Control-Flow Analysis in Cubic Time

Flemming Nielson[1] and Helmut Seidl[2]

[1] Computer Science Department, Aarhus University (Bldg. 540), Ny Munkegade, DK-8000 Aarhus C, Denmark, fn@daimi.au.dk
[2] FB IV – Informatik, Universität Trier, D-54286 Trier, Germany, seidl@uni-trier.de

Abstract. It is well-known that context-independent control flow analysis can be performed in cubic time for functional and object-oriented languages. Yet recent applications of control flow analysis to calculi of computation (like the π-calculus and the ambient calculus) have reported considerably higher complexities. In this paper we introduce two general techniques, the use of *Horn clauses with sharing* and the use of *tiling of Horn clauses*, for reducing the worst-case complexity of analyses. Applying these techniques to the π-calculus and the ambient calculus we reduce the complexity from $\mathcal{O}(n^5)$ to $\mathcal{O}(n^3)$ in both cases.

Keywords: Program analysis, Horn clauses with sharing, tiling of Horn clauses, π-calculus, ambient calculus, 0-CFA.

1 Introduction

Program analyses often can be separated into two phases. In the first phase, the program to be analyzed is translated into a suitable constraint system describing *safe* information about the program, and where the unknowns represent the desired information. In the second phase, a solution for the unknowns (typically the least) is produced by an appropriate constraint solver.

Accordingly, there are also two common sources of inefficiency for a program analyzer constructed in this way. Clearly, efficiency cannot be hoped for if already the presentation of the system itself is extremely large. Therefore, a constraint formalism should be chosen which is expressive enough to represent the generated constraints succinctly. Even so, efficiency might be lost when the constraint formalism is "stronger than necessary", meaning that the solving procedure for the selected class of constraints incurs a large though otherwise unnecessary overhead.

As an example, consider the analysis of the π-calculus as presented in [2]. For a program of size n this analysis succeeds in generating a constraint system of size $\mathcal{O}(n^3)$ using set inclusion constraints. Thus from a practical point of view, even if a cubic worst case behavior is inevitable, such an extensive constraint system is unsatisfactory as it prohibits simpler programs to be analyzed faster. Actually, the presentation of the analysis used in [2] is even less likely to scale to larger programs as the generated constraint system, when fed into an off-the-shelf solver, would consume $\mathcal{O}(n^5)$ steps of solving time in the worst case. In a similar way, the analysis of the ambient calculus as presented in [10] generates a constraint system of size $\mathcal{O}(n^4)$ using set membership constraints and the same $\mathcal{O}(n^5)$ worst case constraint solving time.

The goal of this paper is to improve on these methods. As a general framework within which these problems can be addressed, we propose the concept of *Horn clauses with sharing* (HCS's for short). While being much more succinct than classical Horn clauses, they still admit rather efficient constraint solving techniques. We demonstrate the usefulness of this concept in several ways. By using Horn clauses with sharing instead of ordinary ones we are able to generate linear size constraint systems for analyses of the π-calculus and for the ambient calculus as they have been published in the literature. By using state-of-the-art solvers for Horn clauses with sharing, we bring down the complexity of the 0-CFA analysis of the π-calculus as presented in [2] from $\mathcal{O}(n^5)$ to $\mathcal{O}(n^3)$. It turns out that these methods still do not suffice to get a similar improvement for the ambient calculus. Therefore, we develop *tiling* as a source-to-source transformation of Horn clauses; indeed, tiling may be of independent interest also for other applications. In our application it allows us to reduce the complexity for the ambient calculus from $\mathcal{O}(n^5)$ to $\mathcal{O}(n^3)$ as well. In practical terms, $\mathcal{O}(n^3)$ is likely to be sufficiently good that it will be possible to analyse medium-sized programs whereas lower complexities are called for to analyse large programs.

2 Horn Clauses with Sharing

There are several formalisms around in which to specify constraints for program analyses; two of the more widely used ones are conditional set constraints (see e.g. [1]) and Horn clauses (see e.g. [7]). We base our work on Horn clauses in order to build on the techniques for complexity estimation presented in [7].

A system of *Horn clauses* (abbreviated: HC's) usually is a set of implications where the conclusion is a single relation, and the antecedent is a conjunction of relations. In order to facilitate the introduction of sharing we shall represent a system of Horn clauses as a formula derived by the nonterminal clause' in the grammar below:

$$\begin{array}{lcl} \mathsf{pre}' & ::= & R\,(x_1, \cdots, x_k) \quad | \quad \mathsf{pre}'_1 \wedge \mathsf{pre}'_2 \\ \mathsf{clause}' & ::= & R\,(x_1, \cdots, x_k) \quad | \quad \mathbf{1} \quad | \quad \mathsf{clause}'_1 \wedge \mathsf{clause}'_2 \\ & & \quad | \quad \mathsf{pre}' \Rightarrow R\,(x_1, \cdots, x_k) \quad | \quad \forall x : \mathsf{clause}' \end{array}$$

Here we assume that we are given a fixed countable set $X = \{x, x_1, \cdots\}$ of variables and a finite ranked alphabet $\mathcal{R} = \{R, R_1, \cdots\}$ of relation symbols of predicates. In this notation we are explicit about the otherwise implicit universal quantification in Horn clauses, and $\mathbf{1}$ is the always true clause.

To obtain *Horn clauses with sharing* (abbreviated: HCS's) we extend this formalism by allowing

- disjunctions and existential quantification in pre-conditions, and
- conjunctions of clauses in conclusions.

Disjunctions have been added merely for technical convenience. Existential quantification in pre-conditions, however, allows us to limit the scopes of variables, whereas conjunctions of clauses in conclusions allow us to merge multiple conclusions without the technical inconvenience of introducing auxiliary predicates. The set of HCS's are defined by the nonterminal clause in the grammar below:

$$
\begin{array}{lcl}
\mathsf{pre} & ::= & R\,(x_1, \cdots, x_k) \quad | \quad \mathsf{pre}_1 \wedge \mathsf{pre}_2 \quad | \quad \mathsf{pre}_1 \vee \mathsf{pre}_2 \quad | \quad \exists x : \mathsf{pre} \\
\mathsf{clause} & ::= & R\,(x_1, \cdots, x_k) \quad | \quad \mathbf{1} \quad | \quad \mathsf{clause}_1 \wedge \mathsf{clause}_2 \\
 & & \quad | \quad \mathsf{pre} \Rightarrow \mathsf{clause} \quad | \quad \forall x : \mathsf{clause}
\end{array}
$$

Occurrences of $R(\cdots)$ in pre-conditions are also called *queries*, whereas the others are called *assertions* of predicate R.

Given a universe $\mathcal{U}$ of atomic values (or atoms) together with interpretations ρ and σ for relation symbols and free variables, respectively, we define the satisfaction relations $(\rho, \sigma) \models \mathsf{pre}$ and $(\rho, \sigma) \models \mathsf{clause}$ as follows (where t is a pre-condition or clause):

$$
\begin{array}{lll}
(\rho, \sigma) \models \mathbf{1} & \text{iff} & \text{true} \\
(\rho, \sigma) \models R\,(x_1, \cdots, x_k) & \text{iff} & (\sigma\, x_1, \cdots, \sigma\, x_k) \in \rho\, R \\
(\rho, \sigma) \models \mathsf{pre}_1 \vee \mathsf{pre}_2 & \text{iff} & (\rho, \sigma) \models \mathsf{pre}_1 \quad \text{or} \quad (\rho, \sigma) \models \mathsf{pre}_2 \\
(\rho, \sigma) \models t_1 \wedge t_2 & \text{iff} & (\rho, \sigma) \models t_1 \quad \text{and} \quad (\rho, \sigma) \models t_2 \\
(\rho, \sigma) \models \mathsf{pre} \Rightarrow \mathsf{clause} & \text{iff} & (\rho, \sigma) \models \mathsf{clause} \quad \text{whenever} \quad (\rho, \sigma) \models \mathsf{pre} \\
(\rho, \sigma) \models \forall x : \mathsf{clause} & \text{iff} & (\rho, \sigma \oplus \{x \mapsto a\}) \models \mathsf{clause} \qquad \text{for all } a \in \mathcal{U} \\
(\rho, \sigma) \models \exists x : \mathsf{pre} & \text{iff} & (\rho, \sigma \oplus \{x \mapsto a\}) \models \mathsf{pre} \qquad \text{for some } a \in \mathcal{U}
\end{array}
$$

In the sequel, we will view the free variables occurring in a HCS (or HC) as *constant* symbols or *atoms* from the *finite* universe $\mathcal{U}$. Thus, given an interpretation σ of the constant symbols in clause, we call an interpretation ρ of the relational symbols $\mathcal{R}$ a *solution* provided $(\rho, \sigma) \models \mathsf{clause}$.

Let $\Delta_\sigma = \{\rho \mid (\rho, \sigma) \models \mathsf{clause}\}$ denote the set of solutions of clause (given a fixed σ). Then Δ_σ is partially ordered in the natural way by the componentwise ordering $\sqsubseteq$. It is standard (see e.g. [9, Subsection 3.2.3]) that Δ_σ is a Moore family, i.e. closed under greatest lower bounds $\sqcap$, and we conclude that Δ_σ has a least element which we call the *least solution* of clause. It is well-known [7, 5] that in the case of HC's this solution can be computed efficiently. The following result establishes a similar result[1] for HCS's.

Proposition 1. *Given an interpretation of the constant symbols, the least solution of a HCS formula $c_1 \wedge \cdots \wedge c_m$ can be computed in time*

$$\mathcal{O}(N^{r_1} \cdot n_1 + \cdots + N^{r_m} \cdot n_m)$$

where N is the number of atoms in $\mathcal{U}$, n_i is the size of c_i, and r_i is the maximal nesting depth of quantifiers in c_i.

Proof. See Appendix A for an algorithm whose worst case complexity is as stated. We are currently experimenting with a solver having the same worst case complexity but a potentially much lower best case complexity.

[1] All our complexity bounds refer to the RAM model with a uniform cost measure.

3 The Virtues of Sharing

We now show how sharing facilitates developing a cubic time algorithm for performing control flow analysis [3, 2] for the π-calculus [8].

3.1 Example: The π-calculus

Introduction to the π-calculus. Let $\mathcal{N}$ be an infinite set of *names* ranged over by $a, b, \cdots, x, y, \cdots$ and let τ be a distinguished element not in $\mathcal{N}$. Then *processes* $P \in \mathcal{P}$ are built from names according to the following syntax:

$$
\begin{aligned}
P &::= \mathbf{0} \mid \mu.P \mid P + P \mid P|P \mid (\nu x^{\chi})P \mid [x = y]P \mid !P \\
\mu &::= x(y^{\beta}) \mid \overline{x}y \mid \tau
\end{aligned}
$$

The prefix μ is the first atomic action that the process $\mu.P$ can perform. The input prefix $x(y^{\beta})$ binds the name y in the prefixed process and corresponds to a name y that is received along the link named x. The superscript β is a "variable type"; we write $\mathcal{B}$ for the set of variable types. The output prefix $\overline{x}y$ does not bind the name y and corresponds to the name y that is sent along x. The silent prefix τ denotes an action which is invisible to an external observer of the system.

Turning to the processes, $P + Q$ behaves either as P or as Q whereas $P|Q$ performs P and Q simultaneously and also allows them to communicate with each other (as when one performs an input and the other an output on the same common link). The restriction operator $(\nu x^{\chi})P$ binds the name x in the process P that it prefixes, in such a way that x is a unique name in P that is different from all external names. The agent $(\nu x^{\chi})P$ behaves as P except that sending along $\overline{x}$ and receiving along x is blocked. The superscript χ is a "channel type" in the manner of the "variable type" discussed above; we write $\mathcal{C}$ for the set of channel types. Matching $[x = y]P$ is an if-then operator: process P is activated if $x = y$. Finally, replication $!P$ behaves as $P|P|\cdots$ as many times as needed.

Flow Logic specification of 0-CFA. The result of control flow analyzing a process P is a pair (R, K) (called (ρ, κ) in [3, 2]). The first component, $R : \mathcal{B} \to \wp(\mathcal{C})$, is an abstract environment which gives information about the set of channels to which names can be bound. The second component, $K : \mathcal{C} \to \wp(\mathcal{C})$, is an abstract channel environment which gives information about the set of channels that can flow over given channels. The correctness of a proposed solution (R, K) is validated by a set of clauses operating upon judgments of the form, $(R, K) \models_{me} P$, where the functionality of $R : \mathcal{B} \to \wp(\mathcal{C})$ is extended[2] to $R : (\mathcal{B} \cup \mathcal{C}) \to \wp(\mathcal{C})$ by stipulating that $\forall \chi \in \mathcal{C} : R(\chi) = \{\chi\}$. As in [3, 2] the control flow analysis is developed relative to a "marker environment" $me : \mathcal{N} \to (\mathcal{B} \cup \mathcal{C})$ that maps names to their variable type (in $\mathcal{B}$) or channel type (in $\mathcal{C}$) as appropriate; in the

[2] This seemingly ad-hoc definition is made because the π-calculus does not make a syntactic distinction between "variables" and "channels".

(R,K)	$\models_{me} \mathbf{0}$	iff *true*
(R,K)	$\models_{me} \tau.P$	iff $(R,K) \models_{me} P$
(R,K)	$\models_{me} \overline{x}y.P$	iff $(R,K) \models_{me} P \wedge$ $\forall \chi \in R(me(x)) : R(me(y)) \subseteq K(\chi)$
(R,K)	$\models_{me} x(y^{\beta}).P$	iff $(R,K) \models_{me[y \mapsto \beta]} P \wedge$ $\forall \chi \in R(me(x)) : K(\chi) \subseteq R(\beta)$
(R,K)	$\models_{me} P_1 + P_2$	iff $(R,K) \models_{me} P_1 \wedge (R,K) \models_{me} P_2$
(R,K)	$\models_{me} P_1 \vert P_2$	iff $(R,K) \models_{me} P_1 \wedge (R,K) \models_{me} P_2$
(R,K)	$\models_{me} (\nu x^{\chi})P$	iff $(R,K) \models_{me[x \mapsto \chi]} P$
(R,K)	$\models_{me} [x = y]P$	iff $(R(me(x)) \cap R(me(y)) \neq \emptyset$ $\Rightarrow (R,K) \models_{me} P$
(R,K)	$\models_{me} !P$	iff $(R,K) \models_{me} P$

Table 1. Flow Logic for the π-calculus (taken from [2]).

interest of simplicity we sometimes simplify explanation by pretending that me is the identity.

The Control Flow Analysis is given by the Flow Logic in Table 1. All the rules dealing with a compound process require that the components are validated, apart from the one for matching. Moreover, the second conjunct of the rule for output requires that the set of channels that can be communicated along each element of $R(x)$ (pretending here that me is the identity) includes the channels to which y can evaluate. Symmetrically, the rule for input demands that the set of channels that can pass along x is included in the set of channels to which y can evaluate. The condition for matching says that the continuation P needs to be validated if there is at least one channel to which both x and y can evaluate. Similar "reachability" considerations can be performed also for input and output without invalidating Theorem 1 below. We refer to [2] for further explanation of the analysis and for proofs of its semantic correctness.

An algorithm for obtaining the least solution in[3] time $\mathcal{O}(n^5)$ in the size n of processes is given in [2].

Horn Clauses with Sharing for 0-CFA. To generate HCS's corresponding to the Flow Logic specification in Table 1 we shall perform the following systematic transformations in order to adhere to the format of Horn clauses with sharing:

- A set inclusion of the form $X \subseteq Y$ is expressed using set memberships of the form $\forall u : u \in X \Rightarrow u \in Y$.

[3] In [3] it is conjectured that the constraints can be solved in $\mathcal{O}(n^3)$ bit-vector operations which corresponds to overall time $\mathcal{O}(n^4)$ but no algorithm is given.

$$
\begin{aligned}
\mathcal{G}[\![\mathbf{0}]\!]_{me} &= \mathbf{1} \\
\mathcal{G}[\![\tau.P]\!]_{me} &= \mathcal{G}[\![P]\!]_{me} \\
\mathcal{G}[\![\overline{x}y.P]\!]_{me} &= \mathcal{G}[\![P]\!]_{me} \wedge \\
&\quad \forall u : \forall v : (R(u, me(x)) \wedge R(v, me(y))) \Rightarrow K(v, u) \\
\mathcal{G}[\![x(y^{\beta}).P]\!]_{me} &= \mathcal{G}[\![P]\!]_{me[y \mapsto \beta]} \wedge \\
&\quad \forall u : \forall v : (R(u, me(x)) \wedge K(v, u)) \Rightarrow R(v, \beta) \\
\mathcal{G}[\![P_1 + P_2]\!]_{me} &= \mathcal{G}[\![P_1]\!]_{me} \wedge \mathcal{G}[\![P_2]\!]_{me} \\
\mathcal{G}[\![P_1 | P_2]\!]_{me} &= \mathcal{G}[\![P_1]\!]_{me} \wedge \mathcal{G}[\![P_2]\!]_{me} \\
\mathcal{G}[\![(\nu x^{\chi})P]\!]_{me} &= \mathcal{G}[\![P]\!]_{me[x \mapsto \chi]} \wedge R(\chi, \chi) \\
\mathcal{G}[\![[x = y]P]\!]_{me} &= (\exists u : R(u, me(x)) \wedge R(u, me(y))) \Rightarrow \mathcal{G}[\![P]\!]_{me} \\
\mathcal{G}[\![!P]\!]_{me} &= \mathcal{G}[\![P]\!]_{me}
\end{aligned}
$$

Table 2. Horn Clauses with Sharing for the π-calculus.

- A set membership of the form $u \in R(v)$ is written using a binary predicate of the form $R(u, v)$.

To obtain a finite algorithm we shall restrict our attention to a finite universe, $\mathcal{C}_\star$, containing all the relevant channels; this corresponds to the set $\mathcal{U}_\star \cap \mathcal{C}$ considered in [2]. The constraint generation in Table 2 differs from the one in [2] because Horn clauses with sharing facilitate a more succinct representation of constraints. In particular, in the clause $[x = y]P$ we directly generate the condition $(\exists u : R(u, x) \wedge R(u, y))$ (once more pretending that me is the identity) shared for all of P without the need to duplicate it for each individual constraint (as would be needed to generate constraints in the form of Horn clauses). Also we enforce the convention that $R(\chi) = \{\chi\}$ by generating the constraint $\chi \in R(\chi)$ when appropriate; the desired equality then holds in the least solution.

We state without proof that the two formulations of the analysis are equivalent (using the notational conventions explained above):

Lemma 1. $(R, K) \models_{me} P$ *holds if and only if* $\mathcal{G}[\![P]\!]_{me}$.

For a universe of size $\mathcal{O}(n)$ we prove in Theorem 1 below that the resulting constraints can be solved in cubic time.

3.2 The Complexity of Constraint Specifications

The complexity of the control flow analysis can be established by applying Proposition 1 to the constraints generated for a program but it is more convenient to argue directly in terms of the constraint generation function itself. As will become clear in the next section it is convenient to define a *constraint specification* to be a triple $(\mathcal{T}, \alpha, c)$ where $\mathcal{T}$ is a compositionally defined constraint generation function (like $\mathcal{G}$ in Table 2), c is a global constraint (absent above, hence could be taken to be $\mathbf{1}$), and α is an initial context for the constraint generation function. Here, contexts are supposed to consist of a bounded number of atoms from the universe together with a bounded number of functions to extract atoms from pieces of syntax (like me above). Given a program P the constraint generated then is $\mathcal{T}[\![P]\!]_\alpha \wedge c$.

A constraint specification $(\mathcal{T}, \alpha, c)$ is said to be *linear HCS* if each defining equation of $\mathcal{T}$ takes the form

$$\mathcal{T}[\![\phi'(P_1, \cdots, P_{m'})]\!]_{\alpha'} = c' \wedge \bigwedge_{i=1}^{m''} p'_i \Rightarrow \mathcal{T}[\![P_i]\!]_{\alpha'_i}$$

where $m'' \leq m'$, the P_i are distinct and non-overlapping components of the program $\phi'(P_1, \cdots, P_{m'})$ and α'_i is computed from α' and ϕ'; the formulae c' and p'_i chosen for ϕ' may contain free variables $\tilde{z}$ occurring in α' or extracted from ϕ' (using the extraction functions in α'). The constraint specification is *linear HC* when additionally all clauses (c', p'_i and c) are formulae of HC.

A constraint specification $(\mathcal{T}, \alpha, c)$ is said to have *cost coefficient* r if r is minimal such that each defining equation of $\mathcal{T}$ have quantifiers nested at most to depth $r-1$ and if the global constraint c has quantifiers nested at most to depth r; note that r will always be greater than zero.

Proposition 2. *Given a linear HCS constraint specification* $(\mathcal{T}, \alpha, c)$ *of cost coefficient* r*, a program* P *of size* $\mathcal{O}(n)$ *and a universe of size* $\mathcal{O}(n)$*; the constraint* $\mathcal{T}[\![P]\!]_\alpha \wedge c$ *has size* $\mathcal{O}(n)$ *and its least solution can be found in time* $\mathcal{O}(n^r)$*.* ☐

Proof. Clearly $\mathcal{T}[\![P]\!]_\alpha$ has size $\mathcal{O}(n)$ with quantifiers nested at most to depth $r-1$ and c has size $\mathcal{O}(1)$ with quantifiers nested at most to depth r. The result then follows from Proposition 1. ☐

Theorem 1. *Control Flow Analysis for the* π*-calculus (as in [2]) can be done in cubic time.* ☐

Proof. Clearly $(\mathcal{G}, me, \mathbf{1})$ is a linear HCS constraint specification with cost coefficient 3. Also the universe has size linear in the program because it only consists of entities explicitly mentioned in the program. The result then follows from Proposition 2. ☐

4 The Virtues of Tiling

We now show how tiling facilitates developing a cubic time algorithm for performing control flow analysis [10] for the ambient calculus [4].

4.1 Example: The Ambient Calculus

Introduction to mobile ambients. The syntax of processes $P \in \mathbf{Proc}$, capabilities $M \in \mathbf{Cap}$ and namings $N \in \mathbf{Nam}$ is given by:

$$
\begin{array}{llll}
P ::= (\nu\, n^{\mu})P & \text{restriction} & M ::= \mathsf{in}^t N & \text{enter } N \\
\quad \mid \ \mathbf{0} & \text{inactivity} & \quad \mid \ \mathsf{out}^t N & \text{exit } N \\
\quad \mid \ P \mid P' & \text{composition} & \quad \mid \ \mathsf{open}^t N & \text{open } N \\
\quad \mid \ !P & \text{replication} & & \\
\quad \mid \ N^l[P] & \text{ambient} & N ::= n & \text{name} \\
\quad \mid \ M.\,P & \text{movement} & &
\end{array}
$$

Processes contain a number of constructs known from the π-calculus; an example is the restriction operator where $\mu \in \mathbf{SNam}$ is the "ambient type" (in the manner of the "variable type" and "channel type" considered above) called "stable name" in [10]. The final two constructs are unique to the ambient calculus. An ambient is a process operating inside a named border. Movement of ambients is governed by capabilities. The in-capability directs the enclosing ambient to enter a sibling named N. The out-capability directs the enclosing ambient to move out of its parent named N. The open-capability dissolves the border around a sibling ambient named N. Finally, namings are names. Much as in [10] we have placed labels $l \in \mathbf{ALab}$ on ambients and labels $t \in \mathbf{TLab}$ on capabilities (or transitions) in order to have explicit notation for the various subterms.

Flow Logic specification. An ambient will be identified by its label $l \in \mathbf{ALab}$ and a transition by its associated *capability type* $\tilde{m} \in \mathbf{SCap}$ called "stable capability" in [10]; capability types are given by

$$\tilde{m} ::= \mathsf{in}^t \mu \mid \mathsf{out}^t \mu \mid \mathsf{open}^t \mu$$

and correspond to capabilities except that names have been replaced by ambient types. The analysis records which ambients and transitions occur inside what ambients in the component $I : \mathbf{ALab} \to \wp(\mathbf{ALab} \cup \mathbf{SCap})$. We also use the "inverse" mapping $I^{-1} : (\mathbf{ALab} \cup \mathbf{SCap}) \to \wp(\mathbf{ALab})$ that returns the set of ambients in which the given ambient or transition might occur; formally $z \in I(l)$ if and only if $l \in I^{-1}(z)$.

Each occurrence of an ambient has an ambient type and to keep track of this information the analysis also contains the component $H : \mathbf{ALab} \to \wp(\mathbf{SNam})$. As above we use the "inverse mapping" $H^{-1} : \mathbf{SNam} \to \wp(\mathbf{ALab})$ that returns the set of ambients that might have the given ambient type.

The acceptability of the analysis is defined by the following four predicates defined by the Flow Logic in Table 3:

$(I, H) \models^l_{me} P$	for checking a process $P \in \mathbf{Proc}$;
$(I, H) \rhd_{me} M : \tilde{M}$	for translating a capability $M \in \mathbf{Cap}$ into a set $\tilde{M} \in \wp(\mathbf{SCap})$ of capability types;
$(I, H) \Vert\!\equiv_{me} N : \tilde{N}$	for decoding a naming $N \in \mathbf{Nam}$ into a set $\tilde{N} \in \wp(\mathbf{SNam})$ of ambient types;
$(I, H) \models\!\!\!\equiv^l \tilde{m}$	for checking a capability type. $\tilde{m} \in \mathbf{SCap}$.

$$
\begin{array}{rcl}
(I,H) \models^l_{me} (\nu\, n^\mu)P & \text{iff} & (I,H) \models^l_{me[n\mapsto\mu]} P \\[1ex]
(I,H) \models^l_{me} \mathbf{0} & \text{iff} & \text{true} \\[1ex]
(I,H) \models^l_{me} P \mid P' & \text{iff} & (I,H) \models^l_{me} P \;\wedge\; (I,H) \models^l_{me} P' \\[1ex]
(I,H) \models^l_{me} !P & \text{iff} & (I,H) \models^l_{me} P \\[1ex]
(I,H) \models^l_{me} N^{l'}[P] & \text{iff} & (I,H) \models^{l'}_{me} P \;\wedge\; l' \in I(l) \;\wedge \\
& & (I,H) \Vvdash_{me} N : \tilde{N} \;\wedge\; \tilde{N} \subseteq H(l') \\[1ex]
(I,H) \models^l_{me} M.\,P & \text{iff} & (I,H) \models^l_{me} P \;\wedge \\
& & (I,H) \rhd_{me} M : \tilde{M} \;\wedge\; \forall \tilde{m} \in \tilde{M} : (I,H) \models^l \tilde{m} \\
\hline
(I,H) \rhd_{me} \mathsf{in}^t N : \tilde{M} & \text{iff} & (I,H) \Vvdash_{me} N : \tilde{N} \;\wedge\; \tilde{M} \supseteq \{\mathsf{in}^t \mu \mid \mu \in \tilde{N}\} \\[1ex]
(I,H) \rhd_{me} \mathsf{out}^t N : \tilde{M} & \text{iff} & (I,H) \Vvdash_{me} N : \tilde{N} \;\wedge\; \tilde{M} \supseteq \{\mathsf{out}^t \mu \mid \mu \in \tilde{N}\} \\[1ex]
(I,H) \rhd_{me} \mathsf{open}^t N : \tilde{M} & \text{iff} & (I,H) \Vvdash_{me} N : \tilde{N} \;\wedge\; \tilde{M} \supseteq \{\mathsf{open}^t \mu \mid \mu \in \tilde{N}\} \\
\hline
(I,H) \Vvdash_{me} n : \tilde{N} & \text{iff} & \tilde{N} \supseteq \{me(n)\} \\
\hline
(I,H) \models^l \mathsf{in}^t \mu & \text{iff} & \mathsf{in}^t \mu \in I(l) \;\wedge \\
& & \forall l^a \in I^{-1}(\mathsf{in}^t \mu) : \forall l'^a \in I^{-1}(l^a) : \\
& & \quad \forall l''^a \in I(l'^a) \cap H^{-1}(\mu) : l^a \in I(l''^a) \\[1ex]
(I,H) \models^l \mathsf{out}^t \mu & \text{iff} & \mathsf{out}^t \mu \in I(l) \;\wedge \\
& & \forall l^a \in I^{-1}(\mathsf{out}^t \mu) : \forall l'^a \in I^{-1}(l^a) \cap H^{-1}(\mu) : \\
& & \quad \forall l''^a \in I^{-1}(l'^a) : l^a \in I(l''^a) \\[1ex]
(I,H) \models^l \mathsf{open}^t \mu & \text{iff} & \mathsf{open}^t \mu \in I(l) \;\wedge \\
& & \forall l^a \in I^{-1}(\mathsf{open}^t \mu) : \forall l'^a \in I(l^a) \cap H^{-1}(\mu) : \\
& & \quad \forall l' \in I(l'^a) : l' \in I(l^a)
\end{array}
$$

Table 3. Flow Logic for the ambient calculus (taken from [10]).

Much as before a marker environment $me : \mathbf{Nam} \to_{\text{fin}} \mathbf{SNam}$ is used for mapping names to ambient types. We refer to [10] for further explanation of the analysis and for proofs of its semantic correctness.

An algorithm for obtaining the least solution in[4] time $\mathcal{O}(n^5)$ is given in [10].

Constraint generation. To generate the constraints as simply as possible we note that in the communication-free fragment of the mobile ambients studied here the only possible naming (N) is a name (n). Thus namings can be replaced by names everywhere and this makes the judgement $(I,H) \Vvdash_{me} n : \tilde{N}$ dispensable (essentially by always choosing for $\tilde{N}$ the least choice $\{me(n)\}$).

In a similar way we can dispense with the judgement $(I,H) \rhd_{me} M : \tilde{M}$ if we arrange that the translation from names to ambient types also becomes the duty of the judgement $(I,H) \models^l \tilde{m}$ that thus takes the form $(I,H) \models^l_{me} M$.

[4] In [10] it is conjectured that a more sophisticated implementation will be able to achieve $\mathcal{O}(n^4)$ but no details are provided.

$$
\begin{aligned}
\mathcal{G}[\![(\nu\, n^{\mu})P]\!]^{l}_{me} &= \mathcal{G}[\![P]\!]^{l}_{me[n\mapsto\mu]}\\
\mathcal{G}[\![\mathbf{0}]\!]^{l}_{me} &= \mathbf{1}\\
\mathcal{G}[\![P \mid P']\!]^{l}_{me} &= \mathcal{G}[\![P]\!]^{l}_{me} \;\wedge\; \mathcal{G}[\![P']\!]^{l}_{me}\\
\mathcal{G}[\![!P]\!]^{l}_{me} &= \mathcal{G}[\![P]\!]^{l}_{me}\\
\mathcal{G}[\![n^{l'}[P]]\!]^{l}_{me} &= \mathcal{G}[\![P]\!]^{l'}_{me} \;\wedge\; I(l',l) \;\wedge\; H(me(n),l')\\
\mathcal{G}[\![M.\,P]\!]^{l}_{me} &= \mathcal{G}[\![P]\!]^{l}_{me} \;\wedge\; \mathcal{G}'[\![M]\!]^{l}_{me}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{G}'[\![\mathsf{in}^t n]\!]^{l}_{me} = {} & I(\mathsf{in}^t me(n),l) \wedge\\
& \forall l^a : \forall l'^a : \forall l''^a : (I(\mathsf{in}^t me(n),l^a) \;\wedge\; I(l^a,l'^a) \;\wedge\\
& \quad I(l''^a,l'^a) \;\wedge\; H(me(n),l''^a)) \Rightarrow I(l^a,l''^a)\\
\mathcal{G}'[\![\mathsf{out}^t n]\!]^{l}_{me} = {} & I(\mathsf{out}^t me(n),l) \wedge\\
& \forall l^a : \forall l'^a : \forall l''^a : (I(\mathsf{out}^t me(n),l^a) \;\wedge\; I(l^a,l'^a) \;\wedge\\
& \quad H(me(n),l'^a) \;\wedge\; I(l'^a,l''^a)) \Rightarrow I(l^a,l''^a)\\
\mathcal{G}'[\![\mathsf{open}^t n]\!]^{l}_{me} = {} & I(\mathsf{open}^t me(n),l) \wedge\\
& \forall l^a : \forall l'^a : \forall l' : (I(\mathsf{open}^t me(n),l^a) \;\wedge\; I(l'^a,l^a) \;\wedge\\
& \quad H(me(n),l'^a) \;\wedge\; I(l',l'^a)) \Rightarrow I(l',l^a)
\end{aligned}
$$

Table 4. Horn Clauses for the ambient calculus.

This leaves us with the judgements $(I,H) \models^{l}_{me} P$ and $(I,H) \models^{l}_{me} M$ and they give rise to constraint generation functions $\mathcal{G}[\![P]\!]^{l}_{me}$ and $\mathcal{G}'[\![M]\!]^{l}_{me}$, respectively. To satisfy the Horn clause format we perform the following systematic transformations:

- A set membership involving an "inverse" relation of the form $u \in R^{-1}(v)$ is rewritten to the form $v \in R(u)$ thus avoiding "inverse" relations.
- As in Subsection 3.1 a set membership of the form $u \in R(v)$ is written using a binary predicate of the form $R(u,v)$.

Using the notational conventions explained above we state without proof that the formulations of Tables 3 and 4 are equivalent:

Lemma 2. *$(I,H) \models^{l}_{me} P$ holds if and only if $\mathcal{G}[\![P]\!]^{l}_{me}$.*

Clearly $(\mathcal{G},(l,me),\mathbf{1})$ is a linear HCS constraint specification with cost coefficient 4 that operates over a universe of size linear in the size of the program so that by Proposition 2 the constraints can be solved in time $\mathcal{O}(n^4)$; we now develop the notion of tiling in order to obtain a cubic bound.

4.2 Tiling of Constraint Specifications

Tiling applies to a linear HC constraint specification and systematically rewrites it into another with the aim of eventually reducing the cost coefficient. There are two main tricks to be played when tiling a constraint specification $(\mathcal{T},\alpha,c)$:

- to remove quantifiers in c or in the defining equations of $\mathcal{T}$, and

- to transfer sub-formulae from a defining equation of $\mathcal{T}$ into the global constraint c.

We first apply the techniques to the analysis of the mobile ambients and then show how to perform it in general.

Theorem 2. *Control Flow Analysis for the mobile ambients (as in [10]) can be done in cubic time.*

Proof. The constraint specification $(\mathcal{H}, (l, me), c_{\mathcal{H}})$ of Table 5 has cost coefficient 3 and so by Proposition 2 we can solve $\mathcal{H}[\![P]\!]^l_{me} \wedge c_{\mathcal{H}}$ in cubic time.

It remains to show that the least solution to $\mathcal{H}[\![P]\!]^l_{me} \wedge c_{\mathcal{H}}$ equals the least solution to $\mathcal{G}[\![P]\!]^l_{me}$ (ignoring the auxiliary relations). The key idea to reducing the complexity is to ensure that the formulae generated are "tiled" such that subformulae with three nested quantifiers are only generated a constant number of times whereas subformulae with two nested quantifiers may be generated a linear number of times.

Concentrating on the clause for in-capabilities we note that it establishes that l^a and l''^a are siblings because they have the same parent (namely l'^a). Imagine that we have a relation S for expressing the sibling relation: $S(l^a, l''^a)$ if and only if $\exists l'^a : I(l^a, l'^a) \wedge I(l''^a, l'^a)$. Then the clause for $\mathcal{G}'[\![\mathsf{in}^t n]\!]^l_{me}$ is equivalent to the formula:

$$\begin{array}{l} I(\mathsf{in}^t me(n), l) \wedge \\ \forall l^a : \forall l''^a : (I(\mathsf{in}^t me(n), l^a) \ \wedge \ S(l^a, l''^a) \ \wedge \ H(me(n), l''^a)) \Rightarrow I(l^a, l''^a) \end{array}$$

Indeed the relation S can be obtained by generating the Horn clause

$$\forall l^a : \forall l'^a : \forall l''^a : (I(l^a, l'^a) \ \wedge \ I(l''^a, l'^a)) \Rightarrow S(l^a, l''^a)$$

and taking the least solution (assuming that this is the only clause defining S).

The clause for out-capabilities has a slightly different structure so here we make use of a predicate $O(l^a, l'^a)$ for indicating when l^a may be a candidate for moving out of l'^a. Similarly in the clause for open-capabilities we make use of a predicate $P(l'^a, l^a)$ for indicating when l'^a may be a candidate for being opened inside l^a. This concludes the proof. $\square$

In fact it is not necessary to have any deep insights in the analysis in order to perform tiling. To make this clear we now develop a purely mechanical notion of tiling, $\longmapsto$, such that Theorem 2 follows from merely noting that, except for a few additional simplifications,

$$(\mathcal{G}, (l, me), \mathbf{1}) \longmapsto^\star (\mathcal{H}, (l, me), c_{\mathcal{H}})$$

and then relying on Proposition 3 below.

Tiling individual constraints. We begin by considering a tiling transformation on certain individual constraints. It takes the form $c \overset{o}{\mapsto} c_1 \& c_2$ where the idea is that c should be replaced by c_1 and that c_2 should be moved out to the global constraint; the superscript o will be 0 when the constraint c occurs in the

$$\mathcal{H}[\![(\nu\, n^{\mu})P]\!]^{l}_{me} = \mathcal{H}[\![P]\!]^{l}_{me[n\mapsto\mu]}$$

$$\mathcal{H}[\![\mathbf{0}]\!]^{l}_{me} = \mathbf{1}$$

$$\mathcal{H}[\![P \mid P']\!]^{l}_{me} = \mathcal{H}[\![P]\!]^{l}_{me} \;\wedge\; \mathcal{H}[\![P']\!]^{l}_{me}$$

$$\mathcal{H}[\![!P]\!]^{l}_{me} = \mathcal{H}[\![P]\!]^{l}_{me}$$

$$\mathcal{H}[\![n^{l'}[P]]\!]^{l}_{me} = \mathcal{H}[\![P]\!]^{l'}_{me} \;\wedge\; I(l',l) \;\wedge\; H(me(n),l')$$

$$\mathcal{H}[\![M.\,P]\!]^{l}_{me} = \mathcal{H}[\![P]\!]^{l}_{me} \;\wedge\; \mathcal{H}'[\![M]\!]^{l}_{me}$$

$$\begin{aligned}\mathcal{H}'[\![\mathsf{in}^t n]\!]^{l}_{me} = {} & I(\mathsf{in}^t me(n),l) \;\wedge \\ & \forall l^a : \forall l''^a : (I(\mathsf{in}^t me(n),l^a) \;\wedge\; S(l^a,l''^a) \;\wedge \\ & \quad H(me(n),l''^a)) \Rightarrow I(l^a,l''^a)\end{aligned}$$

$$\begin{aligned}\mathcal{H}'[\![\mathsf{out}^t n]\!]^{l}_{me} = {} & I(\mathsf{out}^t me(n),l) \;\wedge \\ & \forall l^a : \forall l'^a : (I(\mathsf{out}^t me(n),l^a) \;\wedge\; I(l^a,l'^a) \;\wedge \\ & \quad H(me(n),l'^a)) \Rightarrow O(l^a,l'^a)\end{aligned}$$

$$\begin{aligned}\mathcal{H}'[\![\mathsf{open}^t n]\!]^{l}_{me} = {} & I(\mathsf{open}^t me(n),l) \;\wedge \\ & \forall l^a : \forall l'^a : (I(\mathsf{open}^t me(n),l^a) \;\wedge\; I(l'^a,l^a) \;\wedge \\ & \quad H(me(n),l'^a)) \Rightarrow P(l',l^a)\end{aligned}$$

$$\begin{aligned}c_{\mathcal{H}} = {} & \forall l^a : \forall l'^a : \forall l''^a : (I(l^a,l'^a) \;\wedge\; I(l''^a,l'^a)) \Rightarrow S(l^a,l''^a) \;\wedge \\ & \forall l^a : \forall l'^a : \forall l''^a : (O(l^a,l'^a) \;\wedge\; I(l'^a,l''^a)) \Rightarrow I(l^a,l''^a) \;\wedge \\ & \forall l^a : \forall l'^a : \forall l' : (P(l'^a,l^a) \;\wedge\; I(l',l'^a)) \Rightarrow I(l',l^a)\end{aligned}$$

Table 5. Tiled Horn Clauses for the ambient calculus.

global constraint and 1 when it occurs in a defining equation for the constraint specification.

The intention is to reduce the quantifier depth of c by possibly generating additional "cheap" clauses; in intuitive terms, reduction of quantifier depth means reducing the number of variables that are "simultaneously active" when expressing the analysis. The general form of a formula c to be tiled is

$$c = \forall y_1 : \cdots \forall y_k : \mathsf{pre}' \Rightarrow R(\tilde{w})$$

where $\tilde{w}$ may contain bound variables from $y_1, \cdots, y_k$ as well as variables occurring in the program; we shall write $\tilde{z}$ for the latter. To define the transformation we first introduce two auxiliary concepts. We shall say that a bound variable y_i is a *candidate* in case it does not occur in $\tilde{w}$; similarly, we shall say that the special symbol $\Box$ is a *candidate* in case no symbol from $\tilde{z}$ occurs in $\tilde{w}$. Furthermore, we say that two distinct bound variables y_i and y_j are *neighbours* in case there is a query $R'(\cdots)$ in pre' that mentions both y_i and y_j; similarly, we shall say that a bound variable y_i and the special symbol $\Box$ are *neighbours* in case there is a query $R'(\cdots)$ in pre' that mentions both y_i and some variable from $\tilde{z}$.

There are three rules defining $c \overset{o}{\mapsto} c_1 \& c_2$, each one removing a candidate having at most 2 neighbours. The first rule removes a bound variable that is a neighbour of $\Box$:

$$\begin{array}{l}\forall y_1 : \cdots \forall y_k : \mathsf{pre}' \Rightarrow R(\tilde{w}) \\ \overset{o}{\mapsto} (\forall y'_1 : \cdots \forall y'_{k-1} : A_{\text{fresh}}(x_1, \cdots, x_d, \tilde{z}) \wedge \mathsf{pre}'_2 \Rightarrow R(\tilde{w})) \wedge \\ \quad (\forall y : \forall x_1 : \cdots \forall x_d : \mathsf{pre}'_1 \Rightarrow A_{\text{fresh}}(x_1, \cdots, x_d, \tilde{z})) \\ \quad \& \ \mathbf{1} \\ \text{if } y \text{ is a candidate with neighbour list } x_1, \cdots, x_d, \Box \text{ and } o + k \geq 4, d \leq 1\end{array}$$

Here y is a bound variable and $y'_1, \ldots, y'_{k-1}$ is an enumeration of the remaining bound variables. Furthermore, pre'_1 denotes the conjunction of all queries from pre' containing y, pre'_2 denotes the conjunction of the remaining ones, and $\tilde{z}$ is an enumeration of the program variables occurring in pre'_1. The auxiliary relation A_{fresh} is chosen fresh for each use of the rule.

The next rule removes a bound variable that is not a neighbour of $\Box$:

$$\begin{array}{l}\forall y_1 : \cdots \forall y_k : \mathsf{pre}' \Rightarrow R(\tilde{w}) \\ \overset{o}{\mapsto} \forall y'_1 : \cdots \forall y'_{k-1} : A_{\text{fresh}}(x_1, \cdots, x_d) \wedge \mathsf{pre}'_2 \Rightarrow R(\tilde{w}) \\ \quad \& \ \forall y : \forall x_1 : \cdots \forall x_d : \mathsf{pre}'_1 \Rightarrow A_{\text{fresh}}(x_1, \cdots, x_d) \\ \text{if } y \text{ is a candidate with neighbour list } x_1, \cdots, x_d \text{ and } o + k \geq 4, d \leq 2\end{array}$$

As before, y is a bound variable and $y'_1, \ldots, y'_{k-1}$ is an enumeration of the remaining bound variables. Also pre'_1 denotes the conjunction of all queries from pre' containing y, pre'_2 denotes the conjunction of the remaining ones, and the auxiliary relation A_{fresh} is chosen fresh for each use of the rule.

The final rule could perhaps be said to remove $\Box$ by transferring the program independent parts of the clause into the global constraint:

$$\begin{array}{l}\forall y_1 : \cdots \forall y_k : \mathsf{pre}' \Rightarrow R(\tilde{w}) \\ \overset{o}{\mapsto} \forall x_1 : \cdots \forall x_d : \mathsf{pre}'_1 \Rightarrow A_{\text{fresh}}(x_1, \cdots, x_d) \\ \quad \& \ \forall y_1 : \cdots \forall y_k : A_{\text{fresh}}(x_1, \cdots, x_d) \wedge \mathsf{pre}'_2 \Rightarrow R(\tilde{w}) \\ \text{if } \Box \text{ is a candidate with neighbour list } x_1, \cdots, x_d \text{ and } o = 1, d \leq 2\end{array}$$

As before pre'_1 denotes the conjunction of all queries from pre' containing some program variable (from $\tilde{z}$), pre'_2 denotes the conjunction of the remaining ones, and the auxiliary relation A_{fresh} is chosen fresh for each use of the rule. (The condition $o = 1$ merely says that the rule cannot be applied to the global constraint.)

Tiling constraint specifications. The tiling transformation $(\mathcal{T}, \alpha, c) \longmapsto (\mathcal{T}', \alpha', c')$ on constraint specifications is defined by the following rules:

$$\begin{array}{l}(\mathcal{T}, \alpha, \cdots \wedge c \wedge \cdots) \longmapsto (\mathcal{T}, \alpha, \cdots \wedge c_1 \wedge c_2 \wedge \cdots) \\ \text{if } c \overset{1}{\mapsto} c_1 \& c_2\end{array}$$

$$\begin{array}{l}(\mathcal{T}, \alpha, c) \longmapsto (\mathcal{T}', \alpha, c \wedge c_2) \\ \text{if } c' \overset{0}{\mapsto} c_1 \& c_2 \text{ and } \mathcal{T}' \text{ is as } \mathcal{T} \text{ except that} \\ \quad \mathcal{T}[\![\phi'(P_1, \cdots, P_m)]\!]_{\alpha'} = \cdots \wedge c' \wedge \cdots \wedge \bigwedge_{i=1}^{m''} \mathsf{pre}'_i \Rightarrow \mathcal{T}[\![P_i]\!]_{\alpha'_i} \\ \quad \mathcal{T}'[\![\phi'(P_1, \cdots, P_m)]\!]_{\alpha'} = \cdots \wedge c_1 \wedge \cdots \wedge \bigwedge_{i=1}^{m''} \mathsf{pre}'_i \Rightarrow \mathcal{T}'[\![P_i]\!]_{\alpha'_i}\end{array}$$

The following result establishes the correctness of the tiling transformation; since tiling is not able always to reduce the complexity to cubic it is important also to show that the non-determinism is purely benign:

Proposition 3. *Let* $(\mathcal{T}, \alpha, c)$ *be a linear CH constraint specification of cost coefficient* r. *If* $(\mathcal{T}, \alpha, c) \longmapsto (\mathcal{T}', \alpha', c')$ *then*

- $(\mathcal{T}', \alpha', c')$ *is a linear CH constraint specification of cost coefficient* $r' \leq r$.
- *For all programs* P *the least solution to* $\mathcal{T}'[\![P]\!]_{\alpha'} \wedge c'$ *equals the least solution to* $\mathcal{T}[\![P]\!]_{\alpha} \wedge c$ *(ignoring the auxiliary relations introduced).*

The $\longmapsto$ *rewrite relation is terminating and if some maximal reduction sequence leads to cost coefficient* r' *then so do all.*

Proof. See Appendix B.

5 Conclusion

The search for the techniques reported here was partly stimulated by the Theorem of Robertson and Seymour (see e.g. [6]) that says that for a large class of properties of graphs (essentially those that are closed under taking subgraphs) it can be decided in cubic time whether or not a graph has the property. While not immediately applicable to the problem of control flow analysis for calculi of computation it nonetheless motivates careful scrutiny of those instances where more than cubic time seems to be needed. Indeed we managed to reduce two previously published bounds from a higher polynomial to cubic and we are currently working on extending the techniques to deal also with the full ambient calculus where communication is admitted.

References

1. A. Aiken. Introduction to set constraint-based program analysis. *Science of Computer Programming*, 35:79–111, 1999.
2. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static analysis for the π-calculus with applications to security. *Information and Computation*, (to appear), 2001.
3. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Control flow analysis for the π-calculus. In *Proceedings of CONCUR'98*, volume 1466 of *LNCS*, pages 84–98. Springer-Verlag, 1998.
4. L. Cardelli and A. D. Gordon. Mobile ambients. In *Proceedings of FoSSaCS'98*, volume 1378 of *LNCS*, pages 140–155. Springer-Verlag, 1998.
5. W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 3:267–284, 1984.
6. J. van Leeuwen. Graph Algorithms. *Handbook of Theoretical Computer Science*, A:525–631, 1990.
7. D. McAllester. On the complexity analysis of static analyses. In *6th Static Analysis Symposium (SAS)*, pages 312–329. LNCS 1694, Springer Verlag, 1999.
8. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (I and II). *Information and Computation*, 100(1):1–77, 1992.

9. F. Nielson, H. Riis Nielson, and C. L. Hankin. *Principles of Program Analysis*. Springer, 1999.
10. F. Nielson, H. Riis Nielson, R. R. Hansen, and J. G. Jensen. Validating firewalls in mobile ambients. In *Proceedings of CONCUR'99*, volume 1664 of *LNCS*, pages 463–477, 1999.
11. J. Rehof and T. Mogensen. Tractable constraints in finite semilattices. *Science of Computer Programming (SCP)*, 35(1):191–221, 1999.
12. M. Yannakakis. Graph-theoretic concepts in database theory. In *9th ACM Symp. on Principles of Database Systems (PODS)*, pages 230–242, 1990.

A Proof of Proposition 1

The proof proceeds in three phases. First we transform $c = c_1 \wedge \cdots \wedge c_n$ to $\tilde{c}$ by replacing every universal quantification $\forall x : \mathsf{clause}$ by the conjunction of all N possible instantiations of clause and every existential quantification $\exists x : \mathsf{pre}$ by the disjunction of all N possible instantiations of pre. The resulting clause $\tilde{c}$ is logically equivalent to c, has size

$$\mathcal{O}(N^{r_1} \cdot n_1 + \cdots + N^{r_m} \cdot n_m) \tag{1}$$

and is *boolean*; the latter just means that there are no variables or quantifications and all literals are viewed as nullary predicates.

For the second phase we now describe a transformation $F \longmapsto F_1, \ldots, F_l$ that for each boolean HCS formula F, produces a sequence of boolean "almost-HC" formulae $F_1, \ldots, F_l$. The transformation first replaces all top-level conjunctions in F with ",". Then it proceeds by successively replacing clauses occurring in the sequence with sequences of simpler ones.

$$\begin{array}{lll}
\mathsf{pre} \Rightarrow \mathsf{clause}_1 \wedge \mathsf{clause}_2 & \longmapsto & \mathsf{pre} \Rightarrow A_{\text{fresh}}, \\
 & & A_{\text{fresh}} \Rightarrow \mathsf{clause}_1,\ A_{\text{fresh}} \Rightarrow \mathsf{clause}_2 \\
\mathsf{pre}_1 \Rightarrow \mathsf{pre}_2 \Rightarrow \mathsf{clause} & \longmapsto & \mathsf{pre}_1 \wedge \mathsf{pre}_2 \Rightarrow \mathsf{clause} \\
\mathsf{pre} \Rightarrow \mathbf{1} & \longmapsto & \mathbf{1}
\end{array}$$

Here A_{fresh} is a new fresh nullary predicate generated for each application of the relevant transformation. The transformation is completed, with result $\tilde{F}$, as soon as none of these rewrite rules can be applied. Clearly the conjunction of the resulting formulae $\tilde{F}$ is logically equivalent to F (ignoring the fresh predicates).

To show that this process terminates and that the size of $\tilde{F}$ is at most a constant times the size of the input formula F, we assign a cost to the formulae. Let us define the *cost* of a sequence of clauses as the sum of costs of all occurrences of predicate symbols and operators (excluding ","). In general, the cost of a symbol or operator is 1 — except implications "$\Rightarrow$" which count 2, and conjunctions in conclusions which count 8. Then the first rule decreases the cost from $k + 10$ to $k + 9$, the second rule decreases the cost from $k + 4$ to $k + 3$, whereas the third rule decreases the cost from $k + 3$ to 1 (for suitable values of k). Since the cost of the initial sequence is at most 8 times the size of F, only a linear number of rewrite steps can be performed. Since each step increases the size at most by a constant, we conclude that the $\tilde{F}$ has increased just by a constant factor. Consequently, when applying this transformation to $\tilde{c}$, we obtain a boolean formula without sharing of size as in (1).

Finally, the third phase consists in solving the resulting system of boolean Horn clauses (possibly with disjunctions). This can be done in linear time using the techniques in e.g. [11]. Alternatively one can remove also the disjunctions, by replacing each $\mathsf{pre}_1 \vee \mathsf{pre}_2$ by A_{fresh} and two new clauses $\mathsf{pre}_1 \Rightarrow A_{\text{fresh}}$ and $\mathsf{pre}_2 \Rightarrow A_{\text{fresh}}$. Assigning all symbols a cost of 1 — except disjunction that counts 6 — suffices for showing that the size does not increase by more than a constant factor here as well. The resulting system can then be solved in linear time by the classical techniques of e.g. [5].

B Proof of Proposition 3

It is straightforward to prove that $(\mathcal{T}', \alpha', c')$ is a linear CH constraint specification of cost coefficient $r' \leq r$ and that the least solution to $\mathcal{T}'[\![P]\!]_{\alpha'} \wedge c'$ equals the least solution to $\mathcal{T}[\![P]\!]_{\alpha} \wedge c$ (ignoring the auxiliary relations introduced).

The key ingredient in showing that $\longmapsto$ is terminating is to note that $\overset{o}{\mapsto}$ is only applied to formula whose cost coefficient is at least 4, that the cost coefficient is reduced by 1, and that all auxiliary clauses generated have cost coefficient at most 3.

The key ingredient in showing that all maximal transformation sequences of $\longmapsto$ lead to the same cost coefficient is to note that $\overset{o}{\mapsto}$ is confluent. To show this we develop a simple graph model and then prove a diamond property. The undirected graph g_c associated with a clause

$$c = \forall y_1 : \cdots \forall y_k : \mathsf{pre}' \Rightarrow R(\tilde{w})$$

has nodes $\{y_1, \cdots, y_k, \Box\}$ and edges between any two nodes that are neighbours (in the sense of Subsection 4.2). We now present three reduction rules on undirected graphs which allow the removal of candidate nodes (in the sense of Subsection 4.2).

Formally, let $g_i = (V_i, E_i)$, $i = 1, 2$, denote two undirected finite graphs and let Y be the complement of the set of candidate nodes. We say that g_1 can be reduced to g_2 (by removal of vertex v), i.e., $g_1 \rightarrow_v g_2$, provided that $v \in V_1 \backslash Y$ and $V_2 = V_1 \backslash \{v\}$, and one of the following conditions are satisfied:

- $\mathsf{deg}\, v = 0$ and $E_2 = E_1$; or
- $\mathsf{deg}\, v = 1$ and $E_2 = E_1 \backslash \{e\}$ where e is the unique edge incident with v; or
- $\mathsf{deg}\, v = 2$ and $E_2 = (E_1 \backslash \{e_1, e_2\}) \cup \{e\}$ where $e_i = \{u_i, v\}$, $i = 1, 2$ are the two unique edges incident with v, and $e = \{u_1, u_2\}$.

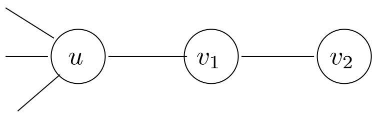
Whenever one of the rules of Subsection 4.2 is applied the effect on the undirected graph is recorded by one of the rules above (ignoring the clauses with cost coefficient at most 3) and vice versa. In particular, this graph formalization reveals that our tiling technique can be seen as a generalization of the reduction of "chain queries" in Datalog as considered in [12].

Let g denote a finite undirected graph with n nodes. Since every reduction step decreases the number of vertices by 1, we conclude that every maximal sequence of reduction steps has length at most $n - |Y|$. Since the reduction is terminating, Proposition 3 follows from the following 1-step diamond property:

Lemma 3. *Assume that for finite undirected graphs g_0, g_1, g_2, and vertices $v_1 \neq v_2$, we have $g_0 \rightarrow_{v_1} g_1$ and $g_0 \rightarrow_{v_2} g_2$. Then there is a finite undirected graph g_3 such that also $g_1 \rightarrow_{v_2} g_3$ and $g_2 \rightarrow_{v_1} g_3$:*

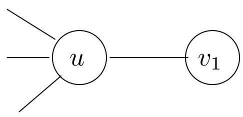
$$\begin{tikzcd}
 & g_0 \arrow[dl, "v_1"'] \arrow[dr, "v_2"] & \\
g_1 \arrow[dr, dotted, "v_2"'] & & g_2 \arrow[dl, dotted, "v_1"] \\
 & g_3 &
\end{tikzcd}$$

Proof. Lemma 3 is proved by case distinction on the various possibilities of relative positions of v_1 and v_2 in g_0. In case, v_1 and v_2 are not neighbours in g_0, the property holds, since their reductions do not interfere. Therefore assume that v_1 and v_2 are neighbours in g_0, i.e., $\{v_1, v_2\}$ is an edge in g_0. Then either both have degree 2, or, one of them has degree 2 whereas the other has degree 1, or both have degree 1. Assume for example that v_1 and v_2 have degrees 2 and 1, respectively. Then there must be an edge $\{u, v_1\}$ in g_0 for some $u \neq v_2$:

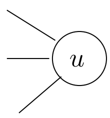


Reduction of v_1 results in a graph g_1 where v_1 has been removed and the two edges $\{u, v_1\}$ and $\{v_1, v_2\}$ have been replaced with an edge $\{u, v_2\}$:

In particular, the degree of v_2 is still 1. Accordingly, reduction of v_2 results in a graph g_2 where v_1 has been removed together with the edge $\{v_1, v_2\}$:



In particular, the degree of v_1 has decreased to 1. Thus, reduction of v_2 in g_1 as well as reduction of v_1 in g_2 results in the same graph g_3 which can be obtained from g_1 by removing both v_1 and v_2 together with the edges $\{v_1, v_2\}$ and $\{u, v_1\}$:



The cases where v_1 and v_2 are neighbours and both have degree 2 or 1 is similar. This concludes the proof of Lemma 3. $\square$