Automatic Decidability

Christopher Lynch and Barbara Morawska

Department of Mathematics and Computer Science Box 5815, Clarkson University, Potsdam, NY 13699-5815, USA, E-mail: clynch@clarkson.edu,morawskb@clarkson.edu **

Abstract. We give a saturation procedure that takes a theory as input, and returns a decision procedure for that theory when it halts. In addition, it returns a bound on the complexity of that theory: $O(n \times lg(n))$ for some theories (such as the theory of lists), polynomial for other theories (including all equational theories that halt), and simply exponential for other theories (such as the theory of arrays).

1 Introduction

Verification problems often involve proving the satisfiability of a universally quantified equational implication[1] modulo a background theory, such as the theory of lists or arrays or some other data structure. For example, in the theory of lists, we may want to prove that $\forall x_1, x_2, x_3, x_4, x_5, x_6 (x_1 = cons(x_3, x_4) \wedge x_2 = cons(x_5, x_6) \wedge car(x_1) = car(x_2) \wedge cdr(x_1) = cdr(x_2) \rightarrow x_1 = x_2)$. By skolemization, satisfiability of a universally quantified equational implication modulo a theory can be shown equivalent to proving the unsatisfiability of a ground conjunction of equations and disequations modulo that theory, which is the problem we will concern ourselves with in this paper.

In particular, we have developed a procedure P that takes a theory E in first order logic with equality, represented as a set of clauses. Our procedure returns an algorithm A which decides the satisfiability of any ground set of equations and disequations modulo E. In addition, it also gives us a bound on the complexity of A. Unfortunately, it is undecidable, in general, to decide the satisfiability of any set of ground equations and disequations modulo E. Therefore, it cannot be the case that P will always halt. However, it often does halt, and we give examples of theories where it does halt, such as the theory of lists and the theory of arrays.[2]

We define a class of theories called *noncylic* where the algorithm A will run in $O(n \times lg(n))$, e.g., the theory of lists. It will run in polynomial time for many theories where P halts, in particular for all equational theories (no clauses) where

** This work was supported by NSF grant number CCR-0098270 and ONR grant number N00014-01-1-0435. Contact the first author at (315) 268-2384, fax number (315) 268-2371.

[1] Predicates can be encoded with equations. There may be function symbols.

[2] See [1] to show how this also applies to the theory of extensional arrays.

P halts. In the other theories where P halts, A runs in simply exponential time, e.g., the theory of arrays.

This paper is based heavily on the work of [1], which provides a good background for the usefulness of this work. Our paper is basically an automation of their result, with some additional complexity results. They gave an algorithm A, based on saturation as an extension of congruence closure, for deciding the satisfiability of a conjunction of ground equations and disequations modulo the theory of lists and (extensional) arrays. They proved the completeness of those algorithms. Our improvement of their result is to give an automated method of developing such algorithms. For us, the algorithm A is developed automatically, the completeness of each individual A is based on the completeness of procedure P which must only be proved once, as opposed to proving is separately for each algorithm. Additionally, the complexity of A must only be proved once for all algorithms derived by P.

Our procedure P is itself a saturation algorithm of clauses containing *constant constraints*, which is a conjunction of constraints of the form $const(t)$, true if and only if t is a constant. Our algorithm A is an extension of the congruence closure algorithm given in [4] (see also [7]). The paper [4] proves a complexity bound of $O(n \times lg(n))$, based on the way the precedence of the constants is chosen. We use the same technique as they do, and show how to extend the congruence closure results to *noncylic* theories. This gives us better complexity results than those given in [1].

Our results are in the spirit of automated decidability and complexity results of theories saturated by ordered paramodulation [5]

2 Preliminaries

We use standard definitions as in [2].

We assume a reduction ordering $\succ$, which is total on ground terms, for example, the lexicographic path ordering[6]. This ordering is extended to equations by considering them as multisets of terms, and then to clauses by considering them as multisets of equations. We define the ordering in such a way that the negative literals are always slightly bigger than the positive ones, constants are smaller than composite expressions, and the constant *true* is the smallest one. Since the ordering is total, we can always orient the ground equations, hence we treat them often as rewrite rules and write $s \to t$ instead of $s \approx t$.

3 Flattening

We want to to decide the satisfiability of a set of ground equations and disequations, modulo a theory. First we provide an algorithm to make the ground equations and disequations *flat*.

Definition 1. *Define a new set of constants called extension constants. A* flat f-rule *is a rule of the form* $f(c_1, \cdots, c_n) \to c_0$*, where each* c_i *is an extension* *constant and $n \geq 0$. An equation is* flat *if it is a flat f-rule or if it contains only extension constants. A disequation is* flat *if it contains only extension constants.*

Let S be a set of equations and disequations. Then S can be flattened by repeated application of the following **Extension** rule:

$$\frac{S \cup \{A[t]\}}{S \cup \{A[c],\, t \to c\}}$$

where c is a fresh extension constant, $A[t]$ is not flat, and t is either

1. a constant a which is not an extension constant, or
2. of the form $f(c_1, \ldots, c_n)$ such that $c_1, \ldots, c_n$ are extension constants,

The number of rules created by this will be the number of function (and constant) symbols in $|S|$.

We will define a function called *depth*, which will be applied to extension constants. When an extension constant c is created by the Extension rule, we set $depth(c) = 0$. The Orient rule, defined later, will modify the depth of an extension constant. The precedence of extension constants is not determined by the original precedence. Precedence between these constants will be determined by the Orient rule, by analyzing depth. It will be determined in such a way that every extension constant has only logarithmically many extension constants smaller than it.

From now on, we assume that S is a set of ground, flat equations.

4 Saturation

In this section we will define a sound and complete set of inference rules for determining the unsatisfiability of a set of clauses. First we define some closure rules in Figure 1, and some deletion rules in Figure 2.

The inference rules use a *selection rule*, which is a function that is applied to a clause C and returns a set of one or more literals in C. Literal in that set are said to be *selected* in C. For completeness, the selection rule must have the following properties:

Selection Rule
In each clause select:

1. a negative literal ("don't care non-determinism") or
2. all maximal literals.

Notice that the closure rules are defined so that a new clause is always added, whereas the deletion rules delete a clause or replace a clause by another one. An *inference system* I is a set of closure and deletion rules. A (possibly infinite) set of clauses T is considered to be *saturated* by an inference system I if any application of a closure rule from I to T results in a clause C such that either

Right Paramodulation:

$$\frac{G \cup \{u[s'] \approx v \vee C, s \approx t \vee D\}}{G' \cup \{(u[t] \approx v \vee C \vee D)\sigma\}}$$

where $G' = G \cup \{u[s'] \approx v \vee C, s \approx t \vee D\}$, $\sigma = mgu(s, s')$, $u[s'] \approx v$ is selected in its clause, $s \approx t$ is selected in its clause, $u[s']\sigma \not\preceq v\sigma$, $s\sigma \not\preceq t\sigma$, and s' is not a variable.

Left Paramodulation:

$$\frac{G \cup \{u[s'] \not\approx v \vee C, s \approx t \vee D\}}{G' \cup \{(u[t] \not\approx v \vee C \vee D)\sigma\}}$$

where $G' = G \cup \{u[s'] \not\approx v \vee C, s \approx t \vee D\}$, $\sigma = mgu(s, s')$, $u[s'] \approx v$ is selected in its clause, $s \approx t$ is selected in its clause, $u[s']\sigma \not\preceq v\sigma$, $s\sigma \not\preceq t\sigma$, and s' is not a variable.

Equational Resolution

$$\frac{G \cup \{u \not\approx v \vee C\}}{G' \cup \{C\sigma\}}$$

where $G' = G \cup \{u \not\approx v \vee C\}$, $\sigma = mgu(u, v)$ and $u \not\approx v$ is selected.

Equational Factoring

$$\frac{G \cup \{s \approx t \vee s' \approx u \vee C\}}{G' \cup \{(s \approx u \vee t \not\approx u \vee C)\sigma\}}$$

where $G' = G \cup \{s \approx t \vee s' \approx u \vee C\}$, $\sigma = mgu(s, s')$, $s \approx t$ is selected, $t\sigma \not\preceq u\sigma$, $s\sigma \not\preceq t\sigma$ and $s'\sigma \not\preceq u\sigma$.

Orient

$$\frac{G \cup \{c \approx d\}}{G[c \mapsto d]^{a}}$$

where c and d are extension constants and $depth(c) \leq depth(d)$. Then we set the precedence $c \succ d$ and set $depth(d) = max\{depth(d), 1 + depth(c)\}$.

[a] This is an abuse of the substitution notation. It means to replace c by d everywhere.

Fig. 1. Rules for Closure

Subsumption:

$$\frac{G \cup \{C, C'\}}{G \cup \{C\}}$$

where $C\sigma \subseteq C'$ for some substitution σ.

Simplification:

$$\frac{G \cup \{C[l'], l \approx r\}}{G \cup \{C[r\sigma], l \approx r\}}$$

where $l' = l\sigma$, $l\sigma \succ r\sigma$, and $C[l'] \succ (l\sigma \approx r\sigma)$.

Tautology Deletion:

$$\frac{G \cup \{t \approx t \vee C\}}{G}, \frac{G \cup \{s \approx t \vee s \not\approx t \vee C\}}{G}$$

Fig. 2. Rules for Deletion

1. C is in T, or
2. a sequence of deletion rules from I to $T \cup \{C\}$ will result in T (i.e., C could be removed, so the inference is not necessary).

If T is a set of clauses, then T' is a *saturation of* T *by* I if T' is saturated by I and T' is obtained by a (possibly infinite) sequence of closure and deletion rules from I applied to T. This mimics what happens in a real theorem prover, where closure rules **must** be applied and deletion rules **may** be applied. In practice, it is better to apply deletion rules whenever possible to cut down on the size of the search space.

Let I be the inference system whose closure rules consist of the closure rules defined in Figure 1. The Orient rule will always be applied eagerly. The deletion rules of I consist of the deletion rules given in Figure 2. We have the following theorem[3].

Theorem 1. *A set T of clauses is unsatisfiable if and only if the empty clause is in a saturation of T by I if and only if the empty clause is in all saturations of T by I.*

In particular, we will consider a set of clauses E that is already saturated by I, and a set of flat ground equations and disequations S. We would like to know whether I halts on $E \cup S$.

5 Meta-saturation

Given a set of clauses E, we would like to show that given any set of ground equations and disequations S, it is decidable whether $E \cup S$ is satisfiable. We will use the inference procedure I to decide the satisfiability. In particular, for a given E, we would like to show that the inference system I will halt on $E \cup S$ for any ground set of equations and disequations S. We will also examine the number of clauses that can be generated, and give an upper bound on the complexity of the decision procedure.

The way we do this is to define a new inference system I_1. Then we will create a set of equations and disequations G_E which depends on E. We will show that if $E \cup G_E$ halts for the inference system I_1 then for any ground set of equations and disequations S, $E \cup S$ will halt for the inference system I.

G_E will contain equations and disequations with *constant constraints*. An *atomic constant constraint* is of the form $const(t)$, and it is true if t is an extension constant. A *constant constraint* is a conjunction of zero or more atomic constant constraints, which is true if each atomic constant constraint in the conjunction is true. A substitution σ *satisfies* a constant constraint φ if $\varphi\sigma$ is true. A *constrained clause* is of the form $C[\![\varphi]\!]$, where C is an (unconstrained) clause and φ is a constant constraint. $C\sigma$ is an *instance* of $C[\![\varphi]\!]$ if σ is a substitution satisfying φ. Orderings on constrained clauses can be defined so that a clause C is bigger than a clause D if all ground instances of C are bigger than all ground instances of D.

Now we can define what G_E is. Given the equational theory E, G_E is the set of all equations of the form $f(x_1, \cdots, x_n) \approx x_0 [\![const(x_0) \wedge \cdots \wedge const(x_n)]\!]$, where f is a function symbol appearing in E, plus the equation $x \approx y [\![const(x) \wedge const(y)]\!]$ and the disequation $x \not\approx y [\![const(x) \wedge const(y)]\!]$.

The inference system I_1 is not dependent on E, and is defined similarly to the way I is defined, except that the clauses in the premise of a conclusion may have constraints (unconstrained clauses have the empty constraint), and the constraints are inherited by the conclusion. If the constraint in the conclusion of an inference is unsatisfiable, then the clause does not need to be added. The closure rules of I_1 are given in Figure 3.

These closure rules are meant to simulate the closure rules of I. We will prove that any inference of $E \cup S$ using inference system I can be simulated with an inference from $E \cup G_E$ using inference system I_1. So, the existence of a clause in the saturation of $E \cup G_E$ only means that an instance of that clause *may* exist in the saturation of $E \cup S$. Therefore, the constrained clauses that are derived by $E \cup G_E$ cannot be used for deletion of another clause. The deletion rules of I_1 are given in Figure 4.

Of course, it is always OK to subsume a clause by a renaming, because then the subsumed clause will be a duplicate of the other one.

We will assume that E is finite and saturated by these inference rules, because our results will require that this is possible.

Before presenting the first important theorem of this paper, we need a couple more definitions. Let $C[\![\varphi]\!]$ be a clause. Then $C\sigma$ is a *constraint instance* of $C[\![\varphi]\!]$ if $Dom(\sigma) = Vars(\varphi)$ and the range of σ only contains extension constants. Notice that σ satisfies φ because all of its constrained variables have been

Right Paramodulation:

$$\frac{G \cup \{u[s'] \approx v \vee C\,[\![\,\varphi\,]\!], s \approx t \vee D\,[\![\,\Psi\,]\!]\}}{G' \cup \{(u[t] \approx v \vee C \vee D\,[\![\,\varphi \wedge \Psi\,]\!])\sigma\}}$$

where $G' = G \cup \{u[s'] \approx v \vee C\,[\![\,\varphi\,]\!], s \approx t \vee D\,[\![\,\Psi\,]\!]\}$, $\sigma = mgu(s, s')$, $u[s'] \approx v$ is selected in its clause, $s \approx t$ is selected in its clause, $u[s']\sigma \not\preceq v\sigma$, $s\sigma \not\preceq t\sigma$, and s' is not a variable unless s' is constrained and D is not empty.

Left Paramodulation:

$$\frac{G \cup \{u[s'] \not\approx v \vee C\,[\![\,\varphi\,]\!], s \approx t \vee D\,[\![\,\Psi\,]\!]\}}{G' \cup \{(u[t] \not\approx v \vee C \vee D\,[\![\,\varphi \wedge \Psi\,]\!])\sigma\}}$$

where $G' = G \cup \{u[s'] \not\approx v \vee C\,[\![\,\varphi\,]\!], s \approx t \vee D\,[\![\,\Psi\,]\!]\}$, $\sigma = mgu(s, s')$, $u[s'] \approx v$ is selected in its clause, $s \approx t$ is selected in its clause, $u[s']\sigma \not\preceq v\sigma$, $s\sigma \not\preceq t\sigma$, and s' is not a variable unless s' is constrained and D is not empty.

Equational Resolution

$$\frac{G \cup \{u \not\approx v \vee C\,[\![\,\varphi\,]\!]\}}{G' \cup \{(C\,[\![\,\varphi\,]\!])\sigma\}}$$

where $G' = G \cup \{u \not\approx v \vee C\,[\![\,\varphi\,]\!]\}$, $\sigma = mgu(u, v)$ and $u \not\approx v$ is selected.

Equational Factoring

$$\frac{G \cup \{s \approx t \vee s' \approx u \vee C\,[\![\,\varphi\,]\!]\}}{G' \cup \{(s \approx u \vee t \not\approx u \vee C\,[\![\,\varphi\,]\!])\sigma\}}$$

where $G' = G \cup \{s \approx t \vee s' \approx u \vee C\,[\![\,\varphi\,]\!]\}$, $\sigma = mgu(s, s')$, $s \approx t$ is selected, $t\sigma \not\preceq u\sigma$, $s\sigma \not\preceq t\sigma$ and $s'\sigma \not\preceq u\sigma$.

Fig. 3. Constrained Rules for Closure

Subsumption:

$$\frac{G \cup \{C, C' [\![\varphi]\!]\}}{G \cup \{C\}}$$

where C is a member of E, and $C\sigma \subseteq C'$ for some substitution σ.

Simplification:

$$\frac{G \cup \{C[l'] [\![\varphi]\!], l \approx r\}}{G \cup \{C[r\sigma] [\![\varphi]\!], l \approx r\}}$$

where $l \approx r$ is a member of E, $l' = l\sigma$, $l\sigma \succ r\sigma$, and $C[l'] \succ (l\sigma \approx r\sigma)$.

Tautology Deletion:

$$\frac{G \cup \{t \approx t \vee C [\![\varphi]\!]\}}{G}, \quad \frac{G \cup \{s \approx t \vee s \not\approx t \vee C [\![\varphi]\!]\}}{G}$$

Fig. 4. Constrained Rules for Deletion

replaced by constants. Therefore, $C\sigma$ is an instance of $C [\![\varphi]\!]$. However, $C\sigma$ may not be ground, because unconstrained variables are not substituted for.

Theorem 2. *Let E be a finite set of clauses, saturated by I. Let G be the set of all clauses generated in a finite saturation of $E \cup G_E$ by I_1. Let S be a set of flat ground equations and disequations. Then we can saturate $E \cup S$ by I such that every clause generated in the saturation is a constraint instance of a clause in G, or is a flat f-rule where f does not appear in E.*

The proof of this theorem, which appears in [8], is basically a lifting argument, which shows that every infernce in the saturation of $E \cup S$ can be lifted to an inference in the saturation of $E \cup G_E$.

For a constrained clause $C [\![\varphi]\!]$, define the *width* of $C [\![\varphi]\!]$, written as $w(C [\![\varphi]\!])$ to be the number of constrained variables in C. For a set of clauses S, define the *width* of S to be $w(S) = max\{w(C) \mid C \in S\}$.

A saturation can be viewed as a graph, so that each clause generated (modulo renaming) labels a node in the graph. If there is an inference in the saturation with $C_1, \cdots, C_n$ as the premises and C as the conclusion (modulo renaming), then there is an edge from each node labeled with C_i to the node labeled with C. We call this graph a *saturation graph.* The saturation is called *cyclic* if the saturation graph contains a cycle, otherwise the saturation is called *noncyclic.*

Suppose we have a noncyclic saturation graph of the saturation of S, and suppose that C is generated in that saturation. Then C will label a node in the saturation graph. We define a *proof* of C inductively, as a subgraph of the saturation graph. If $C \in S$ then a single node labeled with C is a *proof* of C. Suppose there is an inference with premises $C_1, \cdots, C_n$ and conclusion C, and suppose that $\Pi_1, \cdots, \Pi_n$ are proofs of $C_1, \cdots, C_n$ respectively. Then the subgraph containing $\Pi_1, \cdots, \Pi_n$, plus the node labeled by C and the edges from each C_i to C is a proof of C. Note that a proof of C is just one way of deriving C. Since we have said that the saturation graph is noncyclic, there are only a finite number of proofs of C. A proof is actually a dag with C labeling the root and members of S labeling the leaves.

An immediate corollary to Theorem 2 is the following.

Corollary 1. *Let E be a finite set of clauses, saturated by I. Let G be the set of all clauses generated in a finite saturation of $E \cup G_E$ by I_1. Let S be a set of flat ground equations and disequations. Then the satisfiability of $E \cup S$ is decidable in time $O(|S|^{w(G)})$. If the saturation of $E \cup G_E$ is noncyclic, then the satisfiability of $E \cup S$ is decidable in time $O(|S| \times lg(|S|))$.*

Proof. Theorem 2 shows that every clause generated in the saturation of $E \cup S$ will be a constraint instance of a clause in G or a flat f-rule where f does not appear in E. There are at most $|S|$ extension constants. Therefore, if C is a clause in G, then there are at most $|S|^{w(C)}$ constraint instances of C. If all symbols of S appear in E, then this immediately shows that there are $O(|S|^{w(G)})$ clauses appearing in the saturation of $E \cup S$, since there are a constant number of clauses in G.

But suppose there is a symbol f which appears in E but not in S. Then $E \cup S$ will contain some flat f-rules. But the only way a new rule containing the symbol f can be created is if a constant is reduced in a flat f-rule, and then a new flat f-rule results. These flat f-rules are the only clauses generated that are not constraint instances of G. So we need to count how many times the constants of a flat f-rule can be reduced. Note that the precedence of the extension constants has been designed specifically so that there is no reduction sequence $c_1 \succ c_2 \succ \cdots \succ c_n$ of extension constants that is longer than $lg(|S|)$. Since each clause contains no more than a constant number of extension constants, the extension constants in that clause will be reduced by an Orient rule $O(lg(|S|))$ times. Since there are at most $|S|$ flat f-rules, this gives us $O(|S| \times lg(|S|))$ clauses that originate from flat f-rules.

We have shown that there are $O(|S|^{w(G)})$ constraint instances of G, and there are $O(|S| \times lg(|S|))$ clauses originating from flat f-rules. So there are $O(|S|^{w(G)} + (|S| \times lg(|S|)))$ clauses generated in the saturation of $E \cup S$. Since there is a clause $x \approx y \llbracket const(x) \wedge const(y) \rrbracket$ in G_E, then $w(G)$ is at least 2, so the first term dominates the second, and therefore there are $O(|S|^{w(G)})$ clauses generated.

Now, suppose the saturation of $E \cup G_E$ is noncyclic. Consider a clause C in the saturation graph of $E \cup G_E$. We need to consider how many constraint instances of C will appear in the saturation of $E \cup S$. Since the saturation graph is noncyclic, there are only a constant number of proofs of C in the saturation graph. Let's consider one of the proofs of C. Since each clause can be reduced $O(lg(|S|))$ times, and since there are $O(|S|)$ constraint instances of each member of G_E initially, there will be only $O(|S| \times lg(|S|))$ constraint instances of the clauses at the leaves of this dag, which appear in the saturation of $E \cup S$. Every extension constant that appears in a clause labeling another node of this dag must have come from one of the members of G_E. Since the Orient rule implies that all extension constants are reduced in the same order in all clauses, then the constraint instances of any clause in this proof must be directly determined by the constraint instances of the clauses at the leaves of the proof. Therefore, there will be only $O(|S| \times lg(|S|))$ constraint instances of every clause in this proof. We mentioned that there are only a constant number of proofs of C, which means that there can only be $O(S \times lg(|S|)$ constraint instances of each clause in the saturation. And since the size of G is constant, we see that only $O(S \times lg(|S|))$ clauses can be generated in the saturation of $E \cup S$. $\square$

6 Extending the Inference System

There are some theories E that can't be handled by the inference system I_1 given in the previous section. The problem with these theories is the saturation keeps generating longer and longer clauses. But only certain types of literals will appear in these clauses. So we can extend inference system I_1 to I_2 to handle these theories. This will also give us a decision procedure, but the complexity of the decision procedure goes from polynomial to exponential.

We define a new type of literal that represents zero or more literals of the same form. So, if L is a literal, then we will define a *multiliteral* L^* to represent zero or more copies of renamings of L. The technicalities follow.

Given a literal L and a constraint φ, the substitution σ is called a *constraint renaming substitution* of L in φ if $Dom(\sigma) = Vars(L) \cap Vars(\varphi)$. and the range of σ contains only variables.

Now we can define the semantics of a clauses containing multiliterals. First we define an *expansion* of a clause recursively. The clause $C\llbracket \varphi \rrbracket$ has the *expansion* $C\llbracket \varphi \rrbracket$. The clause $L^* \vee C\llbracket \varphi \rrbracket$ has the *expansion* $C\llbracket \varphi \rrbracket$. The clause $L^* \vee C\llbracket \varphi \rrbracket$ also has the *expansion* $L\sigma \vee L^* \vee C\llbracket \varphi \wedge \varphi\sigma \rrbracket$, where σ is a constraint renaming substitution of L in φ such that the range of σ contains only fresh variables. An expansion of a clause is called a *full expansion* if it has no multiliterals. The clause $D\sigma$ is an *instance* (*constraint instance*) of $C\llbracket \varphi \rrbracket$ if $D\llbracket \Psi \rrbracket$ is a full expansion of $C\llbracket \varphi \rrbracket$ and $D\sigma$ is an instance (constraint instance) of $D\llbracket \Psi \rrbracket$. Notice that a clause has a simply exponential number of constraint instances.

Now we are ready to define some inference rules, called **Collapse 1**, **Collapse 2** and **Collapse 3**.

$$\frac{G \cup \{L \vee L\sigma \vee C\llbracket \varphi \rrbracket\}}{G \cup \{L^* \vee C\llbracket \varphi \rrbracket\}}$$

where σ is a constraint renaming substitution of L in φ.

$$\frac{G \cup \{L^* \vee L\sigma \vee C\llbracket \varphi \rrbracket\}}{G \cup \{L^* \vee C\llbracket \varphi \rrbracket\}}$$

where σ is a constraint renaming substitution of L in φ.

$$\frac{G \cup \{L^* \vee (L\sigma)^* \vee C \llbracket \varphi \rrbracket\}}{G \cup \{L^* \vee C \llbracket \varphi \rrbracket\}}$$

where σ is a constraint renaming substitution of L in φ.

Finally a rule called **Breakout**.

$$\frac{G \cup \{L^* \vee C \llbracket \varphi \rrbracket\}}{G \cup \{L^* \vee L\sigma \vee C \llbracket \varphi \wedge \varphi\sigma \rrbracket\}}$$

where σ is a constraint renaming substitution of L in φ, whose range contains only fresh variables. $L\sigma$ is called the *broken out literal*.

Proposition 1. *Any constraint instance of a premise of a Collapse or a Breakout inference is also a constraint instance of the conclusion of the inference.*

Given a closure rule R, we define a closure rule R' to be an *extension* of R if R' is identical to R except that Breakout may have been performed to one or both of the premises before the inference was performed, and the literal involved in the inference is the one that was broken out.

Let I_2 be the inference system whose closure rules are comprised of the closure rules of I_1, plus all extensions of those rules. The deletion rules of I_2 are the same as the deletion rules of I_1, with the addition of the three collapse rules. It is also possible to subsume a clause C' with a clause C if an expansion of C is a renaming of C'. The completeness theorem of I_1 also applies for I_2

Theorem 3. *Let E be a finite set of clauses, saturated by I. Let G be the set of all clauses generated in a finite saturation of $E \cup G_E$ by I_2. Let S be a set of flat ground equations and disequations. Then there is a saturation of $E \cup S$ by I such that every clause in that saturation is a constraint instance of a clause in G or is a flat f-rule, with f a symbol not occurring in E.*

The proof is basically the same as before. The only thing that changes is the complexity result in the corollary.

Corollary 2. *Let E be a finite set of clauses, saturated by I_2. Let G be the set of all clauses generated in a finite saturation of $E \cup G_E$ by I_2. Let S be a set of flat ground equations and disequations. Then the satisfiability of $E \cup S$ is decidable in time $O(2^{|S|})$.*

Proof. As noted above, there can be now be a simply exponential number of constraint instances of a clause. Otherwise, the proof is as before. □

7 Example

We illustrate our results with some examples from the paper [1]. First is the theory of lists.

Let E be the list theory defined by the following equations:

$$car(cons(x, y)) \approx x$$
$$cdr(cons(x, y)) \approx y$$

Then G_E consists of the following constrained equations and disequations:

$$car(x) \approx y \, [\![\, const(x) \wedge const(y) \,]\!]$$
$$cdr(x) \approx y \, [\![\, const(x) \wedge const(y) \,]\!]$$
$$cons(x, y) \approx z \, [\![\, const(x) \wedge const(y) \wedge const(z) \,]\!]$$
$$x \approx y \, [\![\, const(x) \wedge const(y) \,]\!]$$
$$x \not\approx y \, [\![\, const(x) \wedge const(y) \,]\!]$$

Consider a Right Paramodulation inference between the first member of E and the third member of G_E:

$$\frac{car(cons(x, y)) \approx x \quad cons(x', y') \approx z' \, [\![\, const(x') \wedge const(y') \wedge const(z') \,]\!]}{car(z') \approx x' \, [\![\, const(x') \wedge const(z') \,]\!]}$$

We removed the constraint $const(y')$ from the conclusion, since y' does not appear in the clause. The conclusion will be immediately deleted because it is a renaming of a member of G_E. However, this inference will determine two edges in the saturation graph, one from the node labeled by the first premise to the node labeled by the conclusion, and the another one from the node labeled by the second premise to the node labeled by the conclusion.

There is also an inference between the second member of E and the third member of G_E, but that is also deleted because it is a renaming of something in G_E. There are no other possible inferences. Therefore $E \cup G_E$ is the saturation of itself. There were some edges added to the saturation graph, but it did not create a cycle. Therefore, this shows that the problem of the satisfiability of $E \cup S$ for any set of equations and disequations S can be decided in time $O(|S| \times lg(|S|))$.

Now let us consider the case where E is the theory of arrays, also from [1]. In this example, *select* is a function that takes an array and an index position, and returns the element at that index position. *store* is a function that takes an array, an index position and en element, and returns the array that results when that element is put in that position in the array.

$$select(store(x, y, z), y) \approx z$$
$$y \approx w \vee select(store(x, y, z), w) = select(x, w)$$

Then G_E consists of the following constrained equations and disequations:

$$select(x, y) \approx z \, [\![\, const(x) \wedge const(y) \wedge const(z) \,]\!]$$
$$store(x, y, z) \approx w \, [\![\, const(x) \wedge const(y) \wedge const(z) \wedge const(w) \,]\!]$$
$$x \approx y \, [\![\, const(x) \wedge const(y) \,]\!]$$
$$x \not\approx y \, [\![\, const(x) \wedge const(y) \,]\!]$$

After saturation we get the clauses of E and the following clauses. Notice that some of the members of G_E were subsumed by clauses containing multiliterals.

$$select(x,y) \approx z \vee (y \approx w)^* \; [\![\, const(x) \wedge const(y) \wedge const(z) \wedge const(w) \,]\!]$$
$$store(x,y,z) \approx w \vee (x' \approx y')^* \; [\![\, const(x) \wedge const(y) \wedge const(z) \wedge const(w) \wedge const(x') \wedge const(y') \,]\!]$$
$$(x \approx y)^* \; [\![\, const(x) \wedge const(y) \,]\!]$$
$$x \not\approx y \vee (z \approx w)^* \; [\![\, const(x) \wedge const(y) \wedge const(z) \wedge const(w) \,]\!]$$
$$select(x,w) \approx select(z,w) \vee (y \approx w)^* \; [\![\, const(x) \wedge const(y) \wedge const(z) \wedge const(w) \,]\!]$$

8 Conclusion

We consider our results to be an extension of previous results. In [10], there is an algorithm for deciding the theory of lists. In [9] there is a decision procedure for the theory of arrays, and in [11] a decision procedure for the theory of arrays with extensionality. In [1], a general result is given which subsumes the previous results and gives a general procedure which fits everything into a saturation theorem proving framework. What we have done is to automate and generalize the results of [1]. Given a theory for a data structure in first order logic with equality, we provide a procedure such that when it halts it will produce an algorithm for the theory of that data structure. Not only will it give an algorithm, but it will also give a bound on the complexity of that algorithm. For *noncyclic* theories like arrays, the complexity is $O(n \times lg(n))$. For some other theories, including those that have no disjunctions, the complexity is polynomial. For other theories, like the theory of arrays, it is simply exponential. Our results are an extension of the congruence closure algorithm given in [4], which gives a nice proof of an $O(n \times lg(n))$ bound for the empty theory. We showed that their algorithm can be extended to *noncyclic* theories, like the theory of arrays. Our work is also in the spirit of [5], because it gives an automatic method to derive decision procedures and complexity results, that of saturation of a particular theory.

Acknowledgments

Many of these results were obtained while the first author was visiting MPI Saarbrücken, and we thank Harald Ganzinger for many helpful discussions. Also, this work is based heavily on the work of [1]. We thank Alessandro Armando, Silvio Ranise and Michaël Rusinowitch for discussions of their paper.

References

1. A. Armando, S. Ranise and M. Rusinowitch. Uniform Derivation of Decision Procedures by Superposition. In *Proceedings 15th Workshop on Computer Science Logic.*, LNCS vol. 2142, 513-527, 2001.
2. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge, 1998.
3. L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. In *Journal of Logic and Computation* 4(3), 1-31, 1994.

4. L. Bachmair and A. Tiwari. Abstract Congruence Closure and Specializations. In *Proceedings 17th Conference on Automated Deduction.* LNCS vol. 1831, 64-78, 2000.
5. D. Basin and H. Ganzinger. Automated complexity analysis based on ordered resolution. In *J. Association for Computing Machinery* 48(1), 70-109, 2001.
6. N. Dershowitz. Termination of Rewriting. In *Journal of Symbolic Computation* 3, 69-116, 1987.
7. D. Kapur. Shostack's Congruence Closure as Completion. In *Proceedings 8th Conference on Rewriting Techniques and Applications.* LNCS vol. 1232, 23-37, 1997.
8. C. Lynch and B. Morawska. Automatic Decidability `http://www.clarkson.edu/~clynch/papers/auto_full.ps/`, 2002.
9. G. Nelson. Techniques for Program Verification. In *Technical Report CSL-81-10*, Xerox Palo Alto Research Center, June 1981.
10. G. Nelson and D. C. Oppen. Fast Decision Procedures Based on Congruence Closure. In *J. Association for Computing Machinery* 27(2), 356-364, 1980.
11. A. Stump, D. L. Dill, C. W. Barrett and J. Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *Proceedings 16th IEEE Symposium on Logic in Computer Science,* IEEE Computer Society Press, 29-37, 2001.