A New Meta-Complexity Theorem for Bottom-up Logic Programs

Harald Ganzinger[1] and David McAllester[2]

[1] MPI Informatik, D-66123 Saarbrücken, Germany, hg@mpi-sb.mpg.de
[2] AT&T Labs-Research, Florham Park NJ 07932, USA, dmac@research.att.com

Abstract. Nontrivial meta-complexity theorems, proved once for a programming language as a whole, facilitate the presentation and analysis of particular algorithms. This paper gives a new meta-complexity theorem for bottom-up logic programs that is both more general and more accurate than previous such theorems. The new theorem applies to algorithms not handled by previous meta-complexity theorems, greatly facilitating their analysis.

1 Introduction

McAllester has recently shown that the running time of a bottom-up logic program can be bounded by the number of "prefix firings" of its inference rules [10]. A prefix firing of a rule is a derivable instantiation of a prefix of the antecedents of that rule. This single nontrivial meta-complexity theorem simplifies the presentation and complexity analysis of a variety of parsing and static analysis algorithms. Many other algorithms, however, seem to fall outside of the range of this theorem. In particular, algorithms based on union-find or congruence closure can not be analyzed. A second meta-complexity theorem for the analysis of union-find based algorithms is also given in [10]. While this second theorem applies to a broader class of algorithms, it yields running time bounds that are often worse by logarithmic factors than algorithm-specific bounds — bounds proved without the use of a meta-complexity theorem. Here we prove a more accurate and more general meta-complexity theorem. The increased generality is achieved by proving the theorem for logic programs with priorities and deletions. Priorities and deletions allow the simulation of arbitrary classical control structures. So the new meta-complexity theorem has, in some sense, universal coverage. The new theorem yields improvements in meta-complexity-derived bounds for a variety of algorithms including union-find and congruence closure. As an example, section 5 presents an algorithm for determining the satisfiability of a ground set of Horn clauses with equality. The new meta-complexity theorem allows the simple derivation of a very tight running time bound for this algorithm. Proving the same bound for this problem without the the use of our new theorem appears to be significantly more difficult.

2 Inference Rules with Priorities and Deletions

We use the term *inference rule* to mean a first-order Horn clause, i.e., a formula of the form $A_1 \wedge \ldots \wedge A_n \rightarrow C$ where C and each A_i is a first-order atom. We will use *assertion* to mean a ground atom and use the term *data base* to mean a set of assertions. If R is a set of rules and D is a data base, then we let $R(D)$ denote the set of ground atoms derivable from the ground set D using the rules in R.

Here we are interested in expressing algorithms with prioritized inference rules with deletion. An inference rule with deletion is an expression of the form $A_1 \wedge \ldots \wedge A_n \rightarrow C$ where C is an atom and each A_i is either an atom or an expression of the form $[A]$ where A is an atom. Intuitively, the marking $[\ldots]$ means that the premise is to be deleted as soon as the rule is run. Deletion makes the behavior of the algorithm nondeterministic. For example, consider the following rules with deletion.

$$P \Rightarrow Q \qquad [Q] \Rightarrow S \qquad [Q] \Rightarrow W$$

Suppose the initial data base contains only P. The first rule fires adding the assertion Q. Now either the second or third rule can fire. Since each of these rules deletes Q, once one of them fires the other is blocked. Hence the final data base is nondeterministically either $\{P,\ S\}$ or $\{P,\ W\}$. When viewing rules with deletions as algorithms this nondeterminism is viewed as "don't care" nondeterminism — the choices are made arbitrarily and not backtracked. (In many cases this kind of don't-care nondeterminism can be justified by a suitable notion of redundancy, cf. Section 8.) Suppose now that we have additional rules through which W entails a large number of additional facts whereas the absence of W does not. Then, in order to obtain a more efficient run of the rules, we should prefer to fire the second rule rather than the third rule, which we could achieve by giving higher priority to the second rule. In summary, allowing for deletion makes deduction nondeterministic, and hence priorities are needed for indicating which choices are to be made in order to increase efficiency, or to avoid unwanted results.

The proof of the meta-complexity theorem requires that deletion be permanent — once an assertion is deleted further attempts to reassert it have no effect. (If deletion is based on a notion of redundancy such as the one proposed in [2], once an assertion has become redundant it remains so for the remainder of the computation.) To see the problem with deletion consider the simple pair of rules $[P] \rightarrow Q$ and $[Q] \rightarrow P$. If deletion is can be revoked by subsequent assertion, the rules can oscillate between a database containing P and a database containing Q and fail to terminate. To formalize this notion of permanent deletion we take a *state* of the computation to be a set S of literals (atoms and negations of atoms). The presence of a negated atom $\neg A$ in a state indicates that A should be considered deleted. Hence, we say that an atom A is *visible* in a state S if $A \in S$ and $\neg A \notin S$, while a negative literal $\neg A$ is called *visible* in S whenever $\neg A \in S$. If σ is a ground substitution and $[A]$ is a deleted antecedent then we define $\sigma([A])$ to be the ground atom $\sigma(A)$. Now let S be a state, and let r be an inference rule with deletions. We write $S \stackrel{r}{\to} S'$ if $S \neq S'$ and r is a rule $A_1 \wedge \ldots \wedge A_n \to C$ such that there exists a ground substitution σ defined on all the variables in r such that $\sigma(A_i)$ is visible in S, and S' is $S \cup \{\sigma(C), \neg\sigma(A_{i_1}), \ldots, \neg\sigma(A_{i_k})\}$ where $A_{i_1}, \ldots, A_{i_k}$ are the deleted antecedents of the rule. We say that a rule r is *applicable* at the state S if there exists a state S' (which must be different from S) such that $S \stackrel{r}{\to} S'$.

Now let R be a set of rules with deletions where each rule in R is associated with a positive rational number called its priority. We call R a *rule set with priorities and deletions*. For technical simplicity we may assume that priorities are unique in that no two rules have the same priority. We say that a state S is *visible* to a rule $r \in R$ if no higher priority rule in R is applicable at S. We write $S \stackrel{R}{\to} S'$ if there exists a rule $r \in R$ such that S is visible to r and $S \stackrel{r}{\to} S'$. We will say that a state S is *saturated* under R if it is a normal form, i.e., there is no S' such that $S \stackrel{R}{\to} S'$. An R-computation from a database D is a sequence S_0, $S_1, \ldots, S_T$ such that $S_0 = D$, $S_t \stackrel{R}{\to} S_{t+1}$. An R-computation is called *complete* if the final state S_T is saturated. If there is a complete R-computation from D ending in S_T then we say that S_T is an R-saturation of D. A rule set R is said to terminate on input database D if there is no infinite R-computation from D.

A *prefix firing* in an R-computation $\mathcal{C}$ is a triple $\langle r, \sigma, i\rangle$, where $r \in R$ is a rule $A_1 \wedge \ldots \wedge A_n \Rightarrow C$ such that the computation $\mathcal{C}$ contains a state S visible to r and σ is a ground substitution defined on the variables in the antecedent prefix $A_1, \ldots, A_i$ such that the $\sigma(A_j)$, for $1 \leq j \leq i$, are visible in S. Note that the set of prefix firings of a given rule is determined by the set of states visible to that rule. For any R-computation $\mathcal{C}$ we let $p(\mathcal{C})$ be the number of prefix firings in $\mathcal{C}$. We will call a rule range-restricted if every variable in the conclusion appears in some antecedent. Bottom-up logic programs are generally range-restricted and for simplicity we only consider range-restricted rules. In the following, by $|D|$ we denote the *size* of a database which is the number of nodes in its fully shared graphical representation by a dag.

Theorem 1. *For any given set R of range-restricted rules with priorities and deletions there exists an algorithm mapping an input database D to an R-saturation $R(D)$ of D whose running time is $O(|D| + \max_{\mathcal{C}} p(\mathcal{C}))$ where the maximization is over all R-computations $\mathcal{C}$ from D.*[1]

The theorem extends the one in [10] to inference rules with priorities and deletion showing essentially that no penalty has to be paid for these extensions. The complexity can again be linearly bounded by the number of prefix firings.

Before giving a proof of this theorem, in the next sections we will present a variety of applications. Before discussing those, the following example is given in order to clarify one of the more subtle issues behind our definitions. Consider

[1] Note that if there is no bound on the length of computations then the algorithm need not terminate.

the rules $r1$ and $r2$, where

$$r1:\ r(x,y),\ [p(x)] \Rightarrow s(x) \qquad r2:\ p(x),\ q(x,y) \Rightarrow r(x,y)$$

with priorities from left to right, on a database D consisting of facts $p(i)$, for $1 \leq i \leq n$, and $q(i,j)$, for $1 \leq i,j \leq n$. In any computation from D, whenever rule r2 produces an r-fact $r(i,j)$, in the next step r1 takes priority over r2, and the $p(i)$ is deleted so that no other fact $r(i,j')$ can by produced thereafter. Hence any computation takes at most $2n$ steps. However, the number of prefix firings of rule r2 is $n+n^2$, and that is the upper bound on the time complexity provided by the meta-complexity theorem above. A more refined meta-complexity theorem, based on refined notions of prefix firings, could be stated. However in this paper we deliberately confine ourselves to the simpler version. The additional technical complexity does not appear to be required for the examples that we are interested in at present.

3 A Union-Find Algorithm

This section presents an $O(n \log n)$ union-find algorithm given as a rule set with priorities and deletions. This union-find algorithm both gives an example of a use of theorem 1 and serves as a foundation for other algorithms given in later sections of this paper. The union-find algorithm in itself is perhaps not significantly simpler than classical presentations using pointers and recursive procedures. However its direct relation to the usual inference rules for Knuth-Bendix completion makes correctness arguments more straightforward.

The union-find algorithm is used to represent equivalence relations. In the inference rule union-find algorithm U in Figure 1 we assume a binary predicate union such that the assertion union$(x,\ y)$ means that x and y are to be made equivalent — the procedure is to compute the least equivalence relation such that if the data base contains union$(x,\ y)$ then x and y are equivalent. The find function is defined in terms of a more basic rewrite relation which we represent here as a set of assertions of the form $x \xrightarrow{f} y$. We define the "find" of x to be the normal form of x under the rewrite relation $\xrightarrow{f}$. Storing this relation explicitly as assertions in the data base defines in a more logical manner what is usually implemented with pointer structures.

The union-find inference system implements Knuth-Bendix completion for the simple case of equations between constants. The equations are represented by the union facts. The rules (F1) and (F2) compute the normal forms of terms. (U2)–(U4) orient equations into rewrite rules using an ordering that is dynamically determined by the weight computation in rules (U3) and (U4). If $\xrightarrow{f}*$ is the reflexive-transitive closure of $\xrightarrow{f}$, the weight of y is the number of nodes x such that $x \xrightarrow{f}* y$. An assertion $x \xrightarrow{f} y$ is to be added only for irreducible y for which $y \xrightarrow{f}! y$. The rule (F1)–(F2) are to run at a higher priority than any other rules mentioning the predicates find, $\xrightarrow{f}$ or $\xrightarrow{f}!$. This ensures that, at any state visible to other rules mentioning these relations, the relation $\xrightarrow{f}!$ is the fixed "normal form relation" determined by the $\xrightarrow{f}$ relation.

$$(\mathtt{F1})\ \frac{\text{find}(x)}{\begin{array}{l} x \xrightarrow{f}! x, \\ \text{weight}(x,1) \end{array}} \qquad\qquad (\mathtt{F2})\ \frac{\begin{array}{l} [x \xrightarrow{f}! y] \\ y \xrightarrow{f} z \end{array}}{x \xrightarrow{f}! z}$$

$$(\mathtt{U1})\ \frac{\text{union}(x,\ y)}{\begin{array}{l} \text{find}(x), \\ \text{find}(y) \end{array}} \quad (\mathtt{U2})\ \frac{\begin{array}{l} [\text{union}(x,\ y)] \\ x \xrightarrow{f}! z \\ y \xrightarrow{f}! z \end{array}}{T} \quad (\mathtt{U3})\ \frac{\begin{array}{l} [\text{union}(x,\ y)] \\ x \xrightarrow{f}! z_1 \\ y \xrightarrow{f}! z_2 \\ \text{weight}(z_1, w_1) \\ [\text{weight}(z_2, w_2)] \\ w_1 < w_2 \end{array}}{\begin{array}{l} z_1 \xrightarrow{f} z_2, \\ \text{weight}(z_2, w_1 + w_2) \end{array}} \quad (\mathtt{U4})\ \frac{\begin{array}{l} [\text{union}(x,\ y)] \\ x \xrightarrow{f}! z_1 \\ y \xrightarrow{f}! z_2 \\ [\text{weight}(z_1, w_1)] \\ \text{weight}(z_2, w_2) \\ w_1 \geq w_2 \end{array}}{\begin{array}{l} z_2 \xrightarrow{f} z_1, \\ \text{weight}(z_1, w_1 + w_2) \end{array}}$$

Fig. 1. Module U for union-find. We write rules vertically as the antecedents followed by a horizontal line followed by the conclusions. The rules are listed in decreasing priority. Multiple conclusions $A_1, \ldots, A_k$ should be viewed as a single atom $a(A_1 \ldots, A_k)$ with auxiliary rules (of highest priority) generating the individual conjuncts A_j.

All of the rules represented by (U1) run at higher priority than (U2), (U3), or (U4). This implies that in any state visible to (U2), (U3), or (U4) we have that find(x) and find(y) have been asserted and hence the normal forms of x and y have been computed and the weights have been initialized. Rule (U2) is given higher priority than either (U3) or (U4). This implies that at any state visible to (U3) or (U4) we have that x and y have distinct normal forms (otherwise the state would be visible to rule (U2) which would then delete the link assertion and assert the trivial "true" assertion T).

The use of addition in the rules (U3) and (U4) is outside of the formal language defined in section 2. However, the use of addition in the conclusion can be replaced by an additional final antecedent of the form $w_3 = w_1 + w_2$. Theorem 1 can be generalized to handle constraint antecedents provided that the set of assignments of values to the unassigned variables (those not appearing in earlier antecedents) can be computed in time proportional to the number of such assignments.

One should think of the rules (F1)–(U4) as a module U that takes as *input* assertions of the form find(x) and union(x, y) and produces as *output* assertions of the form $x \xrightarrow{f}! z$ such that z is the normal form of x in a canonical rewrite system generated from the equations union(x, y). The input can be extended dynamically by new assertions of the form find(x) and union(x, y) generated from additional rules that are *compatible* with U.

Definition 1. *A module consists of a rule set with priorities and deletions plus specified input and output predicate symbols. All other predicates of the module are called local. A rule set R will be called* compatible *with a module M provided that it does not mention local predicates of M, and*

- *no output predicate of M appears in any conclusion or deleted antecedent of a rule in R,*
- *no input predicate of M appears in any deleted antecedent of R,*
- *and every rule in R containing an output predicate of M in an antecedent has priority lower priority than all rules in M.*

An initial database D will be called compatible with a module M if it does not mention any predicates of M other than input predicates.

Theorem 2. *The union-find module U has the property that for any rule set R and initial database D, where R and D are both compatible with U, and any $(R \cup U)$-computation $\mathcal{C}$ from D, the total number of prefix firings in $\mathcal{C}$ of the rules in U is $O(m + n \log n)$ where m is the number of* union *assertions in $\mathcal{C}$ or produced by R, and n is the number of distinct terms appearing in union or find assertions.*

Proof. Note that each non-redundant union operation generates a single new assertion of the form $x \xrightarrow{f} y$ where the weight of y prior to the addition of this assertion is at least as large as the weight of x. This implies that weight at least doubles as one moves across any assertion of the form $x \xrightarrow{f} y$. So for a given x the set of y such that $x \xrightarrow{f}* y$ can have at most $\log n$ elements. (At most $n-1$ rewrite rules $x \xrightarrow{f} y$ can be generated until all terms become equal.)

Now we show that each of the rules in U has an appropriate number of prefix firings. The rule (F1) has at most n firings. It follows from the above comments that there are at most $n \log n$ assertions ever generated of the form $x \xrightarrow{f}! y$. This, and the fact that out-degree of $\xrightarrow{f}$ is at most one, imply that rule (F2) has at most $n \log n$ prefix firings. Rule (U1) has at most m firings. All states containing the assertion union$(x,\ y)$ and visible to any of the rules (U2), (U3), or (U4) must assign the same unique normal forms and weights to x and y. This implies that the rules (U2), (U3), and (U4) also have at most m prefix firings.

It is possible to give a more complex inference rule implementation of union-find that runs in $O(n\alpha(n))$ time where α is the inverse of Ackermann's function. However, many algorithms based on union-find run in $O(n \log n)$ time even when using an $O(n\alpha(n))$ implementation of union-find. For such applications an $O(n \log n)$ implementation of union-find suffices.

4 Congruence Closure

The congruence closure problem is to determine whether an equation $s = t$ between ground terms is provable from a given set of equations between ground

$$(\texttt{C1})\ \frac{\mathrm{find}(\langle x,\ y\rangle)}{\mathrm{find}(x),\ \mathrm{find}(y),\ \mathrm{init}_{\Rightarrow}(\langle x,\ y\rangle)} \qquad (\texttt{C2})\ \frac{[\mathrm{init}_{\Rightarrow}(z)] \quad z \Rightarrow w}{T} \qquad (\texttt{C3})\ \frac{[\mathrm{init}_{\Rightarrow}(z)]}{z \Rightarrow z}$$

$$(\texttt{C4})\ \frac{[\langle x,\ y\rangle \Rightarrow z] \quad x \xrightarrow{f}! \, x' \quad \langle x',\ y\rangle \Rightarrow z'}{\mathrm{union}(z,\ z')} \qquad (\texttt{C5})\ \frac{[\langle x,\ y\rangle \Rightarrow z] \quad x \xrightarrow{f}! \, x'}{\langle x',\ y\rangle \Rightarrow z} \qquad (\texttt{C6})\ \frac{[\langle x,\ y\rangle \Rightarrow z] \quad y \xrightarrow{f}! \, y' \quad \langle x,\ y'\rangle \Rightarrow z'}{\mathrm{union}(z,\ z')} \qquad (\texttt{C7})\ \frac{[\langle x,\ y\rangle \Rightarrow z] \quad y \xrightarrow{f}! \, y'}{\langle x,\ y'\rangle \Rightarrow z}$$

Fig. 2. Rules for congruence closure listed in order of decreasing priority

terms using the reflexivity, symmetry, transitivity and congruence rules for equality. Here we assume that expressions are represented using constants and a single pairing function. The congruence property of the pairing function states that if $u_1 = w_1$ and $u_2 = w_2$ then $\langle u_1,\ u_2\rangle = \langle w_1,\ w_2\rangle$.

The inference rules in Figure 2 are compatible with the union-find module U (assuming that rules in U have priority lower than the rules (C4)–(C7) that use the output predicate $\xrightarrow{f}!$ of U). Combined with U, they give an $O(n \log n)$ algorithm for congruence closure. The rules are related to rules given in [3] which view congruence closure as a form of ground completion. The ternary atoms $\langle \_, \_\rangle \Rightarrow \_$ represent the signatures in [5], or the definitions in [3]. Note that, by contrast to [3], we do not introduce new constants to denote the subterms of the input equations. The terms on the right side in $\Rightarrow$ play the role of these constants. This explains rule (C3) where $z \Rightarrow z$ gives us z as the handle to all terms that are semantically equal to z. These rules have lower priority than all rules in the union-find module and the priority between rules corresponding to the order in which the rules are given with (C1) having highest priority and (C7) having lowest. The precedence of (C2) over (C3), (C4) over (C6) and (C5) over (C7) ensure the invariant that for any pair $\langle x,\ y\rangle$ there is at most one assertion of the form $\langle x,\ y\rangle \Rightarrow z$. Furthermore, one can check that in any state visible to these rules, and hence where no union-find rules are still to be run, we have that if the state contains $\langle x,\ y\rangle \Rightarrow z$ then $\langle x,\ y\rangle$ and z have the same find value. Furthermore, the rules maintain the invariant that in states visible to (C4) through (C7), if the state contains find($\langle x,\ y\rangle$) then there exists an x' and y' such that $x \xrightarrow{f}! \, x'$ and $y \xrightarrow{f}! \, y'$ and the state contains $\langle x',\ y'\rangle \Rightarrow z$ where z is equivalent to (has the same find as) $\langle x,\ y\rangle$. In any final (saturated) state we must have that x' and y' are normal forms under the equivalence generated by the union assertions. This implies that in the final state, if we have find($\langle x_1,\ y_1\rangle$) and find($\langle x_2,\ y_2\rangle$) where x_1 and x_2 are equivalent and y_1 and y_2 are equivalent we must also have $\langle f_1,\ f_2\rangle \Rightarrow z$ where f_1 is the common find of x_1 and x_2, f_2 is the common find of y_1 and y_2, and where z is equivalent to both input pairs, and hence the input pairs are equivalent to each other.

The union-find module satisfies the condition that for any given x there are at most $\log n$ terms y such that $x \xrightarrow{f}! \ y$. This implies that an initial assertion $\langle x,\ y\rangle \Rightarrow z$ can generate at most $2\log n$ "descendents" of the form $\langle x',\ y'\rangle \Rightarrow z$. This implies that at most $2n\log n$ assertions of this form are ever generated and this implies that each of the rules (C4), (C5), (C6), and (C7) have at most $2n\log n$ prefix firings. The other rules have at most $O(n)$ prefix firings. Hence we have the following theorem.

Let C be the module with input predicates union and find and output predicate $\xrightarrow{f}!$, resulting from combining the union-find module with the congruence closure rules.

Theorem 3. *If R is a rule set and D an initial database where both R and D are compatible with C then, in any computation from D of $R \cup C$, the total number of prefix firings of rules in C is $O(m + n\log n)$ where m is the number of input union assertions, that is, union assertions in D or generated by R, and n is the number of terms x appearing in* find *assertions (either in input find assertions or find assertions generated by (C1)).*

Note that in this case, n is proportional to the number of the different subterms in input find assertions. The complexity bound given by the theorem is the same as the one given in [5]. The latter paper, however, ignores the work needed for processing the input equations. Our inference rules do include this preprocessing and, therefore, come with an additional additive $O(m)$ term in the complexity bound.

5 Satisfiability of Ground Horn Clauses with Equality

We now extend congruence closure (in a compatible manner) to handle ground (object-level) Horn clauses represented as assertions in the input database D. (The meta-level Horn clauses are called inference rules.) More specifically we want to construct an algorithm for computing the deductive closure of a set of ground assertions of the form input$(\Phi \to A)$ where the possible expressions for Φ and A are defined by the following grammar where c ranges over constants and p ranges over binary predicate symbols including the special symbol $\doteq$ for denoting formal equality.

$$\Phi ::= A \mid \Phi_1 \wedge \Phi_2 \qquad A ::= T \mid p(t_1,\ t_2) \qquad t ::= c \mid \langle t_1,\ t_2\rangle$$

The algorithm takes as input a set D of ground assertions of the form $\Phi \to A$. The algorithm uses the congruence closure module and all inference rules in this section run at priority higher than those in the congruence closure module.

We start with the following linear time module for ground Horn clauses without equality. The module may be viewed as a high-level implementation of the algorithm in [4]. The main idea in this set of rules is that atoms appearing in the antecedent of clauses are first detached from their clauses. This has the effect that even if an atom has many occurrences in antecedents of clauses it is nevertheless only derived once.

$$(\mathtt{I1})\frac{\mathrm{input}(\Phi \to A)}{\mathrm{antecedent}(\Phi),\ \mathrm{conclusion}(A),\ \mathrm{true}(\Phi \to A)} \qquad (\mathtt{I2})\frac{\mathrm{antecedent}(\Phi_1 \wedge \Phi_2)}{\mathrm{antecedent}(\Phi_1),\ \mathrm{antecedent}(\Phi_2)}$$

$$(\mathtt{I3})\frac{}{\mathrm{true}(T)} \qquad (\mathtt{I4})\frac{\mathrm{true}(\Phi \to \Psi) \quad \mathrm{true}(\Phi)}{\mathrm{true}(\Psi)} \qquad (\mathtt{I5})\frac{\mathrm{antecedent}(\Phi_1 \wedge \Phi_2) \quad \mathrm{true}(\Phi_1),\ \mathrm{true}(\Phi_2)}{\mathrm{true}(\Phi_1 \wedge \Phi_2)}$$

A natural way to extend these rules to handle equality would be to treat atoms themselves as terms and apply congruence closure. This would give a simple $O(m \log m)$ algorithm for conditional equations, where m is the size (in dag representation) of the set of clauses. If m is quadratic in the number n of different terms appearing in the set, that would give the bound $O(n^2 \log n)$. However, for the particular application given in section 6 a more refined bound, and more refined algorithm, is needed. We handle equality with the following rules.

$$(\mathtt{I6})\frac{\mathrm{true}(\doteq(s,\ t))}{\mathrm{union}(s,\ t)} \qquad (\mathtt{I7})\frac{\mathrm{antecedent}(p(s,\ t))}{\mathrm{find}(s),\ \mathrm{find}(t),\ \mathrm{push}(p(s,\ t))} \qquad (\mathtt{I8})\frac{\mathrm{conclusion}(p(s,\ t))}{\mathrm{find}(s),\ \mathrm{find}(t),\ \mathrm{push}(p(s,\ t))}$$

$$(\mathtt{I9})\frac{[\mathrm{push}(p(s,\ t))] \quad s \xrightarrow{f}!\ s'}{\mathrm{true}(p(s',\ t) \to p(s,\ t)),\ \mathrm{true}(p(s,\ t) \to p(s',\ t)),\ \mathrm{push}(p(s',\ t))} \qquad (\mathtt{I10})\frac{[\mathrm{push}(p(s,\ t))] \quad t \xrightarrow{f}!\ t'}{\mathrm{true}(p(s,\ t') \to p(s,\ t)),\ \mathrm{true}(p(s,\ t) \to p(s,\ t')),\ \mathrm{push}(p(s,\ t'))}$$

$$(\mathtt{I11})\frac{[\mathrm{push}(\doteq(s,\ s))]}{\mathrm{true}(\doteq(s,\ s))}$$

The rules have priority in the order given with (I1) having highest priority and (I11) having lowest but with all rules at lower priority than any rules in the congruence closure module. We leave it to the reader to verify that any saturation of these rules contains a given conclusion if and only that conclusion follows from the input under the standard interpretation of equality. Here we focus on the run time (number of prefix firings) of these rules.

Let m be the number of antecedents as derived by rules (I1) and (I2), plus the number of clauses in the input, and let n and a, respectively, be the number of different terms and atoms appearing there. Clearly, a is in $O(m)$, and also in $O(n^2)$, with $m, n \leq |D|$. The number of prefix firings of rules (I1), (I2), (I3), (I5), (I7), and (I8) are all proportional to m. The number of prefix firings of (I4) is proportional to m plus the number of firings of (I9) and (I10). The number of prefix firings of (I6) is proportional to m plus the number of firings of rule (I11). The number of prefix firings of (I11) is bounded by to the number of prefix firings of (I7) and (I8) (which is m) plus the number of prefix firings of (I9) and (I10). So the total number of prefix firings is proportional to m plus the number of firings of the two rules (I9) and (I10). Since $\xrightarrow{f}!$ has out-degree at most one we immediately get that these two rules have at most n^2 firings where n is the number of subterms appearing in the input. By the properties of union-find we also get that those rules have at most $a \log n$ firings. The number of union operations generated by rule (I6) is at most $a \log n$, hence in $O(m \log n)$. The number of prefix firings inside the congruence-closure module is therefore $O((m + n) \log n)$. So the total number of prefix firings is $O(m + n \log n + \min(m \log n,\ n^2))$.

Theorem 4. *Satisfiability of ground Horn clauses with equality can be decided in time $O(|D|+n \log n+\min(m \log n,\ n^2))$ where m is the number of antecedents and input clauses and n is the number of terms.*

The above bound is better than $O(m \log m)$ in any family of problems where m is $\Omega(n^2)$. In that case, also $|D|$ is in $\Omega(n^2)$ and the algorithm becomes linear in the size of the input. In cases where the length of antecedents is bounded by a constant, m is proportional to the number of input clauses in D.

6 Henglein's Quadratic Typability Algorithm

Following the exposition by [10], the typability problem in a variant of the Abadi-Cardelli object calculus [1] considered by [8] can be taken to consist of a given set of assertions of the form $\sigma \leq \tau$ and $accepts(\sigma,\ l)$ and $notaccepts(\sigma,\ l)$, where σ and τ are type names and l is a message name. The instance is acceptable (solvable) provided that the following rules do not derive *fail*. We also assume that $type(\sigma)$ is derivable for those type terms σ that appear in [*not*]*accepts*- or $\leq$-facts in the input, or which are of the form $\tau.l$ with $accepts(\tau, l)$ appearing in the input. Moreover, we assume the standard reflexivity, symmetry, transitivity, and congruence properties of equality.

$$\frac{type(\sigma)}{\sigma \sqsubseteq \sigma} \qquad \frac{\begin{array}{c}\sigma \leq \tau\\ type(\rho)\\ \tau \sqsubseteq \rho\end{array}}{\sigma \sqsubseteq \rho} \qquad \frac{\begin{array}{c}accepts(\sigma, l)\\ accepts(\tau, l)\\ \sigma \sqsubseteq \tau\end{array}}{\sigma.l \doteq \tau.l} \qquad \frac{\begin{array}{c}type(\sigma)\\ accepts(\sigma, l)\\ notaccepts(\sigma, l)\end{array}}{fail}$$

To determine solvability of a problem instance we can now simply build all ground Horn clauses that result from resolving the bodies of the first three rules with the [*not*]*accepts*- and the $\leq$-facts in the input, and also with the the *type*-facts derived from the input database. For the last rule we generate the ground instances by resolving with the *type*-facts and instantiating with all label terms in the input. If the size of the problem instance is m, the input contains $O(m)$ *accepts*- and *input*-facts and terms. Hence we obtain $O(m^2)$ resolvents which are ground clauses in which $O(m)$ terms appear. We now have that the input is solvable if and only if one cannot derive *fail* from these ground clauses together with the facts in the input.[2] Applying theorem 4 we get a novel and simple proof of Henglein's result that solvability is decidable in $O(m^2)$ time.

7 Proof of the Meta-Complexity Theorem

In this section we prove Theorem 1. It turns out to be convenient to prove the theorem for a slightly more general language. We define a *literal-based rule* to be a rule of the form $A_1 \wedge \cdots \wedge A_n \rightarrow C$ where each A_i and C are literals, i.e., either atoms or negations of atoms. We write $S \stackrel{r}{\rightarrow} S'$ if there exists a ground substitution σ defined on all the variables in r such that for each antecedent A of r we have that $\sigma(A)$ is visible in S and S' is $S \cup \{\sigma(C)\}$ where C is the conclusion of r. We then define the notion of a rule is applicable to a state, a state being visible to a rule, an R-computation, and an R-saturated state, and a prefix firing as in the case for rules with deletions (in both cases we allow rules to have priorities). We now show that any rule set with priorities and deletions can be translated to a literal-based rule set in a way that allow computations to also be translated in a way that preserves the number of prefix firings up to a constant factor. In particular, we translate a rule with deletions of the form $A_1 \wedge \cdots \wedge A_n \rightarrow C$ to the following set of literal-based rules where p is a fresh predicate symbol, $x_1, \ldots, x_n$ are all variables in the rule, and $A_{i_1}, \ldots, A_{i_k}$ are all the deleted antecedents.

$$\begin{aligned}
A_1 \wedge \cdots \wedge A_n &\rightarrow p(x_1, \ldots, x_n) \\
p(x_1, \ldots, x_n) &\rightarrow \neg A_{i_1} \\
&\cdots \\
p(x_1, \ldots, x_n) &\rightarrow \neg A_{i_k} \\
p(x_1, \ldots, x_n) &\rightarrow C
\end{aligned}$$

The first rule above has the same priority as the translated rule. The other rules are called “transient” rules. Note that an “atomic” invocation of one of the original rules gets translated into a sequence of intermediate states where some,

[2] One needs to show that further ground clauses are redundant. Suppose, for instance, we have $accepts(s, l)$ and $accepts(t, l)$ in the input. Then by the process just described we generate the ground clause $s \sqsubseteq t \rightarrow s.l \doteq t.l$. If we later derive $s \doteq s'$, the clause $s' \sqsubseteq t \rightarrow s'.l \doteq t.l$ is a consequence of the clause we already have.

but not all, of the deletions and insertions have been made. We must ensure that these "transient" states are not visible to other rules in the system. This can be done by assigning all transient rules higher priority than all rules in the set being translated. It now suffices to prove theorem 1 for rules over literals rather than rules with deletions.

We now perform source to source transformations on rules over literals to put the rules in a simplified form without increasing the number of prefix firings by more than a constant factor. First we convert rules such that they have at most three literals. If r is a rule over literals $A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow C$ with $n > 2$ then we replace r by the following set of rules where P_1, P_2, $\ldots$ P_n, are fresh predicate symbols and x_1, $\ldots$, x_{k_i} are the variables occurring in the first i antecedents. The predicate P_i represents the relation defined by the first i antecedents, and $\neg P_i$ represents the negation (retraction) of P_i.

$$\begin{aligned}
A_1 &\rightarrow P_1(x_1, \ldots, x_{k_1}) \\
\neg A_1 &\rightarrow \neg P_1(x_1, \ldots, x_{k_1}) \\
P_1(x_1, \ldots, x_{k_1}) \wedge A_2 &\rightarrow P_2(x_1, \ldots, x_{k_2}) \\
P_2(x_1, \ldots, x_{k_2}) \wedge \neg P_1(x_1, \ldots, x_{k_1}) &\rightarrow \neg P_2(x_1, \ldots, x_{k_2}) \\
P_2(x_1, \ldots, x_{k_2}) \wedge \neg A_2 &\rightarrow \neg P_2(x_1, \ldots, x_{k_2}) \\
&\vdots \\
P_{n-1}(x_1, \ldots, x_{k_1}) \wedge A_n &\rightarrow P_n(x_1, \ldots, x_{k_2}) \\
P_n(x_1, \ldots, x_{k_2}) \wedge \neg P_{n-1}(x_1, \ldots, x_{k_1}) &\rightarrow \neg P_n(x_1, \ldots, x_{k_2}) \\
P_n(x_1, \ldots, x_{k_2}) \wedge \neg A_n &\rightarrow \neg P_n(x_1, \ldots, x_{k_2}) \\
P_n(x_1, \ldots, x_{k_n}) &\rightarrow C
\end{aligned}$$

Since we are now proving a version of theorem 1 for rules over literals we must consider the case where the rule being translated has negative antecedents. In that case the above rules might include rules with doubly negated antecedents. Such rules are simply dropped from the translation since negative literals can not be deleted (or overruled). The last rule above is given the same priority as the rule being translated. All other rules are given higher priority (but lower than the priority of any original rule with priority higher than the rule that is translated) where the priority is in the order given, i.e., the first rule has highest priority and so on. This is possible since the original rules have all different priorities.

To prove the version of theorem 1 for rules over literals it suffices to show that any computation of the translated rule set can be mapped back to a computation of the original rule set with no more than a constant factor reduction in the number of prefix firings. In particular, if the new rule set derives $P_i(x_1, \ldots, x_{k_i})$ then there must exist a single state in the computation of the original rule set where $A_1, \ldots, A_i$ all hold under the corresponding variable substitution. This follows from the observation that the priority assignment guarantees that in states visible to the rule deriving $P_i(x_1, \ldots, x_{k_i})$ the predicate P_{i-1} is guaranteed to have the appropriate meaning as a function of the predicates used in the original antecedents.

We have now shown that we can assume without loss of generality that each rule contains at most two antecedents. We now put the rules in an even more restricted form. For any rule r with two antecedents $A_1 \wedge A_2 \rightarrow C$ we replace r by the following set of rules where $x_1, \ldots, x_n$ are all variables occurring in A_1 but not A_2, $y_1, \ldots, y_m$ are all variables that occur in both A_1 and A_2, and $z_1, \ldots, z_k$ are all variables that occur in A_2 but not A_1. The predicates P, and Q, and the function symbols f, g, and h are all fresh.

$$\begin{aligned} A_1 &\rightarrow P(f(x_1, \ldots, x_n), g(y_1, \ldots, y_m)) \\ \neg A_1 &\rightarrow \neg P(f(x_1, \ldots, x_n), g(y_1, \ldots, y_m)) \\ A_2 &\rightarrow Q(g(y_1, \ldots, y_m), h(z_1, \ldots, z_k)) \\ \neg A_2 &\rightarrow \neg Q(g(y_1, \ldots, y_m), h(z_1, \ldots, z_k)) \end{aligned}$$

$$P(f(x_1, \ldots, x_n), g(y_1, \ldots, y_m)) \wedge Q(g(y_1, \ldots, y_m), h(z_1, \ldots, z_k)) \rightarrow C$$

The rules are given priority in the order given with the last rule having the same priority as the rule being translated. Again the validity of the translation relies on the observation that in any state visible to a rule using one of the newly introduced predicates P and Q in antecedents, these predicates must have the intended meaning as a function of the underlying original predicates and hence there is a corresponding firing of the original rule.

Without loss of generality we now need only prove the theorem for prioritized inference rules over literals where each rule either has only a single antecedent or is of the form $P(t_1, t_2) \wedge Q(t_2, t_3) \rightarrow C$ where t_1, t_2, and t_3 do not share variables and where the rules maintain the invariant that for all derivable ground assertions of the form $P(s_1, s_2)$ we have that s_1 is a substitution instance of t_1 and s_2 is a substitution instance of s_2, and for all derivable ground assertions of the form $Q(s_2, s_3)$ we have that s_2 is a substitution instance of t_2 and s_3 is a substitution instance of t_3. For such rule sets we can use the algorithm shown below to compute an R-saturation of a given initial database D.

Algorithm to Compute $R(D)$:

Assume that D is in fully shared dag representation in which term equality can be checked in constant time. We maintain queues Q_p and R_p for each priority p in R. Initialize S to be D and place every element of D on every queue Q_p. Initialize all queues R_p to be empty.

While some queue is nonempty do the following:

> Let p be the highest priority such that either Q_p or R_p is nonempty. (The current state is visible to rules of priority p.) If Q_p is nonempty then remove a literal Φ from Q_p and if Φ is visible in S then notice Φ at priority p using the procedure given below. If Q_p is empty then remove a pair $\langle r, \sigma \rangle$ from R_p. (Here r is a rule of priority p and σ is a substitution assigning ground values to all variables of r such that for each antecedent of A of r we have $\sigma(A) \in S$.) If $\sigma(A)$ is visible in S for each antecedent A of R then let Ψ be the assertion $\sigma(C)$ where C is the conclusion of R, add Ψ to S, and place Ψ on all queues of the form $Q_{p'}$.

Algorithm to Notice Φ at priority p:

(The current state is visible to all rules of priority p and Φ is visible in S.)

1. For each single-antecedent rule of priority p of the form $A \to C$ determine whether there is a substitution σ such that $\sigma(A) = \Phi$ and, if so, add the pair $\langle r, \sigma \rangle$ to R_p.
2. For each two-antecedent rule of priority p of the form $P(t_1, t_2) \wedge Q(t_2, t_3) \to C$ do the following:
   (a) If Φ has the form $P(s_1, s_2)$ then for each s_3 such that $Q(s_2, s_3)$ is visible in S add the pair $\langle r, \sigma \rangle$ to R_p where σ is the substitution mapping t_1 to s_1, t_2 to s_2, and t_3 to s_3. (We are guaranteed that t_1, t_2, and t_3 do not share variables and that t_1 matches s_1, t_2 matches s_2, and t_3 matches s_3.)
   (b) If Φ has the form $Q(s_2, s_3)$ then for each s_1 such that $P(s_1, s_2)$ is visible in S add the pair $\langle r, \sigma \rangle$ to R_p where σ is defined as in (a). (Analogous guarantees also exist in this case.)

We leave it to the reader to verify the correctness and running time of this algorithm. The main feature of the algorithm is that the processing of a given rule r is restricted to states visible to r. By incrementally maintaining appropriate indices it is possible to run steps (2a) and (2b) in time proportional to the number of values of s_3 and s_1 defined in those steps respectively. For instance the set of substitutions s_3 such that $Q(s_2, s_3)$ is visible in S (cf. step 2a) has to be indexed using term s_2 as key, so that upon adding a new Q-atom to S the index can be updated in constant time. Note that since we have assumed dag representations for expressions under which equality testing is a unit time operation, all matching operations for patterns in R take unit time.

8 Future Work

We have presented a more refined concept of logic programming where the emphasis is on guaranteed execution time bounds linear in the number of prefix firings of its rules. We are optimistic that this logic will continue to prove itself useful in the design of algorithms. We have demonstrated some of the potential of the method by giving a novel algorithm for testing satisfiability of ground Horn clauses with equality and shown that in many cases its complexity is not worse that (unconditional) congruence closure. On top of this we have implemented an abstract version of Henglein's type analysis, confirming the quadratic upper bound that Henglein obtained before. In fact we believe that program analysis is a particularly fruitful area for applying our method. This point was illustrated in detail in [10]. With the methods in the present paper we are able to also deal with congruences that appear in such analyses in a logical way.

There are many directions into which this work should be extended. Theorem 4 should be generalized to cases of input clauses with variables. However, already in the given form it is useful for local (equational) theories in the sense of [7, 9, 6].

The relation between deletions and redundancy elimination, with redundancy in the sense of [2] as entailment from smaller atoms, should be explored. For instance, the rule (I9) deletes $\mathrm{push}(s,t)$ if s can be reduced to s' after the reduced atom $\mathrm{push}(s',t)$ has been generated. The deleted atom is "entailed" by the "smaller" assertions $\mathrm{push}(s',t)$ and $s \xrightarrow{f}! \, s'$. The elimination of such redundancies is stable under enrichments of a state or deletions of other redundant atoms. Therefore, if only redundant premises are deleted, priorities are irrelevant for the correctness of the algorithm and only affect its complexity.

The concept of priorities for rules should be refined in an instance-based manner, allowing different instances of a rule to have different priorities. That would give one direct means of formalizing algorithms that would normally have to be defined via priority queues. For instance, minimal spanning trees can be computed by a two-rule program on top of union-find if rules referring to edges in graphs could be processed in an order related to their associated costs.

References

1. M. Abadi and L. Cardelli. *A Theory of Objects.* Springer, New York, Berlin, Heidelberg, 1996.
2. L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *J. Logic and Computation*, 4(3):217–247, 1994. Revised version of Research Report MPI-I-91-208, 1991.
3. Leo Bachmair and Ashish Tiwari. Abstract congruence closure and specializations. In David McAllester, editor, *Automated Deduction – CADE-17, 17th International Conference on Automated Deduction*, LNAI 1831, pages 64–78, Pittsburgh, PA, USA, June 17–20, 2000. Springer-Verlag.
4. William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Logic Programming*, 3:267–284, 1984.
5. P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpressions problem. *J. Association for Computing Machinery*, 27(4):771–785, 1980.
6. H. Ganzinger. Relating semantic and proof-theoretic concepts for polynomial time decidability of uniform word problems. In *Proc. 16th IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press, 2001. To appear.
7. R. Givan and D. McAllester. New results on local inference relations. In *Principles of Knowledge Representation and reasoning: Proceedings of the Third International Conference (KR'92)*, pages 403–412. Morgan Kaufmann Press, 1992.
8. F. Henglein. Breaking through the n^3 barrier: Faster object type inference. *Theory and Practice of Object Systems*, 5(1):57–72, 1999. A preliminary version appeared in FOOL4.
9. D. McAllester. Automatic recognition of tractability in inference relations. *J. Association for Computing Machinery*, 40(2):284–303, 1993.
10. David McAllester. On the complexity analysis of static analyses. In A. Cortesi and R. Filé, editors, *Static Analysis — 6th International Symposium, SAS'99*, LNCS 1694, pages 312–329, Venice, Italy, September 1999. Springer-Verlag.