Finding small counter examples for abstract rewriting properties

Hans Zantema[1,2]

[1] Department of Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands, email: H.Zantema@tue.nl

[2] Institute for Computing and Information Sciences, Radboud University Nijmegen, P.O. Box 9010, 6500 GL Nijmegen, The Netherlands

Abstract. Rewriting notions like termination, normal forms, and confluence can be described in an abstract way referring to rewriting only by a binary relation. Several theorems on rewriting, like Newman's lemma, can be proved in this abstract setting. For investigating possible generalizations of such theorems, it is fruitful to have counter examples showing that particular generalizations do not hold. In this paper we develop a technique to find such counter examples fully automatically, and we describe our tool `Carpa` that follows this technique. The basic idea is to fix the number of objects of the abstract rewrite system, and to express the conditions and the negation of the conclusion in a satisfiability (SAT) formula, and then call a current SAT solver. In case the formula turns out to be satisfiable, the resulting satisfying assignment yields a counter example to the encoded property. We give several examples of finite abstract rewrite systems having remarkable properties that are found in this way fully automatically.

1 Introduction

Rewriting occurs in several flavors: first order term rewriting, graph rewriting, string rewriting, higher order term rewriting, conditional rewriting, rewriting with respect to strategies or priorities, and so on. But in all kinds of rewriting one considers a set A of objects that may be rewritten, and one considers a *rewrite relation* that is a binary relation on A that describes rewrite steps. Most times there is a notion of *computation*: apply rewrite steps as long as possible. If after a finite number of steps an object is obtained on which no rewrite step is possible any more, such an object is called a *normal form*, the result of the computation. One can state that computation coincides with rewriting to normal form. In this framework several things can go wrong. For instance, computation may go on forever, without reaching a normal form. This can be avoided by requiring that the rewrite system is *terminating*, that is, it does not allow infinite computations. In doing rewriting with respect to a terminating rewrite system one will always reach a normal form. But without extra restrictions this normal form may be not unique, while in most applications it is desirable that the result of a computation is unique, that is, does not depend on the choice of a rewrite step for objects that allow different rewrite steps. These non-unique normal forms can be avoided by requiring that the rewrite system is *confluent*, that is, if an object x can be rewritten to both y and z, then there exists an object w such such that both y and z can be rewritten to w.

A huge amount of research has been done on these basic notions termination and confluence and several variants. In particular, several results exist for concluding termination ([2–4, 11]) and confluence ([9, 10]) in abstract settings, that is, for binary relations on arbitrary sets, independent on the structure of the objects. In searching for theorems for rewriting properties it would be very convenient to have machinery for automatically finding counter examples for variants of the theorems, for instance, to check whether conditions are essential. In case this machinery finds such a counter example after removing a condition, one may conclude that this condition is essential, and inspecting the counter example may help for understanding why this is the case.

Developing such a machinery is exactly the topic of this paper. We focus on the situation where the set A is finite. In applications of rewriting, most time the sets of objects are infinite. But in case some property does not hold, often this can be shown by a finite counter example, and often the smallest possible counter example gives the best insight why the property does not hold. So we will not only focus on counter examples in which A is finite, we will also focus on counter examples in which $\#A$ is as small as possible. As our abstract rewrite relations are just binary relations on a set A, in this paper we will mainly speak about binary relations rather than rewrite systems.

As a binary relation on a set of n elements can be expressed by n^2 Boolean variables, our problem area can be seen as a class of constraint problems on Boolean variables. Our goal is to express rewriting properties like termination and confluence in propositional logic in such a way that we may express our problems as propositional SAT(isfiability) problems, by which we may exploit current powerful SAT solvers.

We succeeded in expressing all abstract rewriting properties of our interest. Often this can be done in several ways. For instance, a relation R on a finite set is terminating if and only if an irreflexive transitive relation S exists such that $R \subseteq S$, as we will see in Theorem 3. This characterization is the basis of how termination is characterized in our machinery. Other correct characterizations of termination of R are obtained by replacing the requirement of S being transitive by $R \cdot S \subseteq S$, or by $S \cdot R \subseteq S$. Alternatively, one also checks that a relation R on a finite set is terminating if and only if R^+ is irreflexive, for R^+ being the transitive closure of R, for which we also develop a characterization in propositional logic. We did quite some experiments on implementations based on these various characterizations, and finally chose for this paper only to present the characterizations for which the implementations yielded results most efficiently. For these characterizations we prove correctness in this paper. None of the proofs are very deep, the main effort was in finding the right characterizations.

Based on these characterizations we developed our tool `Carpa` (Counter examples for Abstract Rewriting Produced Automatically). In this tool a list of desired properties for a number of binary relations can be entered, and then for a given number $n = \#A$ it either gives an example in which the desired properties hold, or concludes that such an example does not exist. Internally this tool first builds a formula expressing the desired properties based on the characterizations presented in this paper, then it calls an external SAT solver. In case the SAT solver concludes that the formula is unsatisfiable, the tool concludes that there is no solution, and in case the SAT solver concludes that the formula is satisfiable, the tool investigates the corresponding satisfying assignment and extracts an example out of it that satisfies the given properties. Although internally this SAT solving is crucial, the user of `Carpa` does not see this and only sees the automatic creation of an example of a set of binary relations that satisfies the given list of desired properties.

This paper is organized as follows. In Section 2 some preliminaries are given. In Section 3 we describe the basic encoding of relations in propositional formulas. In Section 4 we describe how to characterize termination. In Section 5 we describe how to specify transitive closures. Based on this, in Section 6 we describe how to characterize confluence. Completeness is defined to be the conjunction of termination and confluence. In Section 7 we see how this can be characterized much more efficiently than by taking the conjunction of the characterizations of termination and confluence. In all of these sections we give several examples, for many of which it would be a hard job to find them by hand. These examples were all found by our tool `Carpa` as it is presented in Section 8. Most times these examples are the smallest possible, again shown by our implementation by yielding an unsatisfiable formula when decreasing the value of $n = \#A$. We conclude in Section 9.

2 Preliminaries

We start by recalling some basic notions that we will use throughout the paper.

A *binary relation* R on a set A is defined to be a subset of $A \times A$. For $x, y \in A$ we will use xRy as an abbreviation of $(x, y) \in R$.

A binary relation R on a set A is called *reflexive* if $\forall x \in A : xRx$ holds.

A binary relation R on a set A is called *irreflexive* if $\forall x \in A : \neg(xRx)$ holds.

A binary relation R on a set A is called *symmetric* if $\forall x, y \in A : xRy \rightarrow yRx$ holds.

A binary relation R on a set A is called *transitive* if $\forall x, y, z \in A : (xRy \wedge yRz) \rightarrow yRz$ holds.

For two binary relations R, S on a set A its *composition* $R \cdot S$ is defined to be the relation

$$R \cdot S \ = \ \{(x, z) \in A \times A \mid \exists y \in A : (xRy \wedge yRz)\}.$$

The *identity relation* I on A is defined by $I = \{(x, x) \mid x \in A\}$.

For $i \geq$ the relation R^i is defined inductively by

$$R^0 = I, \ \ R^{i+1} = R \cdot R^i.$$

It is easily proved by induction that $R^i \cdot R^j = R^{i+j}$ for all $i, j \geq 0$.

For a binary relation R on a set A its *transitive closure* R^+ is defined by

$$R^+ = \bigcup_{i=1}^{\infty} R^i;$$

it is the smallest transitive relation that contains R. More precisely, it satisfies the following well-known property for which we give a proof for being self-contained.

Lemma 1. *If R and S are binary relations on a set A for which $R \subseteq S$ and S is transitive, then $R^+ \subseteq S$.*

Proof. We prove by in induction on i that $R^i \subseteq S$. For $i = 1$ this is given, for $i > 1$ choose $(x, y) \in R^i$ arbitrary. Since $R^i = R \cdot R^{i-1}$ there exists z such that xRz and $zR^{i-1}y$. Using $R \subseteq S$ we obtain xSz, using the induction hypothesis we obtain zSy, using transitivity of S we obtain xSy, concluding the proof. □

For a binary relation R on a set A its *transitive reflexive closure* R^* is defined by

$$R^* = \bigcup_{i=0}^{\infty} R^i = I \cup R^+.$$

An element $x \in A$ is called a *normal form* with respect to R if $\neg(xRy)$ for all $y \in A$. For an element $x \in A$ an element $y \in A$ is called a *normal form of x* with respect to R if xR^*y and y is a normal form with respect to R.

A binary relation R on a set A is called *terminating* or *well-founded* if no infinite sequence $a_1, a_2, \ldots$ of elements in A exists such that $a_i R a_{i+1}$ for all $i \geq 1$. For A being finite this is equivalent to irreflexivity of R^+. With respect to a terminating binary relation, every element has at least one normal form.

The union of two terminating relations does not need to be terminating. For instance, for $R = \{(1, 2)\}$ and $S = \{(2, 1)\}$ both R and S are terminating but $R \cup S$ is not. The following theorem due to Doornbos and Von Karger [4] gives an extra condition by which termination of the union can be concluded.

Theorem 1. *Let R and S be terminating relations on a set A for which*

$$R \cdot S \subseteq R \cup (S \cdot (R \cup S)^*).$$

Then $R \cup S$ is terminating.

For a binary relation R its *inverse* R^{-1} is defined by

$$(x, y) \in R^{-1} \iff (y, x) \in R.$$

A binary relation R is called *confluent* if $(R^*)^{-1} \cdot R^* \subseteq R^* \cdot (R^*)^{-1}$.

It is easy to see that with respect to a confluent binary relation, every element has at most one normal form.

A binary relation R is called *locally confluent* if $R^{-1} \cdot R \subseteq R^* \cdot (R^*)^{-1}$. It is well-known that there are non-terminating relations R that are locally confluent but not confluent; in Example 3 we will see how to find such an example fully automatically. The following well-known theorem is usually called Newman's Lemma. For a proof we refer to standard texts like [1, 8].

Theorem 2. *A terminating relation is confluent if and only if it is locally confluent.*

A binary relation that is both terminating and confluent is called *complete*.

3 Basic Encoding

We fix a number n. We will consider binary relations on the set $\{1, 2, \ldots, n\}$. These binary relations are numbered from 1 to m, and are denoted by $R_1, R_2, \ldots, R_m$. Typically the first one, two or three are the relations referred to in the given property, the others are introduced for being able to express these properties. As we want to express the given property in a SAT problem, these m binary relations have to be expressed by Boolean variables. We do this by introducing n^2m Boolean variables $R(k, i, j)$ for $k = 1, \ldots, m$ and $i, j = 1, \ldots, n$. If these Boolean variables have Boolean values, they define the corresponding relation $R_1, R_2, \ldots, R_m$ as follows

$$(i, j) \in R_k \iff R(k, i, j) \text{ is true.}$$

Using this encoding, the following standard properties of binary relations are expressed as propositional formulas in the variables $R(k, i, j)$.

A relation R_k is reflexive if and only if $\mathsf{refl}(k)$ defined by

$$\mathsf{refl}(k) \equiv \bigwedge_{i=1}^{n} R(k, i, i)$$

holds.

A relation R_k is irreflexive if and only if $\mathsf{irrefl}(k)$ defined by

$$\mathsf{irrefl}(k) \equiv \bigwedge_{i=1}^{n} \neg R(k, i, i)$$

holds.

A relation R_k is transitive if and only if $\mathsf{trans}(k)$ defined by

$$\mathsf{trans}(k) \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \bigwedge_{p=1}^{n} ((R(k, i, j) \wedge R(k, j, p)) \rightarrow R(k, i, p))$$

holds.

Next we express some basic set theoretic concepts. A relation R_k is a subset of a relation $R_{k'}$ if and only if $\mathsf{subset}(k, k')$ defined by

$$\mathsf{subset}(k, k') \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} (R(k, i, j) \rightarrow R(k', i, j))$$

holds.

A relation R_k is the union of a relation $R_{k'}$ and a relation $R_{k''}$ if and only if $\mathsf{union}(k, k', k'')$ defined by

$$\mathsf{union}(k, k', k'') \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} (R(k,i,j) \leftrightarrow (R(k',i,j) \vee R(k'',i,j))$$

holds.

A relation R_k is the intersection of a relation $R_{k'}$ and a relation $R_{k''}$ if and only if $\mathsf{intersect}(k, k', k'')$ defined by

$$\mathsf{intersect}(k, k', k'') \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} (R(k,i,j) \leftrightarrow (R(k',i,j) \wedge R(k'',i,j))$$

holds.

Next we define relation composition. A relation R_k is the composition of a relation $R_{k'}$ and a relation $R_{k''}$ if and only if $\mathsf{compose}(k, k', k'')$ defined by

$$\mathsf{compose}(k, k', k'') \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} (R(k,i,j) \leftrightarrow \bigvee_{p=1}^{n} (R(k',i,p) \wedge R(k'',p,j)))$$

holds.

Note that the size of all of these formulas is $O(n^3)$.

Now we are ready to give our first example

Example 1. For any sub-relation R of a strict order, can we conclude that R^2 is transitive? We will build a formula expressing this question. Let $R_1 = R$, let R_2 be the strict order, that is, it is transitive and irreflexive, and let $R_3 = R^2$. Our formula will consist of the conjunction of all conditions and the negation of the conclusion:

$$\mathsf{subset}(1,2) \wedge \mathsf{trans}(2) \wedge \mathsf{irrefl}(2) \wedge \mathsf{compose}(3,1,1) \wedge \neg(\mathsf{trans}(3)).$$

Applying a SAT solver to this formula for various n shows that for $n \leq 4$ this formula is unsatisfiable, but for $n = 5$ it is satisfiable. After renumbering the elements $1,2,3,4,5$, the resulting satisfying assignment can be interpreted as $1 < 2 < 3 < 4 < 5$ for $<$ being the order R_2, and $R = R_1$ is a relation for which $1R2$, $2R3$, $3R4$ and $4R5$ all hold, showing $1R^23$ and $3R^25$, but for which $1R^25$ does not hold, indeed showing that R^2 is not transitive.

This kind of questions admits all kinds of extensions and variations. For instance, in a similar way one finds two sub-relations R, S of a strict order for which both R^2 and S^2 are not transitive, but for which both $R \cup S$ and $R \cap S$ are transitive. Here $n = 6$ is the smallest value yielding a satisfiable formula, hence yielding a solution.

4 Termination

In this section we present a way how to express the property of termination (= well-foundedness) of a binary relation on a finite set as a propositional formula. First we give the main theorem.

Theorem 3. *A binary relation R on a finite set is terminating if and only a binary relation S on the same set exists that is transitive and irreflexive, and for which $R \subseteq S$.*

Proof. (only if) Let R be terminating, and choose $S = R^+$. Then S is transitive and satisfies $R \subseteq S$. So it remains to prove that $S = R^+$ is irreflexive. Assume it is not, then there is an element a satisfying aR^+a. This yields an infinite reduction $aR^+aR^+aR^+ \cdots$, contradicting termination of R.

(if) Assume S satisfies $R \subseteq S$ and is transitive and irreflexive. Assume R is not terminating. Then R admits an infinite reduction, and since the set is finite there is an element a occurring more than once in this infinite reduction. Hence aR^ka for some $k > 0$. We prove by induction on k that $R^k \subseteq S$. For $k = 1$ this follows from $R \subseteq S$. For $k > 0$ and xR^ky there exists z such that $xR^{k-1}z$ and zRy. By the induction hypothesis we conclude xSz and zSy. Since S is transitive we obtain xSy, concluding the induction proof.

Since aR^ka and $R^k \subseteq S$ we conclude aSa, contradicting that S is irreflexive. This contradicts the assumption that R is not terminating, concluding the proof. □

Note that in Theorem 3 finiteness is only used in the 'if'-direction. There it is essential: the relation $R \ = <$ on the natural numbers is both irreflexive and transitive, so choosing $S = R$ satisfies all requirements, while R is not terminating.

Theorem 3 can be used for expressing termination of R_k in a SAT formula: simply add a fresh relation $R_{k'}$, so choose k' to be one higher than the highest relation number in use, and generate

$$\mathsf{trans}(k') \wedge \mathsf{irrefl}(k') \wedge \mathsf{subset}(k, k').$$

For example, the part $\mathsf{subset}(1,2) \wedge \mathsf{trans}(2) \wedge \mathsf{irrefl}(2)$ of the formula in Example 1 exactly expresses that R_1 is terminating.

We stress that Theorem 3 can not be used for expressing the negation of termination: for A being finite it is the case that R is non-terminating if and only if R^+ is not irreflexive, but from $R \subseteq S$ and S is transitive, we can not conclude that $S = R^+$, only $R^+ \subseteq S$. In Section 5 we will see how to fully specify R^+, by which non-termination can be expressed.

5 Transitive Closure

For a given binary relation R on a finite set A we want to fully specify its transitive closure R^+. The relation R^+ can be seen seen as the smallest fixed point of the equation

$$X = R \cup (R \cdot X).$$

However, by only giving this equation R^+ is not fully specified: in general this equation has more solution than only R^+. For instance, for R being the identity relation I on a set A, not only $R^+ = I$ satisfies the above equation, but also $A \times A$.

Instead we will specify R^+ by non-recursive equations.

Theorem 4. *Let R be a relation on a finite set A. Let $k \geq 1$ satisfy $2^k \geq \#A$. Let R_i be relations on A for $i = 1, 2, \ldots, k$, satisfying*

$$R_1 = R \cup R^2, \quad \textit{and} \quad R_{i+1} = R_i \cup R_i^2 \quad \textit{for } i = 1, \ldots, k-1.$$

Then $R_k = R^+$.

Proof. First we prove the following claim.

Claim: For $i = 1, 2, \ldots, k$ we have

$$R_i \; = \; \bigcup_{j=1}^{2^i} R^j.$$

This claim is prove by induction on i. For $i = 1$ it holds by definition. For the induction step we have to prove that

$$R_i \cup R_i^2 \; = \; \bigcup_{j=1}^{2^{i+1}} R^j,$$

using the induction hypothesis $R_i \; = \; \bigcup_{j=1}^{2^i} R^j$.

'$\subseteq$': If $(x, y) \in R_i$, then by the induction hypothesis we have $(x, y) \in \bigcup_{j=1}^{2^i} R^j \subseteq \bigcup_{j=1}^{2^{i+1}} R^j$. If $(x, y) \in R_i^2$ then there exists z such that $(x, z) \in R_i$ and $(z, y) \in R_i$. Using the induction hypothesis twice yields $(x, z) \in R^j$ and $(z, y) \in R^{j'}$ for $j, j' \leq 2^i$. Hence $j + j' \leq 2^{i+1}$, so $(x, y) \in R^{j+j'} \in \bigcup_{j=1}^{2^{i+1}} R^j$.

'$\supseteq$': Let $(x, y) \in \bigcup_{j=1}^{2^{i+1}} R^j$, then $(x, y) \in R^j$ for some $j \leq 2^{i+1}$. If $j \leq 2$ then $(x, y) \in \bigcup_{j=1}^{2^i} R^j = R_i$. If $j > 2$ then one can write $j = j' + j''$ for $1 \leq j', j'' \leq 2^i$, so $R^{j'} \subseteq R_i$ and $R^{j''} \subseteq R_i$. Hence

$$(x, y) \in R^j = R^{j'+j''} = R^{j'} \cdot R^{j''} \subseteq R_i \cdot R_i = R_i^2,$$

concluding the proof of the claim.

It remains to prove that $R_k = \bigcup_{j=1}^{2^k} R^j = R^+ = \bigcup_{j=1}^{\infty} R^j$. Here '$\subseteq$' is obvious; for the converse we have to prove that if $(x, y) \in R^j$ for any $j > 0$, then $(x, y) \in R^{j'}$ for some $j' \leq 2^k$. Let $(x, y) \in R^j$ for some $j > 2^k$. Then there are

$x_0, x_1, \ldots, x_j$ such that $x_0 = x$, $x_j = y$ and $x_i R x_{i+1}$ for all $i = 0, \ldots, j-1$. Since $j > 2^k \geq \#A$, among the j elements $x_0, \ldots, x_{j-1}$ from A at least one element occurs at least twice, by which $(x, y) \in R^{j'}$ can be concluded for some $j' < j$. Repeating this argument will yield $j' \leq 2^k$ for which $(x, y) \in R^{j'}$, concluding the proof. □

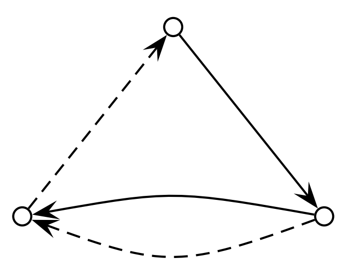
The bound $2^k \geq \#A$ in Theorem 4 is sharp, as is shown by the following example. Let $\#A = 2^k + 1$, say, $A = \{1, 2, \ldots, 2^k + 1\}$. Let R describe a cycle of length $2^k + 1$, say $R = \{(1,2), (2,3), \ldots, (2^k, 2^k + 1), (2^k + 1, 1)\}$. Then $(1,1) \in R^{2^k+1} \subseteq R^+$, but $(1,1) \notin R_k$ since 1 can not be reached from 1 in less than $2^k + 1$ steps.

Note that for expressing termination of a relation R by means of Theorem 3 only one auxiliary relation is required, while for expressing the transitive closure of a relation R by means of Theorem 4 the number of required auxiliary relations is logarithmic in $\#A$.

Example 2. A direct consequence of Theorem 1 is that the union of two terminating relations R and S is terminating if $R \cdot S \subseteq R \cup S^+$. One may wonder whether this still holds if this requirement is relaxed to $R \cdot S \subseteq R^+ \cup S^+$. It turns out to be not: a counter example can already be found for $\#A = 3$. A way to find such a counter example automatically is by calling a SAT solver on a formula expressing the requirements. According to Theorem 4 for $\#A \leq 4$ and $R_1 = R$ we obtain $R_3 = R^+$ by the formula

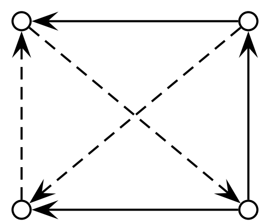
$$\mathsf{compose}(4,1,1) \wedge \mathsf{union}(2,1,4) \wedge \mathsf{compose}(5,2,2) \wedge \mathsf{union}(3,2,5),$$

in which we use $R_4 = R_1 \cdot R_1$, $R_2 = R_1 \cup R_4$ and $R_5 = R_2 \cdot R_2$ as auxiliary relations. Similarly we can express S^+ and $(R \cup S)^+$. As we need these transitive closures anyhow, we may also use them for expressing termination, and therefore will not use Theorem 3. The total formula consists of the conjunction of the formulas defining transitive closures, formulas expressing irreflexivity of R^+ and S^+ (for termination of R and S), a formula expressing the negation of irreflexivity of $(R \cup S)^+$ (for non-termination of $R \cup S$, and a formula expressing $R \cdot S \subseteq R^+ \cup S^+$. In Section 8 we will see how such a formula can be created fully automatically, only entering the requirements in a high-level format. Applying a SAT solver to this formula yields satisfiability, and from the satisfying assignment the following picture is extracted, in which R steps are denoted by solid arrows and S steps are denoted by dashed arrows.



Indeed, one easily checks that for both ways a solid arrow is followed by a dashed arrow, a path with the same start and end can be found either consisting of only solid arrows or only dashed arrows, showing $R \cdot S \subseteq R^+ \cup S^+$, while only the solid arrows are terminating, the same for dashed arrows, but the combination admits a cycle.

In this example we see that R and S are not disjoint: at the bottom there is both an R step and an S step from right to left. If we moreover require disjointness of R and S then for $\#A = 3$ there is no solution any more, but for $\#A = 4$ the same approach yields the following example:



The following theorem shows that the bound $2^k \geq \#A$ in Theorem 4 may be omitted in case R satisfies some extra condition, namely, that $R_k = \bigcup_{j=1}^{2^k} R^j$ is transitive. For taking k large enough, namely, $k \geq \log_2(\#A)$, this always holds due to Theorem 4, but often this already holds for smaller values of k.

Theorem 5. *Let R be a relation on a finite set A and let $k >= 1$. Let R_i be relations on A for $i = 1, 2, \ldots, k$, satisfying*

$$R_1 = R \cup R^2, \quad \text{and} \quad R_{i+1} = R_i \cup R_i^2 \quad \text{for } i = 1, \ldots, k-1.$$

Assume R_k is transitive. Then $R_k = R^+$.

Proof. From the proof of Theorem 4 we use the claim, and conclude $R_k = \bigcup_{j=1}^{2^k} R^j$. Hence $R_k \subseteq \bigcup_{j=1}^{\infty} R^j = R^+$. Conversely, we have $R \subseteq R_k$, and by Lemma 1 and transitivity of R_k we conclude $R^+ \subseteq R_k$. □

In case we need R^* rather than R^+ a simple way to specify this is by first specifying R^+ by means of Theorem 4 or Theorem 5, and then define $R^* = I \cup R^+$. Although this is correct, experience shows that in case only R^* is needed it is often much more efficient to specify R^* based on the following theorem, of which the proof is completely similar to the proofs of Theorem 4 and Theorem 5. For details on this efficiency claim we refer to Section 8.

Theorem 6. *Let R be a relation on a finite set A and let $k >= 1$. Let R_i be relations on A for $i = 1, 2, \ldots, k$, satisfying*

$$R_1 = I \cup R \cup R^2, \quad \text{and} \quad R_{i+1} = R_i \cup R_i^2 \quad \text{for } i = 1, \ldots, k-1.$$

Assume that either $2^k \geq \#A$ or R_k is transitive. Then $R_k = R^*$.

6 Confluence

For specifying confluence and local confluence of a binary relation R we need $R^* = I \cup R^+$. In Section 5 we described how for a given relation R this relation R^* can be described using a logarithmic number of auxiliary relations by means of Theorem 6. In the remainder of this section we assume that apart from R we also have access to the relation R^*.

Apart from composition it is convenient to specify peak and valley. For two binary relations R and S on a set A we write $\mathsf{peak}(R, S)$ for $R^{-1} \cdot S$, so

$$(x, y) \in \mathsf{peak}(R, S) \iff \bigvee_{z \in A} (zRx \wedge zSy).$$

Similarly, we write $\mathsf{valley}(R, S)$ for $R \cdot S^{-1}$, so

$$(x, y) \in \mathsf{valley}(R, S) \iff \bigvee_{z \in A} (xRz \wedge ySz).$$

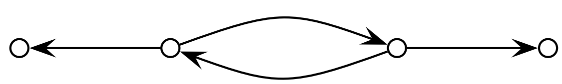
Now by definition a relation R is confluent if and only if

$$\mathsf{peak}(R^*, R^*) \subseteq \mathsf{valley}(R^*, R^*),$$

and a relation R is locally confluent if and only if

$$\mathsf{peak}(R, R) \subseteq \mathsf{valley}(R^*, R^*).$$

Example 3. We look for a binary relation on four elements that is locally confluent but not confluent. We do this by building a SAT formula being the conjunction of the specification of R^*, the requirement $\mathsf{peak}(R, R) \subseteq \mathsf{valley}(R^*, R^*)$, and the requirement $\neg(\mathsf{peak}(R^*, R^*) \subseteq \mathsf{valley}(R^*, R^*))$. Applying a SAT solver on this formula yields that the formula is satisfiable, and the corresponding satisfying assignment yields

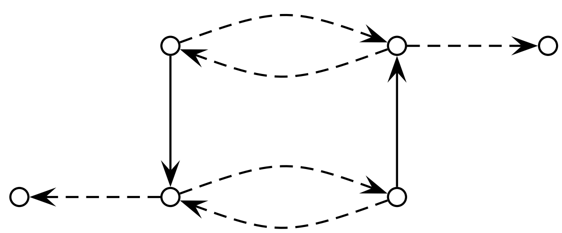


together with some self-loops that are redundant for the example. This is the well-known standard example of a locally confluent system that is not confluent, but now it has been found fully automatically. For details how this was done by our tool we refer to Section 8.

Example 4. By taking $R = \{(1, 2)\}$ and $S = \{(1, 3)\}$ one easily sees that the union of two confluent relations R and S does not need to be confluent. But in case moreover R, S-peaks converge, that is, $R^{-1} \cdot S \subseteq (R \cup S)^* \cdot ((R \cup S)^*)^{-1}$, one may conclude local confluence, and one may wonder whether then confluence of the union may be concluded. This is not the case: take $R = \{(1,2),(2,3)\}$ and $S = \{(2,1),(1,4)\}$: essentially the same example as in Example 3 in which the arrows from left to right are R steps and the arrows from right to left are S steps. One easily checks that in this example one has $R^{-1} \cdot S \subseteq (R^2)^{-1} \cup S^2$. Next we wonder whether this requirement of peak convergence may be strengthened to $R^{-1} \cdot S \subseteq S \cdot R^* \cdot (R^{-1})^*$, which means roughly speaking that the S steps always shift to the left and never disappear or duplicate. We build a formula being the conjunction of the specifications of R^*, S^* and $(R \cup S)^*$, and the requirements

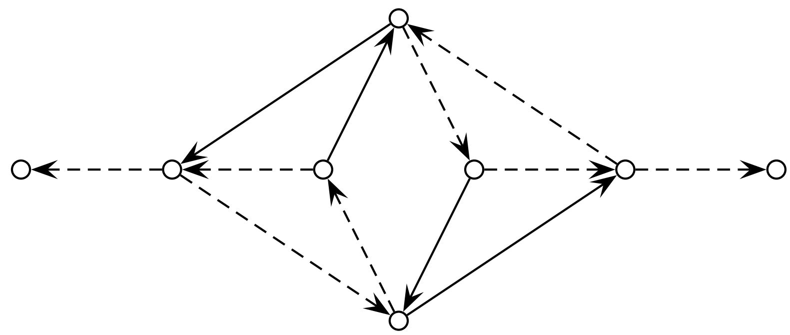
- $\mathsf{peak}(R^*, R^*) \subseteq \mathsf{valley}(R^*, R^*))$, stating that R is confluent,
- $\mathsf{peak}(S^*, S^*) \subseteq \mathsf{valley}(S^*, S^*))$, stating that S is confluent,
- $\neg(\mathsf{peak}((R \cup S)^*, (R \cup S)^*) \subseteq \mathsf{valley}((R \cup S)^*, (R \cup S)^*))$, stating that $R \cup S$ is not confluent, and
- $\mathsf{peak}(R, S) \subseteq \mathsf{valley}(S \cdot R^*, R^*))$, expressing the strengthened peak convergence requirement.

Applying a SAT solver on this formula yields that the formula is unsatisfiable for $n = \#A \leq 5$, but for $n = 6$ it is satisfiable, and the corresponding satisfying assignment yields



Here again R steps are denoted by solid arrows and S steps are denoted by dashed arrows. Indeed one easily checks that both the solid arrows and the dashed arrows are confluent, while the union is not since there are two distinct normal forms that are connected. Also the R, S-peak requirement holds, even $R^{-1} \cdot S \subseteq S \cdot R$. Note that since local confluence easily follows from our requirements, due to Newman's Lemma (Theorem 2) in every such example $R \cup S$ will be non-terminating.

Example 5. In [6] confluence was studied of a system combining simply typed lambda calculus and type computation, using the technique of decreasing diagrams [9, 10]. This work was extended to [7], where the abstract properties leading to confluence were further investigated. It turned out that for R being β-reduction in simply typed lambda calculus and S consisting of type computation steps, these relations are both confluent and satisfy $R^{-1} \cdot S \subseteq (S \cup R^*) \cdot (R^*)^{-1}$, while the goal was to prove that $R \cup S$ is confluent. So in this setting it was a natural question whether this could already be concluded from these abstract properties. It turned out to be not, as is shown by the following example in which again R and S are denoted solid and dashed, respectively:



This example was found by expressing variants of the requirements in a propositional formula, apply a SAT solver on it and inspect the resulting satisfying assignment. In fact this was the starting point of the current research. Directly expressing all requirements yields no result on less than 8 elements, and yields an example on 8 elements. Applying the same approach on a slightly stronger requirement, namely $R^{-1} \cdot S \;\subseteq\; (S \cup R) \cdot (R^*)^{-1}$, yields the slightly simpler and more symmetric example given above.

7 Completeness

A binary relation is called *complete* if it is both terminating and confluent. Due to Theorem 2 (Newman's Lemma) this is equivalent to being both terminating and locally confluent. Since termination implies that every element has at least one normal form and confluence implies that every element has at most one normal form, completeness implies that every element has exactly on normal form, which is a desirable property in many situations. One way to specify completeness in propositional logic is simply by both specifying termination by the approach of Section 4 and specifying confluence or local confluence by the approach of Section 6. However, this needs one auxiliary relation for termination and a logarithmic number of auxiliary relations for (local) confluence. Alternatively, one could specify the transitive closure of the relation as described in Section 5, and require this to be irreflexive, and specify confluence based on this transitive closure. But also this approach needs a logarithmic number of auxiliary relations. The next theorem describes a way to specify completeness using only two auxiliary relations, and which is much more efficient in the sense that SAT solvers find solutions much faster. This approach is based on discussions between Bas Joosten and the author.

Theorem 7. *A binary relation R on a set A is complete if and only if two binary relations S and T on A exist such that the following properties hold:*

1. $R \;\subseteq\; S$,

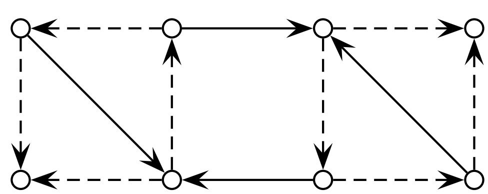
2. *S is transitive and irreflexive,*
3. $\bigwedge_{x \in A}(xTx \ \vee \ \bigvee_{y \in A} xRy)$,
4. $\bigwedge_{x,y \in A}((xSy \wedge yTy) \rightarrow xTy)$,
5. $\bigwedge_{x,y,z \in A, y \neq z} \neg(xTy \wedge xTz)$.

Proof. (only if) Let R be complete, then every element $x \in A$ has a unique normal form with respect to R. Define $S = R^+$ and define T by xTy if and only if y is the normal form with respect to R. Now property 1 is obvious and property 2 follows from termination of R. For property 3 we have to prove that xTx holds for every normal form x; this holds since x is its own normal form. For proving property 4 assume that xSy and yTy. Then xR^+y according to the definition of S, and y is a normal form with respect to R since yTy. Hence y is a normal form of x, hence xTy, concluding the proof. Finally, for proving property 5 assume that xTy and xTz for $x \neq y$. Then due to the definition of T the element x has two distinct normal forms y and z, contradicting completeness.

(if) Assume S and T satisfy properties 1 to 5. By properties 1 and 2 and Theorem 3 we conclude that R is terminating. It remains to prove that R is confluent. Assume xR^*y and xR^*z, we have to find w such that yR^*w and zR^*w. If $x = y$ we may choose $w = z$ and if $x = z$ we may choose $w = y$, so it remains to consider xR^+y and xR^+z. Since R is terminating, y has a normal form y' and z has a normal form z'. Since xR^+y and xR^+z we conclude xR^+y' and xR^+z'. Due to Lemma 1 we obtain xSy' and xSz'. Since y' and z' are normal forms, from property 3 we conclude that $y'Ty'$ and $z'Tz'$. Now from property 4 we conclude xTy' and xTz'. So using property 5 yields $y' = z'$. Now choosing $w = y' = z'$ concludes the proof. □

Using Theorem 7 the following approach specifies a relation R to be complete in a SAT formula: introduce two fresh relations S and T and add the requirements of Theorem 7 to the SAT formula. We give a few examples exploiting this approach.

Example 6. In Example 4 we saw that for two confluent relations R and S satisfying $R^{-1} \cdot S \ \subseteq \ S \cdot R^* \cdot (R^{-1})^*$ the union may be non-confluent. But in the given example the relation S is not terminating. Now we wonder whether the same holds for R and S both being terminating. We do so by building a formula in the same way as in Example 4, but now the requirements for R and S being confluent are replaced by R and S being complete, expressed by the requirements of Theorem 7. Applying a SAT solver on this formula yields that the formula is unsatisfiable for $n = \#A \leq 7$, but for $n = 8$ it is satisfiable, and the corresponding satisfying assignment yields

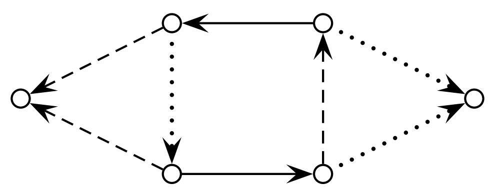


Here again R steps are denoted by solid arrows and S steps are denoted by dashed arrows. Indeed all requirements hold, in fact even

$$R^{-1} \cdot S \subseteq S \cup (S \cdot R \cdot R^{-1}).$$

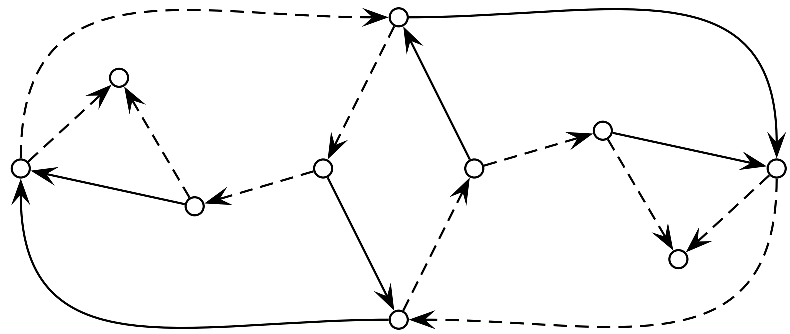
In Section 8 we describe in more detail how this example was obtained.

Example 7. If we have a set of binary relations of which the union of any two is complete, then it is easily seen that the union of all of them is locally confluent, as the union of any two is locally confluent. But can we conclude confluence of this union? This turns out to be not the case: we will give an example of three binary relations for which the the union of any two of them is complete, but for which the union of all three is not confluent. The requirements are expressed in a formula: for the union of any two we specify completeness according to the requirements in Theorem 7; for non-confluence we specify that for the transitive reflexive closure of the union of all three the peak is not a subset of the valley. It turns out that this formula is unsatisfiable for $n = \#A \leq 5$, but for $n = 6$ it is satisfiable, and the corresponding satisfying assignment yields



in which the three relations are indicated by solid, dashed and dotted arrows, respectively.

Example 8. In Example 5 we saw that for two confluent relations R and S satisfying $R^{-1} \cdot S \subseteq (S \cup R^*) \cdot (R^{-1})^*$ the union may be non-confluent, being a stronger requirement than in Example 4. Both in Example 4 and Example 5 the relation S is not terminating. So also here it is a natural question whether confluence of the union may be concluded if moreover both R and S are terminating. Remember that in the origin of this question R corresponds to β-reduction in simply typed lambda calculus and S corresponds to type computation, both being terminating. The following example, in which again R and S are denoted solid and dashed, respectively, shows that even for R and S both being terminating, the union does not need to be confluent.



This example was found several times. A first approach before the representation of completeness based on Theorem 7 was invented used the approach of Sections 4 and 6 to express termination and confluence of R and S. In this approach the SAT solver ran for hours without giving any result. This was the case for several variants of the problem, e.g. by slightly strengthening the peak requirement, or by requiring a term having two distinct normal forms rather than requiring non-confluence. By that time the result of Example 5 was already found, and the symmetry in this result was observed. So in a next attempt not only the given requirements were expressed in the formula, also a requirement of symmetry. More precisely, for $A = \{1, 2, 3, \ldots, 10\}$, for every $x, y \in A$ the requirements $(x, y) \in R \iff (11 - x, 11 - y) \in R$ and $(x, y) \in S \iff (11 - x, 11 - y) \in S$ were added. In this way the SAT solver found a solution within seconds, from which the above example was extracted.

Using the representation of completeness based on Theorem 7 it turned out that without adding symmetry requirements also a solution could be found within seconds, for several variants of the specification of the problem. All solutions that we found on 10 elements turned out to coincide with the example given above, sometimes after removing redundant arrows.

One can wonder whether this number of 10 elements is minimal. For proving so, the formula for $n = 9$ should be unsatisfiable. After a few hours of computation indeed this was concluded case for a formula expressing that R and S are complete, some element has two distinct normal forms with respect to $R \cup S$, and $R^{-1} \cdot S \subseteq (S \cup R) \cdot (R^{-1})^*$. Here $k = 3$ was chosen: from the requirement that $R \cup S$ has distinct normal forms it can be concluded that $R^* = \bigcup_{i=0}^{8} R^i$ for $n = 9$.

8 Implementation

We made an implementation called `Carpa` (Counter examples for Abstract Rewriting Produced Automatically) for entering a list of properties of binary relations, and then either builds a set of binary relations on the specified number of elements that satisfies these properties, or shows that this is impossible. The tool `Carpa` can be downloaded from

`http://www.win.tue.nl/~hzantema/carpa.html` including the source code, a Linux executable, and encodings of all examples in this paper.

Internally `Carpa` does this via SAT solving and the techniques described in this paper. As the SAT solver it uses `Yices` [5], which is not only a SAT solver, but also an SMT solver (satisfiability modulo theories). Experiments with other SAT solvers like `minisat` showed that for the considered kind of problems there was no substantial difference in efficiency. As for our kind of problems typically either solutions are found within seconds or are not found within several hours, we decided to use `Yices`, accepting the standard SMT format for which it is easy to generate formulas.

We defined an input format in which all properties discussed in this can paper can be specified directly, abstracting from the auxiliary relations that have to be introduced internally.

The input for `Carpa` always starts by three numbers n, k, m. Here $n = \#A$ is the cardinality of the set A on which we search for binary relations. The number k is the number of iterations required to define transitive closures and transitive reflexive closure based on Theorems 4, 5 and 6. In case the specification does not refer to these closures, this parameter is ignored. Finally, the number m is the number of basic relations in the specification, internally referred to numbers $1, \ldots, m$. So if we look for a single relation R with a given set of properties we choose $m = 1$, and if we look for two relation R and S with a given set of properties we choose $m = 2$.

The rest of the input consists of a number of lines each begin either a predicate or an assignment. In the following R, S refer to binary relations on A, and x, y refer to elements of A. The possible predicates are

- `subs`, where `subs`(R, S) means that $R \subseteq S$,
- `nsubs`, where `nsubs`(R, S) means that $\neg(R \subseteq S)$,
- `disj`, where `disj`(R, S) means that $R \cap S = \emptyset$,
- `trans`, where `trans`(R) means that R is transitive,
- `ntrans`, where `ntrans`(R) means that R is not transitive,
- `irr`, where `irr`(R) means that R is irreflexive,
- `nirr`, where `nirr`(R) means that R is not irreflexive,
- `symm`, where `symm`(R) means that R is symmetric,
- `sn`, where `sn`(R) means that R is terminating,
- `compl`, where `compl`(R) means that R is complete,
- `nf`, where `nf`(x, R) means that x is a normal form with respect to R, and
- `red`, where `red`(x, y, R) means that $(x, y) \in R$.

It looks strange to have separate predicates for the negations of other predicates instead of having an operator for negation. We decided to do so since predicates like `sn` and `compl` internally introduce auxiliary relations as described in Sections 4 and 7, by which taking the negation of the generated formula is not equivalent to the negation of the intended property.

Assignments always consist of a variable name followed by the symbol '=', followed by an operation applied on a number of arguments. Here for variable names we always choose 'x' followed by a number, and the possible operations are

- `union`, where `union`(R, S) represents the relation $R \cup S$,
- `inters`, where `inters`(R, S) represents the relation $R \cap S$,
- `comp`, where `comp`(R, S) represents the relation $R \cdot S$,
- `peak`, where `peak`(R, S) represents the relation $R^{-1} \cdot S$,
- `val`, where `val`(R, S) represents the relation $R \cdot S^{-1}$,
- `tc`, where `tc`(R) represents the transitive closure R^+ of R,
- `rc`, where `rc`(R) represents the reflexive closure $R \cup I$ of R, and
- `trc`, where `trc`(R) represents the transitive reflexive closure R^* of R.

Here the relations R, S should be either one of the basic relations, numbered $1, \ldots, m$, or a variable name that has been defined in an earlier assignment.

Our tool `Carpa` reads a list of requirements in this format, and builds a formula for it representing these requirements in the way as described in this paper. For every assignment a new binary relation is created. For every call of `sn` a new binary relation is created to represent S in Theorem 3, and to generate the corresponding requirements. For every call of `compl` two new binary relations are created to represent S and T in Theorem 7, and to generate the corresponding requirements. For every call of `tc` and `trc` k new binary relations are created to generate the requirements as described in Theorems 4, 5 and 6.

In this format we described the requirements for all examples as they occur in this paper, in fact all of the examples were found by applying our tool on the specifications written in this format.

For instance, for finding Example 3, a locally confluent relation on four elements that is not confluent, we choose $n = 4$ being the number of elements, $k = 2$ since that is the smallest value for which $2^k \geq n$, and $m = 1$ since we look for a single relation. For describing confluence and local confluence we need `x1` = `trc`(`1`), being R^* for R being the basic relation indicated by number 1. Further, we need the local peak `x2=peak(1,1)`, the valley `x3=val(x1,x1)`, and the global peak `x4=peak(x1,x1)`. Local confluence states that `x2` should be a subset of `x3`, while the negation of confluence states that `x4` should not be a subset of `x3`. Further, in order to avoid self-loops, we add the requirement that the relation is irreflexive. Combining all these ingredients yields the following input:

```
4
2
1
x1=trc(1)
x2=peak(1,1)
x3=val(x1,x1)
subs(x2,x3)
x4=peak(x1,x1)
nsubs(x4,x3)
irr(1)
```

On this input `Carpa` first generates a propositional formula of 531 lines, describing exactly the given requirements. Then it calls the SAT solver on this formula which yields satisfiability within a fraction of a second. Finally, `Carpa` inspects the satisfying assignment generated by the SAT solver and gives as output the desired relation:

```
Relation 1:
(1,2)
(1,3)
(2,1)
(2,4)
```

for which indeed coincides with Example 3.

Although internally the SAT solver plays a crucial role, the user of `Carpa` does not see this: he only calls `./carpa ex1` where `ex1` is a file containing the above input, and receives the above output immediately.

Next we consider Example 6; one way to achieve it is the following. Instead of non-confluence of the union we specify a slightly stronger requirement, namely that there exists an element with two distinct normal forms. As the input we define

```
8
3
2
compl(1)
compl(2)
x1=union(1,2)
nf(2,x1)
nf(3,x1)
x2=tc(x1)
red(1,2,x2)
red(1,3,x2)
x1=trc(1)
x2=comp(2,x1)
x3=peak(1,2)
x4=val(x2,x1)
subs(x3,x4)
```

in which it is specified that the element 1 has two distinct normal forms 2 and 3 with respect to the union `x1` of the two basic relations. For the rest this input only consists of the requirements that 1 and 2 are complete, and that $R^{-1} \cdot S \subseteq S \cdot R^* \cdot (R^{-1})^*$ for R being 1 and S being 2. Note that in this example variable names are reused: at some point the union `x1` of 1 and 2 was not needed any more, and `x1` was redefined by `x1=trc(1)`. On this input `Carpa` generates a formula of 8119 lines within a fraction of a second, on which the SAT solver needs a few seconds to establish satisfiability. From the corresponding satisfying assignment `Carpa` generates the output

```
Relation 1:
(1,4)
(5,4)
(7,6)
(8,6)
Relation 2:
(1,3)
(4,3)
(4,8)
(5,7)
(6,2)
(6,5)
(7,2)
(8,1)
```

that indeed can be represented by the picture given in Example 6.

9 Conclusions

This paper proposes a method for automatically finding finite counter examples for any list of abstract rewriting properties. The basic idea is to fix the number n of elements of the set on which binary relations are searched for, and then build a propositional formula describing the properties. On this formula a SAT solver is applied. If the formula is unsatisfiable then no example exists satisfying the given properties. If the formula is satisfiable then from the corresponding satisfying assignment an example is extracted satisfying the given properties. An implementation following this approach shows to be successful for various examples, typically up to around $n = 10$, including examples that are very hard to find by hand. The formulas are made in such a way that for every line in the list of properties at most $O(n^2 \log n)$ fresh Boolean variables are created, and the contribution to the size of the formula is at most $O(n^3 \log n)$. Although in SAT solving a restricted size of the formulas does not guarantee a quick solution at all, it is convenient not to have a combinatorial explosion in the size of the formula.

In our implementation `Carpa` we restricted to basic notions like termination, confluence, completeness, normal forms, transitive closures, and properties that can be expressed as compositions, peaks and valleys that, transitive closures, and subset relations on compositions, peaks and valleys. The main reason for this is that the main properties of our interest can be expressed in these notions. As soon other notions come up that can be expressed in a set-theoretic way, our approach and implementation may be easily extended accordingly.

A general observation in SAT solving is that among similar formulas some of which are satisfiable and other are not, proving unsatisfiability is harder than proving satisfiability. In our experiments this was confirmed: for n being the smallest number for which there exists a example for a given list of properties, typically finding such an example by our tool is done much faster than proving that such an example does not exist for $n - 1$.

In our encodings for termination we needed one auxiliary relation, for completeness we needed two, and for transitive closure and confluence we needed $\log(n)$ auxiliary relations. We conjecture that it is not possible to fully specify transitive closure or confluence only using a constant number of auxiliary relations.

Acknowledgement We want to thank Bas Joosten for his contribution to Theorem 7.

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
2. L. Bachmair and N. Dershowitz. Commutation, transformation and termination. In J. Siekmann, editor, *Proceedings of the 8th International Conference on Automated Deduction (CADE8)*, volume 230 of *Lecture Notes in Computer Science*, pages 5–20. Springer, 1986.
3. F. Bellegarde and P. Lescanne. Termination by completion. *Applicable Algebra in Engineering, Communication and Computing*, 1(2):79–96, 1990.
4. H. Doornbos and B. von Karger. On the union of well-founded relations. *Logic Journal of the IGPL*, 6(2):195–201, 1998.
5. B. Dutertre and L. de Moura. Yices: An SMT solver. Available at `http://yices.csl.sri.com/`.
6. A. Stump, G. Kimmell, and R. El Haj Omar. Type Preservation as a Confluence Problem. In Manfred Schmidt-Schauß, editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications (RTA)*, volume 10 of *LIPIcs*, pages 345–360, 2011.
7. A. Stump, G. Kimmell, H. Zantema, and R. El Haj Omar. A rewriting view of simple typing. 2012. submitted.
8. Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.
9. V. van Oostrom. Confluence by decreasing diagrams. *Theoretical Computer Science*, 126(2):259–280, 1994.
10. V. van Oostrom. Confluence by Decreasing Diagrams, Converted. In A. Voronkov, editor, *Rewriting Techniques and Applications*, pages 306–320, 2008.
11. V. van Oostrom and H. Zantema. Triangulation. 2012. submitted.