Tactic-Based Theorem Proving and Knowledge-Based Forward Chaining: An Experiment with Nuprl and Ontic

Wilfred Z. Chen*

TR 92-1276
March 1992

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*The author would like to acknowledge support for this work under grant CCR-9108062 and ONR grant N000014-88K-0409.

Tactic-based Theorem Proving and Knowledge-based Forward Chaining: an Experiment with Nuprl and Ontic

Wilfred Z. Chen*

Department of Computer Science
Cornell University
Ithaca, NY 14853 USA
chen@cs.cornell.edu

Abstract. We explore a new approach to interactive theorem proving which combines a knowledge-based notion of obvious reasoning with a tactic-based notion of obvious reasoning. We study the interplay of two particular systems and apply our approach to a proof of the Fundamental Theorem of Arithmetic. We achieve both shorter and more robust proofs. It is our opinion that the kind of control information contained in interactive proofs is a more important issue than their mere size. We analyze our proof scripts in terms of the control information they contain and suggest that stronger knowledge-based notions of obviousness and declarative representations of tactics are needed to further reduce low-level control information.

1 Introduction

Tactic-based theorem proving has received a lot of attention lately [16, 10, 15, 14, 31]. Forward chaining theorem proving has also been studied [9, 22]. In this paper, we explore the potential for combining these two approaches. In particular, we study the combination of **Nuprl** and **Ontic**.

The motivation for studying the combination of these two different approaches is the hope that their strengths and weaknesses will compensate each other and produce proofs that are both shorter and more robust.

Although the current collection of tactics in **Nuprl** has provided indispensable assistance in developing proofs [17, 26, 2], these tactics are often sensitive to minor variations in the formulation of the problem, to changes in the lemma library and to improvements in some standard tactics – the Autotactic, for example. This problem is acute – strengthening a tactic or the lemma collection can damage previously checked proof scripts. It is also general – other **LCF** descendants, such as **HOL**, suffer similarly: providing low-level control information to the system is not only extremely tedious, it also makes proofs harder to reuse. By applying knowledge in the form of a large lemma collection, we hope to reduce the amount of detail.

In contrast, **Ontic** proof scripts replay under any strengthening of the lemma collection, modulo practical feasibility considerations. But there is no mechanism to

* The author would like to acknowledge support for this work under NSF grant CCR-9108062 and ONR grant N000014-88K-0409.

encode simple patterns of inference, so similar subproofs are duplicated. So we hope to find regular patterns of inferences – clichés – that might be captured using tactics.

We expect fruitful cooperation between the tactic-style theorem proving and the forward chaining technique because the former is suitable for capturing and combining dynamic patterns in inferences while the latter does the same to static patterns, in the form of lemmas. Both forms of knowledge are necessary in problem solving.

How are these (dynamic) patterns discovered? Our first step is to implement a knowledge-based forward chaining autotactic and gain some experience by applying it to compress some known proofs in **Nuprl**.

This paper contains an introduction to the main ideas of the knowledge-intensive forward chaining technique and a report on a case study (the fundamental theorem of arithmetic). We obtain shorter and more robust proof scripts by combining the tactical approach and the forward chaining approach.

Proof length is not the only thing we seek to reduce, excessive control information should also be eliminated. Knowledge-intensive forward chaining reduces only one kind of control information. We have written tactics that implement simple heuristics to invoke the forward chaining tactic and greatly reduced both the lengths of proofs and the amount of control information they contain. How to further reduce low-level control information in proof scripts is also discussed at some length.

The paper is organized as follows: Section 2 lays down the general background of this work, Section 3 is an introduction to the forward chaining approach, to prepare the reader for a case study of integrating forward chaining into a tactic-based environment in Section 4, finally we summarize and expand in Section 5.

2 The Theorem Proving Context

2.1 Tactical Theorem Proving

Tactic-based theorem proving systems (**Nuprl**, **HOL**, **Isabelle**) are descendants of **LCF**. Although their object languages differ, a common feature is the meta-language ML [2]. Tactics are programs written in ML to refine a (sequent style) goal into (presumably) simpler subgoals. The simplest tactics are the primitive refinement rules. The higher-order programming style of ML facilitates the combination of simple tactics into complex ones.

In these systems, a collection of tactics defines a notion of obvious inferences.

In our work, we use a subset of the (object) language of **Nuprl**. The **Nuprl** subset we use is essentially an order-sorted first order logic with subset, pairing and lists. Equality reasoning in this language is simpler than in the full language of **Nuprl**. In particular, it permits the use of a single global equality [19].

2.2 Knowledge-Intensive Forward Chaining

There are two important issues in applying a large number of lemmas to a reasoning task: selecting the relevant lemmas and ensuring termination. **Ontic** addresses both

[2] **Oyster**, the Edinburgh version of **Nuprl**, uses PROLOG as its meta-language.

issues with the concept of focusing. The design objectives of **Ontic** are: (1) to obtain a practically effective proof verification system and (2) to model low-level human mathematical reasoning. It has been used to verify the Stone representation theorem for boolean lattices. Most existing interactive proof development systems do not share the second objective of **Ontic**.

Ontic has a knowledge-based (declarative) counterpart to the tactic-based (procedural) notion of obvious inference in the systems mentioned above.

We explain **Ontic** in more detail below.

3 The Forward Chaining Tactic

This section serves two purposes: to explain the basic ideas of **Ontic** and to present, at a high level, a particular **Ontic**-like system used in our work. Our version of **Ontic** removes some of the features of **Ontic** that are not necessary for our experiment – among them, automatic universal generalization [3].

3.1 Ontic in a Nutshell

The decision procedure **Ontic** is defined by a set of inference rules and a set of lemmas. Basically, inference rules are applied under a *mention restriction*, where only those instances containing subterms in the subterm closure of all input formulas are allowed; lemmas are applied under *focus restriction*, where all members in a certain closure of the set of lemmas are instantiated on a given set of terms, the *focus set*.

The reader is referred to [22] for a complete description of **Ontic** and to [23] for a general theory of tractable inferences.

3.2 Forward Chaining Inference Relations

We present the forward chaining autotactic as a family of tractable inference relations indexed by a finite set of terms, called *focus terms*.

The inference relation presented below is not new. It is different from those in the original **Ontic** mainly in the constructive validity of the inference rules and the use of (almost) standard first order syntax (**Ontic** takes advantage of non-standard syntax to obtain a larger tractable fragment [21]).

Let $\vdash$ denote the usual logical derivability relation in first order logic, specialized to our particular language. Let $\vdash\!\!\circ_{\mathcal{F}}$ denote the parameterized tractable approximations to $\vdash$ that we shall define. Let $\vdash\!\!\circ$ abbreviate $\vdash\!\!\circ_{\{\}}$.

For each $\mathcal{F}$, $\vdash\!\!\circ_{\mathcal{F}}$ is quickly (polynomial time) decidable fragment of $\vdash$. A *high level proof* consists of steps of single inference rules that define $\vdash$ mixed with steps of $\vdash\!\!\circ_{\mathcal{F}}$.

$\vdash\!\!\circ_{\mathcal{F}}$ will be presented in two steps: first the inference rules that make up $\vdash\!\!\circ_{\mathcal{F}}$, and second, the restrictions on when these rules are applicable. For the algorithms that implement these rules and restrictions, we refer the reader to the original papers of McAllester.

[3] There are two reasons for this: (1) tactics already perform introduction of universal quantifiers and, more subtly, (2) adding universal generalization reduces the efficiency of forward chaining considerably.

3.3 The Inference Rules

Here we give a set of inference rules that are constructively valid, so that forward chaining proofs using them can be converted to **Nuprl** proofs.

The inference rules that go into $\vdash\!\circ_{\mathcal{F}}$ shall be presented in sequent calculus style. So a typical rule would have the form

$$\frac{\Sigma \vdash\!\circ_{\mathcal{F}} \Phi \quad \Sigma' \vdash\!\circ_{\mathcal{F}'} \Psi}{\Sigma'' \vdash\!\circ_{\mathcal{F}''} \Theta}.$$

However, when the Σ's and $\mathcal{F}$'s are identical in all places, the simpler form $\frac{\Phi \quad \Psi}{\Theta}$ is used.

Propositional Rules Following are the so-called Boolean Constraint Propagation rules:

$$\frac{\Phi \quad \Psi}{\Phi \& \Psi} \quad \frac{\Phi \& \Psi}{\Phi} \quad \frac{\Phi \& \Psi}{\Psi} \quad \frac{\Phi}{\Phi \vee \Psi} \quad \frac{\Psi}{\Phi \vee \Psi} \quad \frac{\Phi \quad \Phi \Rightarrow \Psi}{\Psi} \quad \frac{\perp}{\Phi}$$

These rules do not define a complete inference relation for intuitionistic propositional logic because $\vee$–elim and $\Rightarrow$–intro are absent. These rules require searching or guessing hence are not used automatically in forward chaining.

We also have some derived rules:

$$\frac{\neg\Phi}{\neg(\Phi \& \Psi)} \quad \frac{\neg\Psi}{\neg(\Phi \& \Psi)} \quad \frac{\neg(\Phi \vee \Psi)}{\neg\Phi} \quad \frac{\neg(\Phi \vee \Psi)}{\neg\Psi} \quad \frac{\neg\Phi \quad \neg\Psi}{\neg(\Phi \vee \Psi)} \quad \frac{\neg\Phi \quad \Phi \vee \Psi}{\Psi} \quad \frac{\neg\Psi \quad \Phi \vee \Psi}{\Phi}$$

$$\frac{\Psi}{\Phi \Rightarrow \Psi} \quad \frac{\neg\Phi}{\Phi \Rightarrow \Psi} \quad \frac{\Phi \quad \neg\Psi}{\neg(\Phi \Rightarrow \Psi)} \quad \frac{\neg\Psi \quad \Phi \Rightarrow \Psi}{\neg\Phi} \quad \frac{\neg(\Phi \Rightarrow \Psi)}{\neg\Psi}$$

These are mostly those contrapositives of the primitive rules that are both constructively valid and do not require search. Although negation is defined via implication, for efficiency, it is also included as a primitive.

Equality Rules The forward chaining rules for equality reasoning are reflexivity, symmetry, transitivity and congruence (substitution of equals for equals). A complete decision procedure for these rules is congruence closure [20, 29, 11] which is used by all implementations of **Ontic**. Congruence closure is valid for the fragment of **Nuprl** used here [19].

Quantifier Rules

$$\frac{\Sigma \vdash\!\circ_{\mathcal{F}} \forall x : T.\Phi(x) \qquad \Sigma \vdash\!\circ_{\mathcal{F}} (a \textbf{ in } T)}{\Sigma \vdash\!\circ_{\mathcal{F} \cup \{a\}} \Phi(a)} (\forall\text{-E}) \qquad \frac{\Sigma \vdash\!\circ_{\mathcal{F}} \Phi(a) \qquad \Sigma \vdash\!\circ_{\mathcal{F}} (a \textbf{ in } T)}{\Sigma \vdash\!\circ_{\mathcal{F} \cup \{a\}} \exists x : T.\Phi(x)} (\exists\text{-I})$$

$\forall$-E is much more important than $\exists$-I in practice, and it will receive more attention in the following explanations.

Note the absence of $\forall$-I and $\exists$-E. $\forall$-I is usually applied by tactics in the beginning of a refinement style proof. $\exists$-E introduces new constants that the rest of the proof somehow refers to. If this is done by the forward chaining tactic, the rest of the proofs would have no access to these constants.

Type Inference Rules The following rules allow the inference of types of focus terms in the presence of set types:

$$\frac{}{\forall x : T.\, x \,\mathbf{in}\, T} \qquad \frac{\Sigma \vdash\!\!\circ_{\mathcal{F}} \Phi(a) \qquad \Sigma \vdash\!\!\circ_{\mathcal{F}} (a \,\mathbf{in}\, T)}{\Sigma \vdash\!\!\circ_{\mathcal{F}\cup\{a\}} a \,\mathbf{in}\, \{x : T | \Phi(x)\}}$$

Note that the first rule cannot be a lemma schema, because lemmas are only applied to focus terms and the application requires that '$x \,\mathbf{in}\, T$' be true.

3.4 Restrictions on Rule Applicability

In addition to the *focus restriction* which is implicit in the quantifier rules, all rules are subjected to the so-called *mention restriction.* These two restrictions together ensure that $\vdash\!\!\circ_{\mathcal{F}}$ is quickly decidable. Focus restriction can be understood by inspecting the quantifier rules carefully; mention restriction is the requirement that for each application of a forward chaining rule, the Σs, Φs, Ψs and Θs must all be subexpressions of the rule set, lemma library, current goal sequent or focus set.

As a consequence, from Φ we shall not infer $\Phi|\Psi_1,\ldots,\Phi|\Psi_n,\ldots$, unless the disjunction is mentioned (not only the Ψs).

Similarly, given $a = f(a)$, we do not infer $a = f(\ldots f(a)\ldots)$, unless the terms involved in the latter are actually mentioned. Note, however, that in this case congruence closure actually represents $a = f(\ldots f(a)\ldots)$ implicitly.

3.5 Examples

Example 1. Let $\Sigma = \{\forall x, y : int.\, x + y = y + x, \forall x : int.\, x + 0 = x\}$. Then we can prove $\Sigma \vdash \forall x : int.\, 0 + x = x$ in two 'high-level' steps:

$$\cfrac{\cfrac{\forall x, y : int.\, x + y = y + x \quad \forall x : int.\, x + 0 = x}{0 + x = x}\ (\vdash\!\!\circ_{\{x,0\}})}{\forall x : int.\, 0 + x = x}\ (\forall\text{-I})$$

Note that in the first step, all that the user has to supply is the focus set $\{x, 0\}$; applications of both lemmas and some equality inferences are done by the system.

In practice, due to the high complexity in terms of the focus set size (see [22]), we often limit the focus set size to that of the deepest quantifier nesting. The next example shows that a larger focus set means fewer steps.

Example 2. Let $f : S \to T$ and let $\Sigma = \{\forall x : T.\, \Phi(x), \forall x : S.\, \Phi(f(x)) \Rightarrow \Psi(x)\}$. Suppose we want to prove $\forall x : S.\, \Psi(x)$. Let a be a new constant of type S, then

$$\frac{\forall x : T.\, \Phi(x)}{\Phi(f(a))}\ (\vdash\!\!\circ_{\{f(a)\}}) \qquad \frac{\forall x : S.\, \Phi(f(x)) \Rightarrow \Psi(x)}{\Phi(f(a)) \Rightarrow \Psi(a)}\ (\vdash\!\!\circ_{\{a\}})$$

and $\Psi(a)$ follows by boolean constraint propagation.

Using the focus set $\mathcal{F} = \{a, f(a)\}$, the above can be done in a single step of $\forall$-E (boolean constraint propagation does not count as a *visible* step).

3.6 Some Properties of Ontic

What is the difference between **Ontic**'s use of lemmas and a naive backchaining search through the entire collection of lemmas? **Ontic** is polynomially bounded in some sense, hence can be regarded as forward chaining rather than searching. However, we wish to be a little more precise about the complexity of **Ontic**. The space complexity of **Ontic** is $\mathcal{O}(n^d)$, where n is the size of the focus set and d is the deepest quantifier nesting. In general, n is at least as large as d. In practice, d is about 3 or 4.

Subformulas Property The focused instantiation rule $\forall$-E lacks subformulas property in a rather strong sense.

Example 3. Let

$$\Sigma = \left\{ \begin{array}{l} \forall x, y, z : \mathcal{N}.\, max3(x,y,z) = max(max(x,y),z), \\ \forall x, y : \mathcal{N}.\, max(x,y) = max(y,x), \\ \forall x, y, z : \mathcal{N}.\, max(max(x,y),z) = max(x, max(y,z)) \end{array} \right\}$$

Then $max3(a,b,c) = max3(a,c,b)$ follows from Σ, with a one step proof which requires focusing on $\{a,b,c\}$. But $max3(a,b,c) = max3(c,a,b)$ requires, in addition, $max(a,b)$ in the focus set for a one step proof. Note that $max(a,b)$ does not occur as a subterm of the goal or any lemmas in the library.

Repeated Use of Lemmas Let $\Sigma = \{\forall x : int.\, x+1 > x\}$. It has the obvious consequence $(\ldots(a+1)+\ldots+1) > a$, but **Ontic** cannot deduce this in a single step because it needs to focus on the intermediate terms. Interestingly, we can infer the same from $a+1>a$, where a lemma about the monotonicity of $+$ is used multiple times, implicitly.

Consider again one of the lemmas in Example 2. Let $\Sigma = \{\forall x : S.\, \Phi(f(x)) \Rightarrow \Phi(x)\}$. Then

$$\frac{\forall x : S.\, \Phi(f(x)) \Rightarrow \Phi(x) \qquad \Phi(f^n(a))}{\Phi(a)} \; (\vdash\!\circ_{\{a, f(a), \ldots, f^{n-1}(a)\}})$$

The conclusion is equally obvious for any n, but the focus set size grows linearly with it. Induction and rewriting are two possible solutions to this.

Sequencing of Focus Sets Because of the space complexity associated with large focus sets, simulating a large focus set with a sequence of smaller focus sets is appealing. Let $\vdash\!\circ_{<\mathcal{F}_1,\ldots,\mathcal{F}_n>}$ denote the inference relation based on the sequence of focus sets.

In order for the above to be different from $\vdash\!\circ_{\mathcal{F}_n}$, facts derived from the previous focus sets must be saved. The definition of $\vdash\!\circ_{<\mathcal{F}_1,\ldots,\mathcal{F}_n>}$ depends on what facts are saved between successive focus sets. A reasonable choice is to use the mention restriction again. Under this notion of *focus sequencing*, $\vdash\!\circ_{<\mathcal{F}_1,\mathcal{F}_2>}$ is not equivalent to $\vdash\!\circ_{<\mathcal{F}_2,\mathcal{F}_1>}$ in general.

Example 4. Suppose that $r(b)$ is mentioned, so that it will be saved if proven true from any focus context (this can be done by adding the trivial hypothesis $r(b) = r(b)$) and consider

$$\Sigma = \{p(a,b), q(b,c), p(x,y) \rightarrow r(y), q(x,y) \& r(x) \rightarrow s(x,y)\} \vdash\!\!\circ_{\mathcal{F}s} s(b,c)$$

The focus set $\{a,b,c\}$ will get the goal in one step, the focus sequence $\mathcal{F}s = [\{a,b\};\{b,c\}]$ will also prove the goal, but the focus sequence $\mathcal{F}s = [\{b,c\};\{a,b\}]$ will not prove the goal.

Moreover, there exists a focus set $\mathcal{F}$ such that no (non-repetitive) focus sequencing over proper subsets of $\mathcal{F}$ is equivalent to focusing on $\mathcal{F}$ itself (rough idea: construct a cycle of dependencies).

Relationship to Unification-based Systems Because $\forall$-E and $\exists$-I are restricted by the focus set, matching and unification can be avoided, and lemma instantiation in **Ontic** is implemented via pre-instantiation on generic constants, focus binding with these constants and congruence closure. This approach gives both the tractability of **Ontic** and some of the weaknesses mentioned above, because lemma application on intermediate results is limited.

What is a better trade-off? This question can only be answered in the context of a theorem proving environment: when several tools are available, the virtue of each may not be its individual power, but rather how well it works with other tools. We cannot give a full answer to this question here.

4 The Fundamental Theorem of Arithmetic: a case study

Here we wish to evaluate the effectiveness of applying mathematical knowledge in the form of a lemma library via focused forward chaining similar to **Ontic**. Our choice of number theory is motivated by the need to improve arithmetical reasoning in **Nuprl**. We want to see whether the knowledge-based approach gives a significant boost to the tactical approach. Furthermore, **Nuprl** has two tactics for arithmetical reasoning: **Arith** and **Mono**. Their presence offers possibilities of cooperation and a limited form was exploited in this study.

Arith is a decision procedure for a fragment of number theory (see [9] for details). It is part of Autotactic. **Mono** is a fixed collection of derived rules of inference involving inequalities.

Below we discuss various aspects of the experiment.

4.1 Implementation

We have implemented a version of the forward chaining decision procedure containing most of the features explained in previous section. Some omissions or limitations are listed below.

We have limited the sizes of focus sets to 4, the deepest quantifier nesting in our lemma library. This is necessary for reasonable response times, due to the space complexity of **Ontic** mentioned earlier.

Type inference was not implemented as part of the forward chaining tactic. The subtype relation is simulated via lemmas. Because there were only a few subtypes involved, this approach was sufficient. The typing lemmas are

$$\begin{array}{l}
\forall x : N. (x \mathbf{\ in\ } int) \& (0 \leq x) \\
\forall x : N+. (x \mathbf{\ in\ } N) \& (x \mathbf{\ in\ } int) \& (1 \leq x) \\
\forall x : \{2..\}. (2 \leq x) \& (x \mathbf{\ in\ } N+) \& (x \mathbf{\ in\ } N) \& (x \mathbf{\ in\ } int) \\
\forall x : int. (0 \leq x) \Rightarrow (x \mathbf{\ in\ } N) \& (1 \leq x) \Rightarrow (x \mathbf{\ in\ } N+) \& (2 \leq x) \Rightarrow (x \mathbf{\ in\ } \{2..\})
\end{array}$$

One limitation of this approach is that types like 'prime factors of x' are not expressible to our forward chainer. We refer to our implementation of the forward chaining tactic as **OnticTac** below.

4.2 The Library

The lemma collection determines the effectiveness of **OnticTac**. Adding more lemmas can make **OnticTac** arbitrarily stronger, but also slower, because of the memory performance of **OnticTac**. Furthermore, adding arbitrary lemmas to the library can also invalidate the result of the experiment. For both of these reasons, we have limited the lemma library to a moderate size and only included relatively general facts.

There are 109 lemmas expressing basic facts of number theory: including type inclusion, basic facts about constants, and basic facts about primitive and defined operators. The rather large total number is due to the number of redundant lemmas included.

The redundancy can be classified by form as follows:

- specializations of some lemmas on constants
- overlappings of certain frequently used lemmas
- obvious consequences of **Arith** and **Mono**, in a sense, **OnticTac** acted as a cache for **Arith** and **Mono**

The justifications for the redundancies can be seen by classifying them according to function:

- Obvious facts not provable by Autotactic.
- Obvious facts that, though provable by Autotactic, need to be chained together with other forward chaining proof steps. This is a form of internalization of some of the power of Autotactic by **OnticTac**. This can also be thought of as using **OnticTac** to cache some of the results of Autotactic. Many simple rewriting or monotonicity lemmas are included for this reason.
- Obvious facts that, though establishable by **Arith**, are not established because the simplification procedure in **Arith** goes further. Using **Arith** to prove these lemmas and then using these lemmas with a particular focus binding can be thought of as forcing **Arith** to retain particular intermediate results.
- Some special version or overlaps of more general lemmas that have frequent applications. These are included to reduce the sizes of focus sets.

For instance, due to the frequent occurrence of integer constants like 0 and 1, many lemmas explicitly stating their properties are included: $0 < 1, 1 < 2$ or $\forall x : int.\, 1 \leq x \iff 0 < x$, $\forall x : int.\, x + x = x * 2 = 2 * x$, etc. The extra equation involving 2 is an example of pre-instantiating commutativity of $+$, this avoids having to focusing on 2. As another example, $\forall x, y, z : int.\, x * (y + z) = (y + z) * x = x * y + x * z$ is an overlaps of distributivity and commutativity. This saves the focus term that corresponds to $y + z$, otherwise needed for commutativity. Monotonicity lemmas, such as $\forall m, n, i : int.\, 0 < i \Rightarrow m < n \Rightarrow i * m < i * n$ also make up a fair portion of the library.

We conjecture that shallow application of a large number of facts might subsume some deep applications of a few facts if the results of deep applications can be packaged properly by introducing meta-constructs.

As an example of when not to include redundant lemmas, imagine adding the instantiations of all lemmas on a particular ground term t (a form of partial evaluation). Although this reduces the focus set in the sense that now we do not ever have to focus on t, the space cost of doing this is at least as high as always adding an extra focus term t, which is a rather inflexible way of using the focusing mechanism.

4.3 Examples, Issues and Observations

All examples shown in this section appear as they do in actual **Nuprl** sessions. `AFC`, `SFC` and `ASFC` are interface tactics to **OnticTac**. `AFC` selects the focus set from the variable declarations among the hypotheses, `SFC` takes an explicitly given list of focus terms with their typing as an argument, `ASFC` does both. Also, '`--*`' and '`--`' are used to indicate an Autotactic wrapper – meaning that the Autotactic is applied to unproven subgoals. `Seq` (a generalized version of the cut rule) sequences in a list of formulas as intermediate goals, producing subgoals with each of these and the original conclusion as conclusions and with previous formulas in the list as additional hypotheses. `THENL` matches a list of tactics to the list of subgoals.

Proofs are shorter

Example 5.

```
>> ∀i,j,p:N+.prime(p) & p|i*j=>p|i ∨ p|j
  BY -- Repeat Intro
  | 1. i:N+
  | 2. j:N+
  | 3. p:N+
  | 4. p prime
  | 5. p|i*j
  |->> p|i ∨ p|j
    BY -- Cases ['p|i';'¬p|i']
          THENL [AFC;ILeft;IRight THEN AFC]
```

A proof without **OnticTac** [18] has a size of 43 top level refinement steps and 828 primitive refinement steps (abbreviated as 43/828); it is too large to be shown here.

The new proof has size (2/68). Even including the proofs of the lemmas

$$\forall p, q : \mathcal{N}^+ . \, prime(p) \& \neg p | q => rel\_prime(p, q) \quad (5/135)$$
$$\forall a, b, c : \mathcal{N}^+ . \, a | b * c \& rel\_prime(a, b) => a | c \quad (4/95)$$
$$\forall i, n : int. \, i | n \vee \neg i | n \quad (13/109)$$

the new proof is significantly shorter than the old one. Two of the lemmas mentioned here are applied by **OnticTac** in a single step for the second case of the case analysis. The proofs of these lemmas have a combined size of (9/230). In comparison, the old proof has a size of (36/661) for this case.

On the whole, when we add up all the significant lemmas leading up to the prime factor existence lemma, the new proof is shorter than the previous proof by a factor of about 2-3, both in terms of top level (visible) refinement steps and the primitive refinement steps [4].

This reduction in proof length is a result of several factors: (1) **Mono** was not available at the time when old proof was completed, (2) **OnticTac** chains together applications of several lemmas in a single step and (3) redundant lemmas increase the effect of (2). The absence of the monotonicity rule turns out not to affect the comparison significantly because in Howe's proof[18], many monotonicity lemma had been used instead.

Proofs are less brittle Using **OnticTac** produces more stable proofs because user directives contain no explicit references to lemma names or hypothesis numbering . Furthermore, provability is monotonic in the set of lemmas and hypotheses – adding or strengthening lemmas or hypotheses does not invalidate a proof step.

A simple example to compare how old tactics and **OnticTac** reference lemmas and hypotheses.

Example 6. Reducing flaky references by using **OnticTac**. We show proofs of the same goal with and without **OnticTac**.

```
1,...,7. {omitted}
8. a<b
>> 0<b-a
  BY AFC
```

```
1,...,7. {omitted}
8. a<b
>> 0<b-a
  BY -- LemmaFromHyps 'add_num_to_l' [8] ['-a']
```

[4] The number of top level refinement steps is the more important measure of the amount of user interaction. The primitive refinement steps are included to rule out the possible biasing effect of collapsing multiple steps into a single step using tacticals.

The lemma **add_num_to_l** states $\forall m, n, i : int.\,(m < n) \Rightarrow (m + i < n + i)$. A similar lemma and $\forall x : int.\, x - x = 0$ are referenced automatically in the forward chaining proof. The latter was generated explicitly as a subgoal of '`LemmaFromHyps`' in the old proof and proven by **Arith** (hidden in the Autotactic wrapper). '...' elides irrelevant hypotheses in the sequent.

This example shows the reduction of extraneous dependencies by forward chaining. Larger proofs benefit even more from this kind of increase in robustness.

Cooperation of OnticTac and Autotactic Rewriting of moderate length plus congruence reasoning with small number of ground equations seems quite obvious to human users, but poses great difficulty to automation. Roughly, Autotactic is weak because it only uses a fixed set of rewrite rules, does not make use of (ground) equational hypotheses in rewriting, and cannot incorporate equational lemmas. Congruence closure based forward chaining uses ground equations very effectively, but is weak in reasoning about equality and inequalities because multiple instances of lemmas are often needed, requiring large focus sets.

It seems that **OnticTac** and Autotactic might be able to compensate for each other's deficiencies through some mechanism of cooperation. We explored one particular form of cooperation: dividing proof burdens through the use of the cut rule (called 'seq' in **Nuprl**) and the 'THENL' construct of the meta-language.

As a first step, mediation is done by the human user by explicitly distributing proof obligations between Autotactic and **OnticTac**. Automation of this turned out to be difficult, for reasons given below.

The human mediation introduced a lot of low level control information into proof scripts. This is undesirable in general.

We illustrate these issues and the user decisions by examples.

Example 7. This is one subgoal in the proof of a lemma that says two numbers are relatively prime iff the greatest-common-divisor of them equals to 1.

```
1. l:int
2. 2<l
3. ∀a,b:N+.a+b<(l-1)&rel_prime(a,b)=>(a,b)=1
4. a:N+
5. b:N+
6. a+b<l
7. rel_prime(a,b)
8. a<b
>> a+(b-a)<l-1
  BY -- Seq ['b<l-a';'1≤a';'l-a≤l-1';'b<l-1']
        THENL [AFC;AFC;ASFC [('1','int')];Idtac;Idtac]
```

Idtac leaves its goal unchanged, hence passing it to the Autotactic wrapper, hence to **Arith**. As a result, $b < l - a, 1 \leq a$ and $l - a \leq l - 1$ are proved by **OnticTac**; $b < l - 1$ and $a + (b - a) < l - 1$ are proved by **Arith**.

Although this proof is robust in the sense of being monotonic in the lemma collection and hypotheses, it still requires a lot of detailed user knowledge about both **OnticTac** and **Arith** in breaking up the focus set and distributing subgoals.

The above goal can also be proved by a single step of **OnticTac**, with the focus set $\{a, b, -a, l, 1, a-1, l-1, a+(b-a)\}$, assuming the standard lemmas about commutativity and associativity of $+$, cancellation with $-$ and transitivity of $<$ and $=$. A focus set of such a size would have taken significantly longer running time than one of size 4 and, even more critically, more space. Moreover, the choice of such a focus set still requires a lot of knowledge about **OnticTac**.

As another example of the kind of knowledge required to guide this kind of proofs by choosing cut formulas and focus sets, here is the complete proof of a lemma mentioned earlier.

Example 8.

```
>> ∀a,b,c:N+.a|b*c&rel_prime(a,b)=>a|c
  BY -- Repeat I
  | 1. a:N+
  | 2. b:N+
  | 3. c:N+
  | 4. a|b*c
  | 5. rel_prime(a,b)
  |->> a|c
    BY -- Seq ['(a,b)=1'] THENL [AFC; OnLastHyp (RPE ['r';'s'])]
    | 6. r:int
    | 7. s:int
    | 8. r*a+s*b=1
    |->> a|c
      BY -- OnLastHyp (Times '0<c')
      | 9. c*(r*a+s*b) = c*1 in int
      |->> a|c
        BY -- Seq ['c=(c*r)*a+s*(b*c)'; 'a|(c*r)*a'; 'a|s*(b*c)']
               THENL
               [Idtac;
                SFC [('a','int');('c*r','int')];
                SFC [('a','int');('s','int');('b*c','int')];
                SFC [('a','int');('(c*r)*a','int');('s*(b*c)','int')]]
```

Let us focus on the last step in the above example and examine: (1) the choice of the cut goals, and (2) the choice of the focus terms for each subgoal.

Again this step is possible by **OnticTac** in a single step with a large focus set (exercise for the reader). However, due to the high cost of large focus sets, we chose to divide the goal into smaller pieces and feed them to **OnticTac** and Autotactic according to their respective strengths.

Selecting cut formulas and focus terms requires detailed knowledge of how the proof will proceed. Take the first cut formulas, for example. $c = (c*r)*a+s*(b*c)$ is a rewrite of hypothesis 9. It was so chosen because **Arith** is capable of proving this by rewrite (if given this formula explicitly) and it matches the two divisibility subgoals better. Without this cut formula, the two divisibility subgoals would require larger focus sets in order to prove the dividends equal to respective subterms in hypothesis 9 by commutativity and associativity.

The choice of these cut goals requires

- knowledge of the workings of the Autotactic (**Arith** and **Mono**)
- knowledge of **OnticTac**
- knowledge of how the proof will proceed

Although requiring some knowledge about the capabilities of these tactics is reasonable, one would like to minimize this requirement as much as possible. And detailed knowledge about how the proof will proceed really subverts the idea of having the machine filling in tedium. At present, we know of no general purpose interactive theorem proving systems that do not require the user to have fairly detailed knowledge of the hidden subproofs that these systems supposedly 'fill-in'. Rewriting steps may be an exception, but simplifying rewrite only plays a limited role in these systems.

These examples raise serious questions about the relationship between **OnticTac** and **Arith/Mono**. One way to pose these questions is as a choice among:

1. Putting enough lemmas into the library so that, with very large focus sets, **OnticTac** can supersede **Arith/Mono**.
2. Using a library similar to the above but dividing proof burdens among different incarnations (*i.e.*, different focus sets) of **OnticTac** to reduce the focus set size
3. Dividing proof burdens among **OnticTac**, **Arith** and **Mono**, as we have done in this experiment
4. Integrating **Arith**, **Mono** and other useful decision procedures directly into **Ontic**.

(1), besides having high space cost, also has limitations: **Arith** applies a few facts deeply while **Ontic** applies many facts shallowly. It would be a test of the hypothesis that results of deep applications of some important facts can be packaged into lemmas and applied shallowly. This approach requires extending the object language with meta-constructs. Moreover, it is unlikely that either style of reasoning can subsume the other.

(2) is different from focus sequencing because the different incarnations of **OnticTac** can exchange facts among themselves, thus there is no dependence on the ordering as in focus sequencing. This may not require focus sets as large as for (1), but is not expected to overcome the limitations of (1).

(3) requires intelligent decisions, either by the user or some tactics. These are difficult options: while machine intelligence is difficult to achieve, applying human intelligence in making nitty-gritty decisions often reduces proof reusability.

At the least, (4) requires combining rewrite with congruence closure, an interesting research direction in its own right [8, 32, 25]. Whether integrating rewrite and 'obvious' induction [24] into **Ontic** will add enough deep applications of facts to provide a natural notion (approximating human judgement) of obviousness remains to be seen. In general, it also raises the question of how to integrate decision procedures. Cooperation among disjoint and convex decision procedures is studied in [30, 28] .

Focus Set Selection Large focus sets are impractical for currently known algorithms implementing the tractable inference relations defined by **Ontic**. Lacking the subformulas property implies that heuristics for automatically choosing focus sets are likely to be inaccurate. These two facts make automatic selection of focus terms difficult: poor performance of large focus sets rules out over-estimates, lack of subformulas property makes over-estimates highly likely. On the hopeful side, automatic focus set selection opens up the possibility of simulating large focus sets with a sequences of small focus sets.

Let us observe that each lemma defines a 'redex' for 'reduction' and focus sets that do not contain redices are useless. For example, applying the commutativity lemma requires two focus terms that are operands of a binary operator. Such redices usually generate small focus sets. By preprocessing the lemma library, we can obtain the redices and use them to select small focus sets. Because of the order dependence of focus set sequencing, it is necessary to either choose a particular order to sequence or to try all orderings. Not all pairs of focus sets are order sensitive, many are independent of each other. It seems possible, at the time of pre-processing of the lemma library, to identify those 'redices' that have order dependence.

A crude heuristic for selecting small focus set can be found by noting that most 'redices' are 'close relatives' in the term structure tree. [5] A crude heuristic for ordering is to sequence the focus sets in a bottom-up order. The combination of these simple heuristics works relatively well for many cases, but is not always the right order to sequence focus sets. These simple heuristics have been tested to cut down large focus sets selected by the user and the experimental result is somewhat encouraging: focus sets with up to 10 terms have been specified and the sequencing of them takes several minutes of real time. [6] In contrast, a single small focus step typically takes on the order of a few seconds to half a minute, and a focus set of 10 terms would have taken on the order of an hour.

In particular, examples 7 and 8, among others of similar complexity, are proved in a single step with focus sequencing on large focus sets chosen by the user. However, focus sequencing requires additional terms be mentioned in order to save enough facts between each subset focusing. These additional terms correspond to the subterms in the cut formulas of the proof shown earlier, plus others needed to permit focusing to simulate rewriting. Moreover, choosing the focus terms and the additional mention terms require knowledge of the proof, to about the same degree of detail as before.

In general, the choice of focus terms tends to involve the terms/subterms of the hypotheses and conclusion of the current sequent. Similar observations were also made in [13, 12].

For a more limited forward chaining tactic, Elkan was able to use the simple subterm heuristic mentioned above [12]. There seems to be a three-way trade-off among the power of the forward chaining tactic, its computational complexity and the ease of applying it in an automated fashion.

[5] If the logical connective '&' appears in a lemma, then members of a redex are not necessarily close neighbors and are not easily recognized. So this approximation picks out those cases that are efficiently recognizable.

[6] Real time is a better measure because it includes the impact of paging, a significant factor in this experiment. Time is measured on a Sparcstation 1.

Recognition versus Indexing Whether a tactic is applicable in a given situation is the *recognition problem.* What tactics are applicable and what parameters to apply them with (in the case of parameterized tactics, such as focusing) is the *indexing problem.* Search can reduce indexing to that of recognition, but the search space can be quite large, sometimes infinite, when tactics take parameters.

A minimum requirement to a solution of the recognition problem is some kind of declarative representation of tactics. In [5, 4, 6], a refinement of the tactic-based approach is taken, where tactics are given partial specifications in the form of pre-conditions and post-effects. In contrast to McAllester's notion of 'obvious by known facts', Bundy's approach can be characterized as defining a notion of 'obvious by known methods'. The relationship between these two notions is extremely intricate and does not seem to be explored in the literature.

A further requirement of a declarative representation may be a small meta-language to facilitate reasoning – the full expressive power of ML is not required to express simple idioms and also makes it difficult to reason about tactics. Meta-reasoning about tactics is one of the motives behind on-going work on reflection in the **Nuprl** group [1].

The recognition problem itself requires theorem proving. A sensible solution might be to use another notion of 'obviousness', to perform limited inference in recognizing pre-conditions of tactics. Moreover, the internal data-structure used by an **Ontic**-like tactic might also provide a solution to the indexing problem.

Tactics that are like decision procedures tend to have difficult recognition problems. In this sense, **Arith** and **OnticTac** are both too powerful. The focus set parameter of **OnticTac** also makes the indexing problem difficult. In contrast, Bundy advocates synthesizing decision procedures out of simple 'proof plans' in [7]. Perhaps some of our focus selection and sequencing heuristics can be formulated as simple 'proof plans' with pre-conditions.

Focus sequencing heuristics blur the distinction between mechanisms to achieve a basic notion of obviousness and proof plans based on that. It is not clear whether proof plans should be used to repair defects in a minimum obviousness decision procedure or should rely on one. Perhaps the distinction is not real.

However, it is not clear whether combining proof plans is a good way to achieve a basic notion of obviousness or having a basic notion of obviousness facilitates proof plan combinations.

There seems to be a parallel between the relationship of facts and methods on the one hand, and forward chaining and backward chaining as evaluation/inference mechanisms on the other. Combinations of forward and backward chaining at the same syntactic level have been studied in the context of deductive databases [34, 27] and of theorem proving [33]. However, lemmas and plans do not seem to belong to the same class of objects.

Monotonicity Maintaining monotonicity under tactical control is non-trivial. By their very nature, tactics make decisions in the course of searching for a proof. The general problem is: given a collection of declarative facts and a collection of control heuristics, how to add new facts and new heuristics monotonically, *i.e.* without disabling old solutions. And more ambitiously, how to add new facts and heuristics without increasing the complexity of old solutions.

There seems to be two notions of monotonicity: *autotactic monotonicity* is the property of autotactics with respects to additions of lemmas and tactics; *interactive monotonicity* is the property of a proof script (like we have seen in this paper) that contains decisions made by the user. Interactive monotonicity is much harder to achieve, because it requires both autotactic monotonicity and a disciplined approach to constructing proof scripts. Moreover, these two are not independent factors to control.

Relation to logic programming based approaches Logic programming and unification based techniques, such as used in **Isabelle**[31], has been suggested as another way to reduce tedium. Logic programming is typically a top-down search procedure. Top-down reasoning with caching seems to perform the same abstract computation as bottom-up reasoning. See for example [3, 33]. A more detailed discussion of the relationship between bottom-up database evaluation and forward chaining theorem proving is beyond the scope of this paper.

Performance As we have noted above, typical response time is under half a minute for single focus step and a few minutes for sequencing moderately sized focus sets. The current implementation leaves *much* room for optimization.

As can be expected, expanding a proof using **OnticTac** often takes longer than expanding one constructed with **Nuprl**'s tactics. Since practical theorem proving requires good response time, efficient implementation of knowledge-intensive forward chaining is a challenge for the future.

In our experiments we have noticed a performance degradation, when going from a Sun-4 with 40Mb to one with 16Mb. This indicate a need to look for algorithms for forward chaining with better memory reference locality.

5 Conclusions and Future Work

We have seen that applying a knowledge-based tactic can significantly shorten interactive proofs and increase robustness in the sense of replayability.

Restricting the size of focus sets keeps the response time acceptably low, but forces us to resort to either: (1) cooperation with other tactics, such as **Arith** or (2) sequencing small subsets of a larger focus set. Currently, both approaches require a fair amount of control information.

We have applied some simple heuristics for selecting and ordering focus sets and had some success. We are currently exploring heuristics that will identify 'relevant' terms to mention during sequencing. This is a less daunting task because the complexity of **Ontic** only depends on the mentioned terms polynomially, hence over-estimating mention terms is not as big a problem as over-estimating focus sets. More sophisticated heuristics for selecting and ordering focus sets are currently being explored. We would also like to settle the question of whether good heuristics for sequencing focus sets will replace cooperation with **Arith** and **Mono**.

Improving **OnticTac** by simply adding more lemmas is limited because of the shallow applications of them by **OnticTac**. However, Adding more lemmas can complement the focus sequencing heuristics. Our current library was not designed with focus sequencing in mind. We expect that focus sequencing heuristics will compensate for defects in the lemma library and additional lemmas in the library will also make it easier to find simple sequencing heuristics that are effective. It would be desirable to find coordinated policies of adding useful lemmas to the library and of choosing focus sets.

Focus sequencing and refinements via lemma library pre-processing seems to be a promising direction to extend our work.

Improving **OnticTac** by incorporating rewriting and/or induction are important research problems just beginning to be explored. We expect progress in these directions to complement our approach.

In general, it is difficult to combine several weak tactics to obtain stronger tactics because recognizing conditions of applicability of arbitrary tactics can be as hard as theorem proving.

Cooperation by distributing subgoals has the same recognition problem. Proofs in which this distribution is made by the human user, contain much control information.

Shorter proofs sometimes contain more control knowledge from the user – *e.g.*, the selection of which tactic to apply with what parameters, or more specifically, the selection of which terms to focus on and in what order. The significant role of this control knowledge can been appreciated by noting that almost all existing theorem provers only exhibit their power in the hands of experts (poor user-interface play a significant, but smaller, part, in my opinion).

An important open question remaining is the relationship between decision procedures and **Ontic**-like procedure. This work started as a study of how the tactical approach and the **Ontic**-like approach might complement each other and ends with a vision containing three components with different characteristics: decision procedures, **Ontic**-like extensible knowledge-based procedures and tactics that express high-level idioms in proof construction. A practical system is likely to require combination of all three.

Acknowledgement

The author is grateful to Robert Constable, Charles Elkan, Douglas Howe, David McAllester and one anonymous referee for helpful comments on the paper and to Bill Aitken, James Allan and Keith Gasser for reading earlier drafts.

References

1. Stuart F. Allen, Robert L. Constable, Douglas J. Howe, and William B. Aitken. The semantics of reflected proof. In *Proceedings of the Fifth Annual Symposium on Logic and Computer Science*, pages 95–107. IEEE Computer Society, June 1990.
2. D. Basin. An environment for automated reasoning about partial functions. In R. Lusk and R. Overbeek, editors, *Ninth International Conference on Automated Deduction*,

volume 310 of *Lecture Notes in Computer Science*, pages 101–110. Springer Verlag, May 1988. Also as Cornell CS TR 87-884.

3. F. Bry. Query evaluation in recursive databases: bottom-up and top-down reconciled. *Data & Knowledge Engineering*, 5(4):289–312, 1990.
4. A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, page (in press) Earlier version available from Edinburgh as Research Paper No 413, 1988.
5. Alan Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *Ninth International Conference on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 111–120. Springer Verlag, 1988. Longer version available from Edinburgh as Research Paper No. 349.
6. Alan Bundy. A science of reasoning. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, Cambridge, MA, 1991.
7. Alan Bundy. The use of proof plans for normalization. Report, Dept. of Artificial Intelligence, University of Edinburgh, 1991.
8. Leslie Paul Chew. An improved algorithm for computing with equations. In *the 21st Annual Symposium of Foundations of Computer Science*, pages 108–117. IEEE Computer Society Press, 1980.
9. Robert L. Constable, Scott D. Johnson, and Carl D. Eichenlaub. *Introduction to the PL/CV2 Programming Logic*, volume 135 of *Lecture Notes in Computer Science*. Springer-Verlag, New York, 1982.
10. Robert L. Constable, et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.
11. Peter J. Downey, Ravi Sethi, and Robert E. Tarjan. Variations on the common subexpression problem. *JACM*, 27(4):758–771, October 1980.
12. Charles Elkan. Personal communication, 1991.
13. Charles Elkan and David McAllester. Automated inductive reasoning about logic programs. In Kenneth Bowen and Robert Kowalski, editors, *Fifth International Conference on Logic Programming*, volume 2, pages 876–892, Seattle, Washington, August 1988. MIT Press.
14. A. Felty and D. Miller. Specifying theorem provers in a higher-order logic programming language. In R. Lusk and R. Overbeek, editors, *Ninth International Conference on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 61–80. Springer Verlag, 1988.
15. M. Gordon. A proof generating system for higher-order logic. In *Proceedings of the Hardware Verification Workshop*, 1989.
16. Michael J. Gordon, Robin Milner, and Christopher P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.
17. D.J. Howe. *Automating Reasoning in an Implementation of Constructive Type Theory.* PhD thesis, Cornell University, Ithaca, NY, April 1988.
18. Douglas J. Howe. Implementing number theory, an experiment with Nuprl. In *Eighth International Conference on Automated Deduction*, volume 230 of *Lecture Notes in Computer Science*, pages 404–415. Springer Verlag, 1986.
19. Douglas J. Howe. Equality in lazy computation systems. In *Proceedings of the Fourth Annual IEEE Symposium on Logic in Computer Science*, pages 198–203. IEEE Computer Society, June 1989.
20. Dexter C. Kozen. Complexity of finitely presented algebras. In *Proceedings of the Ninth Annual ACM Symposium on the Theory of Compututation*, pages 164–177, 1977.

21. D. McAllester, R. Givan, and T. Fatima. Taxonomic syntax for first order inference. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 289–300, 1989. To Appear in JACM.
22. David McAllester. *Ontic: A Knowledge Representation System For Mathematics.* MIT Press, Cambridge, Massachusetts, 1989.
23. David McAllester. Automatic recognition of tractability in inference relations. Memo 1215, MIT Artificial Intelligence Laboratory, February 1990. To appear in JACM.
24. David McAllester. Some observations on cognitive judgements. In *AAAI-91*, pages 910–915. Morgan Kaufmann Publishers, July 1991.
25. David A. McAllester. Grammar rewriting. In *Eleventh International Conference on Automated Deduction*. Springer Verlag, 1992.
26. C. Murthy. An evaluation semantics for classical proofs. In *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 96–107, Amsterdam, The Netherlands, July 1991.
27. Jeffery F. Naughton and Taghu Ramakrishnan. Bottom-up evaluation of logic programs. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 640–700. MIT Press, Cambridge, MA, 1991.
28. Greg Nelson. Combining satisfiability procedures by equality-sharing. In *Automated Theorem Proving: After 25 Years*, pages 201–211. American Mathematical Society, 1983.
29. Greg Nelson and Derek Oppen. Fast decision procedures based on congruence closure. *JACM*, 27(2):356–364, April 1980.
30. Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.
31. L. Paulson. Isabelle: The next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–385. Academic Press, 1990.
32. David J. Sherman. Lazy directed congruence closure. Tech report 90-028, University of Chicago, September 1990.
33. Mark E. Stickel. Upside-down meta-interpretation of the model elimination theorem proving procedure for deduction and abduction. Technical Report TR-664, Institute for New Generation Computer Technology, Tokyo, Japan, May 1991.
34. Jeffery D. Ullman. *Principles of Database and Knowledge-base Systems.* Computer Science Press, 1989. Chapter 13.