted into a fast set in $O(1)$ time, by first determining the relevant word and performing the above, constant-time word insertion. We can furthermore compute a list representing the set-difference of two sets in $O(n/\log n + v)$ time (with v being the number of elements in the result), by computing the set-difference of each pair of words. For each word in the result, repeatedly locate the most significant bit and turn it off. Suppose we can locate the most significant bit of a word in constant time, this will take time proportional to the size of the result. Even if such an operation is not available, we can pre-compute it and store the result in a table. Table 1 summarizes the above operations.

Rytter's subcubic 2NPDA decision procedure works by enumerating the machine states of the automaton into a set, for which he investigates reachability between initial and final states. Based on the above operations and by compressing the rows and columns of a standard adjacency matrix representation, he achieves an $O(n^3/\log n)$ algorithm for 2NPDA recognition.

operation	type	time complexity
$\in$	$int \times fset \to bool$	$O(1)$
INSERT	$int \times fset \to unit$	$O(1)$
DIFF	$fset \times fset \to int\ list$	$O(n/\log n + v)$

Table 1: Type and worst-case time complexity of fast set operations

$$\frac{input(e)}{e \to e}\ (\text{REFL}) \qquad \frac{input(e_1\ e_2) \qquad e_1 \to \lambda x.\ e}{x \to e_2,\ e_1\ e_2 \to e}\ (\text{APP})$$

$$\frac{e_0 \to e_1 \qquad e_1 \to e_2}{e_0 \to e_2}\ (\text{TRANS})$$

Fig. 1: Inference rules for the system $\vdash$

2.3 Control Flow Analysis and Control-Flow Reachability

We now recall Heintze and McAllester's analysis formulation and the corresponding decision problem. The input is given as a pure lambda calculus expression. Expressions are variables, lambda abstractions, and applications.

$$Expr \ni e ::= x \mid \lambda x.\ e \mid e\ e \qquad \text{(expressions)}$$

Heintze and McAllester [1997b] formulate CFA as a graph reachability problem over a set of inference rules. We recall their inference rules in Fig. 1 and write $\vdash$ to mean provability by the inference system. Intuitively, an edge $e \to e'$ can be understood as an inclusion of values, i.e., all values that can result from evaluating e' can also result from evaluating e. The inference rules rely on a predicate *input*, which is defined in Fig. 2. The predicate holds of all subexpressions and bound variables of the input program. The analysis merges all bindings to the same variable (as out-going edges from the variable node) thereby creating the effect of a global environment. It furthermore confuses all calling contexts of the same function (as in-going edges into the node representing the function body). The common name *0-CFA* [Shivers, 1991, Nielson et al., 1999] is by now used for analyses with these two properties. Rather than implementing instances of the inference rules as constraints, we will use the inference system as a reference proof system. The analysis is formulated as a decision problem termed *control-flow reachability*, by accepting two additional subexpressions e_{start} and e_{finish} as input and determining whether the latter is reachable from the former. Heintze and McAllester [1997b] show how to reduce a control-flow reachability problem to an equivalent 2NPDA recognition problem.

$$\frac{}{input(e_p)} \qquad \frac{input(e_1\ e_2)}{input(e_1),\ input(e_2)} \qquad \frac{input(\lambda x.\ e)}{input(x),\ input(e)}$$

Fig. 2: Input predicate

3 Indirect Algorithms: Connecting the Dots

We examine two indirect CFA algorithms from the research literature with potential to have subcubic worst-case time complexity. The first algorithm is the graph-based analysis from Sect. 2.3. The second algorithm is a constraint-based or *set-based analysis*, termed *ML set constraints* by Melski and Reps [2000].

Indirect Control-Flow Reachability (1)

1. reduce control-flow reachability to 2NPDA recognition [Heintze and McAllester, 1997b]
2. apply Rytter's improved algorithm [Rytter, 1985]
3. map result(s) back to control-flow reachability

In the above, Heintze and McAllester [1997b] encode variables as bit strings. As a consequence, the encoded lambda-term input incurs a blowup of a logarithmic factor. Ironically, this logarithmic factor cancels the logarithmic factor gained by Rytter [1985], resulting in a cubic-time algorithm. Hence a logarithmic factor improvement to 2NPDA recognition is lost in translation, so to speak.

Indirect Control-Flow Reachability (2)

1. reduce ML set constraints to CFL-reachability [Melski and Reps, 2000]
2. apply Chaudhuri's improved CFL-reachability algorithm [Chaudhuri, 2008]
3. map result(s) back to control-flow reachability

The direct reduction by Melski and Reps [2000] into a CFL-reachability problem requires $O(n^4)$ time (in the original input). However, Melski and Reps [2000] observe a rendundancy in the context-free grammars resulting from their reduction. They augment their algorithm with a normalization phase that removes the redunancy and recovers $O(n^3)$ worst-case time complexity. Since Chaudhuri's algorithm is given as an improvement to the CFL-reachability algorithm of Melski and Reps [2000], we believe the above constitutes the first indirect subcubic flow analysis algorithm.[3]

[3] This indirect algorithm was kindly pointed out to us by Thomas Reps.

```
CUBIC-CFA(e)
 1  Q, R ← {e → e | input(e)}                                  ▷ Case refl
 2  while Q ≠ ∅
 3        do (e_i → e_j) ← DELETE(Q)
 4           for e_k → e_i ∈ R such that (e_k → e_j) ∉ R      ▷ Case trans 1
 5                 do INSERT(e_k → e_j, R)
 6                    INSERT(e_k → e_j, Q)
 7           for e_j → e_k ∈ R such that (e_i → e_k) ∉ R      ▷ Case trans 2
 8                 do INSERT(e_i → e_k, R)
 9                    INSERT(e_i → e_k, Q)
10           if e_j = λx. e and input(e_i e_w)                ▷ Case app
11              then if (x → e_w) ∉ R
12                         then INSERT(x → e_w, R)
13                              INSERT(x → e_w, Q)
14                   if (e_i e_w → e) ∉ R
15                      then INSERT(e_i e_w → e, R)
16                           INSERT(e_i e_w → e, Q)
17  return R
```

Fig. 3: Cubic algorithm

4 A Direct Algorithm

In Fig. 3 we give an initial direct algorithm implementing the analysis of Heintze and McAllester. The algorithm is structured as a worklist algorithm. All newly added edges are also added to the worklist, Q. Intuitively, for each newly added edge, we investigate all the rules which could potentially apply because of the new addition. The algorithm represents reachability between two expressions e and e' as membership $(e \to e') \in R$ in an explicit set (or its isomorphic binary map). By representing R as an adjacency matrix, we obtain a cubic time algorithm as we shall now see. This upper bound coincides with the best known bound for control flow analysis in the research literature [Ayers, 1992, Heintze, 1994, Palsberg and Schwartzbach, 1995, Nielson et al., 1999].

4.1 Complexity

Time: The initialization of Q and R can be done as a linear traversal. All edges inserted in R are inserted in Q simultaneously. Once an edge is in R, it will never be reinserted in Q. As a consequence, all edges are only inserted in Q once. Hence the number of iterations of the **while**-loop is upper-bounded by the number of edges, i.e, $O(n^2)$. The execution time of the body of the **while**-loop is dominated by the **for**-loops. The two **for**-loops in the body can be implemented as linear scans of a matrix column and a matrix row, respectively. The **for**-loop bodies each require only constant time, hence the algorithm is cubic, $O(n^3)$.

Space: The worklist Q requires at most $O(n^2)$ space. Implementing the relation R as an adjacency matrix, we require $O(n^2)$ space. The latter can be

improved to $O(n^2/\log n)$ on a RAM with $\Theta(\log n)$ word size, by packing bits into words. However, overall the algorithm still requires $O(n^2)$ space.

4.2 Correctness

We now prove that the algorithm implements the inference rules correctly.

Theorem 1. $(e_0 \to e_1) \in \textsc{Cubic-CFA}(e_p) \iff \vdash e_0 \to e_1$

Proof. In the left-to-right direction, let e_p be given. We prove the statement $(e_0 \to e_1) \in R \implies \vdash e_0 \to e_1$ from which this direction of the theorem follows at the exit of the while loop. We use the loop invariant:

$$(e_0 \to e_1) \in R \implies \vdash e_0 \to e_1 \quad \wedge \quad (e_0 \to e_1) \in Q \implies \vdash e_0 \to e_1$$

Before entering the loop, $(e_0 \to e_1) \in R$ and $(e_0 \to e_1) \in Q$ implies $e_0 = e_1$ and $\mathit{input}(e_0)$, in which case the REFL rule applies. Assume the invariant holds before an iteration. We argue that it is preserved across a loop iteration, i.e., all new edge insertions into R and Q can be proved by the inference rules. At the first **for**-loop, it follows from the invariant that $\vdash e_i \to e_j, e_k \to e_i$, hence by TRANS, $\vdash e_k \to e_j$. At the second **for**-loop, it also follows from the invariant that $\vdash e_j \to e_k$, hence by TRANS, $\vdash e_i \to e_k$. For the last conditional to be true, $\vdash e_i \to \lambda x.\, e$ and $\mathit{input}(e_i\ e_w)$ must hold, so by APP, $\vdash x \to e_w, e_i\ e_w \to e$.

In the right-to-left direction, the proof relies on two observations: (1) insertions into R and Q are always simultaneous, and (2) R grows increasingly over time, since elements are never deleted from it.

The run of the CUBIC-CFA algorithm consists of a sequence of assignment pairs. We model the memory modifications to Q and R as a sequence $(Q_0, R_0) \ldots (Q_n, R_n)$, such that (a) two consecutive Q's differ only by one inserted or deleted edge, and (b) two consecutive R's differ only by an inserted edge. Since edges are never removed from R, we prove the statement

$$\vdash e_0 \to e_1 \implies \text{exists a least } i, \text{ such that } (e_0 \to e_1) \in R_i \ \wedge\ (e_0 \to e_1) \in Q_i$$

from which the right-to-left direction of the theorem follows. The proof proceeds by structural induction on the inference tree. For derivation trees of $\vdash e_0 \to e_1$ using the REFL axiom, we have $e_0 = e_1$ and there exists R_0 and Q_0 such that $(e_0 \to e_1) \in R_0$ and $(e_0 \to e_1) \in Q_0$ from the initialization in line 1. Assume the statement holds for all structurally smaller derivation trees. There are now two remaining cases. If $\vdash e_0 \to e_1$ by the TRANS rule, there exists a term e, such that $\vdash e_0 \to e, e \to e_1$ and hence there exists a least i and j such that $(e_0 \to e) \in R_i \ \wedge\ (e_0 \to e) \in Q_i$ and $(e \to e_1) \in R_j \ \wedge\ (e \to e_1) \in Q_j$ by the induction hypothesis. If $i = j$, the same edge is inserted in both R and Q, and $e_0 = e = e_1$. The conclusion therefore coincides with the induction hypothesis. In case $i < j$, when $(e \to e_1)$ is eventually deleted from the queue, $(e_0 \to e) \in R$. If $(e_0 \to e_1) \notin R$, we insert into both by *trans 1*, thereby proving existence. Otherwise, $(e_0 \to e_1) \in R$ and there exists (Q_k, R_k) at which point it

was inserted in both. In case $i > j$, when $(e_0 \to e)$ is eventually deleted from the queue, $(e \to e_1) \in R$. If $(e_0 \to e_1) \notin R$, we insert into both by *trans 2* thereby proving existence. Otherwise $(e_0 \to e_1) \in R$, and there exists (Q_k, R_k) at which point it was inserted in both.

In the APP case, we can assume that $\mathit{input}(e_1\ e_2)$ and $\vdash e_1 \to \lambda x.\, e$. Hence, there exists a least i such that $(e_1 \to \lambda x.\, e) \in R_i \ \wedge\ (e_1 \to \lambda x.\, e) \in Q_i$. We consider one sub-case for each of the two conclusions. In case $\vdash x \to e$, if $(x \to e) \notin R$, we insert the edge into both by *app* thereby proving existence. If on the other hand, $(x \to e) \in R$, it was inserted at some point (Q_k, R_k) into both thereby proving existence. In case $\vdash e_1\ e_2 \to e$, if $(e_1\ e_2 \to e) \notin R$, we insert the edge into both by *app*, proving existence. If on the other hand, $(e_1\ e_2 \to e) \in R$, it was inserted at some point (Q_k, R_k) into both, thereby proving existence. □

5 A Subcubic Algorithm

We formulate a subcubic algorithm as a refinement of the initial algorithm. The basic idea is to change the representation of R into a sequence of rows and columns, each of which is represented as a fast set. First we assume a numbering of all sub-expressions of the program, similar to the labelling found in text book presentations of CFA [Nielson et al., 1999]. To simplify the presentation we will identify a term with its unique label. Assume the labels constitute the set $\{0, \ldots, n-1\}$. We assume the following two operations:

$$\textsc{Column}(j) = \{k \mid (k \to j) \in R\} \quad \text{where} \quad 0 \le j \le n-1$$
$$\textsc{Row}(j) = \{k \mid (j \to k) \in R\} \quad \text{where} \quad 0 \le j \le n-1$$

$\textsc{Row}(j)$ represents the set of nodes reachable from j. $\textsc{Column}(j)$ represents the set of nodes that can reach j. Each insertion $\textsc{Insert}((e_i \to e_j), R)$ in the original algorithm is implemented with two instructions: $\textsc{Insert}(i, \textsc{Column}(j))$ and $\textsc{Insert}(j, \textsc{Row}(i))$. Queries $(e_i \to e_j) \notin R$ are implemented as: $i \notin \textsc{Column}(j)$ **and** $j \notin \textsc{Row}(i)$. Finally, we substitute the **for**-loop condition

4 **for** $(e_k \to e_i) \in R$ **such that** $(e_k \to e_j) \notin R$ ▷ Case trans 1

with the equivalent condition: **for** $k \in \textsc{Diff}(\textsc{Column}(i), \textsc{Column}(j))$ and similarly the second **for**-loop condition is rewritten based on ROW. The final algorithm is given in Fig. 4. Insertions into R are marked with an asterisk, but otherwise the algorithm is kept in its original form.

5.1 Complexity

Time: Pre-computing a table of most-significant bits can be done as a linear traversal of all possible $\Theta(\log n)$ length words, hence it can be done in $O(2^{\log n}) = O(n)$ time. Compared to the cubic-time algorithm, we have only changed the **for**-loops. We consider the complexity of the loops and their bodies separately. The DIFF operation can be computed in $O(n/\log n)$ time. The insertion into Q, R,

```
SUBCUBIC-CFA(e)
 1  Q, R ← {e → e | input(e)}                                   ▷ Case refl
 2  while Q ≠ ∅
 3        do (e_i → e_j) ← DELETE(Q)
 4           for k ∈ DIFF(COLUMN(i), COLUMN(j))                 ▷ Case trans 1
 5                 do INSERT*(e_k → e_j, R)
 6                    INSERT(e_k → e_j, Q)
 7           for k ∈ DIFF(ROW(j), ROW(i))                       ▷ Case trans 2
 8                 do INSERT*(e_i → e_k, R)
 9                    INSERT(e_i → e_k, Q)
10           if e_j = λx. e and input(e_i e_w)                  ▷ Case app
11              then if (x → e_w) ∉ R
12                         then INSERT*(x → e_w, R)
13                              INSERT(x → e_w, Q)
14                   if (e_i e_w → e) ∉ R
15                      then INSERT*(e_i e_w → e, R)
16                           INSERT(e_i e_w → e, Q)
17  return R
```

Fig. 4: Subcubic algorithm

COLUMN, and ROW will only be executed once per edge. Hence the total time complexity is dominated by $O(n^3/\log n)$.

Space: Pre-computing the table of most-significant bits requires $O(2^{\log n}) = O(n)$ space. For each term of the source program we require a COLUMN and ROW set. Each such set requires $O(n/\log n)$ space, hence their total space consumption is $O(n^2/\log n)$. The representation and hence the space requirements from Q remain unchanged hence the overall space complexity is $O(n^2)$.

5.2 Conclusion

Since $\vdash e_{\text{start}} \to e_{\text{finish}}$ holds if and only if $(e_{\text{start}} \to e_{\text{finish}}) \in \text{SUBCUBIC-CFA}(e_p)$ holds, we thereby reach the first contribution: a significantly simpler proof of a theorem which should be attributed to Chaudhuri [2008] and Melski and Reps [2000].

Theorem 2. *Control flow analysis of size n and its associated decision problem can be determined in worst case $O(n^3/\log n)$ time on a RAM.*

6 A First Analysis with Reachability

We now incorporate reachability into the analysis algorithm. Extending control flow analysis with reachability, e.g., for *dead-code elimination*, is a known improvement [Ayers, 1993, Palsberg and Schwartzbach, 1995, Biswas, 1997, Heintze and McAllester, 1997a, Gasser et al., 1997, Midtgaard and Jensen, 2008], which

$$\frac{\textsc{scope}(\lambda x.\, e) \in \text{Reach}}{\lambda x.\, e \to \lambda x.\, e}\ (\textsc{refl-lam})$$

$$\frac{e_1 \to \lambda x.\, e \qquad \textsc{scope}(e_1\ e_2) \in \text{Reach}}{\textsc{scope}(e) \in \text{Reach},\ x \to e_2,\ e_1\ e_2 \to e}\ (\textsc{app}) \qquad \frac{e_0 \to e_1 \qquad e_1 \to e_2}{e_0 \to e_2}\ (\textsc{trans})$$

Fig. 5: Inference rules with reachability for the system $\vdash_r$

$$\overbrace{(\lambda x.\ \underbrace{x\,x}_{s_1})\ (\lambda y.\ \underbrace{y\,y}_{s_2})}^{s_0}$$

$$\begin{aligned}
&\textsc{scope}((\lambda x.\, x\,x)\,(\lambda y.\, y\,y)) \\
&\quad = \textsc{scope}(\lambda x.\, x\,x) = \textsc{scope}(\lambda y.\, y\,y) = s_0 \\
&\textsc{scope}(x\,x) = \textsc{scope}(x) = s_1 \\
&\textsc{scope}(y\,y) = \textsc{scope}(y) = s_2
\end{aligned}$$

Fig. 6: Distinct lexical scopes and the SCOPE operator on the Ω-combinator

gives rise to a more precise analysis. In addition to detecting dead code, the extended analysis avoids analysing unreachable code.

We formulate in Fig. 5 an improved analysis as a set of inference rules and write $\vdash_r$ to mean provability in this system. The improved analysis conservatively determines distinct, reachable *lexical scopes* in the input program. The inference rules assume the availability of an operation $\textsc{scope}(e)$ returning a unique representative of an expression e's lexical scope. Each subexpression in the input program can only belong to one lexical scope. Two nested lexical scopes are considered to be different. Subexpressions of a nested lexical scope instead belong to the nested scope. Figure 6 provides an example thereof. Formulated as an equivalence relation, we have $\textsc{scope}(e_0\ e_1) = \textsc{scope}(e_0)$ and $\textsc{scope}(e_0\ e_1) = \textsc{scope}(e_1)$. Distinct lexical scopes can be understood as the induced partitioning of the set of expressions by this equivalence relation.

Given a program e_p we assume the axiom $\textsc{scope}(e_p) \in \text{Reach}$ and repeatedly apply the inference rules. The corresponding decision problem accepts two additional subexpressions e_{start} and e_{finish} as input and determines whether the latter is reachable from the former. The analysis strongly resembles one of Heintze and McAllester [1997a] which we will recall and formally relate to in Sec. 8.

6.1 A First Algorithm with Reachability

We now turn to formulating the inference system algorithmically. The algorithmic formulation naturally falls into two steps: a preprocessing step and a main step. We describe the latter first.

Main step: We formulate in Fig. 7 a worklist algorithm over *edges and lexical scopes*. When deleting an element from the worklist, we perform a case analysis. The basic idea is (again) to reconsider every rule that can potentially 'fire' because of the element addition. Reachable scopes are remembered in the

operation	type	time complexity
SCOPE	$Expr \to Scope$	$O(1)$
NODESINSCOPE	$Scope \to Expr\ list$	$O(1)$
$\in$	$Scope \times Scopeset \to bool$	$O(1)$
INSERT	$Scope \times Scopeset \to unit$	$O(1)$

Table 2: Type and worst-case time complexity of scope operations

set *Reach*. For fast insertion and membership testing, *Reach* can be represented as a bit set.

Preprocessing step: For each expression e, add a link to SCOPE(e). This can be chosen as the outermost expression in a top-down traversal of the input program. In lambda nodes, where a new scope begins, a new scope is passed down to its children. At the same time, we add each expression to a linked list NODESINSCOPE(s) for each scope s. The list represents all expressions of a specific lexical scope. We summarize the resulting operations in Table 2. We assume that the scope operations satisfy the following:

$$\forall s, e : e \in \text{NODESINSCOPE}(s) \implies \text{SCOPE}(e) = s$$
$$\forall e : e \in \text{NODESINSCOPE}(\text{SCOPE}(e))$$

6.2 Complexity

Time: Preprocessing an input term can be done as a linear time traversal, hence the running time of the algorithm is dominated by the main worklist loop. In the outer **while**-loop, every scope and edge is considered only once upon deletion from Q, of which there are at most $O(n)$ and $O(n^2)$, respectively.

For each considered scope, we iterate over its subexpressions, hence each expression is only investigated once in the iteration over NODESINSCOPE. The total time complexity spent in the scope case is dominated by the inner **for**-loop in *app 1*. The inner **for**-loop iterates at most $O(n)$ times, hence the total time complexity of the scope case is $O(n^2)$. The total time spent in the edge case is dominated by the two consecutive (inner) **for**-loops, both of which are worst-case linear. Hence the overall time-complexity is $O(n^3)$.

Space: Maintaining SCOPE takes only $O(n)$ since there is one entry for each subexpression of the source program. Likewise NODESINSCOPE takes only $O(n)$ space since every subexpression belongs to only one scope. Since both edges and scopes can belong to the worklist Q, it takes at most $O(n^2) + O(n) = O(n^2)$ space. R takes $O(n^2)$ space and *Reach* takes $O(n)$ space. Hence the algorithm requires $O(n^2)$ space.

6.3 Correctness

We prove that the new algorithm implements the refined inference rules correctly.

```
CUBIC-REACHABILITY-CFA(e)
 1  Q ← {SCOPE(e)}
 2  R ← ∅
 3  Reach ← {SCOPE(e)}
 4  while Q ≠ ∅
 5        do elem ← DELETE(Q)
 6           case elem of
 7           s : ▷ ***** Scope *****
 8                    for e_i ∈ NODESINSCOPE(s)
 9                         do if e_i = λx. e and (e_i → e_i) ∉ R        ▷ Case refl-lam
10                               then INSERT(e_i → e_i, R)
11                                    INSERT(e_i → e_i, Q)
12                            if e_i = e_1 e_2                          ▷ Case app 1
13                               then for (e_1 → λx. e_0) ∈ R
14                                         do if SCOPE(e_0) ∉ Reach
15                                               then INSERT(SCOPE(e_0), Reach)
16                                                    INSERT(SCOPE(e_0), Q)
17                                            if (x → e_2) ∉ R
18                                               then INSERT(x → e_2, R)
19                                                    INSERT(x → e_2, Q)
20                                            if (e_1 e_2 → e_0) ∉ R
21                                               then INSERT(e_1 e_2 → e_0, R)
22                                                    INSERT(e_1 e_2 → e_0, Q)
23           (e_i → e_j): ▷ ***** Edge *****
24                    for (e_k → e_i) ∈ R such that (e_k → e_j) ∉ R   ▷ Case trans 1
25                         do INSERT(e_k → e_j, R)
26                            INSERT(e_k → e_j, Q)
27                    for (e_j → e_k) ∈ R such that (e_i → e_k) ∉ R   ▷ Case trans 2
28                         do INSERT(e_i → e_k, R)
29                            INSERT(e_i → e_k, Q)
30                    if e_j = λx. e and SCOPE(e_i e_w) ∈ Reach          ▷ Case app 2
31                       then if SCOPE(e) ∉ Reach
32                               then INSERT(SCOPE(e), Reach)
33                                    INSERT(SCOPE(e), Q)
34                            if (x → e_w) ∉ R
35                               then INSERT(x → e_w, R)
36                                    INSERT(x → e_w, Q)
37                            if (e_i e_w → e) ∉ R
38                               then INSERT(e_i e_w → e, R)
39                                    INSERT(e_i e_w → e, Q)
40  return R, Reach
```

Fig. 7: Cubic reachability algorithm

Theorem 3. *Let* $R, \mathit{Reach} = \textsc{Cubic-Reachability-CFA}(e_p)$.

$$(e_0 \to e_1) \in R \iff \vdash_r e_0 \to e_1$$

The proof proceeds similarly to that of Theorem 1. Details are available in Appendix A.

6.4 A First, Subcubic Algorithm with Reachability

We obtain Subcubic-Reachability-CFA, a subcubic algorithm for CFA with reachability, again by changing the representation of R, and its associated operations into a column-row representation implemented with fast sets. We rewrite the **for**-loop conditions with Diff and change all insertions into R into column-row insertions using Insert* instead, substituting lines 24–29 in Fig. 7 with the following lines:

```
24  for k ∈ Diff(Column(i), Column(j))                ▷ Case trans 1
25        do Insert*((e_k → e_j), R)
26           Insert((e_k → e_j), Q)
27  for k ∈ Diff(Row(j), Row(i))                      ▷ Case trans 2
28        do Insert*((e_i → e_k), R)
29           Insert((e_i → e_k), Q)
```

6.5 Complexity

Time: The time complexity analysis proceeds as the previous. Again we have only changed the representation of R and the **for**-loops conditions, hence, as for the first algorithm, the total time complexity is dominated by $O(n^3/\log n)$.

Space: Pre-processing for the scope operations takes $O(n)$ space as does *Reach*. Again, the new representation of R takes $O(n^2/\log n)$ space, hence the overall space complexity is still dominated by Q. A straightforward representation hereof requires $O(n^2)$ space.

6.6 Conclusion

Let $R, \mathit{Reach} = \textsc{Subcubic-Reachability-CFA}(e_p)$. Since $\vdash_r e_{\text{start}} \to e_{\text{finish}}$ holds if and only if $(e_{\text{start}} \to e_{\text{finish}}) \in R$ holds, we thereby reach the second contribution.

Theorem 4. *Control flow analysis with reachability of size* n *and its associated decision problem can be determined in worst case* $O(n^3/\log n)$ *time on a RAM.*

$$\frac{\textsc{scope}(\lambda x.\, e) \in \text{Reach}}{\lambda x.\, e \to \lambda x.\, e,\ \lambda x.\, e \in \text{HasVals}}\ (\textsc{refl-lam})$$

$$\frac{e_1 \to \lambda x.\, e \qquad \textsc{scope}(e_1\ e_2) \in \text{Reach} \qquad e_2 \in \text{HasVals}}{\textsc{scope}(e) \in \text{Reach},\ x \to e_2,\ e_1\ e_2 \to e}\ (\textsc{app})$$

$$\frac{e_0 \to e_1 \qquad e_1 \to e_2}{e_0 \to e_2}\ (\textsc{trans}) \qquad \frac{e_0 \to e_1 \qquad e_1 \in \text{HasVals}}{e_0 \in \text{HasVals}}\ (\textsc{has-val})$$

Fig. 8: Refined inference rules with reachability for the system $\vdash_{r'}$

7 A More Precise Analysis with Reachability

The analysis in Fig. 5 will not analyse the body of a function if it is never called. However, the analysis will mark the body of a lambda as reachable, even if, e.g., the operands at all its call sites diverge. By augmenting the analysis further such a property can be obtained. Such an augmentation is the topic of this section.

In order to analyse function bodies only when values can flow to the operands at its call sites, we track expressions that can have a value flow to them, or expressed in graph-based terminology: *expressions that can reach a value*. As "can reach a value" is not a static property, we will compute it dynamically by augmenting the analysis with a set of expressions 'HasVals'. In the resulting analysis in Fig. 8, we have added the condition $e_2 \in \text{HasVals}$ to the APP rule and added the conclusion $\lambda x.\, e \in \text{HasVals}$ to the REFL-LAM rule. We have furthermore added a rule HAS-VAL to propagate the property between expressions. Given a program e_p we again assume the axiom $\textsc{scope}(e_p) \in \text{Reach}$ and repeatedly apply the inference rules. We write $\vdash_{r'}$ to mean provability in this inference system. The corresponding decision problem again accepts two additional subexpressions e_{start} and e_{finish} as input and determines whether the latter is reachable from the former.

7.1 A More Precise Algorithm with Reachability

Since we added a new (dynamic) condition, we extend the corresponding worklist algorithm in Fig. 9 to work over edges, scopes, and "expressions with values". When deleting an element from worklist Q, we perform a case analysis over the three. The algorithm furthermore computes a set of expressions *HasVals* modelling HasVals from the inference rules. We assume this set is represented, e.g., as a bit set, with constant time insertion and membership testing. Again the idea underlying the algorithm is to test all rules that can potentially apply because of a new addition.

In order to retain a reasonable size of the resulting algorithm we have factored tests and insertions into three helper procedures CHECKED-INSERT-EDGE, CHECKED-INSERT-SCOPE, and CHECKED-INSERT-EXPR given in Fig. 10.

```
REFINED-CUBIC-REACHABILITY-CFA(e)
 1  Q ← {SCOPE(e)}
 2  R ← ∅
 3  Reach ← {SCOPE(e)}
 4  HasVals ← ∅
 5  while Q ≠ ∅
 6        do elem ← DELETE(Q)
 7           case elem of
 8           e_j : ▷ ***** Expression *****
 9                 for (e_i → e_j) ∈ R                          ▷ Case has-val 1
10                     do CHECKED-INSERT-EXPR(e_i)
11                 if SCOPE(e_i e_j) ∈ Reach                    ▷ Case app 1
12                    then for (e_i → λx. e_0) ∈ R
13                               do CHECKED-INSERT-SCOPE(SCOPE(e_0))
14                                  CHECKED-INSERT-EDGE(x → e_j)
15                                  CHECKED-INSERT-EDGE(e_i e_j → e_0)
16           s : ▷ ***** Scope *****
17                 for e_i ∈ NODESINSCOPE(s)
18                     do if e_i = λx. e                        ▷ Case refl-lam
19                           then CHECKED-INSERT-EDGE(e_i → e_i)
20                                CHECKED-INSERT-EXPR(e_i)
21                        if e_i = e_1 e_2 and e_2 ∈ HasVals     ▷ Case app 2
22                           then for (e_1 → λx. e_0) ∈ R
23                                      do CHECKED-INSERT-SCOPE(SCOPE(e_0))
24                                         CHECKED-INSERT-EDGE(x → e_2)
25                                         CHECKED-INSERT-EDGE(e_1 e_2 → e_0)
26           (e_i → e_j): ▷ ***** Edge *****
27                 if e_j ∈ HasVals                             ▷ Case has-val 2
28                    then CHECKED-INSERT-EXPR(e_i)
29                 for (e_k → e_i) ∈ R such that (e_k → e_j) ∉ R  ▷ Case trans 1
30                     do INSERT(e_k → e_j, R)
31                        INSERT(e_k → e_j, Q)
32                 for (e_j → e_k) ∈ R such that (e_i → e_k) ∉ R  ▷ Case trans 2
33                     do INSERT(e_i → e_k, R)
34                        INSERT(e_i → e_k, Q)
35                 if e_j = λx. e and SCOPE(e_i e_w) ∈ Reach and e_w ∈ HasVals
36                    then CHECKED-INSERT-SCOPE(SCOPE(e))       ▷ Case app 3
37                         CHECKED-INSERT-EDGE(x → e_w)
38                         CHECKED-INSERT-EDGE(e_i e_w → e)
39  return R, Reach
```

Fig. 9: More precise, cubic time reachability algorithm

```
CHECKED-INSERT-EDGE(e_i → e_j)
1  if (e_i → e_j) ∉ R
2     then INSERT(e_i → e_j, R)
3          INSERT(e_i → e_j, Q)

CHECKED-INSERT-SCOPE(s)
1  if s ∉ Reach
2     then INSERT(s, Reach)
3          INSERT(s, Q)

CHECKED-INSERT-EXPR(e)
1  if e ∉ HasVals
2     then INSERT(e, HasVals)
3          INSERT(e, Q)
```

Fig. 10: Factored helper procedures

7.2 Complexity

Time: Each of the three procedures in Fig. 10 are constant time as they do a constant time test and two constant time insertions. We now analyse each of the three cases separately.

Expression case: all expressions are considered only once. The two inner loops are both linear, hence the total time spent in the expression case take $O(n^2)$ time. Scope case: the first **for**-loop will iterate in total at most n times since each expression will only belong to one scope. Each such iteration uses $O(n)$ time, hence the total time spent is $O(n^2)$. Edge case: There are $O(n^2)$ possible edges, for which we may run two linear **for**-loops, hence the total time spent is $O(n^3)$.

Space: The space complexity of *HasVals* is $O(n)$. The space requirements of the remaining algorithm remains the same, hence the total space complexity is still $O(n^2)$.

7.3 Correctness

Theorem 5. *Let* $R, \mathit{Reach} = \text{REFINED-CUBIC-REACHABILITY-CFA}(e_p)$.

$$(e_0 \to e_1) \in R \iff \vdash_{r'} e_0 \to e_1$$

The proof proceeds similarly to that of Theorems 1 and 3. Details are available in Appendix B.

7.4 A More Precise, Subcubic Algorithm with Reachability

A subcubic algorithm, REFINED-SUBCUBIC-REACHABILITY-CFA, for the refined analysis is obtained by again changing the representation of R and its associated operations into a column-row representation implemented with fast sets. Again we rewrite the **for**-loop conditions with DIFF and change all insertions into R into column-row insertions, substituting lines 29–34 in Fig. 9 with the following lines:

```
29  for k ∈ Diff(Column(i), Column(j))                    ▷ Case trans 1
30      do Insert*((e_k → e_j), R)
31         Insert((e_k → e_j), Q)
32  for k ∈ Diff(Row(j), Row(i))                          ▷ Case trans 2
33      do Insert*((e_i → e_k), R)
34         Insert((e_i → e_k), Q)
```

7.5 Complexity

Time: The time complexity of the operations on R and hence the majority of the refined algorithm remain unaffected by the change. As before the algorithm therefore has $O(n^3/\log n)$ time complexity.

Space: Compared to the previous subcubic reachability algorithm we have only added *HasVals* taking at most $O(n)$ space. The addition therefore does not affect the overall $O(n^2)$ space complexity.

7.6 Conclusion

Let $R, \mathit{Reach} = \textsc{Refined-Subcubic-Reachability-CFA}(e_p)$. Since the reachability $\vdash_{r'} e_{\text{start}} \to e_{\text{finish}}$ holds if and only if $(e_{\text{start}} \to e_{\text{finish}}) \in R$ holds, we thereby reach the third contribution.

Theorem 6. *Control flow analysis with refined reachability of size n and its associated decision problem can be determined in worst case $O(n^3/\log n)$ time on a RAM.*

8 Equivalence

In this section, we relate the three analyses presented thus far and prove each successive analysis is a refinement of its predecessors. We then relate the refined reachability analysis of Fig. 8 to a known flow analysis from the literature, namely the set-based abstraction (SBA) analysis of Heintze and McAllester [1997a]. By doing so, we are able to appeal to the soundness result of set-based analysis to prove soundness for the analyses presented here. Moreover, we show decision problems for set-based analysis can be decided in subcubic time.

Theorem 7. $\vdash_r e \to e' \Longrightarrow \vdash e \to e'$

Proof. We prove the statement,

$$\vdash_r e \to e' \Longrightarrow \vdash e \to e' \ \wedge\ \vdash_r \textsc{scope}(e) \in \text{Reach} \Longrightarrow \mathit{input}(e),$$

by mutual induction on the derivation of $\vdash_r e \to e'$ and $\vdash_r \textsc{scope}(e) \in \text{Reach}$.

In the base case, $\vdash_r \textsc{scope}(e_p) \in \text{Reach}$ implies $\mathit{input}(e_p)$ by the definition of *input*. Each inductive case holds by appealing to the induction hypothesis, the definition of *input*, and applying the corresponding $\vdash$ rule. □

$$\frac{\textsc{eval}(e_1\ e_2)}{\textsc{eval}(e_1),\ \textsc{eval}(e_2)}\ (\textsc{beta'}) \qquad \frac{\textsc{eval}(e_1\ e_2) \qquad e_1 \to \lambda x.\ e \qquad e_2 \to v}{x \to v,\ e_1\ e_2 \to e}\ (\textsc{beta})$$

$$\frac{\textsc{eval}(\lambda x.\ e)}{\lambda x.\ e \to \lambda x.\ e}\ (\textsc{ident}) \qquad \frac{e_1 \to e_2}{\textsc{eval}(e_2)}\ (\textsc{call}) \qquad \frac{e_1 \to e_2 \qquad e_2 \to v}{e_1 \to v}\ (\textsc{return})$$

Fig. 11: Heintze and McAllester [1997a] SBA inference rules with reachability

Theorem 8. $\vdash_{r'} e \to e' \Longrightarrow \vdash_r e \to e'$

Proof. We prove the statement,

$$\vdash_{r'} e \to e' \Longrightarrow \vdash_r e \to e' \ \wedge\ \vdash_{r'} \textsc{scope}(e) \in \text{Reach} \Longrightarrow \vdash_r \textsc{scope}(e) \in \text{Reach},$$

by mutual induction on the derivation of $\vdash_{r'} e \to e'$ and $\vdash_{r'} \textsc{scope}(e) \in \text{Reach}$. The base case holds trivially. Each inductive case holds by appealing to the induction hypothesis and applying the corresponding $\vdash_r$ rule. □

We now turn to the SBA analysis of Heintze and McAllester [1997a]. In Fig. 11 we recall the SBA inference system. Given a program e_p we assume the axiom $\textsc{eval}(e_p)$ and repeatedly apply the inference rules. We write $\vdash_{sba}$ to mean provability in this system. First, we prove that if one expression reaches another under SBA, it does so under our refined reachability analysis, and therefore under all of the analyses considered. This is sufficient to establish the soundness of all the analyses presented here since SBA is sound.

We first prove a needed lemma showing that if an expression reaches a value, it is in HasVals.

Lemma 1. $\vdash_{r'} e \to v \Longrightarrow \vdash_{r'} e \in \text{HasVals}$

Proof. By induction on the derivation of $\vdash_{r'} e \to v$. The REFL-LAM case is trivial. The TRANS case follows by the induction hypothesis and HAS-VAL. In the APP case, we have $\vdash_{r'} x \in \text{HasVals}$ by HAS-VAL. If $\vdash_{r'} e_1\ e_2 \to e$ where e is a value, then REFL-LAM applies and $\vdash_{r'} e_1\ e_2 \in \text{HasVals}$ follows by HAS-VAL. □

Theorem 9. $\vdash_{sba} e \to e' \Longrightarrow \vdash_{r'} e \to e'$

Proof. We prove the statement,

$$\vdash_{sba} e \to e' \Longrightarrow \vdash_{r'} e \to e' \ \wedge\ \vdash_{sba} \textsc{eval}(e) \Longrightarrow \vdash_{r'} \textsc{scope}(e) \in \text{Reach},$$

by mutual induction on the derivation of $\vdash_{sba} e \to e'$ and $\vdash_{sba} \textsc{eval}(e)$.

In the base case, $\textsc{eval}(e_p)$ implies $\vdash_{r'} \textsc{scope}(e_p) \in \text{Reach}$ by definition. Case BETA': by the induction hypothesis, $\vdash_{r'} \textsc{scope}(e_1\ e_2) \in \text{Reach}$, and the case holds by the definition of SCOPE. Case IDENT: by the induction hypothesis, $\vdash_{r'} \textsc{scope}(\lambda x.\ e) \in \text{Reach}$, so the case holds by REFL-LAM. Case BETA: by the induction hypothesis, $\vdash_{r'} \textsc{scope}(e_1\ e_2) \in \text{Reach}$, $e_1 \to \lambda x.\ e$, $e_2 \to v$. By Lemma 1,

$\vdash_{r'} e_2 \in \text{HasVals}$, so $\vdash_{r'} x \to e_2, e_1\ e_2 \to e$ follows by APP. Finally, $\vdash_{r'} x \to v$ follows by TRANS, and the case holds. Case CALL: by the induction hypothesis, $\vdash_{r'} e_1 \to e_2$. Since $\vdash_{r'} e \to e'$ implies $\vdash_{r'} \text{SCOPE}(e') \in \text{Reach}$, which is proved by induction on the derivation of $\vdash_{r'} e \to e'$, the case holds. Case RETURN: by the induction hypothesis, $\vdash_{r'} e_1 \to e_2, e_2 \to v$, so the case holds by TRANS. □

The alert reader may have noticed that the conclusion $x \to v$ in the BETA rule in Fig. 11 differs slightly from the corresponding conclusion $x \to e_2$ in the APP rule of Fig. 8. As a consequence all inclusions (reachability arrows) of the refined algorithm cannot be proved by Heintze and McAllester's analysis. We instead prove the following, weaker theorem, relating inclusions between expressions and syntactic values. The proof is available in Appendix C.

Theorem 10. $\vdash_{r'} e \to v \Longrightarrow \vdash_{sba} e \to v$

Putting the above minor difference aside, the final refined analysis $\vdash_{r'}$ can be understood as a two-step version of Heintze and McAllester's analysis in which transitive implications of BETA' are pre-computed exhaustively, thereby enabling the marking of all subexpressions in a lexical scope as reachable as soon as the outermost expression of the same scope becomes reachable.

Heintze and McAllester's analysis may be formulated as a decision problem termed *set-based reachability*, by accepting two additional subexpressions e_{start} and v_{finish} and determining whether the latter syntactic value is reachable from the former expression.

Theorem 11. *Set-based abstraction analysis with refined reachability of size n and its associated decision problem can be determined in worst case $O(n^3/\log n)$ time on a RAM.*

9 Conclusion and Perspectives

Fast sets require an architecture with constant-time bit operations on $\Theta(\log n)$ size words. Compared to the textbook RAM model, this may seem a substantial requirement. Consider the undergraduate binary search algorithm in a sorted array of size n. This requires computing the average of two numbers in the range $[0 \ldots n-1]$. Such numbers require $O(\log n)$ bits, hence we are relying on constant time addition of $O(\log n)$ bit numbers. In this light, requiring constant time for similar operations such as complement or bitwise and, seems less demanding.

Heintze and McAllester [1997b] define a *subcubic algorithm* as a RAM algorithm that operates in time $O(n^k)$ with $k < 3$. By this definition, neither the Rytter [1985] algorithm, nor those presented here would be considered subcubic. However, by examining the limit of $n^3/(n^3/\log n) = \log n$ it is clear that $n^3/\log n = o(n^3)$. Hence we use *subcubic time* to mean *less than cubic time*, as does Chaudhuri [2008], and other researchers in the algorithms research literature [Zwick, 2006, Chan, 2007, Han, 2008]. Instead the term *truly subcubic* is sometimes used to refer to algorithms of time complexity $O(n^k)$ with $k < 3$ [Zwick, 2006, Chan, 2007].

CFA is complete for both 2NPDA [Heintze and McAllester, 1997b] and PTIME [Melski and Reps, 2000, Van Horn and Mairson, 2007], yet it is not clear what the relationship is between these classes. In part, this is because the 2NPDA inclusion proof is sensitive to representation choices and problem formulations. Heintze and McAllester [1997b] use an encoding of lambda-terms that requires a non-standard bit string labelling scheme in which identical subterms have the same labels. They remark that without this labelling scheme, the problem "appears not to be in 2NPDA." Moreover, the notions of reduction for 2NPDA-hard and PTIME-hard relies on different computational models. For a problem to be 2NPDA-hard, all problems in the class must be reducible in $O(nR(\log n))$ time on a RAM, where R is a polynomial [Heintze and McAllester, 1997b], whereas to be PTIME-hard, all problems in the class must be reducible using a $O(\log n)$ space work-tape on a TM.

Historically, researchers have approached the cubic bottleneck from different angles. One such approach is to add input assumptions, e.g., accepting bounded-type programs [Heintze and McAllester, 1997c]. Another approach has been to explore faster but less precise analyses, e.g., almost linear time *simple closure analysis* based on equalities rather than inclusions [Henglein, 1992, Bondorf and Jørgensen, 1993].

We have given the first direct subcubic algorithms for performing both the standard notion of flow analysis and two more precise variants incorporating reachability. In retrospect, the present results were achievable a decade ago. Rather than using 2NPDA-completeness as an argument for dismissing further investigation of algorithmic improvements, we view the above results as an invitation to re-investigate them.

Acknowledgements The authors wish to thank Kristoffer Arnsfelt Hansen for comments. The first author is supported by the Carlsberg Foundation.

Bibliography

A. V. Aho, J. E. Hopcroft, and J. D. Ullman. Time and tape complexity of pushdown automaton languages. *Information and Control*, 13(3):186–206, 1968.

A. E. Ayers. *Abstract Analysis and Optimization of Scheme*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, Sept. 1993.

A. E. Ayers. Efficient closure analysis with reachability. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, and A. Rauzy, editors, *Actes WSA'92 Workshop on Static Analysis*, Bigre, pages 126–134, Bordeaux, France, Sept. 1992. Atelier Irisa, IRISA, Campus de Beaulieu.

J. Basch, S. Khanna, and R. Motwani. On diameter verification and boolean matrix multiplication. Technical report, Stanford University, Stanford, CA, USA, 1995.

S. K. Biswas. A demand-driven set-based analysis. In N. D. Jones, editor, *Proceedings of the 24th Annual ACM Symposium on Principles of Programming Languages*, pages 372–385, Paris, France, Jan. 1997.

A. Bondorf and J. Jørgensen. Efficient analyses for realistic off-line partial evaluation. *Journal of Functional Programming*, 3(3):315–346, July 1993.

T. M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 590–598, San Diego, California, 2007.

S. Chaudhuri. Subcubic algorithms for recursive state machines. In G. C. Necula and P. Wadler, editors, *Proceedings of the 35th Annual ACM Symposium on Principles of Programming Languages*, pages 159–169, San Francisco, California, Jan. 2008.

K. L. S. Gasser, F. Nielson, and H. R. Nielson. Systematic realisation of control flow analyses for CML. In Tofte, pages 38–51.

Y. Han. A note of an $O(n^3/\log n)$ time algorithm for all pairs shortest paths. *Information Processing Letters*, 105(3):114–116, 2008.

N. Heintze. Set-based program analysis of ML programs. In C. L. Talcott, editor, *Proceedings of the 1994 ACM Conference on Lisp and Functional Programming*, LISP Pointers, Vol. VII, No. 3, pages 306–317, Orlando, Florida, June 1994.

N. Heintze and D. McAllester. On the complexity of set-based analysis. In Tofte, pages 150–163.

N. Heintze and D. McAllester. On the cubic bottleneck in subtyping and flow analysis. In G. Winskel, editor, *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science (LICS '97)*, pages 342–351, Warsaw, Poland, June 1997b.

N. Heintze and D. McAllester. Linear-time subtransitive control flow analysis. In R. K. Cytron, editor, *Proceedings of the ACM SIGPLAN 1997 Conference on Programming Languages Design and Implementation*, pages 261–272, Las Vegas, Nevada, June 1997c.

F. Henglein. Simple closure analysis. Technical Report Semantics Report D-193, DIKU, Computer Science Department, University of Copenhagen, 1992.

J. C. Martin. *Introduction to Languages and the Theory of Computation*. McGraw-Hill Higher Education, 1997.

D. A. McAllester. On the complexity analysis of static analyses. *J. ACM*, 49 (4):512–537, 2002.

D. Melski and T. Reps. Interconvertibility of a class of set constraints and context-free-language reachability. *Theoretical Computer Science*, 248(1-2): 29–98, 2000.

J. Midtgaard and T. Jensen. A calculational approach to control-flow analysis by abstract interpretation. In M. Alpuente and G. Vidal, editors, *Static Analysis, 15th International Symposium, SAS 2008*, volume 5079 of *Lecture Notes in Computer Science*, pages 347–362, Valencia, Spain, July 2008. Springer-Verlag.

R. Neal. The computational complexity of taxonomic inference. Available at ftp://ftp.cs.utoronto.ca/pub/radford/taxc.ps, Dec. 1989.

F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999.

J. Palsberg and M. I. Schwartzbach. Safety analysis versus type inference. *Information and Computation*, 118(1):128–141, 1995.

W. Rytter. Fast recognition of pushdown automaton and context-free languages. *Information and Control*, 67(1-3):12–22, 1985.

O. Shivers. *Control-Flow Analysis of Higher-Order Languages or Taming Lambda*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1991. Technical Report CMU-CS-91-145.

M. Tofte, editor. *Proceedings of the Second ACM SIGPLAN International Conference on Functional Programming*, Amsterdam, The Netherlands, June 1997.

D. Van Horn and H. G. Mairson. Relating complexity and precision in control flow analysis. In N. Ramsey, editor, *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming (ICFP'07)*, pages 85–96, Freiburg, Germany, Oct. 2007.

V. Vassilevska. Efficient algorithms for clique problems. *Information Processing Letters*, 109(4):254–257, 2009.

U. Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. *Algorithmica*, 46(2):181–192, 2006.

A Proof of Theorem 3

Proof. Let e_p be given.

1. We prove the statement

$$((e_0 \to e_1) \in R \implies \vdash_r e_0 \to e_1) \ \wedge \ (s \in \mathit{Reach} \implies \vdash_r s \in \text{Reach}),$$

from which the left-to-right direction of Theorem 3 follows. We reason about the algorithm using the following loop invariant:

$$(e_0 \to e_1) \in R \implies \vdash_r e_0 \to e_1 \ \wedge \tag{a}$$
$$(e_0 \to e_1) \in Q \implies (e_0 \to e_1) \in R \ \wedge \tag{b}$$
$$s \in \mathit{Reach} \implies \vdash_r s \in \text{Reach} \ \wedge \tag{c}$$
$$s \in Q \implies \vdash_r s \in \text{Reach} \tag{d}$$

Invariant holds at while-loop entry: Since R is empty and Q contains no edges, only scopes, (a) and (b) trivially hold. Since we assume the axiom $\vdash_r \textsc{scope}(e_p) \in \text{Reach}$ and that all sub-expressions of e_p are part of the input program, (c) and (d) hold as well.

Invariant is preserved by while-loop iteration: Assume the invariant holds; we prove it holds after each of the two cases.
Case s: From our scope operation assumptions, we have $\textsc{scope}(e_i) = s$. From (d), it follows that $\vdash_r s \in \text{Reach}$. There are now two possibilities:
(1) $e_i = \lambda x.\, e$: Parts (c) and (d) remain true as their conditions are untouched. We argue the invariant is preserved for the potential new edge additions to R and Q. By REFL-LAM, $\vdash_r e_i \to e_i$ and hence (a) holds. The edge $(e_i \to e_i)$ belongs to both R and Q after the insertion, so (b) holds.
(2) $e_i = e_1\ e_2$: If $(e_1 \to \lambda x.\, e_0) \in R$, it follows from (a) that $\vdash_r e_1 \to \lambda x.\, e_0$. For each of the three conditionals, we argue the invariant still holds for the potential new additions to R and Q.
(3a) Parts (a) and (b) remain true as their conditions are untouched. Now $\vdash_r \textsc{scope}(e_1\ e_2) \in \text{Reach}$ follows from the above, so by APP, (c) and (d) hold.
(3b) Parts (c) and (d) remain true as their conditions are untouched. Now $\vdash_r x \to e_2$ follows by APP, so (a) holds. The edge $(x \to e_2)$ belongs to both R and Q after the insertion, so (b) holds.
(3c) Parts (c) and (d) remain true as their conditions are untouched. Now $\vdash_r e_1\ e_2 \to e_0$ again follows by APP, so (a) holds. The edge $(e_1\ e_2 \to e_0)$ belongs to both R and Q after the insertion, so (b) holds.

Case $(e_i \to e_j)$: From (b), we have $\vdash_r e_i \to e_j$. For each new addition, we again argue the invariant is preserved.
Case *trans 1*: Parts (c) and (d) remain true as their conditions are untouched. If $(e_k \to e_i) \in R$, from (a) we have $\vdash_r e_k \to e_i$. Hence, $\vdash_r e_k \to e_j$ by TRANS,

so (a) holds. The edge $(e_k \to e_j)$ belongs to both R and Q after the insertion, so (b) holds.
Case *trans 2*: Parts (c) and (d) remain true as their conditions are untouched. If $(e_j \to e_k) \in R$, from (a) we have $\vdash_r e_j \to e_k$. Hence, $\vdash_r e_i \to e_k$ by TRANS, so (a) holds. The edge $(e_i \to e_k)$ belongs to both R and Q after the insertion, so (b) holds.
Case *app 2*: If $e_j = \lambda x.\, e$ and $\text{SCOPE}(e_i\ e_w) \in \mathit{Reach}$, it follows from (c) that $\vdash_r \text{SCOPE}(e_i\ e_w) \in \text{Reach}$.
Case (*app-2-a*): Parts (a) and (b) remain true as their conditions are untouched. Parts (c) and (d) follow immediately from the above by APP.
Case (*app-2-b*): Parts (c) and (d) remain true as their conditions are untouched. Part (a) follows immediately from the above by APP. The edge $(x \to e_w)$ belongs to both R and Q after the insertion, so (b) holds.
Case (*app-2-c*): Parts (c) and (d) remain true as their conditions are untouched. Part (a) follows immediately from the above by APP. The edge $(e_i\ e_w \to e)$ belongs to both R and Q after the insertion, so (b) holds.

2. The right-to-left direction of the proof relies on three observations: (1) insertions into R and Q are always simultaneous. (2) insertions into *Reach* and Q are always simultaneous. (3) R and *Reach* grow increasingly over time, since edges and scopes are never deleted from them.
The run of CUBIC-REACHABILITY-CFA consists of a sequence of modifications to Q, R, and *Reach*, which we model as a sequence of triples

$$(Q_0, R_0, \mathit{Reach}_0) \ldots (Q_n, R_n, \mathit{Reach}_n),$$

such that (a) two consecutive Q's differ only by one inserted or deleted edge or scope node, (b) two consecutive R's may differ only by an inserted edge, and (c) two consecutive *Reach*'s may differ only by an inserted scope node. Since edges are never deleted from R and scope nodes are never deleted from *Reach*, we prove the statement

$$\begin{aligned} \vdash_r e_0 \to e_1 &\implies \text{exists a least } i, \text{ s.t. } (e_0 \to e_1) \in R_i \ \wedge\ (e_0 \to e_1) \in Q_i \\ \vdash_r s \in \text{Reach} &\implies \text{exists a least } i, \text{ s.t. } s \in \mathit{Reach}_i \ \wedge\ s \in Q_i, \end{aligned}$$

from which the right-to-left direction of Theorem 3 follows. The proof proceeds by mutual structural induction on the inference trees for $\vdash_r e_0 \to e_1$ and $\vdash_r s \in \text{Reach}$.
For derivations using the axiom $\vdash_r \text{SCOPE}(e_p) \in \text{Reach}$, clearly there exists Reach_0 and Q_0 such that the above holds from the initialization in lines 1 and 3. Now assume the statement holds for all structurally smaller derivation trees. There are three remaining cases.
Case REFL-LAM: If $\vdash_r e_0 \to e_1$ uses the REFL-LAM rule, we have $e_0 = e_1 = \lambda x.\, e$, and by the induction hypothesis, there exists a $\text{SCOPE}(\lambda x.\, e) \in \mathit{Reach}_i$ and $\text{SCOPE}(\lambda x.\, e) \in Q_i$. Hence when $\text{SCOPE}(\lambda x.\, e)$ is later deleted from Q, $\lambda x.\, e \in \text{NODESINSCOPE}(\text{SCOPE}(\lambda x.\, e))$ holds by the scope operation assumptions. If $(\lambda x.\, e \to \lambda x.\, e) \in R$, it was inserted simultaneously in Q thereby proving existence. Otherwise, we add it to both thereby proving existence.

Case APP: If $\vdash_r s \in \text{Reach}$ uses the APP rule, we have that $s = \text{SCOPE}(e)$ and there exists $e_1\ e_2$ such that $\vdash_r e_1 \to \lambda x.\ e$ and $\vdash_r \text{SCOPE}(e_1\ e_2) \in \text{Reach}$. Hence by the induction hypothesis, there exists a least i such that $(e_1 \to \lambda x.\ e) \in R_i$ and $(e_1 \to \lambda x.\ e) \in Q_i$, and there exists a least j such that $\text{SCOPE}(e_1\ e_2) \in \mathit{Reach}_j$ and $\text{SCOPE}(e_1\ e_2) \in Q_j$. The situation $i = j$ is impossible as insertions always happen pairwise: two scope insertions or two edge insertions. If $i < j$ when $\text{SCOPE}(e_1\ e_2)$ is deleted from Q, we have $e_1\ e_2 \in \text{NODESINSCOPE}(\text{SCOPE}(e_1\ e_2))$ by the scope operation assumptions. At this point in time, $(e_1 \to \lambda x.\ e) \in R$. Hence if $\text{SCOPE}(e) \notin \mathit{Reach}$, we add it to both Reach and Q thereby proving existence. If already $\text{SCOPE}(e) \in \mathit{Reach}$, it was earlier inserted simultaneously into Q as well, thereby proving existence. If $i > j$ when $(e_1 \to \lambda x.\ e)$ is deleted from Q, we have $\text{SCOPE}(e_1\ e_2) \in \mathit{Reach}$. Hence if $\text{SCOPE}(e) \notin \mathit{Reach}$, we add it to both Reach and Q thereby proving existence. If already $\text{SCOPE}(e) \in \mathit{Reach}$, it was earlier inserted simultaneously into Q as well, thereby proving existence.

If $\vdash_r e_0 \to e_1$ uses the APP rule, there are two cases to consider: one for each of the remaining conclusions. If $e_0 \to e_1 = x \to e_2'$, there exists $e_1'\ e_2'$ such that $e_0 = x$, $e_1 = e_2'$, and by the induction hypothesis there exists a least i such that $(e_1' \to \lambda x.\ e)$ belongs to both R_i and Q_i. Also by the induction hypothesis, there exists a least j such that $\text{SCOPE}(e_1'\ e_2')$ belongs to both Reach_j and Q_j. Again the situation $i = j$ is impossible. If $i < j$ when $\text{SCOPE}(e_1'\ e_2')$ is deleted from Q, $e_1'\ e_2' \in \text{NODESINSCOPE}(\text{SCOPE}(e_1'\ e_2'))$ by the scope operation assumptions. At this point in time, $(e_1' \to \lambda x.\ e) \in R$. Hence if $(x \to e_2') \notin R$, we add it to both R and Q thereby proving existence. If already $(x \to e_2') \in R$, it was earlier inserted simultaneously into Q as well, thereby proving existence. If $i > j$ when $(e_1' \to \lambda x.\ e)$ is deleted from Q, we have $\text{SCOPE}(e_1'\ e_2') \in \mathit{Reach}$. Hence if $(x \to e_2') \notin R$, we add it to both R and Q thereby proving existence. If already $(x \to e_2') \in R$, it was earlier inserted simultaneously into Q as well, thereby proving existence.

If $e_0 \to e_1 = e_1'\ e_2' \to e$, we have $e_0 = e_1'\ e_2'$ and $e_1 = e$, and by the induction hypothesis there exists a least i such that $(e_1' \to \lambda x.\ e)$ belongs to both R_i and Q_i. Also by the induction hypothesis, there exists a least j such that $\text{SCOPE}(e_1'\ e_2')$ belongs to both Reach_j and Q_j. Again the situation $i = j$ is impossible. If $i < j$ when $\text{SCOPE}(e_1'\ e_2')$ is deleted from Q, we have $e_1'\ e_2' \in \text{NODESINSCOPE}(\text{SCOPE}(e_1'\ e_2'))$ by the scope operation assumptions. At this point in time, $(e_1' \to \lambda x.\ e) \in R$. Hence if $(e_1'\ e_2' \to e) \notin R$, we add it to both R and Q thereby proving existence. If already $(e_1'\ e_2' \to e) \in R$, it was earlier inserted simultaneously into Q as well, thereby proving existence. If $i > j$ when $(e_1' \to \lambda x.\ e)$ is deleted from Q, we have $\text{SCOPE}(e_1'\ e_2') \in \mathit{Reach}$. Hence if $(e_1'\ e_2' \to e) \notin R$, we add it to both R and Q thereby proving existence. If already $(e_1'\ e_2' \to e) \in R$, it was earlier inserted simultaneously into Q as well, thereby proving existence.

Case TRANS: If $\vdash_r e_0 \to e_1$ uses the TRANS rule, there exists e such that $\vdash_r e_0 \to e, e \to e_1$. By the induction hypothesis, there exists a least i such that $(e_0 \to e)$ belongs to both R_i and Q_i, and there exists a least j such that $(e \to e_1)$ belongs to both R_j and Q_j. If $i = j$, we have $e_0 = e_1 = e$ hence

existence follows from the induction hypothesis. If $i < j$ when $(e \to e_1)$ is deleted from Q, we have $(e_0 \to e) \in R$. If $(e_0 \to e_1) \notin R$, we insert it in both R and Q in *trans 1*, thereby proving existence. If already $(e_0 \to e_1) \in R$, it was earlier inserted simultaneously into both R and Q thereby proving existence. If $i > j$ when $(e_0 \to e)$ is deleted from Q, we have $(e \to e_1) \in R$. If $(e_0 \to e_1) \notin R$, we insert it in both R and Q in *trans 2*, thereby proving existence. If already $(e_0 \to e_1) \in R$, it was earlier inserted simultaneously into both R and Q thereby proving existence. □

B Proof of Theorem 5

Proof. Let e_p be given.

1. We prove the statement

$$\begin{aligned}(e_0 \to e_1) \in R &\implies \vdash_{r'} e_0 \to e_1 \ \wedge \\ s \in \mathit{Reach} &\implies \vdash_{r'} s \in \text{Reach} \ \wedge \\ e_0 \in \mathit{HasVals} &\implies \vdash_{r'} \textsc{scope}(e_0) \in \text{Reach},\end{aligned}$$

from which the left-to-right direction of Theorem 5 follows. We reason about the algorithm using the following loop invariant:

$$\begin{aligned}(e_0 \to e_1) \in R &\implies \vdash_{r'} e_0 \to e_1 \ \wedge && \text{(a)} \\ (e_0 \to e_1) \in Q &\implies (e_0 \to e_1) \in R \ \wedge && \text{(b)} \\ s \in \mathit{Reach} &\implies \vdash_{r'} s \in \text{Reach} \ \wedge && \text{(c)} \\ s \in Q &\implies \vdash_{r'} s \in \text{Reach} \ \wedge && \text{(d)} \\ e_0 \in \mathit{HasVals} &\implies \vdash_{r'} e_0 \in \text{HasVals} \ \wedge && \text{(e)} \\ e_0 \in Q &\implies \vdash_{r'} e_0 \in \text{HasVals} && \text{(f)}\end{aligned}$$

Invariant holds at while-loop entry: Since R is empty and there are no edges in Q, (a) and (b) are trivially satisfied. Since we assume the axiom $\vdash_{r'} \textsc{scope}(e_p) \in \text{Reach}$, and all sub-expressions of e_p are part of the input program, (c) and (d) hold as well. As $\mathit{HasVals}$ is empty, (e) also holds. Finally, there are no expressions in Q, hence (f) holds.

Invariant is preserved by while-loop iteration: Assume the invariant holds. We prove it holds after the insertions in each of the three cases.

Case e_j: From (f), it follows that $\vdash_{r'} e_j \in \text{HasVals}$. For each $(e_i \to e_j) \in R$ in case *has-val 1*, it follows from (a) that $\vdash_{r'} e_i \to e_j$. Hence by HAS-VALS, we have $\vdash_{r'} e_i \in \text{HasVals}$, thereby fulfilling (e) and (f) for each new insertion into $\mathit{HasVals}$ and Q. The conditions for (a),(b),(c), and (d) are untouched and hence still hold.
In case *app 1*, assume that $\textsc{scope}(e_i\ e_j) \in \mathit{Reach}$ and that $(e_i \to \lambda x.\ e_0) \in R$. From (c) and (a), it follows $\vdash_{r'} \textsc{scope}(e_i\ e_j) \in \text{Reach}, e_i \to \lambda x.\ e_0$. By APP

$\vdash_{r'} \textsc{scope}(e_0) \in \text{Reach}, x \to e_j, e_i\ e_j \to e_0$ now follow. After the first insertion, (c) and (d) still hold. As the conditions for (a),(b),(e), and (f) are untouched, they still hold. After the second insertion (a) and (b) still hold. As the conditions for (c),(d),(e), and (f) are untouched, they still hold. Finally, after the third insertion, (a) and (b) still hold. As the conditions for (c),(d),(e), and (f) are untouched, they also still hold.

Case s: From our scope operation assumptions, we have $\textsc{scope}(e_i) = s$. From (d), $\vdash_{r'} s \in \text{Reach}$ follows. There are now two possibilities:
(1) $e_i = \lambda x.\, e$: Parts (c) and (d) remain true as their conditions are untouched and we argue the invariant is preserved for the potential new edge and expression additions to R, *HasVals*, and Q. By REFL-LAM, $\vdash_{r'} e_i \to e_i$ holds and hence (a) follows immediately from the above. The edge $(e_i \to e_i)$ belongs to both R and Q after the insertion, so (b) holds. Furthermore, $\vdash_{r'} e_i \in \text{HasVals}$ holds by REFL-LAM, hence (e) follow immediately from the above. As e_i belongs to both *HasVals* and Q after the insertion, (f) also holds.
(2) $e_i = e_1\ e_2$ and $e_2 \in \mathit{HasVals}$: If $(e_1 \to \lambda x.\, e_0) \in R$, it follows from (a) that $\vdash_{r'} e_1 \to \lambda x.\, e_0$, and from (e) that $\vdash_{r'} e_2 \in \text{HasVals}$. For each of the three conditionals, we argue that the invariant still holds for the potential new additions to R and Q.
(3a) Parts (a),(b),(e), and (f) remain true as their conditions are untouched. $\vdash_{r'} \textsc{scope}(e_1\ e_2) \in \text{Reach}$ follows from the above, hence (c) and (d) hold by APP.
(3b) Parts (c),(d),(e), and (f) remain true as their conditions are untouched. Now $\vdash_{r'} x \to e_2$ follows from the above by APP and hence (a) holds. The edge $(x \to e_2)$ belongs to both R and Q after the insertion, so (b) holds.
(3c) Parts (c),(d),(e), and (f) remain true as their conditions are untouched. Now $\vdash_{r'} e_1\ e_2 \to e_0$ again follows by APP, so (a) holds. The edge $(e_1\ e_2 \to e_0)$ belongs to both R and Q after the insertion, so (b) holds.

Case $(e_i \to e_j)$: From (b) we have that $\vdash_{r'} e_i \to e_j$. For each new addition, we again argue the invariant is preserved.
Case *has-val 1*: If $e_j \in \mathit{HasVals}$, from (e) we have $\vdash_{r'} e_j \in \text{HasVals}$. By HAS-VAL, $\vdash_{r'} e_i \in \text{HasVals}$, so (e) and (f) hold.
Case *trans 1*: If $(e_k \to e_i) \in R$, from (a) we have that $\vdash_{r'} e_k \to e_i$. By TRANS, $\vdash_{r'} e_k \to e_j$, so (a) holds. The edge $(e_k \to e_j)$ belongs to both R and Q after the insertion, so (b) holds. Parts (c),(d),(e), and (f) remain true as their conditions are untouched.
Case *trans 2*: If $(e_j \to e_k) \in R$, from (a) we have that $\vdash_{r'} e_j \to e_k$. By TRANS, $\vdash_{r'} e_i \to e_k$, so (a) holds. The edge $(e_i \to e_k)$ belongs to both R and Q after the insertion, so (b) holds. Parts (c),(d),(e), and (f) remain true as their conditions are untouched.
Case *app 3*: If $e_j = \lambda x.\, e$, $\textsc{scope}(e_i\ e_w) \in \mathit{Reach}$, and $e_w \in \mathit{HasVals}$, $\vdash_{r'} \textsc{scope}(e_i\ e_w) \in \text{Reach}$ follows from (c), and $\vdash_{r'} e_w \in \text{HasVals}$ follows from (e).

Case (*app-3-a*): Parts (a),(b),(e), and (f) remain true as their conditions are untouched. Parts (c) and (d) follow from the above by APP.
Case (*app-3-b*): Parts (c),(d),(e), and (f) remain true as their conditions are untouched. Part (a) follows from the above by APP. $(x \to e_w)$ belongs to both R and Q after the insertion and hence (b) holds.
Case (*app-3-c*): Parts (c),(d),(e), and (f) remain true as their conditions are untouched. Part (a) follows from the above by APP. The edge $(e_i\ e_w \to e)$ belongs to both R and Q after the insertion, so (b) holds.

2. The right-to-left direction of the proof relies on four observations: (1) insertions into R and Q are always simultaneous. (2) insertions into *Reach* and Q are always simultaneous. (3) insertions into *HasVals* and Q are always simultaneous. (4) R, *Reach*, and *HasVals* grow increasingly over time since edges, scopes, and expressions are never deleted from them.
The run of REFINED-CUBIC-REACHABILITY-CFA consists of a sequence of modifications to Q, R, *Reach*, and *HasVals*, which we model as a sequence of quadruples

$$(Q_0, R_0, \mathit{Reach}_0, \mathit{HasVals}_0) \dots (Q_n, R_n, \mathit{Reach}_n, \mathit{HasVals}_n),$$

such that (a) two consecutive Q's differ only by one inserted or deleted edge or scope node, (b) two consecutive R's may differ only by an inserted edge, (c) two consecutive *Reach*'s may differ only by an inserted scope node, and (d) two consecutive *HasVals*'s may differ only by an inserted expression.
Since edges are never deleted from R, and scope nodes are never deleted from *Reach*, we prove the statement

$$\begin{aligned}
\vdash_{r'} e_0 \to e_1 &\implies \text{exists a least } i, \text{ s.t. } (e_0 \to e_1) \in R_i \ \wedge\ (e_0 \to e_1) \in Q_i \\
\vdash_{r'} s \in \text{Reach} &\implies \text{exists a least } i, \text{ s.t. } s \in \mathit{Reach}_i \ \wedge\ s \in Q_i \\
\vdash_{r'} e \in \text{HasVals} &\implies \text{exists a least } i, \text{ s.t. } e \in \mathit{HasVals}_i \ \wedge\ e \in Q_i,
\end{aligned}$$

from which the right-to-left direction of Theorem 5 follows. The proof proceeds by mutual structural induction on the inference trees for $\vdash_{r'} e_0 \to e_1$, $\vdash_{r'} s \in \text{Reach}$, and $\vdash_{r'} e \in \mathit{HasVals}$.
For derivations using the axiom $\vdash_{r'} \text{SCOPE}(e_p) \in \text{Reach}$, clearly there exists Reach_0 and Q_0 such that the above holds from the initialization in lines 1 and 3. Now assume the statement holds for all structurally smaller derivation trees. There are four remaining cases.
Case REFL-LAM: By the induction hypothesis, there exists a least i, such that $\text{SCOPE}(\lambda x.\ e) \in \mathit{Reach}_i$ and $\text{SCOPE}(\lambda x.\ e) \in Q_i$. Hence when $\text{SCOPE}(\lambda x.\ e)$ is later deleted from Q, by the scope operation assumptions we have $\lambda x.\ e \in \text{NODESINSCOPE}(\text{SCOPE}(\lambda x.\ e))$. For the conclusion $\vdash_{r'} e_0 \to e_1$, if $(\lambda x.\ e \to \lambda x.\ e) \in R$, it was inserted simultaneously in Q, thereby proving existence. Otherwise, we add it to both thereby proving existence. For the conclusion $\vdash_{r'} \lambda x.\ e \in \text{HasVals}$, if $\lambda x.\ e \in \mathit{HasVals}$, it was inserted simultaneously in Q, thereby proving existence. Otherwise, we add it to both, thereby proving existence.

Case APP: If $\vdash_{r'} s \in \text{Reach}$ uses the APP rule, we have that $s = \text{SCOPE}(e)$ and there exists $e_1\ e_2$ such that $\vdash_{r'} e_1 \to \lambda x.\, e$, $\text{SCOPE}(e_1\ e_2) \in \text{Reach}$, and $e_2 \in \text{HasVals}$. Hence by the induction hypothesis, there exists a least i such that $(e_1 \to \lambda x.\, e) \in R_i$ and $(e_1 \to \lambda x.\, e) \in Q_i$, and there exists a least j such that $\text{SCOPE}(e_1\ e_2) \in \mathit{Reach}_j$ and $\text{SCOPE}(e_1\ e_2) \in Q_j$, and there exists a least k such that $e_2 \in \mathit{HasVals}_k$ and $e_2 \in Q_k$. Situations $i = j$, $i = k$, and $j = k$ are impossible as insertions always happen pairwise: two scope insertions, two edge insertions, or two expression insertions. If j is greatest ($j > i$ and $j > k$) when $\text{SCOPE}(e_1\ e_2)$ is deleted from Q, we have $e_1\ e_2 \in \text{NODESINSCOPE}(\text{SCOPE}(e_1\ e_2))$ by the scope operation assumptions. At this point, $(e_1 \to \lambda x.\, e) \in R$ and $e_2 \in \mathit{HasVals}_k$. Hence if $\text{SCOPE}(e) \notin \mathit{Reach}$, we add it to both Reach and Q, thereby proving existence. If already $\text{SCOPE}(e) \in \mathit{Reach}$, it was earlier inserted simultaneously into Q as well, thereby proving existence. If i is greatest ($i > j$ and $i > k$) when $(e_1 \to \lambda x.\, e)$ is deleted from Q, we have $\text{SCOPE}(e_1\ e_2) \in \mathit{Reach}$ and $e_2 \in \mathit{HasVals}_k$. Hence if $\text{SCOPE}(e) \notin \mathit{Reach}$, we add it to both Reach and Q, thereby proving existence. If already $\text{SCOPE}(e) \in \mathit{Reach}$, it was earlier inserted simultaneously into Q as well, thereby proving existence. If k is greatest ($k > i$ and $k > j$) when e_2 is deleted from Q, we have $(e_1 \to \lambda x.\, e) \in R$ and $\text{SCOPE}(e_1\ e_2) \in \mathit{Reach}$. Hence if $\text{SCOPE}(e) \notin \mathit{Reach}$, we add it to both Reach and Q, thereby proving existence. If already $\text{SCOPE}(e) \in \mathit{Reach}$, it was earlier inserted simultaneously into Q as well, thereby proving existence.

If $\vdash_{r'} e_0 \to e_1$ uses the APP rule, there are two cases to consider: one for each of the remaining conclusions. If $e_0 \to e_1 = x \to e_2'$, there exists $e_1'\ e_2'$ such that $e_0 = x$, $e_1 = e_2'$, and by the induction hypothesis, there exists a least i such that $(e_1' \to \lambda x.\, e)$ belongs to both R_i and Q_i. Also by the induction hypothesis, there exists a least j such that $\text{SCOPE}(e_1'\ e_2')$ belongs to both Reach_j and Q_j. Finally, by the induction hypothesis, there exists a least k such that e_2' belongs to both $\mathit{HasVals}_k$ and Q_k.

Again situations $i = j$, $i = k$, and $j = k$ are impossible. If j is greatest ($j > i$ and $j > k$) when $\text{SCOPE}(e_1'\ e_2')$ is deleted from Q, we have $e_1'\ e_2' \in \text{NODESINSCOPE}(\text{SCOPE}(e_1'\ e_2'))$ by the scope operation assumptions. At this point, $(e_1' \to \lambda x.\, e) \in R$ and $e_2' \in \mathit{HasVals}$. Hence if $(x \to e_2') \notin R$, we add it to both R and Q, thereby proving existence. If already $(x \to e_2') \in R$, it was earlier inserted simultaneously into Q as well, thereby proving existence. If i is greatest ($i > j$ and $i > k$) when $(e_1' \to \lambda x.\, e)$ is deleted from Q, we have $\text{SCOPE}(e_1'\ e_2') \in \mathit{Reach}$ and $e_2' \in \mathit{HasVals}$. Hence if $(x \to e_2') \notin R$, we add it to both R and Q, thereby proving existence. If already $(x \to e_2') \in R$, it was earlier inserted simultaneously into Q as well, thereby proving existence. If k is greatest ($k > i$ and $k > j$) when e_2' is deleted from Q, we have $(e_1' \to \lambda x.\, e) \in R$ and $\text{SCOPE}(e_1'\ e_2') \in \mathit{Reach}$. Hence if $(x \to e_2') \notin R$, we add it to both R and Q, thereby proving existence. If already $(x \to e_2') \in R$, it was earlier inserted simultaneously into Q as well, thereby proving existence. If $e_0 \to e_1 = e_1'\ e_2' \to e$, we have $e_0 = e_1'\ e_2'$ and $e_1 = e$, and by the induction hypothesis, there exists a least i such that $(e_1' \to \lambda x.\, e)$ belongs to both R_i and Q_i. Also by the induction hypothesis, there exists a least j such

that SCOPE($e_1'\ e_2'$) belongs to both $Reach_j$ and Q_j. Finally, by the induction hypothesis, there exists a least k such that e_2' belongs to both $HasVals_k$ and Q_k.

Again situations $i = j$, $i = k$, and $j = k$ are impossible. If j is greatest ($j > i$ and $j > k$) when SCOPE($e_1'\ e_2'$) is deleted from Q, we have $e_1'\ e_2' \in$ NODESINSCOPE(SCOPE($e_1'\ e_2'$)) by the scope operation assumptions. At this point, $(e_1' \to \lambda x.\ e) \in R$ and $e_2' \in HasVals$. Hence if $(e_1'\ e_2' \to e) \notin R$, we add it to both R and Q, thereby proving existence. If already $(e_1'\ e_2' \to e) \in R$, it was earlier inserted simultaneously into Q as well, thereby proving existence. If i is greatest ($i > j$ and $i > k$) when $(e_1' \to \lambda x.\ e)$ is deleted from Q, we have SCOPE($e_1'\ e_2'$) $\in Reach$ and $e_2' \in HasVals$. Hence if $(e_1'\ e_2' \to e) \notin R$, we add it to both R and Q thereby proving existence. If already $(e_1'\ e_2' \to e) \in R$, it was earlier inserted simultaneously into Q as well, thereby proving existence. If k is greatest ($k > i$ and $k > j$) when e_2' is deleted from Q, we have $(e_1' \to \lambda x.\ e) \in R$ and SCOPE($e_1'\ e_2'$) $\in Reach$. Hence if $(e_1'\ e_2' \to e) \notin R$, we add it to both R and Q, thereby proving existence. If already $(e_1'\ e_2' \to e) \in R$, it was earlier inserted simultaneously into Q as well, thereby proving existence.

Case TRANS: If $\vdash_{r'} e_0 \to e_1$ uses the TRANS rule, there exists e such that $\vdash_{r'} e_0 \to e, e \to e_1$. By the induction hypothesis, there exists a least i such that $(e_0 \to e)$ belongs to both R_i and Q_i, and there exists a least j such that $(e \to e_1)$ belongs to both R_j and Q_j. If $i = j$, we have that $e_0 = e_1 = e$, hence existence follows from the induction hypothesis. If $i < j$ when $(e \to e_1)$ is deleted from Q, we have that $(e_0 \to e) \in R$. If $(e_0 \to e_1) \notin R$, we insert it in both R and Q in *trans 1*, thereby proving existence. If already $(e_0 \to e_1) \in R$, it was earlier inserted simultaneously into both R and Q, thereby proving existence. If $i > j$ when $(e_0 \to e)$ is deleted from Q, we have that $(e \to e_1) \in R$. If $(e_0 \to e_1) \notin R$, we insert it in both R and Q in *trans 2*, thereby proving existence. If already $(e_0 \to e_1) \in R$, it was earlier inserted simultaneously into both R and Q, thereby proving existence.

Case HAS-VAL: If $e \in$ HasVals uses rule HAS-VAL, there exists e_1 such that $\vdash_{r'} e \to e_1, e_1 \in$ HasVals. By the induction hypothesis, there exists a least i such that $(e \to e_1)$ belongs to both R_i and Q_i, and there exists a least j such that e_1 belongs to both $HasVals_j$ and Q_j. Again the case $i = j$ is impossible. If $i < j$ when e_1 is deleted from Q, $(e \to e_1) \in R$. If $e \notin HasVals$, we insert it in both $HasVals$ and Q in *has-val 1*, thereby proving existence. If already $e \in HasVals$, it was earlier inserted simultaneously into both $HasVals$ and Q, thereby proving existence. If $i > j$ when $(e \to e_1)$ is deleted from Q, $e_1 \in HasVals$. If $e \notin HasVals$, we insert it in both $HasVals$ and Q in *has-val 2*, thereby proving existence. If already $e \in HasVals$, it was earlier inserted simultaneously into both $HasVals$ and Q, thereby proving existence.

□

C Proof of Theorem 10

Let $\vdash_{sba'}$ denote provability in an inference system as given in Fig. 11, but without CALL and with the following in place of BETA:

$$\frac{\textsc{eval}(e_1\ e_2) \qquad e_1 \to \lambda x.\ e \qquad e_2 \to v}{\textsc{eval}(e),\ x \to v,\ e_1\ e_2 \to e}\ (\textsc{beta}^\star)$$

Lemma 2. $\vdash_{sba'} e' \to e \implies \vdash_{sba'} \textsc{eval}(e)$.

Proof. By induction on the derivation of $\vdash_{sba'} e' \to e$. Case IDENT is trivial. Case RETURN follows by the induction hypothesis. Case BETA* follows by the induction hypothesis for $\vdash_{sba'} \textsc{eval}(v)$ and by assumption for $\vdash_{sba'} \textsc{eval}(e)$. □

The resulting system $\vdash_{sba'}$ is equivalent to the existing SBA system $\vdash_{sba}$.

Lemma 3. $\vdash_{sba'} \textsc{eval}(e) \iff \vdash_{sba} \textsc{eval}(e) \ \wedge\ \vdash_{sba'} e_1 \to e_2 \iff \vdash_{sba} e_1 \to e_2$

Proof. In the left-to-right direction, the only case to consider is BETA*: from the induction hypothesis, BETA applies, leaving $\textsc{eval}(e)$ to be proved, which follows from CALL. In the right-to-left direction, the only case to consider is CALL, which follows by the induction hypothesis and Lemma 2. □

First we realize that any inference tree in $\vdash_{r'}$ using rule TRANS can be rotated into an equivalent inference tree in which the first condition of TRANS is not itself an instance of TRANS. Intuitively this can be understood as letting the reachability relation $\to$ associate from the right.

Lemma 4. *A proof of* $\vdash_{r'} e \to e'$, $\vdash_{r'} \textsc{scope}(e) \in \text{Reach}$, *and* $\vdash_{r'} e \in \text{HasVals}$ *can be transformed into an equivalent proof not containing two consecutive instances of* TRANS *on a left diagonal:*

$$\dfrac{\dfrac{\dfrac{T_0}{e_0 \to e_0'} \qquad \dfrac{T_1}{e_0' \to e_1}}{e_0 \to e_1}\ (\textsc{trans}) \qquad \dfrac{T_2}{e_1 \to e_2}}{e_0 \to e_2}\ (\textsc{trans})$$

Proof. The proof proceeds by mutual structural induction on the height of the involved inference trees. All cases except TRANS follow immediately from the induction hypothesis. For TRANS, there are two cases to consider. If the first (left) condition is not an instance of TRANS, the conclusion follows immediately from the induction hypothesis. If the first (left) condition is an instance of TRANS (as above), we first construct an equivalent inference tree:

$$\dfrac{\dfrac{T_0}{e_0 \to e_0'} \qquad \dfrac{\dfrac{T_1}{e_0' \to e_1} \qquad \dfrac{T_2}{e_1 \to e_2}}{e_0' \to e_2}\ (\textsc{trans})}{e_0 \to e_2}\ (\textsc{trans})$$

By the induction hypothesis, we can transform the proofs of $e_0 \to e_0'$ and $e_0' \to e_2$ accordingly. □

Finally, we need a simple helper lemma equating values reachable from values.

Lemma 5. $\vdash_{sba'} v \to v' \implies v = v'$

Proof. There are only two rules from which conclusions of the above form can arise. The case IDENT is immediate. The case RETURN follows from the induction hypothesis and transitivity of equality. □

We now prove the following lemma, from which Theorem 10 follows.

Lemma 6.

$$\begin{aligned} \vdash_{r'} e \to v &\implies \vdash_{sba'} e \to v \ \wedge \\ \vdash_{r'} \text{SCOPE}(e) \in \text{Reach} &\implies \vdash_{sba'} \text{EVAL}(e) \ \wedge \\ \vdash_{r'} e \in \text{HasVals} &\implies \exists\, v : \vdash_{sba'} e \to v \end{aligned}$$

Proof. By Lemma 4, we can assume that no sub-trees contain two consecutive instances of TRANS on a left diagonal. The proof now continues by simultaneous structural induction on the inference trees.
Base case: $\vdash_{sba'} \text{EVAL}(e_p)$ follows immediately from the corresponding $\vdash_{sba'}$ program axiom.
Case REFL-LAM: If $\vdash_{r'} \lambda x.\, e \to \lambda x.\, e$ by REFL-LAM, $\vdash_{sba'} \text{EVAL}(\lambda x.\, e)$ follows from the induction hypothesis, hence by IDENT $\vdash_{sba'} \lambda x.\, e \to \lambda x.\, e$ If $\vdash_{r'} e \in \text{HasVals}$ by REFL-LAM, there exists $\lambda x.\, e$ such that $\vdash_{sba'} \lambda x.\, e \to \lambda x.\, e$ again by IDENT.
Case TRANS: We case-analyse the first (left) condition:

- Sub-case REFL-LAM: By the induction hypothesis on the second (right) condition, the requested result follows immediately: $\vdash_{sba'} \lambda x.\, e \to v$ (without applying RETURN).
- Sub-case APP: From the conditions of the APP rule, we have $\vdash_{r'} e_1 \to \lambda x.\, e$, $\text{SCOPE}(e_1\ e_2) \in \text{Reach}$, and $e_2 \in \text{HasVals}$. We consider each possible conclusion separately:
  - Sub-case x: If $\vdash_{r'} x \to e_2$, from the second (right) condition of TRANS we have $\vdash_{r'} e_2 \to v$. By two applications of the induction hypothesis, we have $\vdash_{sba'} e_1 \to \lambda x.\, e$ and $\vdash_{sba'} \text{EVAL}(e_1\ e_2)$. By the induction hypothesis, we furthermore have $\vdash_{sba'} e_2 \to v$. By BETA*, it now follows that $\vdash_{sba'} x \to v$.
  - Sub-case $e_1\ e_2$: If $\vdash_{r'} e_1\ e_2 \to e$ from the second (right) condition of TRANS we have $\vdash_{r'} e \to v$. By three applications of the induction hypothesis, we have $\vdash_{sba'} e_1 \to \lambda x.\, e$, $\vdash_{sba'} \text{EVAL}(e_1\ e_2)$, and that there exists v' such that $\vdash_{sba'} e_2 \to v'$. Hence by BETA* we have $\vdash_{sba'} e_1\ e_2 \to e$. By another application of the induction hypothesis we have $\vdash_{sba'} e \to v$. Hence $\vdash_{sba'} e_1\ e_2 \to v$ by RETURN.

Case APP: From the conditions, we have $\vdash_{r'} e_1 \to \lambda x.\, e$, $\text{SCOPE}(e_1\ e_2) \in \text{Reach}$, and $e_2 \in \text{HasVals}$. By three applications of the induction hypothesis, we have $\vdash_{sba'} e_1 \to \lambda x.\, e$, $\vdash_{sba'} \text{EVAL}(e_1\ e_2)$, and there exists v' such that $\vdash_{sba'} e_2 \to v'$. If $\vdash_{r'} \text{SCOPE}(e) \in \text{Reach}$, we get $\vdash_{sba'} \text{EVAL}(e)$ by BETA*. If $\vdash_{r'} e \to v$ by APP, there are two sub-cases:

– Case x: Hence $\vdash_{r'} x \to e_2$ and $e_2 = v$. By BETA*, we have $\vdash_{sba'} x \to v'$. By the helper lemma, $\vdash_{sba'} v \to v'$ implies $v = v'$, and hence $\vdash_{sba'} x \to v$ as requested.
– Case $e_1\ e_2$: Hence $\vdash_{r'} e_1\ e_2 \to e$ and $e = v$. By BETA*, we have $\vdash_{sba'} e_1\ e_2 \to e$.

□

RECENT RESEARCH REPORTS

#124 Torben Braüner. Hybrid logic and its proof-theory. 318 pp. March 2009, Roskilde University, Roskilde, Denmark.

This thesis has been accepted by Roskilde University for public defence in fulllment of the requirements for the Danish degree doctor scientiarum (dr.scient.). The public defence will take place at Roskilde University in Biografen/The Cinema, Building 41, on Thursday April 23rd 2009, at 1300.

#123 Magnus Nilsson. *Arbejdet i hjemmeplejen: Et etnometodologisk studie af IT-støttet samarbejde i den københavnske hjemmepleje*. PhD thesis, Roskilde, Denmark, August 2008.

#122 Jørgen Villadsen and Henning Christiansen, editors. *Proceedings of the 5th International Workshop on Constraints and Language Processing (CSLP 2008)*, Roskilde, Denmark, May 2008.

#121 Ben Schouten and Niels Christian Juul, editors. *Proceedings of the First European Workshop on Biometrics and Identity Management (BIOID 2008)*, Roskilde, Denmark, April 2008.

#120 Peter Danholt. *Interacting Bodies: Posthuman Enactments of the Problem of Diabetes Relating Science, Technology and Society-studies, User-Centered Design and Diabetes Practices*. PhD thesis, Roskilde, Denmark, February 2008.

#119 Alexandre Alapetite. *On speech recognition during anaesthesia*. PhD thesis, Roskilde, Denmark, November 2007.

#118 Paolo Bouquet, editor. *CONTEXT'07 Doctoral Consortium Proceedings*, Roskilde, Denmark, October 2007.

#117 Kim S. Henriksen. *A Logic Programming Based Approach to Applying Abstract Interpretation to Embedded Software*. PhD thesis, Roskilde, Denmark, October 2007.

#116 Marco Baroni, Alessandro Lenci, and Magnus Sahlgren, editors. *Proceedings of the 2007 Workshop on Contextual Information in Semantic Space Models: Beyond Words and Documents*, Roskilde, Denmark, August 2007.

#115 Paolo Bouquet, Jérôme Euzenat, Chiara Ghidini, Deborah L. McGuinness, Valeria de Paiva, Luciano Serafini, Pavel Shvaiko, and Holger Wache, editors. *Proceedings of the 2007 workshop on Contexts and Ontologies Representation and Reasoning (C&O:RR-2007)*, Roskilde, Denmark, August 2007.

#114 Bich-Liên Doan, Joemon Jose, and Massimo Melucci, editors. *Proceedings of the 2nd International Workshop on Context-Based Information Retrieval*, Roskilde, Denmark, August 2007.