Normalizable Horn Clauses, Strongly Recognizable Relations and Spi

Flemming Nielson[1], Hanne Riis Nielson[1], and Helmut Seidl[2]

[1] Informatics and Mathematical Modelling, Technical University of Denmark, DK-2800 Kongens Lyngby, Denmark, {nielson,riis}@imm.dtu.dk
[2] Universität Trier, FB IV – Informatik, D-54286 Trier, Germany, seidl@uni-trier.de

Abstract. We exhibit a rich class of Horn clauses, which we call $\mathcal{H}1$, whose least models, though possibly infinite, can be computed effectively. We show that the least model of an $\mathcal{H}1$ clause consists of so-called *strongly* recognizable relations and present an exponential normalization procedure to compute it. In order to obtain a practical tool for program analysis, we identify a restriction of $\mathcal{H}1$ clauses, which we call $\mathcal{H}2$, where the least models can be computed in polynomial time. This fragment still allows to express, e.g., Cartesian product and transitive closure of relations. Inside $\mathcal{H}2$, we exhibit a fragment $\mathcal{H}3$ where normalization is even cubic. We demonstrate the usefulness of our approach by deriving a cubic control-flow analysis for the Spi calculus [1] as presented in [14].

Keywords: Program analysis, uniform Horn clauses, strongly recognizable relations, Spi calculus.

1 Introduction

In [16], we proposed ALFP ("alternation-free least fixpoint logic in clausal form") as a specification language for control-flow analyzers. ALFP extends classical Datalog clauses by extra logical features like, e.g., explicit quantification, disjunctions in pre-conditions and conditional clauses. ALFP and related formalisms have shown to be extremely convenient for specifying control-flow analyses for various programming languages [11, 15]. The advantage of such a logic-based approach is that tuning of the analysis boils down to rewriting of formulas whose correctness can be formally proven within the logic. Also, the generic implementation of formula solvers gives rise to complexity meta-theorems [3, 11, 7] thus forming the basis for generators of *predictable program analyses* [16, 13]. The ALFP based approach as well as the restricted classes of Horn clauses used by McAllester in [11], though, have the draw-back that the result of the analysis, i.e., the least model of the specification is necessarily finite.

Here, we try to lift this limitation. A classical approach uses (classes of) set constraints [8, 4, 17, 2]. As a more flexible formalism, Frühwirth et al. proposed a syntactical restriction of Horn clauses which they called *uniform*, guaranteeing that the least model consists of *recognizable sets* of trees and, moreover, can be effectively computed [6]. Recall that a set of trees is *recognizable* iff it can be accepted by a finite tree automaton. In essence, Frühwirth et al. consider unary predicates only and restrict implications to be of one of the following forms:

$$\begin{array}{lcl} p(X) & \Leftarrow & p_1(t_1), \ldots, p_m(t_m) \\ p(a) & \Leftarrow & p_1(t_1), \ldots, p_m(t_m) \\ p(t) & \Leftarrow & p_1(X_1), \ldots, p_n(X_n) \end{array}$$

where a is an atom, t is linear, and t_i are arbitrary terms. So, the implications

$$\begin{array}{lcll} p(a(X,Y)) & \Leftarrow & q(b(X,Y)) & \text{(constructor renaming)} \\ p(a(X,Z)) & \Leftarrow & q_1(a(X,Y)), q_2(a(Y,Z)) & \text{(composition)} \end{array}$$

are *not* uniform. Uniform Horn clauses (like set constraints) are *too weak* since they are not well suited for dealing with *relations*. Neither do they conveniently support Cartesian product, nor transitive closure or projections onto more than just one component. On the other hand, set constraints (like uniform Horn clauses) in general are *too strong* a formalism since computing their least solutions easily becomes exponential-time hard [6, 18, 4].

Here, we follow Frühwirth et al. in [6] by using (subclasses of) ordinary Horn clauses as specifications of analyses and *generalize* it by allowing non-unary predicates. Clearly, without any restrictions the resulting relations will be neither finite nor effectively computable. In order to obtain effective descriptions for relations, we restrict ourselves to *strongly recognizable* relations which we define to be *finite unions of Cartesian products* of recognizable tree sets. We exhibit a rich class of Horn clauses where the least model consists of strongly recognizable tree relations only and present an exponential-time normalization procedure to compute least models. This class, which we call $\mathcal{H}1$, allows renaming and composition as defined above. Furthermore, it conveniently supports relational operations like Cartesian product, transitive closure and arbitrary projections.

As $\mathcal{H}1$ can express unrestricted intersection of recognizable tree sets by a formula of polynomial size, computing the least model of $\mathcal{H}1$ clauses is hard for deterministic exponential time [18]. Thus, the class $\mathcal{H}1$ has the same efficiency problems as set constraints or uniform Horn clauses. Therefore, we exhibit in section 4 a large subclass which we call $\mathcal{H}2$. This subclass still supports, e.g., constructor renaming, Cartesian product, transitive closure and arbitrary projections. The complexity of $\mathcal{H}2$ is polynomial: the exponent, however, depends on the number of variables in individual implications. Therefore, we exhibit in section 5 a subclass $\mathcal{H}3$ of $\mathcal{H}2$ for which a uniformly cubic solving time can be proven. The normalization procedure for $\mathcal{H}3$ then is used in section 6 to obtain a control-flow analysis for the Spi calculus [1] as presented in [14] which is cubic.

2 Basics

A Horn clause is a finite set of *implications*. Every implication r is of the form $h \Leftarrow \alpha$ where h and α are the *head* and the *pre-condition* of r, respectively. Every pre-condition equals a sequence of *queries*. The head and every query are *literals*, i.e., of the form $p(t_1, \ldots, t_k)$ where p is a k-ary predicate and t_i

are terms which are built up from variables (starting with capital letters) and constants through constructor applications. For convenience, we also introduce the abbreviation: $h_1, \ldots, h_n \Leftarrow \alpha$ for the set of implications:

$$h_1 \Leftarrow \alpha \quad \ldots \quad h_n \Leftarrow \alpha$$

all of which share the same pre-condition α. For a signature Σ of constants and constructors, we denote the set of all finite variable-free (*ground*) terms by $\mathcal{T}_\Sigma$. A k-ary *tree relation* is a subset of $\mathcal{T}_\Sigma^k$. Horn clauses are interpreted relative to an interpretation ρ which maps predicate symbols to corresponding tree relations.

Let ρ denote an interpretation of occurring predicates, and θ denote a ground substitution mapping the occurring variables to ground terms. Relative to ρ and θ, we define for literals g, pre-conditions α, implications $h \Leftarrow \alpha$ and Horn clauses c, the *satisfaction* relation "$\models$" by:

(1)	$\rho, \theta \models p(t_1, \ldots, t_k)$	iff	$\theta(t_1, \ldots, t_k) \in \rho(p)$
(2)	$\rho, \theta \models g_1, \ldots, g_m$	iff	$\forall\, j :\ \rho, \theta \models g_j$
(3)	$\rho \quad \models h \Leftarrow \alpha$	iff	$\forall\, \theta :\ \rho, \theta \models \alpha \ \Rightarrow\ \rho, \theta \models h$
(4)	$\rho \quad \models c$	iff	$\rho \models r$ for all implications r in c

An interpretation ρ with $\rho \models c$ is also called *model* of c. It is well-known that the set of models of a Horn clause forms a Moore family. Therefore, there is a unique least model.

3 Normalizable Horn Clauses

Unrestricted Horn clauses have Turing power. In order to arrive at a decidable fragment, we have to impose restrictions. Horn clauses define tree relations. A k-ary tree relation R is *recognizable* iff the set of tuples $(t_1, \ldots, t_k) \in R$ can be characterized by a finite tree automaton running on all k trees simultaneously. Recognizable tree relations have been introduced by Lugiez and Schnoebelen for model-checking of PA processes and deciding first-order transition logic [10]. Recognizable relations enjoy many useful closure properties. In particular, they contain all finite relations and are closed under boolean operations, projection, Cartesian product and composition. They are in general not closed, though, under transitive closure.

Therefore, we consider a subclass of recognizable tree relations only. We call a tree relation $R \subseteq \mathcal{T}_\Sigma^k$ *strongly recognizable* iff R equals a finite union of k-fold Cartesian products of recognizable sets of trees. In particular, unary strongly recognizable tree relations simply are recognizable *sets* of trees. Not every recognizable tree relation, though, is also strongly recognizable. The simplest counter example is the identity relation which is trivially recognizable but not strongly recognizable (unless the universe is finite). Like recognizable relations, strongly recognizable relations are closed under boolean operations, projection, Cartesian product and composition (see chapter 3, section 2 of [5]). As a subclass, strongly recognizable relations enjoy stronger closure properties than general recognizable relations. Here, we show that the transitive closure of a binary strongly recognizable relation is again strongly recognizable (see corollary 1 below). In terms of program analysis, the so-called *independent attribute method* [9] is often taken to be a Cartesian product of powersets of values whereas the so-called *relational method* [9] often is taken to be a powerset of a Cartesian product of values; in this sense, our notion of *strongly recognizable relations* constitutes a "finitary relational method" that is more versatile than the independent attribute method and specializes to the relational method in the case of a finite universe.

We call an implication *normal* iff it is of one of the forms:

$$
\begin{array}{lll}
(N1) & p(b) & \Leftarrow \\
(N2) & p(f(X_1,\ldots,X_k)) & \Leftarrow \quad q_1(X_1),\ldots,q_k(X_k) \\
(N3) & p(X_1,\ldots,X_k) & \Leftarrow \quad q_1(X_1),\ldots,q_k(X_k)
\end{array}
$$

for distinct variables X_i. In the following, we subsume the first case into the second one by allowing also constructors f of arity 0. Also, we call a Horn clause c *normal* iff every implication occurring in c is normal. Every strongly recognizable relation can be described through a normal Horn clause. In fact, the reverse is also true:

Proposition 1. *Assume c is a normal Horn clause. Then the least model of c maps every occurring predicate to a strongly recognizable relation.* □

We call a Horn clause c *normalizable* iff a normal Horn clause c' can be constructed from c which is equivalent to c (up to auxiliary relations). We are interested in large classes of normalizable Horn clauses in order to allow as much flexibility as possible when specifying analyses. In order to characterize such classes, we introduce the following notion. To a sequence α of queries, we associate the (undirected) *variable dependence graph* $D_\alpha = (V_\alpha, E_\alpha)$. The set of vertices of D_α equals the set of queries g occurring in α, whereas the set of edges is given by $\{g_1, g_2\} \in E_\alpha$ iff $g_1 \neq g_2$ and $\mathit{Vars}(g_1) \cap \mathit{Vars}(g_2) \neq \emptyset$. Two variables X_1, X_2 are *connected* (w.r.t. α) iff they occur within queries which are connected in D_α. In particular, variables within the same query are connected.

The implication $h \Leftarrow \alpha$ has property H1 iff

(1) h is linear, i.e., no variable occurs twice in h.
(2) If two variables X, Y in h are connected, then X and Y are siblings in h.

Here, we call two variables *siblings* in a literal or term if they occur as arguments of a common father. So, X, Y are siblings in $p(X,Y)$ and in $p(a(X,b,Y))$ but not siblings in $p(X, a(Y))$. A Horn clause c belongs to the class $\mathcal{H}1$ iff all implications in c are H1.

For the following, we fix a finite set Σ of constructors. In particular, the maximal arity $a[c]$ of a constructor or predicate in the clause c is non-zero but $\mathcal{O}(1)$. We *assume* in our complexity estimations that the maximal number of queries as well as the maximal number $v[c]$ of variable occurrences in a single implication is non-zero and $\mathcal{O}(1)$. Note that this does neither preclude the *number* of implications of c nor the *sizes* of occurring ground subterms to be unbounded. The maximum of 2 and the maximal number of repetitions of a variable in a pre-condition of c is denoted by $r[c]$. Also, we will refer to the *size* $|c|$ of c which we define as the size of a "natural" internal representation of c. For this representation, we use a condensed version of the abstract syntax tree of the clause where ground subterms are maximally shared. We have:

Theorem 1. *For every clause from $\mathcal{H}1$, an equivalent normal clause can be constructed in deterministic exponential time.*

Proof (Sketch). We proceed in two steps. In the first step, we transform the $\mathcal{H}1$-clause into an equivalent clause of a particularly simple form which we then normalize in the second step.

A clause from $\mathcal{H}1$ is *$\mathcal{H}1$-special* iff every implication is of one of the forms:

$$
\begin{array}{lll}
(1) & p(X_1, \ldots, X_k) & \Leftarrow q_1(\underline{Y}_1), \ldots, q_m(\underline{Y}_m) \\
(2) & p(f(X_1, \ldots, X_k)) & \Leftarrow q_1(\underline{Y}_1), \ldots, q_m(\underline{Y}_m) \\
(3) & p(X_1, \ldots, X_k) & \Leftarrow q(Y_1, \ldots, Y_{j-1}, f(Z_1, \ldots, Z_{n'}), Y_{j+1}, \ldots, Y_n), \mathbf{p}
\end{array}
$$

where the variables X_i are pairwise distinct, the $\underline{Y}_i$ are arbitrary sequences of variables, and $\mathbf{p}$ is a sequence of unary queries. We have:

Proposition 2. *For a Horn clause c from $\mathcal{H}1$, a $\mathcal{H}1$-special Horn clause c' can be constructed in linear time which is equivalent to c (up to auxiliary predicates) such that c' has the following properties:*

1. $a[c'] \leq v[c] + a[c] - 1$ *and* $v[c'] \leq v[c] + a[c] - 1$*;*
2. $r[c'] \leq r[c]$*.*

Thus, the transformation from c to c' only moderately increases the number of variable occurrences per implication while not increasing the number of variable repetitions. Here, we do not provide a formal proof of prop. 2. Instead, we illustrate the two crucial steps by examples. First, we need a (linear time) transformation which produces an equivalent Horn clause such that all occurring heads are of one of the following forms:

$$p(X_1, \ldots, X_k) \qquad \text{or} \qquad p(f(X_1, \ldots, X_k))$$

Example 1. Consider the implication: $\quad p(a(X, Y, b), c(Z)) \quad \Leftarrow \quad \alpha$.

By definition of H1, neither X and Z nor Y and Z can be connected. Therefore, we can split α into two sequences α_1, α_2 of queries where α_1 contains all queries containing variables which are connected to X or Y and α_2 contains the remaining ones (in particular, those constraining Z). We replace the implication equivalently with the following four implications:

$$
\begin{array}{lclcl}
p(X_1, X_2) & \Leftarrow & p_1(X_1), p_2(X_2) & h(b) & \Leftarrow \\
p_1(a(X, Y, Z)) & \Leftarrow & h(Z), \alpha_1 & p_2(c(Z)) & \Leftarrow \quad \alpha_2
\end{array}
$$

The first implication separates the implication into two queries, one for each argument of p. The following two implications construct the first parameter of p. Each of these create one constructor in the head. The second implication corresponds to the application of a. The bindings for X and Y are provided by α_1, whereas Z is a new auxiliary variable which receives the ground term b through the query to the additional auxiliary predicate h. Finally, the second parameter of p is provided through the fourth implication. □

Next, we simulate complex queries by simpler ones.

Example 2. Consider the complex query $g \equiv p(a(c(X), X), d(Y))$ occurring in some pre-condition. Then g is replaced with the query $p_0(X, X, Y)$ where p_0 is a new predicate with one parameter position for every variable occurrence of g. The predicate p_0 is then defined through the following traversal over g:

$$\begin{array}{lcl} p_0(X_1, X_2, X_3) & \Leftarrow & p_1(c(X_1), X_2, X_3) \\ p_1(X_1, X_2, X_3) & \Leftarrow & p_2(a(X_1, X_2), X_3) \\ p_2(X_1, X_2) & \Leftarrow & p(X_1, d(X_2)) \end{array}$$

Each of the defining implications corresponds to one constructor occurrence in g. The implication for p_1, e.g., inverts the constructor occurrence of a. □

According to proposition 2, it suffices to normalize $\mathcal{H}1$-special clauses only. Assume that c is $\mathcal{H}1$-special. For every subset S of unary predicates, we introduce a new auxiliary unary predicate m_S. In particular, we rename the unary predicates p with $m_{\{p\}}$. The predicate m_S denotes the conjunction of the predicates $p \in S$. The predicate $m_\emptyset$ denotes the empty intersection, i.e., should be true for all trees. For $S = \emptyset$ and every f of rank k, we hence add the rule:

$$m_\emptyset(f(X_1, \ldots, X_k)) \quad \Leftarrow \quad m_\emptyset(X_1), \ldots, m_\emptyset(X_k)$$

By adding queries to $m_\emptyset$, we may w.l.o.g. assume that all head variables also occur in the corresponding pre-conditions.

The general idea of the normalization procedure is to successively add simpler implications until no further simplifications can be applied. Two properties must be enforced. First, the newly added implications should always be implied by the already existing ones. Second, after saturation, all non-normal implications should become superfluous. In the following, we collect the rules for adding new implications. Assume that $\mathcal{H}$ is the least model of the currently obtained set of *normal* implications. Note that it will grow monotonically as clauses are added.

First, consider *chain rules*, i.e., implications of the form:

$$m_S(X) \quad \Leftarrow \quad m_{S'}(X)$$

Then for every normal implication:

$$m_{S'}(f(X_1, \ldots, X_k)) \quad \Leftarrow \quad m_{S_1}(X_1), \ldots, m_{S_k}(X_k)$$

we add the implication:

$$m_S(f(X_1, \ldots, X_k)) \quad \Leftarrow \quad m_{S_1}(X_1), \ldots, m_{S_k}(X_k)$$

If we are given normal implications:

$$m_{\{p\}}(f(X_1, \ldots, X_k)) \quad \Leftarrow \quad m_{S_{p,1}}(X_1), \ldots, m_{S_{p,k}}(X_k)$$

for all $p \in S$, we add the normal implication:

$$m_S(f(X_1, \ldots, X_k)) \quad \Leftarrow \quad m_{S_1}(X_1), \ldots, m_{S_k}(X_k)$$

where $S_j = \bigcup\{S_{p,j} \mid p \in S\}$, i.e., S_j collects all predicates constraining X_j.

Next, assume that the implication is of form (1) or (2) and contains a non-unary query, i.e., equals

$$h \quad \Leftarrow \quad \mathbf{p}_1, q(Y_1, \ldots, Y_n), \mathbf{p}_2$$

$(n \neq 1)$ where h either equals $p(X_1, \ldots, X_k)$ or $p(f(X_1, \ldots, X_k))$. If q is nullary, i.e., $n = 0$, then we add the implication:

$$h \quad \Leftarrow \quad \mathbf{p}_1, \mathbf{p}_2$$

whenever $\mathcal{H}(q)$ is non-empty. If q has arity at least 2, then for every implication:

$$q(X_1, \ldots, X_n) \quad \Leftarrow \quad m_{S_1}(X_1), \ldots, m_{S_n}(X_n)$$

we add the implication:

$$h \ \Leftarrow \ \mathbf{p}_1, m_{S_1}(Y_1), \ldots, m_{S_n}(Y_n), \mathbf{p}_2$$

Now assume that the pre-condition of an implication consists of unary queries only, i.e., the implication equals $\quad h \ \Leftarrow \ \mathbf{p} \quad$ where $\mathbf{p}$ is a sequence of unary queries. By assumption, all head variables $X_1, \ldots, X_k$ occur in $\mathbf{p}$. We simplify $\mathbf{p}$ in two steps. First, we join together all unary predicates constraining the same variable: If $\mathbf{p}$ is (up to re-ordering) of the form $\quad m_{S_1}(X), \ldots, m_{S_n}(X), \mathbf{p}' \quad$ (X some variable not occurring in $\mathbf{p}'$), we add the implication:

$$h \quad \Leftarrow \quad m_S(X), \mathbf{p}'$$

where $S = S_1 \cup \ldots \cup S_n$.

Next assume that every variable occurs at most once. We aim at removing the non-head variables. Assume that $\mathbf{p}$ is (up to re-ordering) of the form:

$$m_{S'_1}(Y_1), \ldots, m_{S'_l}(Y_l), m_{S_1}(X_1), \ldots, m_{S_k}(X_k)$$

where $Y_1, \ldots, Y_l$ do not occur in the head. If all sets $\mathcal{H}(m_{S'_i})$ are non-empty, then we add the clause:

$$h \quad \Leftarrow \quad m_{S_1}(X_1), \ldots, m_{S_k}(X_k)$$

Finally, we consider non-normal implications of the form (3), i.e.:

$$p(X_1, \ldots, X_k) \ \Leftarrow \ q(Y_1, \ldots, Y_{j-1}, f(Z_1, \ldots, Z_{n'}), Y_{j+1}, \ldots, Y_n), \mathbf{p}$$

and assume that the clause contains the normal implications:

$$\begin{array}{lcl} q(X_1, \ldots, X_n) & \Leftarrow & m_{S_1}(X_1), \ldots, m_{S_j}(X_j), \ldots, m_{S_n}(X_n) \\ m_{S_j}(f(X_1, \ldots, X_{n'})) & \Leftarrow & m_{S'_1}(X_1), \ldots, m_{S'_{n'}}(X_{n'}) \end{array}$$

Then we add the implication:

$$\begin{array}{lcl} p(X_1, \ldots, X_k) & \Leftarrow & m_{S_1}(Y_1), \ldots, m_{S_{j-1}}(Y_{j-1}), \\ & & m_{S'_1}(X_1), \ldots, m_{S'_{n'}}(Z_{n'}), m_{S_{j+1}}(Y_{j+1}), \ldots, m_{S_n}(Y_n), \mathbf{p} \end{array}$$

Each newly added rule does not change the least model of the Horn clause, and $\mathcal{H}$ will continue to be a subset of the least model. The number of distinct normal implications is $2^{\mathcal{O}(|c|)}$. We conclude that also the maximal number of normalization steps for $\mathcal{H}1$-special Horn clauses is $2^{\mathcal{O}(|c|)}$. Now, let c_1 denote a clause which is obtained from c by a maximal sequence of normalization steps and c_0 obtained from c_1 by removing the non-normal clauses. We claim that c_0 and c_1 have the same least models. In order to prove that the equivalent normal Horn clause c_0 can be constructed in deterministic exponential time, we observe that each normalization step can be executed in constant time – given an oracle for non-emptiness of occurring unary predicates. Therefore, it remains to prove that the necessary tests for non-emptiness can be executed efficiently enough. For this we, e.g., may maintain a boolean flag b_S for every encountered set S of unary predicates indicating whether the intersection of the predicates in S has already been shown to be non-empty. This book-keeping amounts to overall costs $2^{\mathcal{O}(|c|)}$. Thus, normalization of $\mathcal{H}1$ clauses can be performed in time $2^{\mathcal{O}(|c|)}$. □

Note that our normalization proof is completely straight-forward — quite in contrast to the corresponding proof for uniform Horn clauses in [6] which refers to *two-way tree automata*. As every Horn clause from $\mathcal{H}1$ can be normalized, it might be argued that $\mathcal{H}1$ has the same expressive power for unary predicates as simpler formalisms like finite tree automata, definite set constraints [8, 4, 17, 2] or uniform Horn clauses. From a practical point of view, however, this is not true. $\mathcal{H}1$ is a more convenient specification formalism as it supports manipulation of *relations*. It allows, e.g., to state implications like:

- $p(X, Y) \Leftarrow q(a(X, Y, Z))$
- $p(X, Y, Z) \Leftarrow q(Y, Z, X)$
- $p(X, Z) \Leftarrow q_1(X, Y), q_2(Y, Z)$

On the other hand, arbitrary projections through constructors, permutations of components and composition of relations as in these examples are not expressible through uniform Horn clauses.

Let us finally remark that the the exponential upper complexity bound which we established for the normalization of $\mathcal{H}1$ clauses is the best one can hope to achieve for such a general class. Any fast normalization procedure allows to construct a fast procedure for testing the emptiness of intersections of sequences of recognizable tree languages (given through finite tree automata) — the latter problem, however, is known to be hard for deterministic exponential time [6, 18]. Note that this lower bound even holds for the family of $\mathcal{H}1$ clauses where all implications are of constant size (while the number of implications varies).

4 The Class $\mathcal{H}2$

In this section, we try to break the exponential lower complexity bound by exhibiting a large subclass of $\mathcal{H}1$ clauses which can be normalized much faster. Our key idea is to additionally restrict *how* pre-conditions may affect head variables.

We call $h \Leftarrow \alpha$ H2 iff it is H1, i.e., satisfies (1) and (2) above and additionally:

(3) Every variable which occurs in h, occurs in α at most once.

A Horn clause c is $\mathcal{H}2$ iff all implications in c are H2.

Example 3. The implication: $p(X,Y) \quad \Leftarrow \quad q_1(a(X,Z),b), q_2(b(Z)), q_3(c,X)$ is H1 but not H2 as the variable X occurs in the pre-condition twice. On the contrary, some of the implications generated by the flow-logic specification [12, 15] for the Spi calculus, are H2. Let us assume that the program to be analyzed is specified by the abstract syntax tree prog. Then we maintain unary predicates *reach* and *occurs* which collect all reachable sub-programs and all reachable (data) subexpressions, respectively. In particular, we assert the fact: $\mathit{reach}(\mathsf{prog}) \quad \Leftarrow$. In the abstract syntax tree, we use a constructor $\mathsf{decrypt}$ of arity 4 for collecting the four subtrees corresponding to the parts e, x, k and t in the sub-program: **case** e **of** $\{x\}_k\ t$. The rules of the control-flow analysis (including reachability) from [14] for decryption can be formalized as:

$$\begin{array}{lcl} \mathit{hasValue}(X,Y), \mathit{reach}(T) & \Leftarrow & \mathit{reach}(\mathsf{decrypt}(E,X,K,T)), \\ & & \mathit{hasValue}(K,V), \\ & & \mathit{hasValue}(E, \mathsf{enc}(Y,V)) \\ \mathit{occurs}(E), \mathit{occurs}(K) & \Leftarrow & \mathit{reach}(\mathsf{decrypt}(E,X,K,T)) \end{array}$$

Here, enc is another constructor which creates encrypted messages from a content and a key. All these four implications are H2. □

Theorem 2. *For every $\mathcal{H}2$ clause c, an equivalent normal clause c' can be constructed in time $\mathcal{O}(|c|^m)$ where $m = r[c] \cdot (v[c] + a[c])$.*

Proof (Sketch). We proceed as in the proof of theorem 1. First, we bring c into $\mathcal{H}1$-special form. In fact, the resulting clause c' still is $\mathcal{H}2$. Since every head variable occurs in the corresponding pre-condition at most once, we can abandon the use of any auxiliary predicate m_S where S has cardinality > 1 – given that we replace the emptiness checks of $\mathcal{H}(m_S)$ (S a set of unary predicates) during $\mathcal{H}1$-normalization with the emptiness test for: $\bigcap\{\mathcal{H}(p) \mid p \in S\}$. In particular, the occurring sets S to be tested have cardinality at most $r[c]$. Therefore, we can construct for c', an equivalent normal clause c' of size $\mathcal{O}(|c|^v)$ where the number of normalization steps is of order $\mathcal{O}(|c|^{v+1})$ for $v = v[c] + a[c]$. All necessary tests for non-emptiness then amount to costs $\mathcal{O}(|c|^m)$ where $m = r[c] \cdot (v[c] + a[c])$. Hence, normalization can be performed in time $\mathcal{O}(|c|^m + |c|^{v+1}) = \mathcal{O}(|c|^m)$. □

As a corollary, we obtain a closure property of strongly recognizable relations which distinguishes this class from general recognizable relations:

Corollary 1. *The transitive closure of a strongly recognizable binary relation is again strongly recognizable. It can be computed in cubic time.*

Proof. Assume that the strongly recognizable binary relation edge is defined by the normal Horn clause c. Then we can define the transitive closure by:

$$\begin{array}{lcl} \mathsf{trans}(X,Y) & \Leftarrow & \mathsf{edge}(X,Y) \\ \mathsf{trans}(X,Z) & \Leftarrow & \mathsf{edge}(X,Y), \mathsf{trans}(Y,Z) \end{array}$$

The resulting Horn clause falls into the class $\mathcal{H}2$. Therefore, it is normalizable, and the defined binary relation trans is strongly recognizable.

Concerning the complexity, we observe that there are at most $|c|$ implications for the relation edge. Accordingly, the first implication for trans incurs only cost $\mathcal{O}(|c|)$. It remains to consider the second implication. The original clause contains at most $|c|$ unary predicates. Since trans is binary, the normalized clause can contain at most $|c|^2$ implications for trans. Each of the linearly many pre-conditions of edge potentially must be combined with each of the pre-conditions of trans. This results in at most $\mathcal{O}(|c|^3)$ normalization steps. The emptiness of all pairwise intersections of unary predicates can be determined in time $\mathcal{O}(|c|^2)$. We conclude that normalization requires time $\mathcal{O}(|c|^3 + |c|^2) = \mathcal{O}(|c|^3)$. □

5 The Class $\mathcal{H}3$

In light of theorem 2, the class $\mathcal{H}2$ is (theoretically) feasible – given that every implication does not contain too many variables and variable repetitions. In our example analysis for Spi, we used implications which introduce as many as 9 variable occurrences of 6 distinct variables. The resulting exponent ($\geq 9+2+1 = 12$) is way beyond what can honestly be judged as *practically* feasible for non-trivial input clauses. Therefore, we see a need for subclasses of $\mathcal{H}2$ where the exponent in the runtime estimation of normalization *uniformly* can be bounded by a small constant. Such a class is exhibited in this section.

The idea of this third class of Horn clauses is to allow tree-like variable dependences only. Thus, $h \Leftarrow \alpha$ is called H3 iff (4) and (5) are satisfied:

(4) h and every literal in α is linear;
(5) The variable dependence graph for the sequence $h\alpha$ is acyclic where adjacent literals have at most one variable in common.

Hence, every variable occurs at most twice and in different literals. Therefore, variables from h occur in α at most once. Since (4) and (5) imply property H1, we conclude that H3 implies even property H2. A Horn clause c is called $\mathcal{H}3$ iff all implications in c are H3. The clauses from the Spi analysis from example 3, though, do *not* conform with the definition of H3 — as the implication for *hasValue* violates property (5). We will return to this point in the next section.

Theorem 3. *For every $\mathcal{H}3$ clause c, an equivalent normal clause can be constructed in time $\mathcal{O}(|c|^3)$ where the resulting clause is of size $\mathcal{O}(|c|^2)$.*

Proof. As in the proof of theorem 2, we proceed by first transforming the clause into an equivalent clause of simpler form – which then is normalized in the second step. Note that the normal form here and the corresponding normalization rules are quite different from the ones used in the proofs of theorems 1 and 2.

An $\mathcal{H}3$ clause c is $\mathcal{H}3$-special iff every non-normal implication in c is of one of the forms:

$$
\begin{array}{llcl}
(1) & p(X_1,\ldots,X_k) & \Leftarrow & q_1(X_1),\ldots,q_k(X_k),\mathbf{q} \\
(2) & p(X_i) & \Leftarrow & q(f(X_1,\ldots,X_k)),\mathbf{p} \\
(3) & p(X_i) & \Leftarrow & q(X_1,\ldots,X_k),\mathbf{p} \\
(4) & p() & \Leftarrow & q(X_1,\ldots,X_k),p_1(X_1),\ldots,p_k(X_k)
\end{array}
$$

where $\mathbf{p}$ in lines (2) and (3) denotes a sequence of unary queries

$$\mathbf{p} \quad \equiv \quad p_1(X_1),\ldots,p_{i-1}(X_{i-1}),p_{i+1}(X_{i+1}),\ldots,p_k(X_k)$$

and $\mathbf{q}$ in line (1) is a (possibly empty) sequence of nullary queries. We have:

Proposition 3. *For a Horn clause c from $\mathcal{H}3$, a $\mathcal{H}3$-special Horn clause c' can be constructed in linear time which is equivalent to c (up to auxiliary predicates).*

Proof. In order to decompose an arbitrary $\mathcal{H}3$ implication $h \Leftarrow \alpha$ into (a linear number of) $\mathcal{H}3$-special implications, we consider the dependence graph D_α of the pre-condition α. As this graph is acyclic, we can *direct* its edges in such a way that the resulting graph D'_α has the following properties:

- D'_α is a forest with edges directed towards the roots;
- every head variable only occurs in roots.

Such an edge orientation can be determined by a DFS traversal of the graph D_α. The occurrence of a head variable in a root is considered as the *definition* of potential bindings of the variable. According to the property H3, each variable X which occurs more than once in α, corresponds to a unique edge (g_1, g_2) in D'_α from g_1 to g_2. The occurrence of X in the literal g_2 is the *use* of X whereas the occurrence of X in the literal g_1 is the *definition* of potential bindings of X.

Example 4. Consider the following implication:

$$p(a(X, Y)) \quad \Leftarrow \quad q_1(d(Z, c(X))), q_2(Y), q_3(Z), s(b, b)$$

The corresponding directed graph D'_α is shown in fig. 1. In this figure, we added

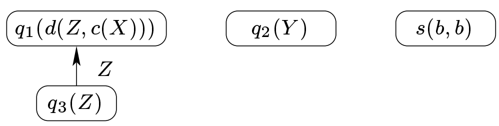

Fig. 1. The directed dependence graph D'_α.

to the single edge the variable by which it is induced. Note that there is one root besides those containing head variables, namely the literal $s(b, b)$. □

Assuming that the variables of distinct implications all are distinct, the preprocessing transformation proceeds as follows:

- For every subterm t of c, we introduce a new predicate h_t. If $t \equiv f(t_1, \ldots, t_k)$, the predicate h_t is defined by the implication:

$$h_t(f(Z_1, \ldots, Z_k)) \quad \Leftarrow \quad h_{t_1}(Z_1), \ldots, h_{t_k}(Z_k)$$

- For every root $g \equiv q(t_1, \ldots, t_k)$ of a pre-condition which does not contain a head variable, we introduce a new nullary predicate h'_g which is defined by the implication:

$$h'_g() \quad \Leftarrow \quad q(X_1, \ldots, X_k), h_{t_1}(X_1), \ldots, h_{t_k}(X_k)$$

– For an implication $p(t_1, \ldots, t_k) \Leftarrow \alpha$, we introduce the implication:

$$p(Z_1, \ldots, Z_k) \quad \Leftarrow \quad h_{t_1}(Z_1), \ldots, h_{t_k}(Z_k), \mathbf{q}$$

where $\mathbf{q}$ is the sequence of queries $h'_g()$, g a query of α without head variable.

– It remains to define the predicates h_X. If X occurs only once, we set:

$$h_X(Z) \quad \Leftarrow \quad h_\top(Z)$$

where $h_\top$ is a unary predicate which holds for all trees t. Since the set of constructors is fixed, $h_\top$ can be defined by a normal clause of size $\mathcal{O}(1)$. Every other variable X has a defining occurrence in some query g in some pre-condition. If $g \equiv q(X)$, then we add the implication:

$$h_X(Z) \quad \Leftarrow \quad q(Z)$$

Otherwise, if $g \equiv q(s_1, \ldots, s_m)$, we introduce new unary predicates g_s for every super-term s of X occurring in g together with the implications:

$$\begin{array}{lcl}
h_X(Z) & \Leftarrow & g_X(Z) \\
g_{s_i}(Z_i) & \Leftarrow & q(Z_1, \ldots, Z_m), h_{s_1}(Z_1), \ldots, h_{s_{i-1}}(Z_{i-1}), h_{s_{i+1}}(Z_{i+1}), \ldots, h_{s_m}(Z_m) \\
 & & \text{if } X \text{ occurs in } s_i \\
g_{s'_j}(Z_j) & \Leftarrow & g_{s'}(f(Z_1, \ldots, Z_{m'})), h_{s'_1}(Z_1), \ldots, h_{s'_{j-1}}(Z_{j-1}), h_{s'_{j+1}}(Z_{j+1}), \ldots, h_{s'_{m'}}(Z_{m'}) \\
 & & \text{if } s' \equiv f(s'_1, \ldots, s'_{m'}) \text{ occurs in } g \text{ and } X \text{ occurs in } s'_j
\end{array}$$

Example 5. Applying these rules to the implication from example 4, we obtain:

$$\begin{array}{lcllcl}
h_{a(X,Y)}(a(X', Y')) & \Leftarrow & h_X(X'), h_Y(Y') & h_Y(Y') & \Leftarrow & q_2(Y') \\
h_X(X') & \Leftarrow & g_X(X') & h_Z(Z') & \Leftarrow & q_3(Z') \\
h_b(b) & \Leftarrow & & & & \\
g_{c(X)}(X') & \Leftarrow & g_{d(Z,c(X))}(d(Z', X')), h_Z(Z') & & & \\
g_X(X') & \Leftarrow & g_{c(X)}(c(X')) & g_{d(Z,c(X))}(X') & \Leftarrow & q_1(X') \\
h'_{s(b,b)}() & \Leftarrow & s(X_1, X_2), h_b(X_1), h_b(X_2) & & & \\
p(X') & \Leftarrow & h_{a(X,Y)}(X'), h'_{s(b,b)}() & & &
\end{array}$$

where g denotes the query $\quad q_1(d(Z, c(X)))$. □

Indeed, this transformation introduces $\mathcal{H}3$-special implications only. Note also that every query may contain the defining occurrence of at most one variable. Therefore, the transformed clause is of size $\mathcal{O}(|c|)$. By fixpoint induction, it can be proven that the transformed clause is equivalent to c. Since the transformation can be implemented in linear time, the statement of proposition 3 follows. □

Given a linear time pre-processing phase, it therefore suffices to construct in time $\mathcal{O}(|c|^3)$, for every $\mathcal{H}3$-special Horn clause c, an equivalent normal clause of size $\mathcal{O}(|c|^2)$. Here, we only collect the necessary normalization rules.

Assume that $\mathcal{H}$ is the least model of the subset of normal implications in c. If an implication equals $p(X_1) \Leftarrow q_1(X_1)$, then for every normal implication:

$$q_1(f(X_1, \ldots, X_k)) \quad \Leftarrow \quad p_1(X_1), \ldots, p_k(X_k)$$

we add the implication:

$$p(f(X_1, \ldots, X_k)) \quad \Leftarrow \quad p_1(X_1), \ldots, p_k(X_k)$$

Now consider non-normal implications of the forms (1) through (4).
Case 1: The implication equals $p(X_1, \ldots, X_k) \quad \Leftarrow \quad q_1(X_1), \ldots, q_k(X_k), \mathbf{q}$ where $\mathbf{q} \equiv f_1(), \ldots, f_m()$ is a non-empty sequence of nullary queries. If $\mathcal{H}(f_j) \neq \emptyset$, for every j, then we add:

$$p(X_1, \ldots, X_k) \Leftarrow q_1(X_1), \ldots, q_k(X_k)$$

Case 2: The implication equals $p(X_i) \Leftarrow q(f(X_1, \ldots, X_k)), \mathbf{p}$ for a sequence

$$\mathbf{p} \quad \equiv \quad p_1(X_1), \ldots, p_{i-1}(X_{i-1}), p_{i+1}(X_{i+1}), \ldots, p_k(X_k)$$

Then for every normal implication:

$$q(f(X_1, \ldots, X_k)) \quad \Leftarrow \quad q_1(X_1), \ldots, q_k(X_k)$$

we add: $p(X_i) \Leftarrow q_i(X_i)$ — provided that $\mathcal{H}(p_j) \cap \mathcal{H}(q_j) \neq \emptyset$ for all $j \neq i$.
Case 3: The implication equals $p(X_i) \Leftarrow q(X_1, \ldots, X_k), \mathbf{p}$ for $k > 1$ and

$$\mathbf{p} \quad \equiv \quad p_1(X_1), \ldots, p_{i-1}(X_{i-1}), p_{i+1}(X_{i+1}), \ldots, p_k(X_k)$$

Then we proceed similar to *Case 2*. Thus, for every normal implication:

$$q(X_1, \ldots, X_k) \quad \Leftarrow \quad q_1(X_1), \ldots, q_k(X_k)$$

we add: $p(X_i) \Leftarrow q_i(X_i)$ — provided that $\mathcal{H}(p_j) \cap \mathcal{H}(q_j) \neq \emptyset$ for all $j \neq i$.
Case 4: The implication equals $p() \Leftarrow q(X_1, \ldots, X_k), p_1(X_1), \ldots, p_k(X_k)$. Then we add: $p() \Leftarrow \epsilon$ — provided that either $k = 1$ and $\mathcal{H}(q)$ and $\mathcal{H}(p_1)$ have a non-empty intersection or $k > 1$ and there is a normal implication:

$$q(X_1, \ldots, X_k) \quad \Leftarrow \quad q_1(X_1), \ldots, q_k(X_k)$$

such that $\mathcal{H}(q_i) \cap \mathcal{H}(p_i) \neq \emptyset$ for all i.

At most $|c|^2$ constraints of the form $p(X) \Leftarrow q(X)$ can be generated. Only $\mathcal{O}(|c|)$ normal constraints are generated for each predicate. So, the total number of generated constraints is $\mathcal{O}(|c|^2)$. Each original constraint as well as as each generated constraint of type (1) triggers at most $|c|$ normalization steps, which altogether can be implemented in time $\mathcal{O}(|c|^3)$ — given that non-emptiness of all (occurring) pairs of unary predicates can be decided in cubic time. The trivial upper bound for these tests is $\mathcal{O}(|c|^4)$. A more detailed analysis, however, reveals that time $\mathcal{O}(|c|^3)$ suffices. Thus, normalization of $\mathcal{H}3$ clauses is cubic. □

6 Application

We have seen that some implications naturally arising in the formalization of the Spi analysis, do not have property H3. It turns out that some further implications even violate H1 (see appendix A for a complete specification of the analysis).

Example 6. The evaluation rule of *hasValue* for the constructor enc is given by:

$$\begin{array}{lcl} \mathit{hasValue}(\mathsf{enc}(E,K),\mathsf{enc}(V,KV)) & \Leftarrow & \mathit{occurs}(\mathsf{enc}(E,K)), \\ & & \mathit{hasValue}(E,V), \\ & & \mathit{hasValue}(K,KV) \end{array}$$

This implication does not comply with H1, since the head variables E, K and V, KV are connected without being siblings. □

Each complicated implication of the Spi analysis, however, contains either a query to one of the predicates *reach* or *occurs*. A closer inspection reveals that both predicates hold only for *subterms* of the ground term prog. Therefore, we obtain an equivalent clause if we instantiate the variables in these queries in all ways such that the arguments of *reach* and *occurs* are subterms of prog.

Example 7. Consider the implications for *hasValue* in the examples 3 and 6. Instantiating the argument positions of *reach* and *occurs* results in the clauses:

$$\begin{array}{lcl} \mathit{hasValue}(x,Y) & \Leftarrow & \mathit{reach}(\mathsf{decrypt}(e,x,k,t)), \\ & & \mathit{hasValue}(k,V), \\ & & \mathit{hasValue}(e,\mathsf{enc}(Y,V)) \\ \mathit{hasValue}(\mathsf{enc}(e,k),\mathsf{enc}(V,VK)) & \Leftarrow & \mathit{occurs}(\mathsf{enc}(e,k)), \\ & & \mathit{hasValue}(e,V), \\ & & \mathit{hasValue}(k,KV) \end{array}$$

for all sub-programs $t' = \mathsf{decrypt}(e,x,k,t)$ and occurring expressions $\mathsf{enc}(y,k)$. Here, x, e, k and t are ground terms. Thus, the resulting implications are H3. □

This instantiation of the Spi analysis indeed results in H3 implications only. Since we use a succinct representation, multiple occurrences of the same ground subterm are represented only once. Therefore, the size of the clause is increased by a constant factor only. We conclude:

Theorem 4. *Control-flow analysis for Spi is cubic.* □

7 Conclusion

We presented a new class $\mathcal{H}1$ of Horn clauses whose least model can be computed exactly, and exhibited subclasses $\mathcal{H}2$ and $\mathcal{H}3$ where this can be done in polynomial time. Due to its expressivity, we find the class $\mathcal{H}1$ interesting in its own right. The most practical class, though, seems to be the class $\mathcal{H}3$, since it admits normalization in cubic time. We exemplified this by applying the normalization procedure for $\mathcal{H}3$ to construct a cubic time analysis for Spi. Since the "natural" formulation of the analysis did not meet the syntactical requirements of $\mathcal{H}3$, we massaged the original formulation by instantiating some parameters in the clause with finitely many ground subterms. The underlying idea can be generalized. It remains for future work to systematically explore corresponding clause transformations w.r.t. their strengths and complexities.

References

1. M. Abadi and A.D. Gordon. A Calculus for Cryptographic Protocols - The Spi Calculus. *Information and Computation*, 148:1–70, January 1999.
2. A. Aiken. Introduction to Set Constraint-Based Program Analysis. *Science of Computer Programming (SCP)*, 35(2):79–111, 1999.
3. D.A. Basin and H. Ganzinger. Complexity Analysis Based on Ordered Resolution. *Journal of the ACM*, 48(1):70–109, 2001.
4. W. Charatonik and A. Podelski. Set Constraints with Intersection. In *12th Ann. IEEE Symp. on Logic in Computer Science (LICS)*, 362–372, 1997.
5. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. Available on: http://www.grappa.univ-lille3.fr/tata, 1999.
6. T. Frühwirth, E. Shapiro, M. Vardi, and E. Yardeni. Logic Programs as Types of Logic Programs. In *6th Symp. on Logic in Computer Science (LICS)*, 300–309, 1991.
7. H. Ganzinger and D.A. McAllester. A New Meta-complexity Theorem for Bottom-Up Logic Programs. In *First Int. Joint Conference on Automated Reasoning (IJCAR)*, 514–528. LNCS 2083, 2001.
8. N. Heintze and J. Jaffar. A Decision Procedure for a Class of Set Constraints. In *5th Ann. IEEE Symp. on Logic in Computer Science (LICS)*, 42–51, 1990.
9. N.D. Jones and S.S. Muchnick. Complexity of Flow Analysis, Inductive Assertion Synthesis, and a Language due to Dijkstra. In Steven S. Muchnick and Neil D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 12, 380–393. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.
10. D. Lugiez and P. Schnoebelen. Decidable First-Order Transition Logics for PA-Processes. In *27th Int. Coll. on Automata, Languages and Programming (ICALP)*, 342–353. LNCS 1853, 2000.
11. D. McAllester. On the Complexity Analysis of Static Analyses. In *6th Static Analysis Symposium (SAS)*, 312–329. LNCS 1694, 1999.
12. F. Nielson, H. Riis Nielson, and C. L. Hankin. *Principles of Program Analysis*. Springer, 1999.
13. F. Nielson, H. Riis Nielson, and H. Seidl. Automatic Complexity Analysis. In *European Symposium on Programming (ESOP)*, 243–261. LNCS 2305, 2002.
14. F. Nielson, H. Riis Nielson, and H. Seidl. Cryptographic Analysis in Cubic Time. In *Electronic Notes in Theoretical Computer Science (ENTCS)*, volume 62. Elsevier Science Publishers, 2002.
15. F. Nielson and H. Seidl. Control-Flow Analysis in Cubic Time. In *European Symposium on Programming (ESOP)*, 252–268. LNCS 2028, 2001.
16. F. Nielson and H. Seidl. Succinct Solvers. Tech. Report 01-12, Trier, 2001.
17. L. Pacholski and A. Podelski. Set Constraints - a Pearl in Research on Constraints. In Gert Smolka, editor, *3rd Int. Conf. on Principles and Practice of Constraint Programming (CP)*, volume 1330 of *Springer LNCS*, 549–561. Springer-Verlag, 1997.
18. H. Seidl. Haskell Overloading is DEXPTIME Complete. *Information Processing Letters (IPL)*, 54:57–60, 1994.

A The Clauses for the Spi Analysis

For simplicity, we consider only *unary* encryptions and *unary* send and receive operations. Here are once again the implications for decryption:

$$\begin{array}{lcl} \mathit{hasValue}(X,Y), \mathit{reach}(T) & \Leftarrow & \mathit{reach}(\mathsf{decrypt}(E,X,K,T)), \\ & & \mathit{hasValue}(K,V), \\ & & \mathit{hasValue}(E,\mathsf{enc}(Y,V)) \\ \mathit{occurs}(E), \mathit{occurs}(K) & \Leftarrow & \mathit{reach}(\mathsf{decrypt}(E,X,K,T)) \end{array}$$

Similar rules model the remaining syntactical constructs. For send and recv, we obtain:

$$
\begin{array}{lcl}
\mathit{hasMessage}(CV, V), \mathit{reach}(T) & \Leftarrow & \mathit{reach}(\mathsf{send}(C, E, T)), \\
 & & \mathit{hasValue}(C, CV), \\
 & & \mathit{hasValue}(E, V) \\
\mathit{occurs}(C), \mathit{occurs}(E) & \Leftarrow & \mathit{reach}(\mathsf{send}(C, E, T)) \\
\mathit{hasValue}(X, V), \mathit{reach}(T) & \Leftarrow & \mathit{reach}(\mathsf{recv}(C, X, T)), \\
 & & \mathit{hasValue}(C, CV), \\
 & & \mathit{hasMessage}(CV, V) \\
\mathit{occurs}(C) & \Leftarrow & \mathit{reach}(\mathsf{recv}(C, \_, \_))
\end{array}
$$

For control constructs, we have:

$$
\begin{array}{lcl}
\mathit{reach}(T) & \Leftarrow & \mathit{reach}(\mathsf{bang}(T)) \\
\mathit{reach}(T_1), \mathit{reach}(T_2) & \Leftarrow & \mathit{reach}(\mathsf{par}(T_1, T_2)) \\
\mathit{reach}(T_0) & \Leftarrow & \mathit{reach}(\mathsf{case}(E, \_, T_0, \_)), \\
 & & \mathit{hasValue}(E, \mathsf{zero}) \\
\mathit{hasValue}(X, V), \mathit{reach}(T_1) & \Leftarrow & \mathit{reach}(\mathsf{case}(E, X, \_, T_1)), \\
 & & \mathit{hasValue}(E, \mathsf{succ}(V)) \\
\mathit{occurs}(E) & \Leftarrow & \mathit{reach}(\mathsf{case}(E, \_, \_, \_)) \\
\mathit{hasValue}(X_1, V_1), \mathit{hasValue}(X_2, V_2), \mathit{reach}(T) & \Leftarrow & \mathit{reach}(\mathsf{let}(E, X_1, X_2, T)), \\
 & & \mathit{hasValue}(E, \mathsf{pair}(V_1, V_2)) \\
\mathit{occurs}(E) & \Leftarrow & \mathit{reach}(\mathsf{let}(E, \_, \_, \_))
\end{array}
$$

Besides encryption and the atom zero, the Spi calculus has two further data constructors, the unary succ and the binary pair. The evaluation rules for these are given by:

$$
\begin{array}{lcl}
\mathit{occurs}(E) & \Leftarrow & \mathit{occurs}(\mathsf{succ}(E)) \\
\mathit{occurs}(E_1), \mathit{occurs}(E_2) & \Leftarrow & \mathit{occurs}(\mathsf{pair}(E_1, E_2)) \\
\mathit{occurs}(E), \mathit{occurs}(K) & \Leftarrow & \mathit{occurs}(\mathsf{enc}(E, K))
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{hasValue}(\mathsf{zero}, \mathsf{zero}) & \Leftarrow & \\
\mathit{hasValue}(\mathsf{succ}(E), \mathsf{succ}(V)) & \Leftarrow & \mathit{occurs}(\mathsf{succ}(E)), \\
 & & \mathit{hasValue}(E, V) \\
\mathit{hasValue}(\mathsf{pair}(E_1, E_2), \mathsf{pair}(V_1, V_2)) & \Leftarrow & \mathit{occurs}(\mathsf{pair}(E_1, E_2)), \\
 & & \mathit{hasValue}(E_1, V_1), \\
 & & \mathit{hasValue}(E_2, V_2) \\
\mathit{hasValue}(\mathsf{enc}(E, K), \mathsf{enc}(V, KV)) & \Leftarrow & \mathit{occurs}(\mathsf{enc}(E, K)), \\
 & & \mathit{hasValue}(E, V), \\
 & & \mathit{hasValue}(K, KV)
\end{array}
$$