Syntactic Equality in Knowledge Representation and Reasoning

Edward P. Stabler*
Canadian Institute for Advanced Research, and
Department of Computer Science
University of Western Ontario

Abstract

Reasoning with equality can be computationally difficult, but is less so when the equality relation for a language is the familiar relation of "syntactic equality:" no two distinct ground terms denote the same thing in the intended model. However, first order theories often contain existentially quantified variables which block the use of the syntactic equality methods. The existence of a decidable set of terms known to have pairwise distinct denotations can nevertheless be a computational advantage. We present an extension of resolution, "SEq resolution," that builds in the syntactic equality axioms for just those terms known to be pairwise distinct. In some cases, there is a computational advantage to respecting the syntactic equality restriction in so far as possible, leaving only Skolem functions and constants as the exceptions, and using SEq resolution.

1 Introduction

In a recent book by Wos et al. [1984], the art of automated reasoning is codified in a number of useful proverbs. One of them is the following:

Employ equality predicates.

This recommendation is surprising to logic programmers who assiduously avoid equality and other congruence (or even equivalence) relations. Consider the following remark from the logic programming literature:

> *One of the reasons why logic programming succeeded where other resolution theorem proving had failed (namely in finding a practical application) was that in logic programming equality was avoided like the plague.*[van Emden and Yukawa , 1980]

Some knowledge is very hard to represent without any equality, though. Consider expressing the simple fact that the only things that have property p are a and b. With equality, this is easily expressed with a formula like $p(X) \leftrightarrow X = a \vee X = b$. Logic programmers have noticed that a very restricted, computationally manageable equality relation suffices to express such things: the familiar "syntactic equality." This relation is so restricted that complete proof methods using it do not need to consider transitive chains of equations or substitutions of distinct terms into any predicate or function expressions. In this paper we generalize this idea to full first order resolution. It is a simple matter to build the syntactic equality theory for any first order language into an inference rule. In fact, we propose a more general idea: access to an efficiently decidable set of terms whose denotations are known to be pairwise distinct can be a real computational advantage. This paper shows how extensions of resolution and similar strategies can exploit such a set of terms for computational advantage even when the set does not include all the terms in the theory. We propose a more specific version of the logic programmers' maxim:

> *Avoid non-syntactic equality. That is, try to minimize the number of terms that are not in the set of terms whose denotations are known to be pairwise distinct.*

An analysis of some examples that support this recommendation about knowledge representation confirm what becomes clear with a little reflection on the matter. The apparent tension between our proverb and

* Slightly revised version of [Stabler 1989]. I am grateful to the Canadian Institute for Advanced Research and the Natural Sciences and Engineering Research Council of Canada for supporting this research. I would like to thank William Demopoulos, Hector Levesque, Alan Mackworth and Ray Reiter for valuable discussions of this material.

Wos's is explained by the observation that while non-syntactic equality can simplify theories and proofs, it can make the search for a proof very difficult. When the search is guided, simplicity is preferable, though it is of course still nice to have a small search space – neither humans nor machines are very good at considering too many alternatives. When search is automatic, a small search space is essential for practical performance. Syntactic equality provides the expressive power needed to make a theory and deductions from it simple, and yet minimizes the search space.

2 A First Example

Consider the following simple theory:

$$nn(0) \qquad\qquad \textit{Theory } \mathit{HNN}$$
$$nn(X) \rightarrow nn(s(X))$$

where, here and throughout, we use such formulas to represent their universal closures, and variable names begin with uppercase letters. Under the obvious interpretation, **N**, this theory says (truly) that 0 and its successors are natural numbers. Suppose that the language contains just one other function symbol, the constant a, and in **N** neither a nor any of its successors names a natural number. Then our theory has the nice properties that $\mathit{HNN} \not\models nn(a)$, and in fact, $\mathit{HNN} \vdash nn(\tau)$ just in case τ names a natural number in **N**. Obviously, though, $\mathit{HNN} \not\models \neg nn(a)$. To get the latter entailment we need, in the first place, a necessary condition for natural numberhood like the following:

$$\textit{Theory } \mathit{NN}$$
$$nn(X) \leftrightarrow \ \big(X = 0 \vee \exists Y(X = s(Y) \wedge nn(Y))\big)$$

And then we need a theory of equality that tells us that our terms are pairwise distinct, and in particular that the term a does not have the same denotation as any other ground term:

$$X = X \qquad\qquad \textit{Theory } \mathit{SEq}(\mathit{NN})$$
$$a \neq 0$$
$$a \neq s(X)$$
$$0 \neq s(X)$$
$$X \neq \tau \qquad \textit{for all terms } \tau \textit{ properly containing } X \qquad (\dagger)$$
$$s(X) = s(Y) \leftrightarrow \ X = Y$$
$$(nn(Y) \wedge X = Y) \rightarrow nn(X)$$
$$(Y_1 = Y_2 \wedge X_1 = Y_1 \wedge X_2 = Y_2) \rightarrow X_1 = X_2$$

Clearly, we now have $\mathit{NN} \cup \mathit{SEq}(\mathit{NN}) \models \neg nn(a)$, and $\mathit{NN} \cup \mathit{SEq}(\mathit{NN}) \models \neg nn(s(a))$.

Notice that this theory contains a schema, (†), with infinitely many instances. In fact this equality theory has a form that is familiar in the logic programming literature: $\mathit{NN} \cup \mathit{SEq}(\mathit{NN})$ is essentially Clark's *completion* of the definite clause theory HNN [Clark, 1978] [Lloyd, 1984]. Clark uses a completion like this in his characterization of the results that can be obtained from HNN when Horn clause resolution is extended with the non-monotonic negation-as-failure rule. We do not need to tangle with negation-as-failure, though.[1] Classical logic gives us entailments like those mentioned, and a sound and complete method like resolution allows us to demonstrate such results.

For present purposes, the interest of equality theories like $\mathit{SEq}(NN)$ is that they serve to force a "free" interpretation of the terms in the language. In other words, the terms of the theory satisfy a "unique names condition," distinct terms (distinct names *and* distinct ground function expressions generally) have distinct denotations. However, the well known proof techniques do not allow us to obtain the computational advantage of syntactic equality in deductions from $NN \cup \mathit{SEq}(NN)$ because the theory contains an existential quantifier. When converted to clausal form, a Skolem function will be introduced for which the syntactic equality restriction cannot be presumed to hold. We will show how to take advantage of the fact that all the other terms respect the unique names assumption. We will briefly describe a very simple sound and complete specialized inference rule for syntactic equality, and then modify it to handle theories like the clausal form of $NN \cup \mathit{SEq}(NN)$ in which a subset of the terms fail to satisfy the syntactic equality restriction.

3 Building in syntactic equality

Because equality vastly increases the search space of a theorem prover, there has been a good deal of work on building the standard equality axioms Eq into special inference rules R in such a way that $T \vdash_R S$ iff $T \cup Eq \models S$. If $T \cup Eq$ is satisfiable, we say T is *E-satisfiable*. A rule R meeting the above condition is said to be a sound and complete method for E-unsatisfiability. Paramodulation is probably the best known method [Chang and Lee, 1973] [Wos *et al.*, 1980]: equalities license the substitution (or "paramodulations") of distinct terms in any literal at any point in the deduction. We can use a similar technique to build in the equality axioms that enforce the

[1] For a consideration of some of the troubles with negation-as-failure in logic programming, see, for example, [Flannagan, 1986] [Sheperdson, 1984].

unique names condition. We have seen an example of these axioms in the case of $SEq(\mathcal{NN})$, above. In general, let the *axiomatization of the unique names assumption* $SEq(T)$ *for the language of* T be defined to comprise the standard equality axioms Eq for the language, pairwise inequalities for all the constants and function symbols, all instances of †, and for each function symbol an axiom saying that it is a 1-1 function. Then we say that T is *SEq-satisfiable* iff $T \cup SEq(T)$ is satisfiable. Obviously, then, $\mathcal{NN} \cup nn(a)$ is SEq-unsatisfiable.[2]

Building in the (often infinitely many) axioms of SEq, we can formulate a simple, clausal refutation method for SEq-unsatisfiability. The sufficient condition for identity in our restricted models is easily expressed. It can be represented simply by the clause: $\{X = X\}$. A resolution step in which this clause plays a role can be easily replaced by a rule that has the same effect – our rule (i) below. The necessary condition on syntactic equality is the tricky part. We want to say that two things are nonidentical just in case they are named by distinct terms. The following simple extensions to resolution suffice:

For any clause C,

(i) If $t \neq t' \in C$ and σ is the mgu of t and t', deduce $(C - \{t \neq t'\})\sigma$;

(ii) If $t = t' \in C$ and for substitution σ, $t\sigma$ and $t'\sigma$ are not unifiable, deduce $(C - \{t = t'\})\sigma$.

Theorem 3.1. There is a derivation of $\square$ from Γ using resolution together with (i) and (ii) iff Γ is not SEq-satisfiable.

Proof: We must show $\Gamma \vdash \square$ iff Γ is SEq-unsatisfiable. The result for derivations that do not unify literals containing the equality predicate was established by Robinson [Robinson, 1965], so we consider only our special treatment of equality.

When a clause C contains a literal $\neg t = t'$ we allow the deduction of $C - \{\neg t = t'\}\sigma$ just in case σ is the most general unifier of t and t'. This is sound since $X = X$ is true, and it is complete because two terms denote the same thing in a model that satisfies our semantic restriction only when the terms are identical.

When a clause C contains a literal $t = t'$ we allow the deduction of $C - \{t = t'\}\sigma$ just in case $t\sigma$ and $t'\sigma$ are not unifiable. This is sound because terms that are not unifiable cannot denote the same thing, under any assignment to variables, in models that satisfy our restriction. And it is complete, because two terms can fail to denote the same thing only if the terms are distinct. ∎

Like the axioms of SEq, inference rule (ii) still poses an efficiency problem. The space of possible substitutions that can make a pair of terms distinct is too large to be managed efficiently. We will make some headway on this problem below, but we can note here the logic programmers' strategy. It is just to delay the evaluation of any positive equation (or any other non-ground positive literal) in a clause as long as possible – try to resolve against all the other literals first, in hopes of instantiating the terms in the equations – and then form a resolvent $C - \{t = t'\}$ just in case t and t' are not unifiable.[3] That is, we can modify rule (ii) of our extension to resolution as follows:

(ii') If $(t = t') \in C$ and t and t' are not unifiable, deduce $C - \{t = t'\}$.

This strategy is more tractable, but it is obviously not complete.

4 Skolem functions

With clausal form theorem provers, all problems must be represented in clausal forms: existentially quantified variables must be replaced by Skolem functions. When these are introduced, they must be new to the theory, and so we do not know whether their denotations are distinct from those of other terms. For example, the following is a clausal form of $\mathcal{NN}$ in which a unary Skolem function $sk1$ has been introduced:

$\{\neg nn(A), A{=}0, A{=}s(sk1(A))\}$ *Theory* $Cl(\mathcal{NN})$
$\{A \neq 0, nn(A)\}$
$\{\neg nn(A), nn(sk1(A)), A{=}0\}$
$\{\neg nn(A), B \neq s(A), nn(B)\}$

The first clause tells us that if $nn(A)$ for some A, then either A is 0 or A is the successor of something in the domain, viz. $sk1(A)$. Obviously, on the intended interpretation of the theory, this function must have numbers as its values, and so we have lost the unique names property. But since the Skolem function is the *only* departure from the unique names restriction, since the theory still contains a set of terms that satisfies the restriction, we can still benefit from a specialized rule of inference that provides standard equality reasoning

[2] We modify the definition of SEq-satisfiability below to handle the case where not all terms satisfy the unique names condition.

[3] Compare, for example, the strategy for soundly implementing negation-as-failure in [Lloyd, 1984], and the more general application of similar strategies on otherwise difficult problems in [Naish, 1985].

when necessary, but which takes advantage of the fact that many of the terms must have distinct denotations.

4.1 RUE-NRF resolution

As noted above, there are a number of techniques, such as paramodulation, for building in the standard equality theories for the language of any theory. We adapt a technique defined by Digricoli [Digricoli, 1983] [Digricoli and Harrison, 1986] called *RUE-NRF resolution* that is like paramodulation in building in standard equality, but it builds in the equality theory by extending unification in a way more similar to the e-unification strategies of Bledsoe [Bledsoe and Hines, 1980] and Morris [Morris, 1969]. ("RUE-NRF " stands for "resolution by unification and equality" using the "negative reflexive function".)

In RUE-NRF resolution, if terms t, t' are not unified, they give rise to inequalities in the resolvent. For example, the RUE resolvent of $\{p(a, f(b, c))\} \cup A$ and $\{\neg p(X, f(X, d))\} \cup B$ is $A\sigma \cup B\sigma \cup D$ where σ is a substitution and D is a *disagreement set* for $p(a, f(b, c))\sigma$ and $\neg p(X, f(X, d))\sigma$. For example, when $\sigma = \{X/a\}$, D can be either of the following two disagreement sets: $\{f(b, c) \neq f(a, d)\}$ or $\{b \neq a,\ c \neq d\}$. The former is the "topmost" disagreement set. When resolving on equations, symmetry is captured by obtaining two RUE resolvents. For example, the RUE resolvents of $\{a_1 = a_2\} \cup A$ and $\{b_1 \neq b_2\} \cup B$ are $\{a_1 \neq b_1,\ a_2 \neq b_2\} \cup A \cup B$ and $\{a_2 \neq b_1,\ a_1 \neq b_2\} \cup A \cup B$. A NRF resolvent is essentially the result of RUE resolution with $\{X = X\}$, which removes an inequality $t \neq t'$ from a clause and introduces a disagreement set for t, t'. The NRF resolvent of $\{f(b, c) \neq f(a, d)\} \cup A$ is $A\sigma \cup D$ where σ is empty and $D = \{b \neq a,\ c \neq d\}$. The formal definition of this intuitive approach is not difficult. Definitions and proofs of the soundness and completeness of RUE-NRF resolution (with factoring) for showing E-unsatisfiability are presented in [Digricoli, 1983] [Digricoli and Harrison, 1986]. (A nice informal account appears in [Stickel , 1986].)

4.2 Making RUE-NRF resolution more efficient

Among the various ways of building the standard equality axioms into an inference rule, the advantage of RUE-NRF resolution for present purposes is that it clearly isolates the problems of restricting the substitutions σ and the disagreement sets D used to define the resolvents without losing completeness. The analogous difficulties with finding restrictive and complete paramodulation strategies show that our options are limited. We consider restrictions on the disagreement sets and restrictions on the substitutions, in turn.

The restrictions on disagreement sets are presented in terms of the following notion:

Definition: A disagreement set D is *viable* in S only if one of the following conditions is satisfied:

(a) D is empty, or

(b) for each term s_i in each equation in D, there is a term t_i in a positive equation in some clause in S such that either s_i unifies with t_i or they have the same principal functor and viable disagreement sets, or

(c) there is a substitution that makes the terms of each inequality in D identical.

Digricoli then considers restrictions on the difference sets that are motivated by the following lemma. We let Eq be the standard equality theory for a theory S.

Definition: An *e-chain* from t to t' in clausal form theory S is a sequence of equations each of which occurs in some clause in S, which is instantiated by some substitution σ,

$$(r_0 = r_1)\sigma,\ (r'_1 = r_2)\sigma,\ (r'_2 = r_3)\sigma,\ \ldots,\ (r_{k-1} = r_k)\sigma$$

where $k > 0$, $r_0\sigma$ is t, $r_k\sigma$ is t' and for all $0 < i < k$ either $r_i\sigma$ is identical to $r'_i\sigma$ or $r_i\sigma$ and $r'_i\sigma$ have the same principal functor and their respective arguments can be proven equal from $S \cup Eq$.

Lemma 4.1. If t and t' are not identical and $t = t'$ can be proven from $S \cup Eq$, then either

(1) $t = t'$ has the form $f(a_1, \ldots, a_k) = f(b_1, \ldots, b_k)$ where for all $0 < i \leq k$, $a_i = b_i$ can be proven from $S \cup Eq$, or

(2) there is an e-chain from t to t' in S.

Digricoli proves this lemma and uses it to justify two important pruning strategies:

(Viability) Use only viable disagreement sets, and when there is a choice among disagreement sets, always choose the topmost.

(Equality Restriction) RUE resolution of $A \cup \{s_1 = s_2\}$ and $B \cup \{t_1 \neq t_2\}$ is permitted only if at least one pair in $\{\langle s_1, t_1\rangle, \langle s_1, t_2\rangle, \langle s_2, t_1\rangle, \langle s_2, t_2\rangle\}$ is such that its members either unify or have both the same principal functor and a viable disagreement set.

It turns out that the substitutions σ that are used in RUE-NRF resolution can be restricted to most general partial unifiers (MGPUs) with two important exceptions. The MGPU is the substitution obtained from the standard (left-to-right) most general unifier (MGU) algorithm modified so that instead of failing at the first nonunifiable expression, it continues to unify as many constituents as possible. For example, $\{X/a\}$ is the MGPU of $p(X,b)$ and $p(a,c)$. However, this strategy must be qualified as follows:

(RUE subst) In computing the MGPU of $\phi, \neg\psi$ in RUE , always unify an occurrence of a variable X and a corresponding occurrence of term t except in the following case: this occurrence of the variable is the argument of a function $f(\ldots, X, \ldots)$ and there is a viable disagreement set of $\phi, \neg\psi$ containing $f(\ldots, X, \ldots) \neq f(\ldots, t, \ldots)$.

(NRF subst) When applying NRF to a literal $X \neq t$, always unify X and t to erase this literal.

When applying NRF to a literal $f(a_1, \ldots, a_k) \neq f(b_1, \ldots, b_k)$, always unify an occurrence of a variable X and a corresponding occurrence of term t except in the following case: this occurrence of the variable is the argument of an inner function $g(\ldots, X, \ldots)$ which occurs as or in some a_i or b_i $0 < i \leq k$, and there is a viable disagreement set of $f(a_1, \ldots, a_k)$, $f(b_1, \ldots, b_k)$, containing $g(\ldots, X, \ldots) \neq g(\ldots, t, \ldots)$.

Let's call the result of restricting RUE-NRF resolution in the ways Digricoli suggests *restricted RUE-NRF resolution.* Digricoli defends the conjecture that restricted RUE-NRF resolution is a sound and complete method for E-unsatisfiability.

4.3 SEq resolution to exploit syntactic equality

Let $\overline{Sk}$ be a set of terms such that the ground instances of the terms in $\overline{Sk}$ have pairwise distinct denotations. Let Sk be the set of terms not in $\overline{Sk}$. To cover the case where Sk is non-empty, we modify our earlier definition of SEq-satisfiability as follows.

Definition: Let the *axiomatization of the unique names assumption $SEq(T)$ for the language of T relative to the set Sk* be defined to comprise the standard equality axioms Eq for the language, pairwise inequalities for all the constants and function symbols in $\overline{Sk}$, all instances of † in which τ is in $\overline{Sk}$, and for each function symbol in $\overline{Sk}$ an axiom saying that it is a 1-1 function.

Now, we say that T is *SEq-satisfiable (relative to Sk)* iff $T \cup SEq(T)$ is satisfiable.

We can immediately strengthen the previous lemma in a potentially useful way:

Lemma 4.2. If t and t' are not identical and $t{=}t'$ can be proven from $S \cup Eq$, then either t, t' are unifiable or at least one of $t, t' \in Sk$.

This follows directly from our definition of Sk. It also follows that there cannot be two elements t, t' in any e-chain that are neither unifiable nor elements of Sk. We can accordingly strengthen condition (b) in the relevant notion of viability, which we will now call "strong viability":

Definition: A disagreement set D is *strongly viable* in S only if one of the following conditions is satisfied:

(a) D is empty, or

(b) no equation in D contains non-unifiable terms neither of which is an element of Sk, and for each term s_i in each equation in D, there is a term t_i in a positive equation in some clause in S such that either s_i unifies with t_i or they have the same principal functor and strongly viable disagreement sets, or

(c) there is a substitution that makes the terms of each inequality in D identical.

We can thus strengthen Digricoli's pruning strategies as follows:

(Strong Viability) Use only strongly viable disagreement sets, and when there is a choice among disagreement sets, always choose the topmost.

(Strong Equality Restriction) RUE resolution of $A \cup \{s_1 = s_2\}$ and $B \cup \{t_1 \neq t_2\}$ is permitted only if at least one pair in $\{\langle s_1, t_1\rangle, \langle s_1, t_2\rangle, \langle s_2, t_1\rangle, \langle s_2, t_2\rangle\}$ is such that its members (i) unify, or (ii) at least one of them is in Sk and they have the same principal functor and a strongly viable disagreement set.

Digricoli conjectures that restricted RUE-NRF resolution is a sound and complete method for E-unsatisfiability.

Theorem 4.2. If restricted RUE-NRF resolution is a sound and complete method for E-unsatisfiability, then so is RUE-NRF resolution with our stronger restrictions.

This result, conditioned on Digricoli's conjecture as it is, is trivial. We have strengthened only the two

pruning restrictions, and in each case this involves only minor changes that are straightforwardly justified by the definition of Sk, the definition of "strong viability," and our Lemma 4.2.

We have still built in only the standard axioms of equality and not the unique names axioms SEq($\overline{Sk}$) for those terms not in Sk. We cannot use (ii) when Sk is nonempty. There are equations that allow a similar treatment, though. Consider the following relation between terms:

Definition: Terms t and t' are *distinguished* iff either

(a) $t, t' \in \overline{Sk}$ are distinct constants, or

(b) $t, t' \in \overline{Sk}$ have either different functors or distinguished arguments.

We can then use the following analog of (ii) to obtain the effect of all of the pairwise inequalities for terms in $\overline{Sk}$:

(iii) If $(t = t') \in C$, deduce $(C - \{t = t'\})\sigma$ if $t\sigma$ and $t'\sigma$ are distinguished, for any substitution σ

This has exactly the desired effect: an equation of two terms that are not in Sk cannot be true, and hence can be eliminated. We can also get a more tractable incomplete form of this rule, in analogy with (ii'):

(iii') If $(t = t') \in C$, deduce $C - \{t = t'\}$ if t and t' are distinguished.

To cover the case where Sk is nonempty, we must also add a rule which provides the effect of the axioms that say that functions $f \notin Sk$ are 1-1. This is needed because the arguments of these functions can be elements of Sk about which we may need to reason using the standard equality rules of RUE-NRF resolution. For example, with pure syntactic equality $s(sk1) = s(sk2)$ would be refuted using (ii) or (ii'), but if s is not a Skolem function and $sk1$ and $sk2$ are Skolem constants in Sk, we cannot refute $s(sk1) = s(sk2)$ using (iii) or (iii'). In order to be able to deduce $sk1 = sk2$, the following rule suffices – it just has the same effect as the inclusion of the relevant axioms would:

(iv) For any clause C, deduce resolvents of C with any instance of $\{f(X_1, \ldots, X_n) \neq f(Y_1, \ldots, Y_n), X_1 = Y_1, \ldots, X_n = Y_n\}$ for any n-ary function f in $\overline{Sk}$.

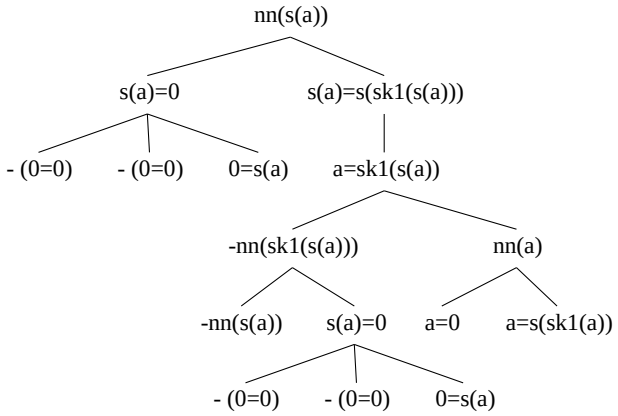

Figure 1: Resolution refutation of $nn(s(a))$

We call the strengthened version of RUE-NRF resolution together with (iii) and (iv) *SEq resolution*. SEq resolution gives us the enormous advantage of syntactic equality whenever the set Sk is empty; it gives us RUE-NRF resolution whenever Sk contains all terms; and it gives us appropriate intermediate strategies for the intermediate cases. We have implemented a version of SEq resolution using the incomplete rule (iii') in a model elimination system [Loveland, 1978] [Stickel , 1986].

5 Comparing resolution and SEq resolution

Consider the problem suggested in §2. This problem is small enough that we can consider it in detail and display resolution and SEq refutations in a tree format, making some aspects of the comparison graphically obvious. In our refutation trees, each literal immediately dominates the literals in the resolvent of the resolution step it is involved in, and all of the substitutions of the proof have been made throughout the tree. The theory $NN \cup SEq(NN) \cup \{nn(s(a))\}$ is infinite, but we can get a refutation using only the following subset of its clausal form:

$$\{\neg nn(A), A=0, A=s(sk1(A))\} \quad (c12)$$
$$\{\neg A=0, nn(A)\} \quad (c13)$$
$$\{\neg nn(A), nn(sk1(A)), A=0\} \quad (c14)$$
$$\{\neg nn(A), \neg B=s(A), nn(B)\} \quad (c15)$$
$$\{A=A\} \quad (c4)$$
$$\{\neg a=0\} \quad (c5)$$
$$\{\neg a=s(A)\} \quad (c6)$$
$$\{\neg 0=s(A)\} \quad (c7)$$
$$\{A=B,\ \neg s(A)=s(B)\} \quad (c8)$$
$$\{s(A)=s(B),\ \neg A=B\} \quad (c9)$$
$$\{\neg nn(A),\ \neg A=B,\ nn(B)\} \quad (c10)$$
$$\{\neg A=B,\ \neg C=A,\ \neg D=B,\ C=D\} \quad (c11)$$

Here is a refutation of this theory:

$$\{nn(s(a))\}$$
clause for refutation
$$\{s(a)=0,\ s(a)=s(sk1(s(a)))\}$$
using literal 1 of (c12)
$$\{-A=0,\ -C=A,\ C=s(a),\ s(a)=s(sk1(s(a)))\}$$
using literal 3 of (c11)
$$\{-C=0,\ C=s(a),\ s(a)=s(sk1(s(a)))\}$$
using literal 1 of (c4)
$$\{0=s(a),\ s(a)=s(sk1(s(a)))\}$$
using literal 1 of (c4)
$$\{s(a)=s(sk1(s(a)))\}$$
using literal 1 of (c7)
$$\{a=sk1(s(a))\}$$
using literal 2 of (c8)
$$\{-A=sk1(s(a)),\ -C=A,\ C=a\}$$
using literal 3 of (c11)
$$\{-C=sk1(s(a)),\ C=a\}$$
using literal 1 of (c4)
$$\{sk1(s(a))=a\}$$
using literal 1 of (c4)
$$\{\neg nn(sk1(s(a))),\ nn(a)\}$$
using literal 2 of (c10)
$$\{\neg nn(s(a)),\ s(a)=0,\ nn(a)\}$$
using literal 2 of (c14)
$$\{s(a)=0,\ nn(a)\}$$
using clause for refutation
$$\{-A=0,\ -C=A,\ C=s(a),\ nn(a)\}$$
using literal 3 of (c11)
$$\{-C=0,\ C=s(a),\ nn(a)\}$$
using literal 1 of (c4)
$$\{0=s(a),\ nn(a)\}$$
using literal 1 of (c4)
$$\{nn(a)\}$$
using literal 1 of (c7)
$$\{a=0,\ s(sk1(a))=0\}$$
using literal 1 of (c12)
$$\{s(sk1(a))=0\}$$
using literal 1 of (c5)
$$\{-A=0,\ -C=A,\ C=s(sk1(a))\}$$
using literal 3 of (c11)
$$\{-C=0,\ C=s(sk1(a))\}$$
using literal 1 of (c4)
$$\{0=s(sk1(a))\}$$
using literal 1 of (c4)
□
using literal 1 of (c7)

This refutation is displayed in Figure 1.

The SEq refutation of the same problem uses only the following theory:

$$\{\neg nn(A), A{=}0, A{=}s(sk1(A))\} \quad (c12)$$
$$\{\neg A{=}0, nn(A)\} \quad (c13)$$
$$\{\neg nn(A), nn(sk1(A)), A{=}0\} \quad (c14)$$
$$\{\neg nn(A), \neg B{=}s(A), nn(B)\} \quad (c15)$$

To shorten the proof, it is convenient to ignore the (RUE subst) restrictions and make use of the following instance of (c12):[4]

$$\{\neg nn(a), a{=}0, a{=}s(sk1(a))\} \quad (c12a)$$

Here is the SEq refutation:

$\{nn(s(a))\}$
clause for refutation
$\{s(a) = 0,\ nn(sk1(s(a)))\}$
using literal 1 *of* $(c14)$
$\{nn(sk1(s(a)))\}$
using (iii')
$\{\neg sk1(s(a)) = a,\ a = s(sk1(a)),\ a = 0\}$
using literal 1 *of* $(c12a)$
$\{\neg s(sk1(s(a))) = s(a),\ a = s(sk1(a)),\ a = 0\}$
using (iv)
$\{s(a) = 0,\ \neg nn(s(a)),\ a = s(sk1(a)),\ a = 0\}$
using literal 3 *of* $(c12)$
$\{\neg nn(s(a)),\ a = s(sk1(a)),\ a = 0\}$
using (iii')
$\{a = s(sk1(a)),\ a = 0\}$
using clause for refutation
$\{a = 0\}$
using (iii')
$\square$
using (iii')

This refutation is displayed in Figure 2.

Notice that only one inequality is introduced by RUE-resolution, in the step that uses (c12a). Also notice that in the SEq refutation, the inequalities at the leaves can all be refuted by the relatively efficient but incomplete rule (iii'). The steps of the shorter SEq resolution refutation are not less intuitive than the steps of the previous proof – on the contrary. This derives from the fact that syntactic equality is such an easy idea to grasp, whereas the explicit derivation of syntactic equality results can be tedious and messy. The fact that the SEq refutation is shorter is nice, but unsurprising, given that SEq resolution builds in many axioms. The difference in proof size shown in this small example will clearly be multiplied many times in deductions that involve extensive reasoning with inequalities. Even in a guided search, it is helpful to keep the reasoning about equations as simple as possible, especially when the number of clauses with equations is large.[5] The existence of a large set $\overline{Sk}$ obviously helps simplify equational reasoning.

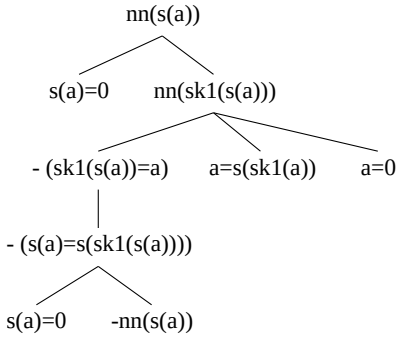

Figure 2: SEq resolution refutation of $nn(s(a))$

The maxims stated in the introduction also make it relevant to compare a problem formulated with syntactic equality vs. the "same" problem formulated without syntactic equality. It is not hard to construct such problems. For example, we can choose to formalize an operation with functions rather than with predicates. Consider the sum function. If we use a sum function symbol in our formalization of arithmetic, then we will have infinitely many different terms denoting each number:

$$s(0) = s(0){+}0 = s(0){+}0{+}0 = \ldots$$

If we use a sum predicate, on the other hand, we can respect the syntactic equality restriction, with the exception of necessary Skolem functions and constants. The equations shown above then correspond roughly to a proposition like (the universal closure of) the following:

[4] Digricoli shows how any refutation R can be transformed into another refutation R' that satisfies the (RUE subst) and (NRF subst) restrictions in §4.4 of [Digricoli, 1983]. Unfortunately, the transformation is infeasible in general, can dramatically increase the length of the refutation, and has not been shown to be effectively computable in general. The transformation is infeasible because it involves finding a new refutation of a clause not refuted in the original proof, where we know such a refutation exists but not how long it will be.

[5] We have applied these techniques to a complex theory involving equations, as discussed in [Stabler, 1988]. The theory considered there is a formalization of a theory of English syntax in which $\overline{Sk}$ contains all of the non-Skolem constants and function symbols.

$$
\begin{aligned}
& s(0) = X \\
\leftrightarrow\ & sum(s(0), 0, X) \\
\leftrightarrow\ & \exists Y\ \big(sum(s(0), 0, Y) \wedge sum(Y, 0, X)\big) \\
\leftrightarrow\ & \ldots
\end{aligned}
$$

Comparative studies of alternatives like these have been done before. It is not really surprising that, even when we do not use a proof method that builds in any axioms as SEq resolution does, proofs using a predicate formulation are sometimes less difficult that proofs using a function formulation. For example, in their work on group theory Bellin and Ketonen [Bellin and Ketonen, 1986] point out that, intuitively, a functional representation highlights "intensional features," and consequently many proofs are substantially simpler when we use the "more abstract" predicate representation.[6] In theories of sequences and trees, we have a choice about whether to formalize *append* as a function or predicate, and we have found a predicate formulation respecting syntactic equality to be the most practical in this case [Stabler, 1988].

6 Conclusions

The thesis of this paper is not surprising. The existence of a substantial, decidable set of terms whose denotations are known to be pairwise distinct can be a computational advantage. We have seen how this advantage can be obtained very naturally in one of the approaches that builds in equality and treats unification as a special kind of equation-solving. We extended this approach, RUE-NRF resolution, to SEq resolution, by building in additional axioms that depend on which terms are known to be pairwise distinct. SEq resolution prunes its search for proofs of equations by making use of available information about terms known to have distinct denotations in the intended model.

References

[Bellin and Ketonen, 1986] Gianluigi Bellin and Jussi Ketonen. Experiments in automatic theorem proving. Technical Report No. STAN-CS-87-1155, Department of Computer Science, Stanford University, 1987.

[Bledsoe and Hines, 1980] W.W. Bledsoe and L.M. Hines. Variable elimination and chaining in a resolution-based prover for inequalities. *Procs. 5th Conf. on Automated Deduction:* 70-87, 1980.

[Chang and Lee, 1973] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.

[Clark, 1978] Keith Clark. Negation as failure. In H. Gallaire and J. Minker, *editors. Logic and Data Bases.* Plenum Press, New York, 1978.

[Digricoli and Harrison, 1986] Vincent J. Digricoli and Malcolm C. Harrison. Equality-based binary resolution. *Journal of the ACM,* 33(2): 253–289, 1986.

[Digricoli, 1983] Vincent J. Digricoli. *Resolution by Unification and Equality.* Ph.D. Thesis, Department of Computer Science, New York University, 1983.

[Flannagan, 1986] Tim Flannagan. The consistency of negation as failure. *Journal of Logic Programming,* 3(2): 93–114, 1986.

[Lloyd, 1984] John W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, New York, 1984.

[Loveland, 1978] Donald W. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, New York, 1978.

[Morris, 1969] J.B. Morris. E-resolution: an extension of resolution to include the equality relation. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 287–294, 1969. International Joint Committee on Artificial Intelligence.

[Naish, 1985] Naish, L. (1985) Automating control for logic programs. *Journal of Logic Programming, 3*: 167-183, 1985.

[Reiter, 1965] Ray Reiter. Equality and domain closure in first-order databases. *Journal of the ACM,* 27: 235-249, 1965.

[Robinson, 1965] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM,* 12: 23–41, 1965.

[Sheperdson, 1984] John C. Sheperdson. Negation as failure: a comparison of Clark's completed data base and Reiter's closed world assumption. *Journal of Logic Programming,* 1(1): 51–79, 1984.

[6] The observation comes from a study of proofs done with a proof-checker for second order logic, EKL. Where permutations are treated as bijections of a finite set into itself, Bellin and Ketonen observe that the predicate representation is better for proofs that the identity function is a permutation and that every permutation has an inverse, but the function representation is better for showing that the composition of two permutations is a permutation.

[Stabler, 1988] Edward P. Stabler, Jr. *The Logical Approach to Syntax.* Forthcoming.

[Stabler, 1989] Edward P. Stabler, Jr. Syntactic equality in knowledge representation and reasoning. In *Proceedings of the First Int. Conf. on Principles of Knowledge Representation and Reasoning, KR '89*: 459-466. Kaufman, Los Altos, CA, 1989.

[Stickel , 1986] Mark Stickel. A prolog technology theorem prover: implementation by an extended Prolog compiler. In *Procs. 8th Int. Conf. on Automated Deduction*, Lecture Notes in Computer Science Volume 230, pages 573–587. Springer-Verlag, New York, 1986.

[Stickel , 1986] Mark Stickel. An introduction to automated deduction. In Wolfgang Bibel and Philippe Jorrand, *editors. Fundamentals of Artificial Intelligence: An Advanced Course,* Lecture Notes in Computer Science Volume 232, pages 75–132. Springer-Verlag, New York, 1986.

[van Emden and Yukawa , 1980] Maarten van Emden and Keitaro Yukawa. Logic programming with equations. *The Journal of Logic Programming,* 4(4): 265-288, 1980.

[Wos *et al.*, 1980] Larry Wos, Ross Overbeek, and Lawrence Henschen. Hyperparamodulation: A refinement of paramodulation. In *Procs. 5th Conf. on Automated Deduction*, pages 208-219. Springer-Verlag, New York, 1980.

[Wos *et al.*, 1984] Larry Wos, Ross Overbeek, Ewing Lusk, and Jim Boyle. *Automated Reasoning: Introduction and Applications.* Prentice-Hall,Englewood Cliffs, New Jersey, 1984.