Automatic Complexity Analysis

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

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

Abstract. We consider the problem of automating the derivation of tight asymptotic complexity bounds for solving Horn clauses. Clearly, the solving time crucially depends on the "sparseness" of the computed relations. Therefore, our asymptotic runtime analysis is accompanied by an asymptotic sparsity calculus together with an asymptotic sparsity analysis. The technical problem here is that least fixpoint iteration *fails* on asymptotic complexity expressions: the intuitive reason is that $O(1) + O(1) = O(1)$ but $O(1) + \cdots + O(1)$ may return any value.

Keywords: Program analysis, Horn clauses, automatic complexity analysis, sparseness.

1 Introduction

A program analyzer workbench should aid the analysis designer in the construction of efficient program analyses. In particular, the workbench has to provide a specification language in which the program properties to be analyzed can be conveniently formalized. Typically, the program analyzer generated from such a specification consists of a frontend for compiling the input program together with the specification into a constraint system which then is solved.

Here, we consider an approach where (some fragment of) predicate logic serves as a specification language for the analysis. Thus, we use predicates to represent program properties and Horn clause-like implications to formalize their inter-dependencies. The notion of predicates denoting relations is stronger than using just classical bit vectors or set constraints as provided by the BANE system [1] and makes the construction of control-flow analyses very easy (see [11, 12] for recent examples). There are three further reasons for the interest in an analyzer workbench based on this approach:

- The task of the frontend is reduced to the extraction of certain input relations from the program which then together with the clause specifying the analysis is supplied to the solver algorithm. Thus, it is possible to rapidly add new frontends for further languages to be analyzed.

- The task of computing the result is reduced to computing the desired model of a formula. As minor syntactical variations of formulas can have major impacts on the efficiency of solving, the task of tuning of the analysis boils down to tuning of formulas. Transformations along these lines were reported in [12] and are currently studied in the context of [13]. And finally,
- The generated program analyzers have *predictable* performance. This is the topic of the present paper.

Clearly, any good algorithm should be *predictable* – although only few are sufficiently well understood. Here, by predictability we mean two things. First, the algorithm should return the expected answers – this has classically been called *correctness*. But second, the algorithm also should return the answer in a reliable amount of time – meaning that the algorithm either always should be fast or, should allow an easy to understand classification of inputs into those which are rapidly doable and others which potentially take longer.

Our goal in this paper is to obtain safe estimations for the asymptotic complexities of the generated analyzers. For ease of presentation we explain our approach for a very simple fragment of predicate logic only, namely, for *Horn clauses*. For these, McAllester has presented a complexity meta-theorem [11] which reduces the complexity estimation for clause solving to counting of "prefix firings". These numbers, however, as well as practical clause solving times crucially depend on the "sparseness" of involved relations. Therefore, we develop an asymptotic sparsity calculus which formalizes this notion and allows to automate the necessary calculations. We use this calculus both to derive an automatic complexity estimator and also to design a sparsity analysis which infers asymptotic sparsity information for predicates for which no sparsity information has been provided by the user. This is particularly important for auxiliary predicates that are not part of the original formulation of the analysis but have been introduced during clause tuning (see, e.g., [12, 13] for an example).

The technical problem here stems from the observation that classical least fixpoint iteration fails on asymptotic expressions: $O(1)+O(1) = O(1)$ but $O(1)+\cdots+O(1)$ may return any value. We overcome this difficulty by relying on an interesting theorem about uniform finite bounds on the number of iterations needed for "nice" functions to reach their *greatest* fixpoints – even in presence of decreasing chains of unbounded lengths.

The paper is organized as follows. We introduce basic notions about Horn clauses in section 2. In section 3 we report on McAllester's complexity meta-theorem. In sections 4 and 5 we present the technical ideas onto which our analysis is based. In section 6 we explain the asymptotic sparsity analysis. In section 7 we sketch our implementation and present results for various benchmark clauses.

2 Horn Clauses

In this section, we recall the classical notion of Horn clauses (without function symbols) as our constraint formalism. A Horn clause is a conjunction of impli-

cations of the form:

$$g_1, \ldots, g_m \Rightarrow r(X_1, \ldots, X_k)$$

where $g_1, \ldots, g_m$ is a (possibly empty) list of assumptions and $r(X_1, \ldots, X_k)$ is the conclusion. W.l.o.g. we assume that the argument tuples of predicates in goals or conclusions are always given by *mutually distinct* variables. Thus, the goals g_i occurring as assumptions either are queries $s(Y_1, \ldots, Y_n)$ to predicates or equality constraints between variables or variables and constants:

$$g \quad ::= \quad s\,(Y_1, \ldots, Y_k) \quad | \quad X = Y \quad | \quad X = a$$

for variables $X, Y, Y_1, \ldots, Y_k$ and atoms a.

Horn clauses are interpreted over a universe $\mathcal{U}$ of atomic values (or atoms). For simplicity (and by confusing syntax and semantics here), we assume that all atoms occurring in the clause are contained in $\mathcal{U}$. Then given interpretations ρ and σ for predicate symbols and (a superset of) occurring variables, respectively, we define the satisfaction relation $(\rho, \sigma) \models t$ (t a goal or clause) as follows.

$$\begin{array}{lll}
(\rho,\sigma) \models r\,(X_1, \ldots, X_k) & \text{iff} & (\sigma\, X_1, \ldots, \sigma\, X_k) \in \rho\, r \\
(\rho,\sigma) \models X = Y & \text{iff} & \sigma\, X = \sigma\, Y \\
(\rho,\sigma) \models X = a & \text{iff} & \sigma\, X = a \\
(\rho,\sigma) \models g_1, \ldots, g_m \Rightarrow r(X_1, \ldots, X_k) & \text{iff} & (\sigma\, X_1, \ldots, \sigma\, X_k) \in \rho\, r \\
 & & \text{whenever } \forall i:\ (\rho,\sigma) \models g_i \\
(\rho,\sigma) \models c_1 \wedge \ldots \wedge c_n & \text{iff} & \forall j:\ (\rho,\sigma) \models c_j
\end{array}$$

In particular, we call an interpretation ρ of the predicate symbols in $\mathcal{R}$ a *solution* of c provided $(\rho, \sigma) \models c$ for all variable assignments σ of the free variables in c.

The set of all interpretations of predicate symbols in $\mathcal{R}$ over $\mathcal{U}$ forms a complete lattice (w.r.t. componentwise set inclusion on relations). It is well-known that the set of all solutions of a clause is a Moore family within this lattice. We conclude that for every clause c and interpretation ρ_0 there is a least solution ρ of c with $\rho_0 \sqsubseteq \rho$. An algorithm which, given c and ρ_0, computes ρ is called a *Horn clause solver*. In the practical application, e.g., of a program analyzer workbench, the initial interpretation ρ_0 assigns to input predicates relations which have been extracted from the program to be analyzed. Depending on the input predicates, the clause c (representing the analysis) defines certain output predicates which return the desired information about the program to be analyzed.

3 Concrete Calculation of Runtimes

In [11], McAllester proposes an efficient Horn clause solver. In particular, he determines the complexity of his algorithm by means of the number of *prefix firings* of the clause. Let ρ be an interpretation of the predicate symbols. Let $p \equiv g_1, \ldots, g_m$ denote a sequence of goals and $A = \mathit{Vars}(p)$ the set of variables occurring in p. Then the set $\mathcal{T}_\rho[p]$ of *firings* of p (relative to ρ) is the set of all variable assignments for which all goals g_i succeed. Thus, this set is given by:

$$\mathcal{T}_\rho[p] = \{\sigma : A \to \mathcal{U} \mid \forall i:\ (\sigma, \rho) \models g_i\}$$

In particular, if p is empty, then $Vars(p) = \emptyset$, and $\mathcal{T}_\rho[p]$ only consists of a single element, namely, the empty assignment (which we denote by $\emptyset$ as well). The set $\mathcal{F}_\rho[c]$ of *prefix firings* of a clause c is given by the set of all firings of prefixes of sequences of assumptions occurring in c.

Let us now without loss of generality assume that each implication $p \Rightarrow r(args)$ is *bottom-up bound*, i.e., each variable occurring in *args* also occurs in the list p of assumptions[1]. McAllester's result then can be stated as follows:

Theorem 1 (McAllester). *Let c be a Horn clause of size $\mathcal{O}(1)$ and ρ_0 an initial interpretation of the predicate symbols occurring in c. Then the least solution ρ of c with $\rho_0 \sqsubseteq \rho$ can be computed in time $\mathcal{O}(|\rho| + |\mathcal{F}_\rho[c]|)$, i.e., asymptotically equals the cardinality of ρ plus the number of prefix firings relative to ρ.* □

Thus, computing the complexity of Horn clause solving reduces to asymptotically counting the number of prefix firings. In simple applications, this can be done manually by looking at the clause and exploiting background knowledge about the occurring relations. In more complicated applications, however, this task quickly becomes tedious — making a mechanization of the asymptotic counting desirable. This is what we are going to do now.

We observe that computing sets of firings can be reduced to the application of a small set of operations on relations and sets of variable assignments: For sets of variables $A \subseteq V$, we define an *extension* operator $\mathsf{ext}_{A,V}$ which maps subsets $\mathcal{E} \subseteq A \to \mathcal{U}$ to subsets of $V \to \mathcal{U}$ by "padding" the variable assignments in all possible ways, i.e.:

$$\mathsf{ext}_{A,V}\,\mathcal{E} \quad = \quad \{\sigma : V \to \mathcal{U} \mid (\sigma|_A) \in \mathcal{E}\}$$

In particular for $A = \emptyset$ and $\mathcal{E} = \{\emptyset\}$, $\mathsf{ext}_{A,V}\,\mathcal{E} = V \to \mathcal{U}$.

For sets of variable assignments $\mathcal{E}_1 \subseteq A \to \mathcal{U}$ and $\mathcal{E}_2 \subseteq B \to \mathcal{U}$, we define an extended *intersection* operation $\cap_{A,B}$ by extending the variable assignments in both sets $\mathcal{E}_i$ to the union $V = A \cup B$ first and computing the intersection then:

$$\mathcal{E}_1 \cap_{A,B} \mathcal{E}_2 \quad = \quad (\mathsf{ext}_{A,V}\,\mathcal{E}_1) \cap (\mathsf{ext}_{B,V}\,\mathcal{E}_2)$$

For simplicity, we omit these extra indices at "$\cap$" if no confusion can arise. Using this generalized intersection operation, we can compute the set of firings of a list p of assumptions inductively by:

$$\mathcal{T}_\rho[] = \{\emptyset\} \qquad\qquad \mathcal{T}_\rho[p, g] = \mathcal{T}_\rho[p] \cap \mathcal{T}_\rho[g]$$

— given that we are provided with the sets of firings for individual goals. Accordingly, the number $\mathcal{C}_\rho[t]$ of prefix firings associated to a list of goals or conjunction of implications t inductively can be computed by:

$$\begin{array}{ll} \mathcal{C}_\rho[] = 1 & \mathcal{C}_\rho[p \Rightarrow r(\ldots)] = \mathcal{C}_\rho[p] \\ \mathcal{C}_\rho[p, g] = \mathcal{C}_\rho[p] + |\mathcal{T}_\rho[p, g]| & \mathcal{C}_\rho[c_1 \wedge \ldots \wedge c_n] = \sum_{j=1}^{n} \mathcal{C}_\rho[c_j] \end{array}$$

[1] Assume that the implication is not bottom-up bound, and X occurs in the conclusion but not in p. Then we simply add the goal $X = X$ to the list of assumptions.

We conclude that, in order to determine the asymptotic behavior of the number of prefix firings, we must find a description for possible asymptotic behaviors of sets of variable assignments which allows us:

1. to get a (hopefully tight) description for the intersection of sets;
2. to extract a safe cardinality estimate for the sets from their descriptions.

We will now proceed in two steps. First, we abstract relations and sets of variable assignments to a description of their quantitative behavior. The latter then is used to obtain the asymptotic description.

4 Abstract Calculation of Runtimes

Our key idea is to use $(k \times k)$-matrices to describe the quantitative dependencies between the components of k-ary relations.

4.1 Sparsity Matrices

For a k-tuple $t = (a_1, \ldots, a_k)$ and j in the range $1, \ldots, k$, let $[t]_j = a_j$ denote the j-th component of t. Let r denote a relation of arity k over some universe $\mathcal{U}$ of cardinality $N \in \mathbb{N}$. To r, we assign the $(k \times k)$-*sparsity matrix* $\beta[r]$ whose coefficients $\beta[r]_{ij} \in \mathbb{N}$ are given by:

$$\begin{array}{lcll} \beta[r]_{ij} & = & \bigvee\{|S_j(i,a)| \mid a \in \mathcal{U}\} & \text{where} \\ S_j(i,a) & = & \{[t]_j \mid t \in r, [t]_i = a\} & \end{array}$$

Here, $\bigvee$ denotes *maximum* on integers. Please note that the value $\beta[r]_{ij}$, i.e., the maximal cardinality of one of the sets $S_j(i,a)$, depends on j in a rather subtle way: for each j we first collect the j-th components from the tuples in the relation r in which a appears on the i-th place (call each such an element an j-witness) and then compute the cardinality, i.e., how many *different* such j-witnesses at most exist. So, for different j's these values can be quite different.

As an example, consider the edge relation e of a directed graph. Then $\beta[e]_{12}$ equals the maximal out-degree of nodes of the graph, $\beta[e]_{21}$ equals the maximal in-degree. The values $\beta[e]_{ii}$ count the maximal number of i-witnesses given a fixed i-th component: thus, they trivially equal 1 (for all non-empty relations). In particular for a binary tree, we obtain the matrix:

$$\begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}$$

Every sparsity matrix m for a non-empty relation satisfies the following three *metric* properties:

$$\begin{array}{lll} m_{ii} = 1 & \text{for all } i & \\ m_{ij} \leq N & \text{for all } i, j & \\ m_{ij} \leq m_{il} \cdot m_{lj} & \text{for all } i, j, l & (\textit{triangular inequality}) \end{array}$$

Let $\mathcal{M}(N)_k$ denote the set of all $(k \times k)$ sparsity matrices which satisfy the three metric properties above. We have:

Proposition 1. *$\mathcal{M}(N)_k$ is a complete lattice with the following properties:*

1. *The least upper bound operation is computed componentwise, i.e., $(a \sqcup b)_{ij} = a_{ij} \vee b_{ij}$ for $a, b \in \mathcal{M}(N)_k$ and all $i, j = 1, \ldots, k$.*
2. *The function β is monotonic, i.e., $R_1 \subseteq R_2 \subseteq \mathcal{U}^k$ implies $\beta[R_1] \sqsubseteq \beta[R_2]$ in $\mathcal{M}(N)_k$.* □

The mapping β induces a Galois connection between (sets of) k-ary relations and sparsity matrices. Instead of giving a concretization function γ (returning downward closed sets of relations), we here prefer to introduce a description relation $\Delta^{(N)}$ between relations and matrices where for $r \subseteq \mathcal{U}^k$ and $m \in \mathcal{M}(N)_k$,

$$r \;\Delta^{(N)}\; m \qquad \text{iff} \qquad \beta[r] \sqsubseteq m$$

Our next step consists in giving abstract versions of necessary operations on relations. First we define for $a, b \in \mathcal{M}(N)_k$, the $(k \times k)$-matrix $a \oplus b$ by:

$$(a \oplus b)_{ij} = \begin{cases} 1 & \text{if } i = j \\ (a_{ij} + b_{ij}) \wedge N & \text{if } i \neq j \end{cases}$$

where "$\wedge$" denotes minimum. We have:

Proposition 2. *1. The operation $\oplus$ is monotonic;*
2. If $r_i \;\Delta^{(N)}\; a_i$ for $i = 1, 2$, then also $(r_1 \cup r_2) \;\Delta^{(N)}\; (a_1 \oplus a_2)$. □

Next, we consider the greatest lower-bound operation which is going to abstract the intersection operation on relations. Opposed to the least upper bound, the greatest lower bound cannot be computed componentwise. As a counterexample consider the two matrices (for $N = 100$):

$$a = \begin{pmatrix} 1 & 2 & 100 \\ 100 & 1 & 100 \\ 100 & 100 & 1 \end{pmatrix} \qquad b = \begin{pmatrix} 1 & 100 & 100 \\ 100 & 1 & 3 \\ 100 & 100 & 1 \end{pmatrix}$$

The componentwise greatest lower bound is given by:

$$a \wedge b = \begin{pmatrix} 1 & 2 & 100 \\ 100 & 1 & 3 \\ 100 & 100 & 1 \end{pmatrix}$$

In particular, $\quad (a \wedge b)_{13} = 100 > 6 = 2 \cdot 3 = (a \wedge b)_{12} \cdot (a \wedge b)_{23}$.

In order to obtain the greatest lower bound of two matrices we additionally have to perform a (reflexive and) *transitive closure* (rt closure for short).

Let m denote a $(k \times k)$-matrix with entries in $\{1, \ldots, N\}$. Then the rt closure $\nu\, m$ is defined by $(\nu\, m)_{ii} = 1$ and:

$$(\nu\, m)_{ij} = \bigwedge \{ m_{ij_1} \cdot m_{j_1 j_2} \cdot \ldots \cdot m_{j_{g-1} j_g} \cdot m_{j_g j} \mid g \geq 0, j_\gamma \in \{1, \ldots, k\} \}$$

for $i \neq j$. In our example, the rt closure of $a \wedge b$ is given by:

$$\nu(a \wedge b) = \begin{pmatrix} 1 & 2 & 6 \\ 100 & 1 & 3 \\ 100 & 100 & 1 \end{pmatrix}$$

It is well-known that the rt closure of a matrix can be computed efficiently. For the greatest lower bound we find:

Proposition 3. *1. The greatest lower bound of $a, b \in \mathcal{M}(N)_k$ is given by*

$$\begin{array}{lll} a \sqcap b & = \nu(a \wedge b) & \text{where} \\ (a \wedge b)_{ij} & = a_{ij} \wedge b_{ij} & \text{for } i, j = 1, \ldots, k \end{array}$$

2. Whenever $r_i \ \Delta^{(N)} \ a_i$, $i = 1, 2$, then also $(r_1 \cap r_2) \ \Delta^{(N)} \ (a_1 \sqcap a_2)$. □

Our domain $\mathcal{M}(N)_k$ is related to the domain of *difference bound matrices* as used, e.g., for the verification of finite state systems with clock variables [5] and for analyzing simple forms of linear dependencies between program variables [14]. In contrast to these applications, we here use positive integers from a bounded range only and also treat this coefficient domain both additively (namely, for abstracting union) and multiplicatively (namely, for abstracting intersection). The key property of our abstraction of relations is that sparsity matrices allow to estimate cardinalities. For $m \in \mathcal{M}(N)_k$ we define:

$$\mathsf{card}\, m = N \cdot \prod \{m_{xy} \mid (x, y) \in T\}$$

where $(\{1, \ldots, k\}, T)$ is a minimal cost spanning tree of the complete directed graph over $\{1, \ldots, k\}$ with edge weights $(i, j) \mapsto m_{ij}$. Consider, e.g., the matrix:

$$c = \begin{pmatrix} 1 & 2 & 6 \\ 100 & 1 & 3 \\ 100 & 100 & 1 \end{pmatrix}$$

The weighted graph for c is depicted in fig. 1 (self loops and edges with weight 100 are omitted). A minimal spanning tree of this graph is given by the edges

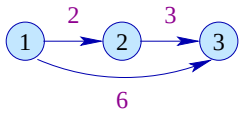

Fig. 1. The weighted graph for the matrix c.

$(1, 2)$ and $(2, 3)$. Therefore, we obtain: $\mathsf{card}\, c = 100 \cdot 2 \cdot 3 = 600$. We have:

Proposition 4. *Assume $r \subseteq \mathcal{U}^k$ and $m \in \mathcal{M}(N)_k$ such that $r \ \Delta^{(N)} \ m$. Then also $|r| \leq \mathsf{card}\, m$.* □

4.2 Computing with Sparsities

In a similar way as to relations, we can assign sparsity matrices to sets $\mathcal{E}$ of variable assignments $\sigma : A \to \mathcal{U}$ for some set A of variables. Here, we deliberately

allow to index the entries of sparsity matrices by the Cartesian product $A \times A$. The sparsity matrix of the non-empty set $\mathcal{E}$ of variable assignments then is given by:

$$\beta[\mathcal{E}]_{xy} = \bigvee \{\, |\{\sigma\, y \mid \sigma \in \mathcal{E}, \sigma\, x = a\}| \;\mid a \in \mathcal{U}\}$$

Let $\mathcal{M}(N)_A$ denote the complete lattice which consists of all matrices with entries from $\{1, \ldots, N\}$ which are indexed with $A \times A$ and satisfy the three metric properties. We introduce an abstract version of the operator $\mathsf{ext}_{A,V}$ by padding the corresponding matrix with the maximally possible value for the so far missing entries. For $A \subseteq V$, we define $\mathsf{ext}^\sharp_{A,V} : \mathcal{M}(N)_A \to \mathcal{M}(N)_V$ by:

$$(\mathsf{ext}^\sharp_{A,V}\, m)_{xy} = \begin{cases} 1 & \text{if } x = y \\ m_{xy} & \text{if } x, y \in A,\ x \neq y \\ N & \text{otherwise} \end{cases}$$

Accordingly, we define for $a \in \mathcal{M}(N)_A$, $b \in \mathcal{M}(N)_B$ and $V = A \cup B$:

$$a \sqcap_{A,B} b = (\mathsf{ext}^\sharp_{A,V}\, a) \sqcap (\mathsf{ext}^\sharp_{B,V}\, b)$$

For convenience, we subsequently drop the subscripts "A, B" at "$\sqcap$". We have:

Proposition 5. *1. If* $r\ \Delta^{(N)}\ a$ *then also* $(\mathsf{ext}_{A,V}\, r)\ \Delta^{(N)}\ (\mathsf{ext}^\sharp_{A,V} a)$.
2. If for $i = 1, 2,$ $r_i\ \Delta^{(N)}\ a_i$ *then also* $(r_1 \cap r_2)\ \Delta^{(N)}\ (a_1 \sqcap a_2)$. $\square$

The greatest matrix $\top_A \in \mathcal{M}(N)_A$ maps the pair (X, Y) to 1 if $X = Y$ and to N otherwise. In order to shorten the presentation, we feel free to specify matrices just by enumerating those entries which deviate from the greatest matrix. Thus for $A = \{X, Y\}$ and $N = 100$, we write $\{(X, Y) \mapsto 5\}_A$ to denote the matrix:

$$\begin{array}{l} \{\ (X, X) \mapsto 1, \quad (X, Y) \mapsto 5, \\ \quad (Y, X) \mapsto 100,\ (Y, Y) \mapsto 1 \quad \} \end{array}$$

Every abstract interpretation $\rho^\sharp$ mapping predicate symbols to corresponding sparsity matrices, gives rise to an abstract description of sets of firings by:

$$\mathcal{T}^\sharp_{\rho^\sharp}[\,] = \{\}_\emptyset \qquad\qquad \mathcal{T}^\sharp_{\rho^\sharp}[p,\, g] = \mathcal{T}^\sharp_{\rho^\sharp}[p] \sqcap \mathcal{T}^\sharp_{\rho^\sharp}[g]$$

where for individual goals,

$$\begin{array}{ll} \mathcal{T}^\sharp_{\rho^\sharp}[r(X_1, \ldots, X_k)] & = \{(X_i, X_j) \mapsto (\rho^\sharp\, r)_{ij} \mid i \neq j\}_{\{X_1, \ldots, X_k\}} \\ \mathcal{T}^\sharp_{\rho^\sharp}[X = Y] & = \{(X, Y) \mapsto 1, (Y, X) \mapsto 1\}_{\{X,Y\}} \\ \mathcal{T}^\sharp_{\rho^\sharp}[X = a] & = \{\}_{\{X\}} \end{array}$$

The treatment of goals $X = a$ as outlined above would record no information about X at all! In order to obtain a better precision here, we consider equalities with constants always together with the preceding goals. We define:

$$\mathcal{T}^\sharp_{\rho^\sharp}[p\ ,\ X = a] = \mathcal{T}^\sharp_{\rho^\sharp}[p] \sqcap \{(Y, X) \mapsto 1 \mid Y \in A\}_{A \cup \{X\}}$$

where A equals the set of variables occurring in the list p. By taking the subsequent extension to the set $A \cup \{X\}$ into account we record that for each value of another variable Y there always can be at most one value of X (namely, a).

Right along the lines of the complexity estimation based on concrete relations and concrete sets of firings, we use abstract descriptions of sets of firings to translate the concrete cost calculation into an abstract one:

$$\begin{array}{ll} \mathcal{C}^\sharp_{\rho^\sharp}[] = 1 & \mathcal{C}^\sharp_{\rho^\sharp}[p \Rightarrow r(\ldots)] = \mathcal{C}^\sharp_{\rho^\sharp}[p] \\ \mathcal{C}^\sharp_{\rho^\sharp}[p,g] = \mathcal{C}^\sharp_{\rho^\sharp}[p] + \mathsf{card}\,(\mathcal{T}^\sharp_{\rho^\sharp}[p,g]) & \mathcal{C}^\sharp_{\rho^\sharp}[c_1 \wedge \ldots \wedge c_n] = \sum_{j=1}^n \mathcal{C}^\sharp_{\rho^\sharp}[c_j] \end{array}$$

Using the estimation of cardinalities of relations according to proposition 4, we can thus calculate an abstract number $\mathcal{C}^\sharp_{\rho^\sharp}[c]$ of prefix firings of a clause c given an assignment $\rho^\sharp$ of predicate symbols to abstract sparsity matrices:

Theorem 2. *Assume c is a Horn clause and ρ an interpretation of the predicate symbols occurring in c. Then we have:*

1. *The number of prefix firings of c (relative to ρ) can be estimated by:*

$$\mathcal{C}_\rho[c] \quad \leq \quad \mathcal{C}^\sharp_{\rho^\sharp}[c]$$

   whenever the universe has cardinality at most N and

$$(\rho\, r) \;\; \Delta^{(N)} \;\; (\rho^\sharp\, r)$$

   for all predicate symbols r occurring in c.
2. *The value $\mathcal{C}^\sharp_{\rho^\sharp}[c]$ can be computed in time polynomial in the size of c.* □

In other words, our abstraction allows to obtain a safe approximation to the number of prefix firings of clauses — given that the solution complies with the assumed sparsity assignment.

5 Asymptotic Calculation of Runtimes

Estimating the runtime of the solver on inputs adhering to a single pre-specified sparsity information, gives us no information on how the runtime scales up when the clause is run on larger relations. The crucial step therefore consists in replacing the abstract calculation from the last section by an asymptotic one. For this, we first introduce the domain of our asymptotic complexity measures. Then we derive the asymptotic runtime calculation and prove its correctness.

5.1 Asymptotic Values

For measuring asymptotic sparsity and complexity we use the abstract value n to refer to the size of the universe. In the application of a program analyzer workbench, the universe typically comprises the set of program points, the names of variables etc. Thus, its cardinality roughly corresponds to the size of

the program to be analyzed. Expressing complexities in terms of powers of the cardinality of the universe, however, often is too coarse. Therefore, we introduce a second value $s \ll n$ which is accounted for in the analysis. The parameter s could, e.g., measure the maximal number of successors/predecessors of a node in a graph. Accordingly, we are aiming at complexity expressions of the form $\mathcal{O}(n \cdot s^2)$. In order to be able to compare such expressions, we must fix (an estimation of) the asymptotic functional relationship between s and n. In particular, we may assume that $s^\eta \sim n$ for some $\eta \in \mathbb{N}$ or even $s \sim \log(n)$ implying that $s^\eta < n$ for all η (at least asymptotically). Let us call η the *dependency exponent* of our analysis. For the following, let $\mathcal{N}$ denote the set of non-negative integers extended by a greatest element ∞. Thus for every exponent $\eta \in \mathcal{N}$, we obtain a linearly ordered lattice $\mathbb{D}_\eta$ of asymptotic complexity measures:

$$\mathbb{D}_\eta = \{n^i \cdot s^j \mid 0 \leq i, 0 \leq j < \eta\}$$

The least element of $\mathbb{D}_\eta$ is given by $n^0 \cdot s^0 = 1$. On $\mathbb{D}_\eta$ we have the binary operations "$\cdot$" (multiplication), "$\sqcup$" (least upper bound) and "$\sqcap$" (greatest lower bound) which are defined in the obvious way. Note that $\mathbb{D}_\eta$ has infinite ascending chains. Also, the lengths of descending chains, though finite, cannot be uniformly bounded.

The set $\mathcal{P}_k(\eta)$ of all asymptotic $(k \times k)$-matrices consists of all $(k \times k)$-matrices a with entries $a_{ij} \in \mathbb{D}_\eta$ such that the following holds:

$$\begin{array}{ll} a_{ii} = 1 & \text{for all } i \\ a_{ij} \sqsubseteq n & \text{for all } i, j \\ a_{ij} \sqsubseteq a_{il} \cdot a_{lj} & \text{for all } i, j, l \end{array}$$

Similar to $\mathcal{M}(N)_k$, $\mathcal{P}_k(\eta)$ forms a complete lattice where the binary least upper bound "$\sqcup$" and greatest lower bound "$\sqcap$" are defined analogously as for $\mathcal{M}(N)_k$. In particular, we can use a completely analogous definition for the card function.

The elements in $\mathbb{D}_\eta$ should be considered as *functions*. Thus, given a concrete argument $N \in \mathbb{N}$, an element p can be evaluated to a natural $[p]_\eta\ N$ by:

$$[n^i \cdot s^j]_\eta\ N = \begin{cases} N^i \cdot \log(N)^j & \text{if } \eta = \infty \\ N^i \cdot N^{j/\eta} & \text{if } \eta < \infty \end{cases}$$

Evaluation at a certain point commutes with the operations "$\cdot$", "$\sqcup$" and "$\sqcap$". Asymptotic statements do not speak about individual elements. Instead, they speak about *sequences* of elements. Let $\underline{x} = (x^{(N)})_{N \in \mathbb{N}}$ denote a sequence of integers $x^{(N)} \in \mathbb{N}$ and $p \in \mathbb{D}_\eta$. Then we write:

$$\underline{x}\ \Delta_\eta\ p \qquad \text{iff} \qquad \exists d \in \mathbb{N} : \forall N \in \mathbb{N} : x^{(N)} \leq d \cdot ([p]_\eta\ N)$$

This formalization captures what we mean when we say that $\underline{x}$ is of order $\mathcal{O}(p)$.

5.2 Computing with Asymptotic Values

In order to analyze the asymptotic runtime complexity of the solver, we not only have to consider sequences of numbers. Besides these, we consider sequences of other objects (all marked by underlining). We introduce:

- sequences of relations $\underline{r} = (r^{(N)})_{N\in\mathbb{N}}$ where $r^{(N)} \subseteq (\mathcal{U}^{(N)})^k$ for universes $\mathcal{U}^{(N)}$ of cardinalities at most N;
- sequences of matrices $\underline{m} = (m^{(N)})_{N\in\mathbb{N}}$ where $m^{(N)} \in \mathcal{M}(N)_k$;
- sequences of interpretations $\underline{\rho} = (\rho^{(N)})_{N\in\mathbb{N}}$ and abstract interpretations $\underline{\rho^\sharp} = (\rho^{\sharp(N)})_{N\in\mathbb{N}}$ of predicate symbols.

Also, we establish a description relation between sequences of matrices $\underline{m}$ and asymptotic matrices:

$$\underline{m} \ \Delta_\eta \ a \text{ iff } (m_{ij}^{(N)})_{N\in\mathbb{N}} \ \Delta_\eta \ a_{ij} \quad \text{for all } i,j$$

In particular, we have:

Proposition 6. *1. Assume $\underline{a}, \underline{b}$ are sequences of sparsity matrices which are asymptotically described by a^* and b^*. Then the following holds:*

$$\begin{array}{lll} (a^{(N)} \sqcup b^{(N)})_{N\in\mathbb{N}} & \Delta_\eta & a^* \sqcup b^* \\ (a^{(N)} \oplus b^{(N)})_{N\in\mathbb{N}} & \Delta_\eta & a^* \sqcup b^* \\ (a^{(N)} \sqcap b^{(N)})_{N\in\mathbb{N}} & \Delta_\eta & a^* \sqcap b^* \end{array}$$

2. If $\underline{a}$ is a sequence of sparsity matrices and $\underline{a} \ \Delta_\eta \ a^*$ then also

$$(\mathsf{card}\,(a^{(N)}))_{N\in\mathbb{N}} \ \Delta_\eta \ \mathsf{card}\,(a^*)$$

□

Similarly to section 4, we now can use the operations on asymptotic sparsity matrices to obtain asymptotic complexity expressions for the runtime of the solver – given an asymptotic description of the sparsities of the computed relations. Thus, for an assignment ρ^* of predicate symbols to asymptotic sparsity matrices, we first infer asymptotic sparsity matrices $\mathcal{T}^*_{\rho^*}[p]$ for sequences of sets of firings of pre-conditions and then calculate the corresponding asymptotic cost functions $\mathcal{C}^*_{\rho^*}[t]$ for the pre-conditions and clauses t. We proceed along the lines for sparsity matrices. The main difference is that we now compute over $\mathbb{D}_\eta$ (instead of $\mathbb{N}$) and that we replace the matrix operation "$\oplus$" with "$\sqcup$".

Example. In [12], we considered a control-flow analysis M_0 for the ambient calculus [3] and showed how to derive an optimized clause M_1 which can be solved in cubic time. For convenience, we described these analyses by using *Horn clauses* extended with explicit quantification and sharing of conclusions — the same analyses, however, can also be described by pure Horn clauses only. In order to illustrate our complexity estimation technique, we take the optimized clause M_1 and pick the abstract description of the In action for ambients. Translating the Flow Logic specification from [12] into plain Horn clauses, we obtain:

$$\mathsf{in}(X,A), \mathsf{father}(X,Y), \mathsf{sibling}(Y,Z), \mathsf{name}(Z,A) \Rightarrow \mathsf{father}(Y,Z)$$

Here, the binary relation in records all pairs (X,A) where A is the label of an In capability of ambients with name A. The binary relation father describes all

pairs (X, Y) of labels where Y is a potential enclosing environment of X. The binary relation $\mathsf{sibling}$ collects all pairs (Y, Z) which potentially have the same father. Finally, the binary relation name records all pairs (Z, A) where Z is the label of an ambient with name A. Thus, the universe $\mathcal{U}$ consists of the labels given to ambient expressions and capabilities occurring in the program together with all occurring names. Therefore, the size n of the universe asymptotically equals the size of the ambient program. By definition, capabilities and ambients are uniquely related to names. Therefore, the relations in and name are asymptotically described by:

$$\rho^* \,\mathsf{in} = \rho^* \,\mathsf{name} = \begin{pmatrix} 1 & 1 \\ n & 1 \end{pmatrix}$$

The binary relation father represents the result of the analysis, i.e., describes all places where an ambient may move to. Let us assume that this relation is "sparse", meaning that each label has only few sons and few fathers. Bounding the numbers of fathers and sons, implies upper bounds for the number of siblings as well. Therefore, we set:

$$\rho^* \,\mathsf{father} = \begin{pmatrix} 1 & s \\ s & 1 \end{pmatrix} \qquad \rho^* \,\mathsf{sibling} = \begin{pmatrix} 1 & s^2 \\ s^2 & 1 \end{pmatrix}$$

Let us assume that the exponent η equals ∞. By varying the exponent η from ∞ (very sparse) down to 1 (dense), we instead could track the impact also of other grades of sparseness onto the complexity of the clause. For the complexity estimation, we first compute the asymptotic descriptions $\mathcal{T}^*_{\rho^*}[p_i]$ for the sets of firings for the prefixes $p_1, \ldots, p_4$ of the pre-condition. The corresponding weighted graphs are shown in fig. 2 (self loops and edges with weight n are omitted). Then we calculate their cardinalities. Starting with $p_1 \equiv \mathsf{in}(X, A)$, we find:

$$\mathcal{T}^*_{\rho^*}[p_1] = \{(X, A) \mapsto 1\}_{\{A,X\}}$$

(see the leftmost graph in fig. 2). A minimal cost spanning tree is given by the edge (X, A). Therefore, $\mathsf{card}\,(\mathcal{T}^*_{\rho^*}[p_1]) = n \cdot 1 = n$.

Next, consider the prefix $p_2 \equiv \mathsf{in}(X, A), \mathsf{hasFather}(X, Y)$. Here, we have:

$$\begin{aligned} \mathcal{T}^*_{\rho^*}[p_2] &= \{(X, A) \mapsto 1\}_{\{A,X\}} \sqcap \{(X, Y) \mapsto s, (Y, X) \mapsto s\}_{\{X,Y\}} \\ &= \nu\,\{(X, A) \mapsto 1, (X, Y) \mapsto s, (Y, X) \mapsto s\}_{\{A,X,Y\}} \\ &= \{(X, A) \mapsto 1, (X, Y) \mapsto s, (Y, A) \mapsto s, (Y, X) \mapsto s\}_{\{A,X,Y\}} \end{aligned}$$

(see the second graph in fig. 2). A minimal spanning tree of this graph consists of the edges (X, A) and (X, Y) which results in the cardinality estimation: $\mathsf{card}\,(\mathcal{T}^*_{\rho^*}[p_2]) = n \cdot 1 \cdot s = n \cdot s$.

Accordingly, we obtain for p_3:

$$\begin{aligned} \mathcal{T}^*_{\rho^*}[p_3] \;=\; \{\; & (X, A) \mapsto 1,\ (X, Y) \mapsto s,\ (X, Z) \mapsto s^3 \\ & (Y, A) \mapsto s,\ (Y, X) \mapsto s,\ (Y, Z) \mapsto s^2 \\ & (Z, A) \mapsto s^3, (Z, X) \mapsto s^3, (Z, Y) \mapsto s^2\ \}_{\{A,X,Y,Z\}} \end{aligned}$$

where a minimal spanning tree is given by the edges (X, A), (X, Y) and (Y, Z) (see the third graph in fig. 2). Therefore, $\mathsf{card}\,(\mathcal{T}^*_{\rho^*}[p_3]) = n \cdot 1 \cdot s \cdot s^2 = n \cdot s^3$.

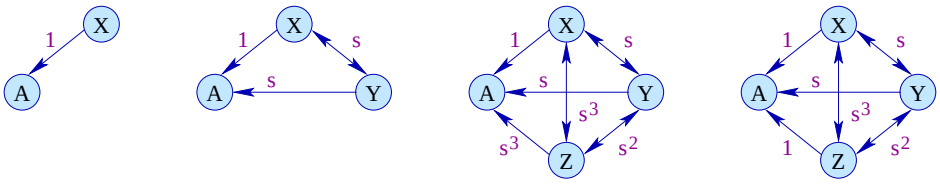

Fig. 2. The weighted graphs for $\mathcal{T}^*_{\rho^*}[p_1]$, $\mathcal{T}^*_{\rho^*}[p_2]$, $\mathcal{T}^*_{\rho^*}[p_3]$ and $\mathcal{T}^*_{\rho^*}[p_4]$.

The asymptotic sparsity matrix for p_4 differs from $\mathcal{T}^*_{\rho^*}[p_3]$ only in the entry for (Z, A) where it has value 1 (instead of s^3) (see fig. 2 to the right), and the asymptotic cardinality stays the same. Summarizing, the contribution of the example clause to the overall complexity is determined as:

$$\begin{aligned} \mathcal{C}^*_{\rho^*}[p_4] &= \bigsqcup_{i=1}^{4} \mathsf{card}\,(\mathcal{T}^*_{\rho^*}[p_i]) \\ &= n \sqcup n \cdot s \sqcup n \cdot s^3 \sqcup n \cdot s^3 = n \cdot s^3 \end{aligned}$$

□

For an asymptotic sparsity assignment ρ^*, let $\mathsf{card}\,\rho^*$ equal the least upper bound of all values $\mathsf{card}\,(\rho^* r)$, $r \in \mathcal{R}$. We obtain our main theorem:

Theorem 3. *For a Horn clause c of size $\mathcal{O}(1)$, let ρ^* denote an asymptotic sparsity assignment for the predicates occurring in c. Then the following holds:*

1. *The solutions $\underline{\rho}$ of c can be computed in time*
$$\mathcal{O}(\mathsf{card}\,\rho^* + \mathcal{C}^*_{\rho^*}[c])$$
provided that the sequence of interpretations $\underline{\rho}$ is asymptotically described by ρ^* (via β and Δ_η), i.e., for every occurring predicate r,
$$(\beta[\rho^{(N)}\ r])_{N\in\mathbb{N}}\ \ \Delta_\eta\ \ (\rho^*\, r)$$
2. *Asymptotic runtime estimates can be computed in time polynomial in the size of clauses.* □

In other words, the runtime analysis predicts correctly the complexity of Horn clause solving for solutions whose sparsity matrices are asymptotically described by ρ^*. Moreover, the estimate itself can be computed efficiently.

6 Asymptotic Sparsity Analysis

Horn clauses will update certain relations by means of assertions. In a general application, we might have knowledge of the (asymptotic) sparsities of some relations whereas others are unknown beforehand or introduced a posteriori during clause tuning. In the control-flow analysis M_1 for Mobile Ambients, this is the case, e.g., for the relation sibling which is defined by the clause:

$$\mathsf{father}(Y, T), \mathsf{father}(Z, T) \Rightarrow \mathsf{sibling}(Y, Z)$$

Clearly, it is both annoying and error-prone if the user has to provide information also for such auxiliary relations – in particular, if these are introduced by some fully automatic clause optimizer.

Therefore, we design a sparsity analysis which takes the partial information provided by the user and tries to infer safe and reasonably precise (asymptotic) sparsity information also for the remaining predicates. We proceed in two steps. First, we infer sparsity matrices and then explain how to do that asymptotically.

6.1 Inferring Sparsity Matrices

For a set $\mathcal{E} \subseteq A \to \mathcal{U}$ of variable assignments and a sequence $args = X_1, \ldots, X_k$ of pairwise distinct variables $X_i \in A$, we define

$$\mathsf{assert}\,(\mathcal{E}, args) \quad = \quad \{(\sigma\, X_1, \ldots, \sigma\, X_k) \mid \sigma \in \mathcal{E}\}$$

Each implication $p \Rightarrow r(args)$ gives rise to the following constraint on ρ:

$$\mathsf{assert}\,(\mathcal{T}_\rho[p], args) \quad \subseteq \quad \rho\, r$$

Thus, the function assert extracts from the firings of the list of assumptions the tuples for the predicate on the right-hand side.

Abstracting this constraint system for solutions of c, we obtain an equation system for the sparsity matrices $(\rho^\sharp\, r), r \in \mathcal{R}$ as follows. For each predicate r in $\mathcal{R}$ of arity k, we accumulate the contributions of assertions onto the sparsity matrix of r where the impact of every implication with matching right-hand side is obtained by abstracting the corresponding concrete constraint. Accordingly, we define an abstract function $\mathsf{assert}^\sharp$ which for $m \in \mathcal{M}(N)_A$ and a list $args = X_1, \ldots, X_k$ of pairwise distinct variables X_i from A, collects the entries from m according to the variable list $args$ to build a $(k \times k)$-matrix from $\mathcal{M}(N)_k$:

$$\mathsf{assert}^\sharp\,(m, args) \quad = \quad \{ij \mapsto m_{X_i, X_j} \mid i, j = 1, \ldots, k\}$$

For $r \in \mathcal{R}$, let $\mathcal{I}[r]$ denote the set of implications in c where r occurs on the right-hand side. Let us furthermore fix an initial interpretation $\rho_0^\sharp$ and a (possibly trivial) upper bound $\rho_1^\sharp$ to the sparsity matrices of occurring predicate symbols. Then we obtain an equation system $\mathcal{S}^\sharp$ for the values $\rho^\sharp\, r, r \in \mathcal{R}$, by:

$$\rho_1^\sharp\, r \;\sqcap\; (\rho_0^\sharp\, r \;\oplus \bigoplus_{p \Rightarrow r(args)\; \in\; \mathcal{I}[r]} \mathsf{assert}\,(\mathcal{T}_{\rho^\sharp}^\sharp[p], args)) \quad = \quad \rho^\sharp r$$

Computing the least model of the clause c is abstractly simulated by the least fixpoint iteration for $\mathcal{S}^\sharp$. This was the easy part. It remains to proceed to asymptotic sparsities.

6.2 Inferring Asymptotic Sparsity Matrices

We define a function assert^* which, given an asymptotic sparsity matrix m and a list $args = X_1, \ldots, X_k$ of pairwise distinct variables X_i, returns

$$\mathsf{assert}^*\,(m, args) \quad = \quad \{ij \mapsto m_{X_i, X_j} \mid i, j = 1, \ldots, k\}$$

Thus, the call $\mathsf{assert}^*\,(m, \mathit{args})$ collects the entries from m according to the variable list args to build a $(k \times k)$-matrix from $\mathcal{P}_k(\eta)$. For a given initial asymptotic assignment ρ_0^* and an upper bound ρ_1^*, we obtain an equation system $\mathcal{S}^*$ for the values $\rho^*\, r, r \in \mathcal{R}$, by:

$$\rho_1^*\, r \;\sqcap\; (\rho_0^*\, r \;\sqcup \bigsqcup_{p \Rightarrow r(\mathit{arg}\,s)\; \in\; \mathcal{I}[r]} \mathsf{assert}\,(\mathcal{T}_{\rho^*}^*[p], \mathit{args})) \quad = \quad \rho^* r$$

The contributions to $\rho^*\, r$ from different implications are now combined by the least-upper-bound operator. Since the left-hand sides of $\mathcal{S}^*$ are monotonic in the asymptotic sparsity assignment, the least as well as the greatest solution of $\mathcal{S}^*$ are well defined. Choosing the least solution, though, is no longer *safe*. Intuitively, this is due to the fact that although $\mathcal{O}(1) + \mathcal{O}(1) = \mathcal{O}(1)$, an arbitrary sum

$$\mathcal{O}(1) + \ldots + \mathcal{O}(1)$$

may return any value. This is reflected in proposition 6.1 (line 2) which speaks about asymptotic descriptions of sequences of *binary* "$\oplus$"-applications only. The incorrectness of the least solution becomes apparent when looking at the Horn clause defining the transitive closure t of an input edge relation e:

$$e(X,Y) \Rightarrow t(X,Y) \qquad \wedge \qquad e(X,Y)\,,\; t(Y,Z) \Rightarrow t(X,Z)$$

If we break down the corresponding system $\mathcal{S}^*$ for the value $(\rho^*\, t)$ to equations for the components $t_{ij} = (\rho^*\, t)_{ij}$, we obtain the following equation for t_{12}:

$$b_{12} \sqcap (e_{12} \sqcup e_{12} \cdot t_{12}) \quad = \quad t_{12}$$

Here, $b_{12} = (\rho_1^*\, t)_{12}$ is the upper bound for t_{12} specified by the user, and $e_{12} = (\rho^*\, e)_{12}$ is the asymptotic maximal out-degree of the input graph. Let us assume that $e_{12} = 1$, i.e., the input graph has asymptotically constant out-degree. Then the equation for t_{12} can be simplified to:

$$b_{12} \sqcap t_{12} \quad = \quad t_{12}$$

The least solution of this equation is $t_{12} = 1$ — implying that the transitive closure of e necessarily has constant out-degree as well: which is *wrong*. In contrast, the *greatest* solution gives us $t_{12} = b_{12}$ — which is reasonable, as it is the the upper bound provided by the user. Sparsity inference, however, through the greatest solution of S^* will not always infer such trivial results.

Example (continued). Consider the definition of the auxiliary predicate sibling and assume that the matrix:

$$\rho_1^*\,\mathsf{father} = \begin{pmatrix} 1 & s \\ s & 1 \end{pmatrix}$$

has been provided as the asymptotic sparsity matrix of the predicate father. Then we calculate for $p \equiv \mathsf{father}(Y,T), \mathsf{father}(Z,T)$ (see fig. 3):

$$\begin{aligned} \mathcal{T}_{\rho^*}^*[p] &= \{(Y,T) \mapsto s, (T,Y) \mapsto s\}_{\{Y,T\}} \sqcap \{(Z,T) \mapsto s, (T,Z) \mapsto s\}_{\{Z,T\}} \\ &= \{\; (Y,T) \mapsto s, (T,Y) \mapsto s, (Y,Z) \mapsto s^2, \\ &\qquad (Z,T) \mapsto s, (T,Z) \mapsto s, (Z,Y) \mapsto s^2 \;\}_{\{Y,Z,T\}} \end{aligned}$$

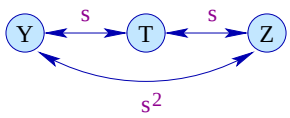

Fig. 3. The weighted graph for $\mathcal{T}^*_{\rho^*}[p]$.

This gives us the value for left-hand side of the equation of $\mathcal{S}^*$ for $\rho^*\,\mathsf{sibling}$:

$$\mathsf{assert}^*(\mathcal{T}^*_{\rho^*}[p],(Y,Z)) = \begin{pmatrix} 1 & s^2 \\ s^2 & 1 \end{pmatrix}$$

Since this matrix is constant, it provides us with the final value of the greatest solution of $\mathcal{S}^*$ for sibling. Indeed, this was precisely the matrix which we had to assert manually in the complexity computation of section 5. □

In summary, we prove:

Theorem 4. *Let c denote a Horn clause of size $\mathcal{O}(1)$ and ρ^* the greatest solution of the equation system $\mathcal{S}^*$. Let $\underline{\rho}_0$ denote a sequence of initial interpretations which is asymptotically described by ρ^*_0 (via β and Δ_η) and $\underline{\rho}$ the sequence of least solutions of c exceeding $\underline{\rho}_0$. Then the following holds:*

1. *Whenever the sequence $\underline{\rho}$ is asymptotically described by ρ^*_1 (via β and Δ_η), then it is also asymptotically described by ρ^*. This means that, whenever*
$$(\beta[\rho^{(N)}\,r])_{N\in\mathbb{N}} \;\; \Delta_\eta \;\; (\rho^*_1\,r)$$
for every predicate r, then also
$$(\beta[\rho^{(N)}\,r])_{N\in\mathbb{N}} \;\; \Delta_\eta \;\; (\rho^*\,r)$$
for every predicate r.
2. *Greatest asymptotic sparsity assignments can be computed in time polynomial in the size of the respective clauses.*

In other words, given a safe assertion about the asymptotic sparsities of (some) predicates, our asymptotic sparsity analysis will provide a possibly better but still safe assertion about the asymptotic sparsities. Thus, it can be seen as a *narrowing* procedure to improve on a given safe information. Theorem 4 holds since, opposed to the least fixpoint, the greatest fixpoint of the system $\mathcal{S}^*$ is reached after a uniformly bounded number of iterations.

Proof. The equation system $\mathcal{S}^*$ can be written as $F\,\rho^* = \rho^*$ where F is the joint function of left-hand sides in $\mathcal{S}^*$. Then we argue as follows.

(1) We safely may apply *one single* fixpoint iteration, i.e., given that ρ^* is a correct asymptotic sparsity assignment, $F\,\rho^*$ is still correct.
(2) We safely may apply any *constant* number of fixpoint iterations, i.e., given that ρ^* is a correct asymptotic sparsity assignment, $F^h\,\rho^*$ is still a correct asymptotic sparsity assignment for any h which may depend on the constraint system – but is independent of the universe and the predicates.
(3) The greatest fixpoint is reached after a finite number of iterations. More precisely, the greatest fixpoint of F is given by $F^h\,\rho^*_1$, with $h \leq |\mathcal{R}| \cdot a^2$ where a is the maximal arity of a predicate from $\mathcal{R}$.

Assertion (1) follows by induction on the structure of pre-conditions. Assertion (2) follows accordingly. Therefore, it remains to prove assertion (3). First, we observe that each component of F defining an entry (i,j) of the asymptotic sparsity matrix for some predicate r is composed of a bounded number of basic operations on $\mathbb{D}_\eta$. Then we rely on the following observation:

Let $\mathbb{D}$ denote a complete lattice. A function $f : \mathbb{D}^m \to \mathbb{D}$ is called *nice* iff f is monotonic and for all pairs of m-tuples $(x_1,\ldots,x_m),(y_1,\ldots,y_m) \in \mathbb{D}^m$, with $x_i \sqsubseteq y_i$ for all i, the following holds:

$$\text{If } \; f(x_1,\ldots,x_m) \sqsubset f(y_1,\ldots,y_m) \; \text{ then } \; \bigsqcap\{x_i \mid x_i \neq y_i\} \sqsubseteq f(x_1,\ldots,x_m).$$

Niceness of functions is a semantical property which generalizes the (partly syntactically defined) property considered by Knuth in [9]. In contrast to Knuth's property, niceness is preserved under composition, and least upper bounds:

Proposition 7. *1. Constant functions $\lambda x_1,\ldots,x_m.c$, $c \in \mathbb{D}$, the identity $\lambda x.x$ as well as the binary operation $\sqcup$ are nice.*
2. In case of linear orderings $\mathbb{D}$, also $\sqcap$ is nice.
3. Nice functions are closed under composition, greatest and least fixpoints. □

A proof of the following theorem is included in the full version of the paper:

Theorem 5. *Consider a system of equations $f_i(x_1,\ldots,x_m) = x_i$, $i = 1,\ldots,m$, where all left-hand sides $f_i : \mathbb{D}^m \to \mathbb{D}$ are nice. Let $F : \mathbb{D}^m \to \mathbb{D}^m$ denote the function $F = (f_1,\ldots,f_m)$. If $\mathbb{D}$ is a linear ordering, then the greatest fixpoint νF of F is reached after m iterations, i.e., $\nu F = F^m(\top,\ldots,\top)$.* □

Let $\mathcal{S}$ denote the constraint system over $\mathbb{D}_\eta$ which is obtained from $\mathcal{S}^*$ by writing the equations componentwise. Thus, the set of variables of $\mathcal{S}$ are given by all $(\rho^* r)_{ij}$, $r \in \mathcal{R}$, where each left-hand side is an expression built up from constants and these variables by means of applications of the operators "$\sqcup$", "$\sqcap$", and "$\cdot$". Since our operation "$\cdot$" is also nice, we conclude from proposition 7 that all left-hand side expressions in $\mathcal{S}$ represent nice functions. Thus, theorem 5 is applicable. As $\mathcal{S}$ has at most $|\mathcal{R}| \cdot a^2$ many variables (a the maximal arity of a predicate in $\mathcal{R}$), our assertion (3) follows. This completes the proof. □

7 Practical Implementation and Experimental Results

The key idea of McAllester's Horn clause solver is to bring clauses into a specific canonical form which then is easy to solve. In order to do so, he introduces auxiliary predicates for prefixes of pre-conditions and employs constructor applications for collecting instantiated variables.

In our applications, we found it rather restrictive to deal with Horn clauses only. Therefore, we extended the Horn clause framework by explicit quantification, conditional clauses and stratified negation. The Horn clause for transitive closure, e.g., could be written in our logic as:

$$\forall X,Y : e(X,Y) \Rightarrow (t(X,Y) \;\wedge\; (\forall Z : t(Y,Z) \Rightarrow t(X,Z)))$$

The logic which we have implemented is known to Logicians as *alternation-free least fixpoint logic* in clausal form [8]. It is more expressive than "Horn clauses with sharing" [12] or Datalog – even with stratified negation [4, 10].

For this richer logic, McAllester's solving method does not suffice any longer. Therefore, we developed and implemented an alternative solving algorithm. In contrast to McAllester's method, our solving procedure does not rely on preprocessing. It also abandons special worklist-like data-structures as are typical for most classical fixpoint algorithms [6]. Still, it meets the same complexity estimation for Horn clauses as McAllester's. For details about this solver algorithm, see [13]. Accordingly, we extended and implemented the complexity estimator described in the preceding sections to this stronger logic and our solver.

Applying the automatic complexity analyzer to the two formulations M_0 and M_1 of control-flow analysis for the Ambient calculus, we succeed in refining the rough complexity estimations from [12] — provided that further assumptions about the resulting control-flow are met.

Besides these refinements for the ambient analyses, we report here also on the results of the complexity estimator on the following benchmarks:

TC: transitive closure of some edge relation;
FL: control-flow analysis of a functional language;
P: control-flow analysis for the pi calculus from [12].

For transitive closure, the results are reported depending on the asymptotic sparsity of the edge relation e. In the sparse case, we use the asymptotic sparsity matrix:

$$\begin{pmatrix} 1 & s \\ s & 1 \end{pmatrix}$$

For all control-flow analyses, we investigate the impact of different assumptions on the asymptotic sparsity of the result relation. The clause M_1 is obtained from the clause M_0 by introduction of various auxiliary predicates [12]. No information has been provided to the complexity analyzer for these — meaning that their asymptotic sparsities are inferred by the system. The following table collects the estimations computed by our analysis:

	dense	sparse
TC	n^3	$n^2 \cdot s$
FL	n^3	$n \cdot s^2$
P	n^3	$n^2 \cdot s$
M_0	n^4	$n \cdot s^3$
M_1	n^3	$n \cdot s^3$

The simplest clause is the one for transitive closure where the tool returns the expected complexities. On CFA for functional languages, it nicely assures that the complexity is low if only few values are found for each expression into which it may develop. The same holds true for the mobile ambients. Interestingly, here both the unoptimized and the optimized analysis give the same asymptotic complexity – provided that the computed relation is sparse.

8 Conclusion

The complexity analysis has benefitted from the pioneering ideas of McAllester [11] and Basin and Ganzinger [2] on the complexity of solving Horn clauses. The contribution of our paper is to *fully automate the necessary calculations*. In particular, the idea of asymptotic sparsity matrices for describing the asymptotic sparseness of relations as well as our narrowing algorithm for inferring asymptotic sparsity matrices for predicates seems to be new.

McAllester himself [11] and together with Ganzinger [7] have provided further complexity meta-theorems for interesting deductive systems which are also candidates for integration into a program analyzer workbench. A challenging open question is whether the necessary complexity calculations for these can be automated as well.

References

1. A. Aiken. Introduction to set constraint-based program analysis. *Science of Computer Programming (SCP)*, 35(2):79–111, 1999.
2. D.A. Basin and H. Ganzinger. Complexity Analysis Based on Ordered Resolution. In *11th IEEE Symposium on Logic in Computer Science (LICS)*, 456–465, 1996. Long version to appear in JACM.
3. L. Cardelli and A.D. Gordon. Mobile ambients. In *Proceedings of FoSSaCS'98*, volume 1378 of *LNCS*, 140–155. Springer-Verlag, 1998.
4. E. Dahlhaus. Skolem normal forms concerning the least fixpoint. In *Computation Theory and Logic*, 101–106. LNCS 270, Springer Verlag, 1987.
5. D. L. Dill. Timing assumptions and verification of finite state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, 197–212. LNCS 407, Springer Verlag, 1989.
6. C. Fecht and H. Seidl. A faster solver for general systems of equations. *Science of Computer Programming (SCP)*, 35(2-3):137–162, 1999.
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, Springer Verlag, 2001.
8. G. Gottlob, E. Grädel, and H. Veith. Datalog LITE: A deductive query language with linear time model checking. *ACM Transactions on Computational Logic*, 2001. To appear.
9. D. E. Knuth. On a generalization of Dijkstra's algorithm. *Information Processing Letters (IPL)*, 6(1):1–5, 1977.
10. P.G. Kolaitis. Implicit definability on finite structures and unambiguous computations (preliminary report). In *5th Annual IEEE Symposium on Logic in Computer Science (LICS)*, 168–180, 1990.
11. D. McAllester. On the complexity analysis of static analyses. In *6th Static Analysis Symposium (SAS)*, 312–329. LNCS 1694, Springer Verlag, 1999.
12. F. Nielson and H. Seidl. Control-flow analysis in cubic time. In *European Symposium on Programming (ESOP)*, 252–268. LNCS 2028, Springer Verlag, 2001.
13. F. Nielson and H. Seidl. Succinct solvers. Technical Report 01-12, University of Trier, Germany, 2001.
14. R. Shaham, K. Kordner, and S. Sagiv. Automatic removal of array memory leaks in Java. In *Compiler Construction (CC)*, 50–66. LNCS 1781, Springer Verlag, 2000.