On Handling Distinct Objects in the Superposition Calculus*

Stephan Schulz** and Maria Paola Bonacina***

Dipartimento di Informatica
Università degli Studi di Verona
Strada Le Grazie 15, 37314 Verona, Italy

Abstract. Many domains of reasoning include a set of *distinct* objects. For general-purpose automated theorem provers, this property has to be specified explicitly, by including distinctness axioms. Since their number grows quadratically with the number of distinct objects, this results in large and clumsy specifications, that may affect performance adversely. We show that object distinctness can be handled directly by a modified superposition-based inference system, including additional inference rules. The new calculus is shown to be sound and complete. A preliminary implementation shows promising results in the theory of arrays.

1 Introduction

Theorem-proving applications often require reasoning in specific domains. A frequent property of these domains is that certain named domain elements are assumed to exist and to be *distinct* from other named elements. One of the more frequent surprises for first-time users of theorem provers coming from PROLOG is that problems as the one in Fig. 1 are not provable for a pure first-order theorem prover, because a valid first-order interpretation can map both `bob` and `ted` to the same domain element. The obvious solution is, of course, to add the inequality `ted!=bob`, and indeed, with this refinement the problem becomes provable.

This solution, however, does not really scale well to large problems, as the number of inequalities grows quadratically with the number of distinct elements. For instance, take applications where a partial table of arithmetic results needs to be encoded (see, e.g., Fig. 2). In that case, all numbers used in the table are supposed to be distinct. However, even if only natural numbers from 0 up to 99 are used, there are 4950 inequalities to specify (each of the 100 numbers is distinct from 99 other numbers, but due to the symmetry of inequality, only one half of the 9900 inequations need to be included in the specification).

Furthermore, in less regular domains it is easy to overlook a necessary inequality. To return to the family relationship example, even the naïvely amended input set does not rule out that `bob` is his own son.

* Supported in part by MIUR grant no. 2003-097383.

** `schulz@eprover.org`
*** `mariapaola.bonacina@univr.it`

```
# Some family relationships: in contrast with some user
# expectations, this set is satisfiable (i.e. not provable).

son(bob, john).
son(ted, john).
brother(X,Y) <- son(X, FATHER), son(Y, FATHER), X!=Y.
?-brother(bob,ted).

# To make it provable, add the following (nonobvious) clause:
# bob!=ted.
```

Fig. 1. Family relationships as an example of (expected) distinctness of constants

```
Sum(  0,    0) =   0 &
Sum(  0,    1) =   1 &
Sum(  0,    2) =   2 &
Sum(  0,    3) =   3 &
Sum(  0,    4) =   4 &
Sum(  0,    5) =   5 &
Sum(  0,    6) =   6 &
Sum(  0,    7) =   7 &
Sum(  0,    8) =   8 &
Sum(  0,    9) =   9 &
Sum(  0,   10) =  10 &
```

Fig. 2. Partial arithmetic table (taken from the EUF test suite used by MathSAT3 [BBC$^+$05b,BBC$^+$05a])

The inequality clauses not only cause specifications to be large and clumsy, but they also need to be processed by the prover. Since they are treated as all other clauses, they take space in term and clause indices, are tested as potential inference partners, and, may, depending on the term ordering, even participate in inferences. Hence, there a significant cost (in memory and CPU time) associated with these clauses, and they can complicate the proof search.

In order to address these problems, we modified the superposition calculus by adding special inference and simplification rules to handle object distincteness without explicit axiomatization. The following sections describe this in some detail.

2 Preliminaries

Let $\Sigma = F \uplus D$ be a finite signature where all elements of D have arity 0. Elements of F are called *free function symbols* (or *free constants* for those with arity 0), and elements of D are called *object identifiers*. Object identifiers are assumed

Superposition (SP)

$$\frac{Q \vee l[u'] \simeq r \quad R \vee u \simeq v}{\sigma(Q \vee R \vee l[v] \simeq r)} \quad (i),\ (ii),\ (iii),\ \forall L \in Q : \sigma(l[u'] \simeq r) \not\preceq \sigma(L)$$

Paramodulation (PM)

$$\frac{Q \vee l[u'] \not\simeq r \quad R \vee u \simeq v}{\sigma(Q \vee R \vee l[v] \not\simeq r)} \quad (i),\ (ii),\ (iii),\ \forall L \in Q : \sigma(l[u'] \not\simeq r) \not\prec \sigma(L)$$

Equality Resolution (ER)

$$\frac{R \vee u' \not\simeq u}{\sigma(R)} \quad \forall L \in R : \sigma(u' \not\simeq u) \not\prec \sigma(L)$$

Equality Factoring (EF)

$$\frac{R \vee u \simeq v \vee u' \simeq v'}{\sigma(R \vee v \not\simeq v' \vee u \simeq v')} \quad (i),\quad \forall L \in (R \vee u' \simeq v') : \sigma(u \simeq v) \not\prec \sigma L$$

where σ is the most general unifier of u and u', u' is not a variable in *(SP)* and *(PM)*, and the following abbreviations hold:

(i) $\sigma(u) \not\preceq \sigma(v)$,
(ii) $\forall L \in R : \sigma(u \simeq v) \not\preceq \sigma(L)$,
(iii) $\sigma(l[u']) \not\preceq \sigma(r)$

Fig. 3. Expansion inference rules of $\mathcal{SP}$.

implicitly to denote distinct domain elements, and are referred to by i, j, k in the following. We assume a countably infinite set V of first order variables, and use upper case letters, usually X, Y, Z, to denote them.

The set of all *terms* over F, D, and V, $\mathit{Term}(\Sigma, V)$ is defined as usual. An equational *literal* is either an equation $t_1 \simeq t_2$ or an inequation $t_1 \not\simeq t_2$ over terms, where we use $\simeq$ for the equality predicate. Equations and inequations are unordered pairs of terms, i.e. the order of terms in the literal does not matter.

A *clause* is a multi-set of literals, interpreted and written as the disjunction of its literals, e.g., $L_1 \vee L_2 \vee \ldots \vee L_n$, where the L_i are literals. If R is a clause $L_1 \vee L_2 \vee \ldots \vee L_n$, we write $R \vee L$ for the clause $L_1 \vee L_2 \vee \ldots \vee L_n \vee L$. All clauses are assumed to be variable-disjoint. The empty clause is written as $\square$. A *substitution* is a mapping $\sigma : V \to \mathit{Term}(\Sigma, V)$ with the property that $\mathit{Dom}(\sigma) = \{X \in V \mid \sigma(X) \neq X\}$ is finite. It is extended to a function on terms, literals and clauses in the usual way.

We define $\mathit{DL}(D) = \{i \not\simeq j \mid \forall i, j \in D, i \neq j\}$ and $\mathit{DC}(D)$ as the set of unit-clauses containing exactly one literal from $\mathit{DL}(D)$. $\mathit{DL}(D)$ and $\mathit{DC}(D)$ are termed the sets of *D–disequality–literals* and *D–disequality–clauses*, respectively.

We assume that $\succ$ is a complete simplification ordering *(CSO)* on terms (i.e., a simplication ordering that is total on ground terms), lifted to literals and clauses via (sign-aware) multiset extension. Then the standard superposition calculus [BG94,BG98,NR01] is given by the inference rules in Fig. 3. In addition to generating rules, the superposition calculus is compatible with several contraction rules that either delete certain *redundant* clauses, or replace them

Strict Subsumption

$$\frac{S \cup \{C, C'\}}{S \cup \{C\}} \quad \text{if for some substitution } \theta,\ \theta(C) \subseteq C' \text{ and for no substitution } \rho,\ \rho(C') = C$$

Rewriting

$$\frac{S \cup \{C[l'], l \simeq r\}}{S \cup \{C[\theta(r)], l \simeq r\}} \quad \text{if } l' = \theta(l),\ \theta(l) \succ \theta(r), \text{ and } C[\theta(l)] \succ (\theta(l) \simeq \theta(r))$$

Deletion

$$\frac{S \cup \{R \vee t \simeq t\}}{S}$$

where S denotes a set of clauses.

Fig. 4. Contraction inference rules of $\mathcal{SP}$.

by simpler ones in some well-founded ordering. Fig. 4 lists some of the most important contraction rules. Let $\mathcal{SP}$ be the inference system given by the union of the sets of inference rules in figures 3 and 4.

The standard superposition calculus is sound and complete. A *fair* derivation starting from a clause set S will eventually derive the empty clause (and hence an explicit inconsistency) if and only if S is unsatisfiable.

3 Handling Object Distinctness in the Calculus

An approach to avoid the inclusion of disequality clauses is to introduce a variant superposition calculus that replaces explicit inferences with clauses from $DC(D)$ by the application of new inference rules. In the following, we require that $\succ$ has the property that all object identifiers are smaller than any other non-variable term, i.e. $s \succ i$ for all $s \notin D \cup V$, $i \in D$. Possible choices are a lexicographic path ordering with a suitable precedence or a Knuth-Bendix ordering with suitable weights and precedence.

Definition 1 *The inference system $\mathcal{SP}'$ is composed of the rules of $\mathcal{SP}$ and the additional rules shown in Fig. 5.*

While (OEC) is stated (and implemented) as a simplifying rule, it is the combination of a superposition and a subsumption inference, and hence (OEC) inferences are necessary for the completeness of $\mathcal{SP}'$.

We will now show that $\mathcal{SP}'$ is sound and complete, i.e. $S \cup DC(D) \vdash^*_{\mathcal{SP}} \Box$ if and only if $S \vdash^*_{\mathcal{SP}'} \Box$.

Theorem 1 (Soundness of $\mathcal{SP}'$) *If $S \vdash_{\mathcal{SP}'} S \uplus \{C\}$, then $S \cup DC(D) \models C$.*

Proof. Any clause that can be derived by (OER1), (OER2) and (OEC) is a logical consequence of the premises and $DC(D)$:

Object equality resolution 1 (OER1)
$$\frac{R \vee X \simeq i}{\sigma(R)} \quad \text{if } \sigma = \{X \leftarrow j\},\ i, j \in D,\ i \neq j,\ \forall L \in \sigma(R) : \sigma(X \simeq i) \not\preceq L,$$
Object equality resolution 2 (OER2)
$$\frac{R \vee X \simeq Y}{\sigma(R)} \quad \text{if } \sigma = \{X \leftarrow i, Y \leftarrow j\},\ i, j \in D,\ i \neq j\ \forall L \in \sigma(R) : \sigma(X \simeq Y) \not\preceq L,$$
Object equality cutting (OEC)
$$\frac{S \cup \{R \vee i \simeq j\}}{S \cup \{R\}} \quad \text{if } i, j \in D, i \neq j$$
Object tautology deletion (OTD)
$$\frac{S \cup \{R \vee i \not\simeq j\}}{S} \quad \text{if } i, j \in D,\ i \neq j$$

Fig. 5. Object identifier rules for $\mathcal{SP}'$

(OER1) Consider a clause $C = R \vee X \simeq i$ and $\sigma = \{X \leftarrow j\}, j \in D, j \neq i$. Since any clause implies all its instances, $\sigma(C) = \sigma(R) \vee j \simeq i$ is implied as well. Resolution between $\sigma(C)$ and $j \not\simeq i \in DC(D)$ generates $\sigma(R)$.
(OER2) Strictly analogous, with $\sigma = \{X \leftarrow i, Y \leftarrow j\},\ i, j \in D, i \neq j$.
(OEC) Again strictly analogous, with empty σ. □

For completeness, since all clauses removed by *(OTD)* are subsumed if $DC(D)$ is part of the clause set, we are only concerned with inferences involving the disequality clauses themselves. We shall see that rules *(OER1)*, *(OER2)* and *(OEC)* compensate for their absence.

The following definitions recapitulate and instantiate a few definitions dealing with redundancy in the superposition calculus.

Definition 2 *Let S be a set of clauses and C be a ground clause.*

- *C is called* redundant *with respect to S (and $\succ$), if there exist ground instances $C_1, \ldots, C_n$ of clauses in S such that $C_1, \ldots, C_n \models C$ and $C \succ C_i$ for all $i \in \{1, \ldots, n\}$. A non-ground clause is* redundant*, if all its ground instances are.*
- *C is called* object identifier redundant (OI-redundant) *with respect to S (and $\succ$), if $C \in DC(D)$ or C is redundant in $S \cup DC(D)$.*

The well-known principle *"once redundant, always redundant"* applies to OI-redundancy as well:

Lemma 1 *Let S be a set of clauses and let C, C' be clauses.*

- *If C is redundant (OI-redundant) in S, then C is redundant (OI-redundant) in $S \cup \{C'\}$.*
- *If C and C' are redundant (OI-redundant) in S, then C is redundant (OI-redundant) in $S \backslash \{C'\}$.*

Proof. See [BG94] for a detailed proof about redundancy. The result for OI-redundancy follows from the result for redundancy and the definition. □

Definition 3 *An* instance *of an inference with premises $C_1, \ldots .C_n$ and conclusion C is an inference with premises $\sigma(C_1), \ldots \sigma(C_n)$ and conclusion $\sigma(C)$ for some substitution σ. A* ground instance *of an inference is an instance such that σ is a grounding substitution for $C_1, \ldots, C_n$ and C. We write $\sigma(I)$ to denote the instance of I with substitution σ.*

The previous definition does not require the same inference rule to be applied in the inference instance. Many instances of the *object equality resolution* rules are applications of *object equality cutting*, as shown by the following lemma:

Lemma 2

1. *For each* object equality resolution *inference with premise clause $C = R \vee X \simeq t$ (where t is either an object identifier i or a variable Y), substitution σ and conclusion $C' = \sigma(R)$, there is an* object equality cutting *inference with premise $\sigma(C)$ and conclusion $\sigma(C')$.*
2. *For each* object equality cutting *inference with premise $C = R \vee i \simeq j$ and conclusion $C' = R$ and for all substitutions σ, there is an* object equality cutting *inference with premise $\sigma(C)$ and conclusion $\sigma(C')$.*

Proof. The result follows from the definitions. □

Inferences are redundant, if some of the participating clauses are:

Definition 4 *A (generating) ground inference with premises $C_1, \ldots, C_n \in S$ and conclusion C is redundant (with respect to S and $\succ$) if any of the premises is redundant, or C is redundant, or $C \in S$. An inference is redundant, if all its ground inferences are.* OI-redundant *inferences are defined analogously, replacing redundant with* OI-redundant.

With the notion of OI-redundant inference the concept of *saturation* is extended to *OI-redundancy*:

Definition 5 *Let S be a set of clauses and let $\mathcal{IS}$ be an inference system. S is* $\mathcal{IS}$-saturated up to redundancy*, if all (generating) $\mathcal{IS}$ inferences with premises in S are redundant. It is called* $\mathcal{IS}$-saturated up to OI-redundancy*, if all (generating) $\mathcal{IS}$ inferences with premises in S are OI-redundant.*

As usual, the completeness proof shows completeness on the ground level first, and then lifts it to the non-ground level. Non-ground inferences in the superposition calculus represent the set of all their ground instances. However, only non-redundant instances are necessary for completeness. The gist of the proof will be to show that $\mathcal{SP}'$ can simulate all non-redundant instances of $\mathcal{SP}$ inferences also in the absence of clauses deleted by *(OTD)*.

Lemma 3 *For any non-redundant ground instance $\sigma(I)$ of a generating $\mathcal{SP}$ inference I over $S = S' \cup DC(D)$ with conclusion $\sigma(C)$, there is a ground instance $\sigma(I')$ of an $\mathcal{SP}'$ inference I' over S' with conclusion $\sigma(C')$, such that $\sigma(C)$ and $\sigma(C')$ are logically equivalent, either $\sigma(C) \succ \sigma(C')$ or $\sigma(C)$ is $\sigma(C')$, and no premise of I' is OI-redundant in S'.*

Proof. Consider an arbitrary $\mathcal{SP}$-inference I with premises $C_1, \ldots, C_n$ and conclusion C and its non-redundant instance $\sigma(I)$. We distinguish the following cases:

- None of $C_1, \ldots, C_n$ is OI-redundant in S'. Then the thesis holds trivially with $I = I'$.
- At least one of the C_l, for $1 \leq l \leq n$, is OI-redundant. Without loss of generality, let $l = 1$. By Definition 2, either C_1 is redundant in $S \cup DC(D)$ or $C_1 \in DC(D)$. If C_1 were redundant in $S \cup DC(D)$, all inferences involving C_1 and all its instances would be redundant, contrary to the assumption that $\sigma(I)$ is not redundant. Thus, we are left with the case where $C_1 \in DC(D)$, i.e. $C_1 = i \not\simeq j$. We assume, also without loss of generality, that $i \succ j$. If we consider the $\mathcal{SP}$ inference rules, it is easy to see that only *Paramodulation* and *Equality Resolution* allow a negative unit clause as a premise. *Equality Resolution* requires that the two sides of the literal are unifiable, which is not the case for a disequality clause. So we only have to consider inferences I where $C_2 = R \vee u \simeq v$ paramodulates into $i \not\simeq j$ with most general unifier σ'. C_2 is not redundant in S (otherwise I would be redundant). Since $C_2 \notin DC(D)$ and C_2 is not redundant in S, C_2 is not OI-redundant in S'.
  Since i is maximal in C_1, u must unify with i for the inference rule to be applicable. Thus, either $u = i$ and σ' is the empty substitution, or $u = X$ and $\sigma' = \{X \leftarrow i\}$. Since $\sigma'(u) = i$ has to be maximal in $\sigma'(u \simeq v)$ and $i \in D$ is smaller than any non-variable term not in D, $\sigma'(v)$ has to be a variable Y or another object identifier $k \in D$ with $i \succ k$. We distinguish these cases:
  $C_2 = R \vee i \simeq k$: If $k \neq j$, then the conclusion of the inference I is $C = R \vee j \not\simeq k$. C is subsumed by $j \not\simeq k \in DC(D)$ and hence redundant. So we can assume $k = j$, and $C = R \vee j \not\simeq j$, which is equivalent to the smaller clause $C' = R$. The same clause is generated by an $\mathcal{SP}'$ inference I' using *Object equality cutting (OEC)* with premise C_2. Lemma 2 ensures the existence of $\sigma(I')$.
  $C_2 = R \vee X \simeq k$: As in the previous case, we can assume $k = j$. Then the conclusion of I is $C = \sigma'(R \vee j \not\simeq j)$, equivalent to $C' = \sigma'(R)$. The same clause is generated by I' applying *(OER1)* with premise C_2 and the same substitution σ'. Lemma 2 again extends this to $\sigma(I')$.
  $C_2 = R \vee i \simeq Y$: In this case, the conclusion of the inference I is $C = R \vee j \not\simeq Y$. Here we have to consider explicitly ground instances of I. Let τ be a grounding substitution for C_2 and C. τ must necessarily map Y to a term smaller than i, otherwise no inference is possible. As above, the only possible instantiation for Y is a smaller object identifier $k \in D$, and again the only choice that does not generate a redundant clause is

$k = j$. So we can write τ as $\tau' \circ \theta$, with $\theta = \{Y \leftarrow k\}$. The conclusion of the ground inference becomes $\tau(R \vee j \not\simeq j)$, which is equivalent to $C'' = \tau(R)$. If we apply *(OER1)* to C_2 with substitution θ, the resulting clause is $C' = \theta(R)$ and $\tau'(C') = C''$. Lemma 2 again guarantees the existence of the proper ground inference.

$C_2 = R \vee X \simeq Y$**:** This case is strictly analogous to the previous one, using *(OER2)* and with the addition that we have to apply the unifier $\sigma' = \{X \leftarrow i\}$ to C_2 first.

□

With this result one can establish the relationship between OI-redundancy and plain redundancy:

Lemma 4 *Let $S = S' \cup DC(D)$ be a clause set. If S' is $\mathcal{SP}'$-saturated up to OI-redundancy, then S is $\mathcal{SP}$-saturated up to redundancy.*

Proof. If S is not $\mathcal{SP}$-saturated up to redundancy, there must be a non-redundant $\mathcal{SP}$-inference I over S, and hence a ground instance of I with non-redundant conclusion C. But then, by Lemma 3, there is an $\mathcal{SP}'$-inference with non-OI-redundant premises from S' and with a ground instance I' with a conclusion C' equivalent to C such that $C = C'$ or $C \succ C'$. If $C \in DC(D)$, then I is OI-redundant. Otherwise, C is not OI-redundant (since it is not redundant in S) and hence C' cannot be OI-redundant. Hence I' is a non-OI-redundant instance of an $\mathcal{SP}'$-inference over S'. But this contradicts the premise that S' is $\mathcal{SP}'$-saturated up to OI-redundancy. Hence no such I exists, and therefore S is $\mathcal{SP}$-saturated up to redundancy. □

The central theorem of the completeness proof follows:

Theorem 2 *Let $S = S' \uplus DC(D)$ be a clause set. If S' is $\mathcal{SP}'$-saturated up to OI-redundancy, then either $\Box \in S'$, or S is consistent.*

Proof. If S' is $\mathcal{SP}'$-saturated up to OI-redundancy, then, by the previous lemma, S is $\mathcal{SP}$-saturated up to redundancy. Then the proof for standard superposition [BG94] applies. □

The next theorem uses standard notions of *derivation* (substituting OI-redundancy for redundancy) and *fairness*:

Theorem 3 *The limit S_∞ of a fair $\mathcal{SP}'$-derivation is saturated up to OI-redundancy.*

Proof. By definition of S_∞, any non-redundant inference is performed eventually and becomes redundant. Removal of OI-redundant clauses does not change this. □

The last theorem wraps up the completeness proof:

Theorem 4 *Let $S = S\prime \cup DC(D)$ be a clause set. If S is unsatisfiable, then a fair $\mathcal{SP}'$-derivation starting from S' will eventually derive the empty clause.*

Proof. It follows from Theorem 3. □

4 Preliminary Implementation and Experiments

We implemented a partial version of $\mathcal{SP}'$ in the equational theorem prover E [Sch02,Sch04]. Unless the feature is disabled by command line options, our experimental version of E treats strings in double quotes (`"Object"`) and positive integers (sequences of digits) as object identifiers. Our preliminary implementation supports only the inference rules (OEC) and (OTD). For the experiments reported here this restriction is irrelevant, because they pertain to the use of E as a decision procedure in the *theory of arrays with extensionality.* A case analysis of all possible clauses that can be generated by $\mathcal{SP}$ from this theory and a set of ground literals shows that (OER1) and (OER2) never apply to such problems [ABRS05]. We expect a full implementation of the calculus to be included in the next public release of E.

The syntactic distinction of free constants, numbers, and objects was accepted also in version 3 of the TPTP syntax [SZS03,SZS04] (see [Sut05] for the latest revision). Thus, we hope that this feature will see more use in future.

We observed the impact that distinctness axioms may have on performance while experimenting with a set of synthetic benchmarks in the theory of arrays [ABR$^+$02]. The problems from the so-called `STORECOMM` family capture the following property: given an array, the result of storing values at distinct indices is independent of the order of the *store* operations. To express this property, one needs to state that the array indices are distinct objects. For each n, `STORECOMM(`n`)` is the family of all problems where n different values are stored at n different indices in an array a. The theorem states that the resulting array in each case is equal to an array achieved by storing the values in some standard order in a. The axiomatization specifies the full theory of arrays with extensionality. As we were evaluating the use of E as a *decision procedure*, we also created *invalid* variants of the problems, where two different values are stored twice at the same index.

For each n, we have generated 9 valid and 9 invalid instances (with different permutations of the assignments) of the problem classes. The problems are reduced to clausal form, flattened, and pre-processed as described in [ABR$^+$02]. The resulting files in TPTP syntax are given to two versions of E. Except for the handling of objects, both version run the same strategy, including clause selection and term ordering. In particular, index constants are smaller than all other non-variable terms for both versions, and the term orderings coincide for all other terms. For comparison purposes, we have also tested CVC [SBD02] and CVC Lite [BB04] on the same problems (in flattened form). Both systems include hard-coded decision procedures for the theory of arrays. CVC is the Stanford *Cooperating Validity Checker*, a highly optimized monolithic system combining a SAT engine with various theory decision procedures. CVC is no longer supported; it was superseded by CVC Lite, a much more modular and programmable system. However, while CVC Lite has many advantages, the original CVC is reportedly still faster on many problems. Our experiments support this claim.

The reported result for each n use the *median* of the run times for all 9 instances. However, variation between different instances is negligible except for CVC Lite, which shows some limited variation for the larger problems.

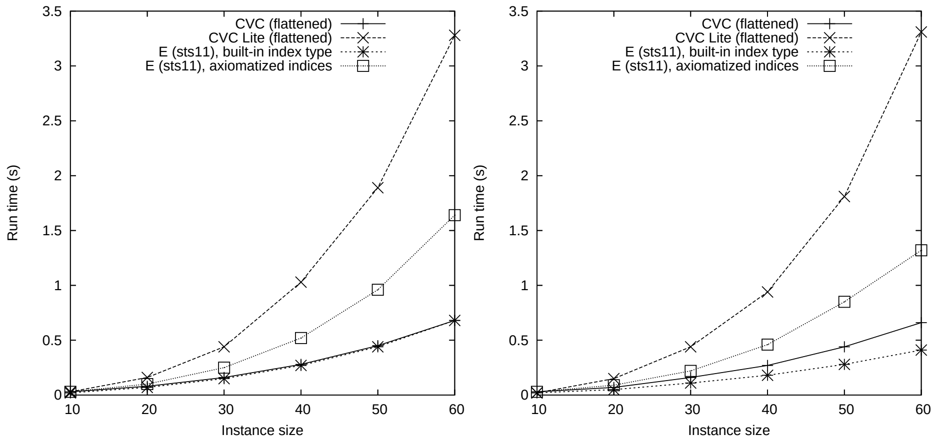

Fig. 6. Performance comparison on valid (left) and invalid (right) STORECOMM array problems

Figure 6 shows the results. For valid instances, the graphs for E with built-in object distinctness and the graph for CVC are essentially identical. Both systems need about 0.7 seconds for the instances of size 60. If the distinctness of the indices is axiomatized, E needs about 1.66 seconds for the same instances, for a speed-up factor of approximately 2.4. Finally, CVC Lite needed a median of 3.3 seconds for the instances of size 60.

For invalid (i.e. satisfiable) instances, the speed-up for E is similar. However, this is enough to make E with built-in support for distinct objects the strongest of the systems, followed by CVC, and then E using axiomatized indices. CVC Lite is again the slowest of the systems.

At least in this domain, the addition of rules for object identifiers yields a significant speed-up, making E with a first-order axiomatization of the theory of arrays competitive with some of the fastest special-purpose decision procedures.

5 Conclusion and Future Work

Object distinctness is a frequent requirement for many application domains. By handling this simple property at the calculus level, significant performance gains are possible at least in some of these domains. Moreover, the specification of problems in these domains becomes easier and the behavior of the prover is more in agreement with user expectations.

The next step will be, of course, the implementation of the full calculus. After that, we plan to evaluate the extended system over a larger range of problems, including finite arithmetic specifications and finite groups.

Another direction is the automatic detection of distinct objects in existing specifications. This would give us a better estimate of how widespread the property is and a larger set of test cases, possibly allowing us to improve the system on existing specifications.

References

[ABR+02] Alessandro Armando, Maria Paola Bonacina, Silvio Ranise, Michael Rusinowitch, and Aditya Kumar Sehgal. High-Performance Deduction for Verification: A Case Study in the Theory of Arrays. In Serge Autexier and Heiko Mantel, editors, *Proc. of the VERIFY Workshop, 3rd FLoC, Copenhagen, Denmark*, 2002. Available from `http://www-ags.dfki.uni-sb.de/verification-ws/verify02.html`.

[ABRS05] Alessandro Armando, Maria Paola Bonacina, Silvio Ranise, and Stephan Schulz. On rewriting-based theorem proving as decision procedure: an experimental appraisal. Manuscript in preparation, 2005.

[BB04] Clark W. Barrett and Sergey Berezin. CVC Lite: A New Implementation of the Cooperating Validity Checker. In Rajeev Alur and Doron A. Peled, editors, *Proc. CAV-16*, volume 3114 of *LNCS*, pages 515–518. Springer, 2004.

[BBC+05a] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. Mathsat: Tight integration of sat and mathematical decision procedures. *Journal of Automated Reasoning*, 2005. (accepted for publication).

[BBC+05b] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi Junttila, Peter van Rossum, Stephan Schulz, and Roberto Sebastiani. An Incremental and Layered Procedure for the Satisfiability of Linear Arithmetic Logic. In *Proc. TACAS-2005*, LNCS. Springer, 2005. (accepted for publication).

[BG94] Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *J. of Logic and Computation*, 4:217–247, 1994.

[BG98] Leo Bachmair and Harald Ganzinger. Equational Reasoning in Saturation-Based Theorem Proving. In W. Bibel and P.H. Schmitt, editors, *Automated Deduction — A Basis for Applications*, volume 9 (1) of *Applied Logic Series*, chapter 11, pages 353–397. Kluwer Academic Publishers, 1998.

[NR01] Robert Niewenhuis and Albert Rubio. Paramodulation-based theorem proving. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 1. Elsevier, 2001.

[SBD02] Aaron Stump, Clark W. Barrett, and David L. Dill. CVC: a Cooperating Validity Checker. In Kim G. Larsen and Ed Brinksma, editors, *Proc. CAV-14*, LNCS. Springer, 2002. See also the web page `http://verify.stanford.edu/CVC/`.

[Sch02] Stephan Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3):111–126, 2002. See also the web page `http://www.eprover.org`.

[Sch04] Stephan Schulz. System Description: E 0.81. In D. Basin and M. Rusinowitch, editors, *Proc. of the 2nd IJCAR, Cork, Ireland*, volume 3097 of *LNAI*, pages 223–228. Springer, 2004.

[Sut05] Geoff Sutcliffe. The TPTP Web Site. `http://www.tptp.org`, 2004–2005.

[SZS03] Geoff Sutcliffe, Jürgen Zimmer, and Stephan Schulz. Communication Formalisms for Automated Theorem Proving Tools. In V. Sorge, S. Colton, M. Fisher, and J. Gow, editors, *Proc. of the IJCAI-18 Workshop on Agents and Automated Reasoning*, pages 53–58, 2003. Available at `http://www.cs.bham.ac.uk/~vxs/ijcai03/index.html#program`.

[SZS04] Geoff Sutcliffe, Jürgen Zimmer, and Stephan Schulz. TSTP Data-Exchange Formats for Automated Theorem Proving Tools. In V. Sorge and W. Zhang, editors, *Distributed and Multi-Agent Reasoning*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2004. (to appear).