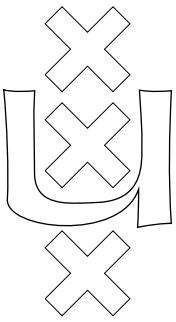
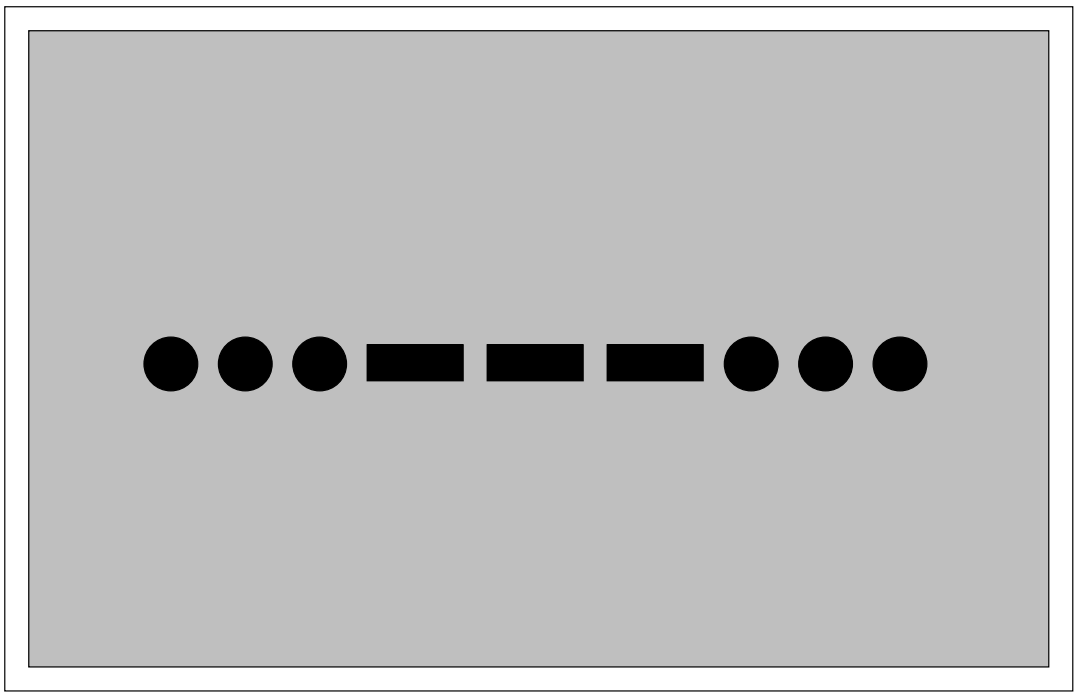


University of Amsterdam

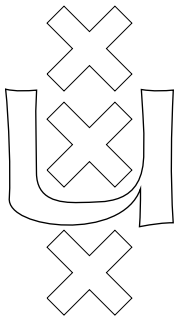
Programming Research Group



An SOS Message: Conservative Extension in Higher-Order Positive/Negative Conditional Term Rewriting

Wan Fokkink, Chris Verhoef

Report P9715 september 1997



University of Amsterdam

Department of Computer Science

Programming Research Group

An sos message: conservative extension in higher-order positive/negative conditional term rewriting

Wan Fokkink, Chris Verhoef

Report P9715 september 1997

W. Fokkink

University of Wales Swansea

Singleton Park
Swansea SA2 8PP
Wales

e-mail: w.j.fokkink@swan.ac.uk

C. Verhoef

Programming Research Group
Department of Computer Science
University of Amsterdam

Kruislaan 403
NL-1098 SJ Amsterdam
The Netherlands

tel. +31 20 525 7581
e-mail: x@wins.uva.nl

Universiteit van Amsterdam, 1997

An SOS Message: Conservative Extension in Higher-Order Positive/Negative Conditional Term Rewriting

Wan Fokkink — *University of Wales Swansea*
Chris Verhoef — *University of Amsterdam*

Abstract. General theorems in structured operational semantics can be transformed into related results in conditional term rewriting. We apply this approach to obtain a conservative extension theorem for higher-order conditional term rewriting systems with negative conditions. This result is useful, for instance, in modular specification of abstract data types, and in software renovation factories.

1 Introduction

There is a strong link between the worlds of conditional term rewriting [23, 7] and of structured operational semantics (SOS) [29]. In fact, from a conceptual level they can be seen as identical. In both fields, terms are built from a set of function symbols. The binary relations on terms, rewrite steps and transitions, both are defined inductively by means of proof rules, called conditional rewrite rules or transition rules, respectively. Those rules, together with the validity, or non-validity, of a number of relations between terms, may imply the validity of another relation between terms. A collection of conditional rewrite rules is called a conditional term rewriting system (CTRS), while a collection of transition rules is called a transition system specification (TSS). In view of their similarity, it was proposed in [2] to unify this terminology to term deduction system.

There is one technical distinction in the notion of provability. In conditional term rewriting, provability is closed under context, in other words, if $s \to t$ is provable, then $Con[s] \to Con[t]$ is provable for every context Con. The set of transitions provable from a TSS does not have to satisfy this characteristic, so in general a TSS cannot be expressed as a CTRS. However, the reverse transposition is possible, that is, for each CTRS there is an equivalent TSS. This transformation is obtained by adding so-called context rules for all function symbols. Owing to this observation, SOS theorems may apply to the field of conditional term rewriting. In this paper, we will state a theorem for CTRSs that we prove using the above correspondence between TSSs and CTRSs.

This correspondence was noted, but not exploited, by Groote and Vaandrager [22, Example 3.5]. They refrain from transposing their congruence format from TSSs to CTRSs, because it would not serve any practical purpose. Namely, although term rewriting and SOS theory are rooted on the same basis, their aims are fundamentally different. Term rewriting seeks for termination and confluence of reductions, while in SOS theory most interesting behaviour is infinite

and non-confluent. Usually, TSSs need to define a congruence relation with respect to a certain semantics, i.e., if two terms s and t are semantically equivalent, then $Con[s]$ and $Con[t]$ are semantically equivalent for all contexts $Con[]$. Several formats for TSSs have been developed which guarantee that they define a congruence relation for bisimulation semantics [22, 8, 2, 35, 16]. Most CTRSs from the literature do not fit these formats.

Conditional rewriting and SOS theory do have some interests in common. For example, there is the question which relation is defined by CTRS and TSS definitions with negative conditions. Negative conditions allow definitions such as $eq(x, y) \to false \Leftarrow x \neq y$. In this paper, we allow negative conditions of the form $s \neg Dt$ for relations D, which expresses that there does not exist a relation sDt. Kaplan [24] gave meaning to CTRSs with such negative conditions by means of quasi-initial models. We give meaning to such negative conditions using the more restricted class of stable models [18, 3, 8]. Recently, van de Pol [30] used semantic definitions for TSSs with negative conditions [8, 19] to provide negative answers to three open questions in term rewriting with priorities [3].

A second common interest is the notion of a conservative extension. When a CTRS (or TSS) is extended with new rules, then the question arises whether or not such an extension influences the rewrite steps (or transitions) of terms in the original domain. Usually, it is desirable that an extension is conservative, in the sense that the rewrite steps (or transitions) for an original term are the same both in the original and in the extended CTRS (or TSS). Several formats have been developed which implicate that an extended TSS is conservative over the original TSS [22, 8, 34, 17, 13]. Groote and Vaandrager [22, Theorem 7.6] proposed the first syntactic restrictions for an original TSS and its extension, which automatically yield that the extension is conservative. Bol and Groote [8] adapted this conservativity format to the setting with negative conditions. Verhoef [34] proposed more general syntactic criteria, which were later on extended to a setting with inequalities in [13]. In [17], Verhoef's format was transposed to higher-order languages. The current paper arose from the final section in [17], where a simplified version of this conservativity format is transposed to the setting of conditional term rewriting. As opposed to the conservativity format for TSSs, in this paper we do not allow the possibility that the left-hand side of a rewrite rule in the extension is an original term. The conservativity format for TSSs could be transposed to CTRSs in its full generality, but the simplifications make it easier to digest and appreciate the conservativity format.

Our conservativity result formulates sufficient syntactic requirements on the form of conditional rewrite rules in CTRSs R_0 and R_1, to ensure that the rewrite relation induced by R_0 on original terms will not be affected by rewrite rules in R_1. It requires for each conditional rewrite rule in R_0 that the variables in the rule are *source-dependent*, meaning that each variable can, in some sense, be related to the left-hand side of the rule. Furthermore, it requires for each conditional rewrite rule in R_1 that the left-hand side of the rule contains a fresh function symbol. These criteria are captured by three standard restrictions on (the union of) CTRSs: variables in the right-hand side of a rewrite rule also

occur in its left-hand side, conditions do not contain fresh variables, and the signatures of two CTRSs are disjoint.

The conservativity format is applicable in the field of abstract data types, where there is a long tradition in specifying by means of modules of CTRSs. In abstract data typing, modular specification means conservative extension, in our terminology. Namely, original modules fix the semantics of original terms, which should not be changed thereafter; new modules give meaning to fresh terms, which did not have a semantics before. See [4] for more information. We will provide examples from the literature, concerning term rewriting, abstract data types, and software renovation factories, to exemplify the conservativity format.

Bergstra and Klop [7] classify the various kinds of CTRSs that occur in the literature into four types: I (semi-equational), II (join), III, and III$_n$ (normal). These types differ in the relations that occur in the conditions of the rewrite rules. We will show how our conservativity theorem applies to CTRSs of type III. Since type III$_n$ is a subtype of III, the conservativity format applies to CTRSs of type III$_n$ as well. Furthermore, CTRSs of type II can be transformed into a CTRS of type III, so the conservativity format can also be applied to CTRSs of type II. For CTRSs of type I an additional requirement is needed, namely, the original and the extended CTRS should be confluent.

Modular properties of term rewriting systems have been studied extensively. These studies all deal with the following question: given two (mostly unconditional) TRSs with a certain desirable property, such as confluence or termination, does the combination of these TRSs also satisfy this property? It is always assumed that the signatures of the two rewrite systems should be disjoint, and that the variables in a rewrite rule all occur in its left-hand side. Therefore, the CTRSs studied in modularity papers automatically satisfy the requirements of our conservativity format. In this paper it is investigated whether the *full* rewriting relation is preserved for terms over only *one* of the signatures, and the signatures of the original CTRS and its extension need *not* be disjoint.

Toyama [31] showed that confluence is a modular property for TRSs [31], but that in general termination is not [32]. Klop and Barendregt gave a counterexample which shows that completeness is not modular, but Toyama, Klop and Barendregt [33] proved that completeness in combination with left-linearity is modular for TRSs. Ohlebusch [28] showed that if a combination of two TRSs does not terminate, then one of the TRSs is not $\mathcal{C}_\mathcal{E}$-terminating, while the other TRS is collapsing. (This generalizes a similar result for finitely branching TRSs of Gramlich [21]). Middeldorp [26] presented a panorama of positive and negative results on modular properties of CTRSs. For example, he showed that confluence constitutes a modular property for CTRSs. Gramlich [20] showed that his main results in [21] extend to CTRSs.

The conservativity format for structured operational semantics also has a direct application to term rewriting, as was noticed in [35]. It can help, for example, to obtain a simple completeness proof for the process algebra ACP [6]. In that paper, completeness of the equations for ACP is derived by means of a term rewriting analysis. In order to prove confluence of the TRS, they need to check

$\pm$400 cases. Completeness of the equations for ACP could also be obtained by the combination of a much simpler completeness result, a conservative extension result for the operational semantics, and an elimination result:

- the equations for the subalgebra BPA are complete;
- the operational semantics for ACP is a conservative extension of the operational semantics for BPA;
- the extra operators of ACP (with respect to BPA) can be eliminated by means of the equations for ACP.

2 Preliminaries

This section introduces some basic definitions of term rewriting [14, 25]. Also, we give meaning to conditional term rewriting systems with negative conditions, using stable models, which were developed, partly independently, in logic programming [18], term rewriting [3] and structured operational semantics [8]. In structured operational semantics negative conditions have the form $s\neg D$, which means that there is no transition sDt for any term t. In conditional term rewriting it is customary to have negative conditions of the form $s\neg Dt$, which means that the rewrite step sDt is not valid. We will use the latter notion.

2.1 The Syntax

We use a notational approach based on [1]; it is the notation for terms in the Nuprl proof development system, see [12]. In the sequel, $\bar{O}$ will denote a sequence $O_1 \dots O_k$ (and $\bar{O}_i$ a sequence $O_{i1} \dots O_{ik}$) for some $k \geq 0$.

Definition 1. A *(many-sorted) signature* Σ consists of a set of sorts, a countably infinite set $\mathcal{V}$ of sorted variables, and a non-empty set of function symbols

$$f : \bar{S}_1.S_1 \times \cdots \times \bar{S}_n.S_n \to S,$$

where the S_{ij} and the S_i and S are sorts.

A function symbol of arity zero is called a *constant*.

Definition 2. Let Σ be a signature. The collection of *(open) terms* $s, t, \dots$ over Σ is defined as the least set satisfying:

- each variable from $\mathcal{V}$ is a term;
- for each function $f : \bar{S}_1.S_1 \times \cdots \times \bar{S}_n.S_n \to S$, $f(\bar{x}_1.t_1, \dots, \bar{x}_n.t_n)$ is a term of sort S, where
  - the terms t_i are of sort S_i,
  - each $\bar{x}_i$ is a sequence of distinct variables $x_{i1} \dots x_{ik_i}$, with x_{ij} of sort S_{ij}.

Definition 3. *Free* occurrences of variables in terms are defined as expected:

- x occurs *free* in x for each $x \in \mathcal{V}$;

- if x occurs *free* in t_i, and x does not occur in the sequence $\bar{x}_i$, then x occurs *free* in $f(\bar{x}_1.t_1, \ldots, \bar{x}_n.t_n)$.

A term is called *closed* if it does not contain any free occurrences of variables.

Definition 4. A *substitution* is a sort-preserving mapping σ from variables to terms. It extends to a mapping from terms to terms as usual; the term $\sigma(t)$ is obtained by replacing each free occurrence of a variable x in t by $\sigma(x)$.

In the definition of substitutions on terms there is a well-known complication. Namely, consider a term $\sigma(t)$, and let x occur free in t. After x in t has been replaced by $\sigma(x)$, variables y that occur in $\sigma(x)$ are suddenly bound in subterms such as $f(y.s)$ of t. A solution for this problem, which originates from the λ-calculus, is to allow unrestricted substitution by applying α-conversion, that is, by renaming bound variables. In the sequel, terms are considered modulo α-conversion, and when a substitution is applied, bound variables are renamed.

2.2 Conditional Rewrite Rules

We assume a signature Σ, and a set $\mathcal{D}$ of relation symbols. The symbols in $\mathcal{D}$ represent binary rewrite relations between closed terms over Σ.

Definition 5. For closed terms s, t over Σ, and $D \in \mathcal{D}$, sDt is called a *positive rewrite step*, and $s \neg Dt$ is called a *negative rewrite step*.

The standard rewrite relation is the one-step relation $\rightarrow$, but we will also encounter its transitive-reflexive closure $\twoheadrightarrow$, the join $\downarrow$, and the equality sign $=$.

Definition 6. A *conditional rewrite rule* is of the form $p \Leftarrow C$, where

- the *conclusion* p is of the form $s[s_1/x_1, ..., s_m/x_m]\, D\, t[t_1/y_1, ..., t_n/y_n]$;
- the *conditions* in C are of the form $s[s_1/x_1, ..., s_m/x_m]\, D\, t[t_1/y_1, ..., t_n/y_n]$ or $s[s_1/x_1, ..., s_m/x_m]\, \neg D\, t[t_1/y_1, ..., t_n/y_n]$;

with $s, s_1, ..., s_m$ and $t, t_1, ..., t_n$ open terms. A *conditional term rewriting system* (CTRS) is a set of conditional rewrite rules.

The substitution harnesses of the form $[s_1/x_1, ..., s_m/x_m]$ in the rewrite rules are pieces of syntax. Intuitively, they replace the variables x_i by the terms s_i, in the term just before the substitution harness. However, a substitution harness only becomes active after a substitution has been applied to the conditional rewrite rule in which it occurs. We give an example.

Example 1. Consider the well-known β-rule from the λ-calculus:

$$(\lambda\, x.y)z \rightarrow y[z/x].$$

In this rule, the substitution harness $[z/x]$ is formal syntax. After application of the substitution σ with $\sigma(y) = x$ and $\sigma(z) = a$ to this rule, it takes the form

$$(\lambda\, x.x)a \rightarrow x[a/x].$$

In this last expression, $[a/x]$ is a substitution, so that $x[a/x]$ evolves into a:

$$(\lambda\, x.x)a \to a.$$

In order to give a rigorous treatment of the meaning of substitution harnesses, it is necessary to distinguish between 'actual' and 'formal' variables. This subtle treatment of variables asks for a more complicated formal framework. Here, we refrain from this distinction, and follow the more conventional set-up, which recognizes only one kind of variables. The interested reader is referred to [17] for more information on the motivation and definition of actual and formal variables.

2.3 Stable Models

Definition 7. A *proof* from a CTRS R of a closed rewrite rule $p \Leftarrow C$ (which contains only closed terms, and no substitution harnesses) consists of an upwardly branching tree in which all upward paths are finite, where the nodes of the tree are labelled by positive rewrite steps, such that:

- the root has label p,
- if some node has label q, and K is the set of labels of nodes directly above this node, then
  1. either $K = \emptyset$, and $q \in C$,
  2. or $q \Leftarrow K$ is a closed substitution instance of a rewrite rule in R.

When there are negative conditions around, it is no longer straightforward to define a sensible rewrite relation. We use the notion of a stable model [18, 3, 8].

Definition 8. A collection of positive rewrite steps $\mathcal{M}$ is a *stable model* for a CTRS R, if the elements of $\mathcal{M}$ are exactly those positive rewrite steps sDs' for which there exists a closed rewrite rule $tDt' \Leftarrow C$ such that:

- $s = Con[t]$ and $s' = Con[t']$ for some context $Con[]$,
- there exists a proof from R for $tDt' \Leftarrow C$,
- C contains only negative rewrite steps,
- for each $s\neg Dt \in C$ we have $sDt \notin \mathcal{M}$.

The definition of stable models has almost faithfully followed the definition of the semantics of TSSs with negative conditions. The only distinction is the first clause in Definition 8, where stable models are closed under context.

The following examples show that there exist CTRSs that allow several stable models, or no stable model at all.

Example 2. Assume two constants a and b, and the rewrite relation $\to$. The CTRS that consists of the rewrite rules $a \to b \Leftarrow b \not\to a$ and $b \to a \Leftarrow a \not\to b$ allows two stable models, namely $\{a \to b\}$ and $\{b \to a\}$.

Example 3. Assume a constants a and the rewrite relation $\to$. The CTRS that consists of the rewrite rule $a \to a \Leftarrow a \not\to a$ does not have a stable model.

3 A Conservative Extension Theorem

3.1 Conservative Extension

Definition 9. Let Σ_0 and Σ_1 be signatures. Their sum (or union) $\Sigma_0 \oplus \Sigma_1$ is *well-defined* if each function symbol and each variable in $\Sigma_0 \cap \Sigma_1$ has the same functionality in both signatures.

We assume two CTRSs R_0 and R_1 over $(\Sigma_0, \mathcal{D}_0)$ and $(\Sigma_1, \mathcal{D}_1)$ respectively, where $\Sigma_0 \oplus \Sigma_1$ is well-defined. Their sum (or union) is denoted by $R_0 \oplus R_1$.

Definition 10. $R_0 \oplus R_1$ is a *conservative extension* of R_0 if for each stable model $\mathcal{M}$ for $R_0 \oplus R_1$, the collection

$$\{sDt \in \mathcal{M} \mid s \text{ a closed term over } \Sigma_0\}$$

is a stable model for R_0.

Not every extension is conservative, which is shown by the following example.

Example 4. Let Σ_0 consist of the constant a, and R_0 of the rewrite rule

$$a \to x \;\Leftarrow\; x \to x.$$

Furthermore, let Σ_1 consist of the constant b, and R_1 of the rewrite rule

$$b \to b.$$

The signatures Σ_0 and Σ_1 are disjoint, but the variable x in the rewrite rule of R_0 does not occur in the left-hand side of the conclusion of this rule. It is easy to see that the rewrite step $a \to b$ is valid in $R_0 \oplus R_1$, but not in R_0. Since a is an original term, it follows that $R_0 \oplus R_1$ is not a conservative extension of R_0.

3.2 Source-Dependency

A variable that occurs free in a rewrite rule is called source-dependent if it is in some sense related to the left-hand side of the conclusion of the rule. Source-dependency will be an important ingredient of the conservativity theorem. In order to conclude that an extension of a CTRS is conservative, we will need to know that the free variables in the original rewrite rules are source-dependent.

Definition 11. For a rewrite rule $p \Leftarrow C$, the *source-dependent* variables in this rule are defined inductively as follows.

- If the left-hand side of p is of the form $s[s_1/x_1, ..., s_m/x_m]$, then all variables that occur free in s are source-dependent.
- If $s[s_1/x_1, ..., s_m/x_m]\, D\, t[t_1/y_1, ..., t_n/y_n]$ is a condition in C, and all free variables in the terms $s, s_1, ..., s_m$ are source-dependent, then all free variables in t are also source-dependent.

A rewrite rule is called source-dependent if all its free variables are so.

Example 5. The conditional rewrite rule in Example 4

$$a \to x \;\Leftarrow\; x \to x$$

is not source-dependent, because the variable x in this rule is not so.

As a next example, we show that so-called context rules are source-dependent.

Example 6. Assume a function symbol f of arity n. The context rules for f are

$$f(\bar{x}_1.y_1, \ldots, \bar{x}_i.y_i, \ldots, \bar{x}_n.y_n) \to f(\bar{x}_1.y_1, \ldots, \bar{x}_i.z, \ldots, \bar{x}_n.y_n) \Leftarrow y_i \to z \quad i = 1, \ldots, n.$$

$y_1, \ldots, y_n$ occur free in the left-hand side of the conclusion, so they are source-dependent. Furthermore, the condition $y_i \to z$ ensures that z is source-dependent.

3.3 The Main Theorem

Theorem 12. *Assume two CTRSs R_0 and R_1 over $(\Sigma_0, \mathcal{D}_0)$ and $(\Sigma_1, \mathcal{D}_1)$ respectively, where $\Sigma_0 \oplus \Sigma_1$ is well-defined. Under the following conditions, $R_0 \oplus R_1$ is a conservative extension of R_0.*

1. *Each rewrite rule in R_0 is source-dependent.*
2. *For each rewrite rule in R_1, the left-hand side $s[s_1/x_1, ..., s_m/x_m]$ of its conclusion contains a function symbol from $\Sigma_1 \backslash \Sigma_0$ in s.*

Proof. This conservativity theorem follows from a similar conservativity theorem in structured operational semantics [17], which implies that an extended TSS $T_0 \oplus T_1$ is conservative over the original TSS T_0 if T_0 and T_1 satisfy the syntactic criteria that we formulated for the CTRSs R_0 and R_1, respectively.

In conditional term rewriting, provability is closed under context, that is, if sDt is provable, then $Con[s]\, D\, Con[t]$ is provable for each context $Con[]$. This is not the case in structured operational semantics. Following [22], we translate the CTRSs R_0 and $R_0 \oplus R_1$ to their equivalent TSSs T_0 and $T_0 \oplus T_1$, by adding context rules. These rules assimilate the notions of provability for CTRSs and TSSs, so that they allow exactly the same stable models. We add explicit context rules to R_0 and R_1 for all function symbols $f \in \Sigma_0$ and $f \in \Sigma_1 \backslash \Sigma_0$, respectively:

$$f(\bar{x}_1.y_1, \ldots, \bar{x}_i.y_i, \ldots, \bar{x}_n.y_n) \to f(\bar{x}_1.y_1, \ldots, \bar{x}_i.z, \ldots, \bar{x}_n.y_n) \Leftarrow y_i \to z \quad i = 1, \ldots, n.$$

In Example 6 it was shown that context rules are source-dependent. Furthermore, the context rules that are added to R_1 all have a function symbol from $\Sigma_1 \backslash \Sigma_0$ in the left-hand side of their conclusions. So the resulting TSSs T_0 and T_1 satisfy the criteria of the conservativity theorem for TSSs. Hence, the TSS $T_0 \oplus T_1$ is conservative over the TSS T_0. Since the TSS T_0 and the CTRS R_0 share the same stable models, and the same holds for $T_0 \oplus T_1$ and $R_0 \oplus R_1$, it follows that the extension $R_0 \oplus R_1$ is conservative over R_0. □

4 Applications to CTRSs of Types III, III$_n$, II and I

A *CTRS of type III* consists of conditional rewrite rules of the form

$$s \to t \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_n \twoheadrightarrow t_n,$$

where $\twoheadrightarrow$ denotes the transitive-reflexive closure of the one-step relation $\to$. It is defined by two rewrite rules, which we add explicitly to each CTRS of type III:

$$x \twoheadrightarrow x \quad \text{and} \quad x \twoheadrightarrow z \Leftarrow x \to y,\ y \twoheadrightarrow z.$$

The first rule is clearly source-dependent. In the second rule, x occurs in the left-hand side of the conclusion, so it is source-dependent. Then y is source-dependent by condition $x \to y$, so z is source-dependent by condition $y \twoheadrightarrow z$.

On a rewrite rule $p \Leftarrow C$ we often see the following three requirements.

A. The left-hand side of p is not a single variable.
B. The free variables in the right-hand side of p also occur as free variables in the left-hand side of p.
C. The variables that occur free in C also occur free in p.

Criteria A and B are natural in the unconditional case, because then they are essential in order to obtain termination. According to Middeldorp [26, page 114], criterion C is often imposed "due to severe technical complications". We left out these criteria, because our results do not require to impose them. Criteria A and B would even be a hindrance, because the two rewrite rules that define the relation $\twoheadrightarrow$ do not satisfy these criteria.

We give an example of an extension of a CTRS of type III, taken from [15], to demonstrate the use of our conservativity result.

Example 7. The CTRS N_0 implements addition on natural numbers. It assumes the sort $\mathbb{N}$ of natural numbers, together with the constant 0, the successor function $S : \mathbb{N} \to \mathbb{N}$, and addition $A : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. Its standard rules are

$$\begin{aligned} A(0, x) &\to x \\ A(S(x), y) &\to S(A(x, y)) \end{aligned}$$

The two rules in the CTRS N_0 are clearly source-dependent, because they do not have conditions, and they satisfy criterion B.

The CTRS N_1 implements the Fibonacci numbers. It assumes the sorts $\mathbb{N}$ and $\mathbb{N} \times \mathbb{N}$, the constant 0, and the functions S and A and $\mathit{Fib} : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$.

$$\begin{aligned} \mathit{Fib}(0) &\to (0, S(0)) \\ \mathit{Fib}(S(x)) &\to (z, A(y, z)) \Leftarrow \mathit{Fib}(x) \twoheadrightarrow (y, z) \end{aligned}$$

The second rule in N_1 is considered difficult, because it does not satisfy criterion B: the variables y and z do not occur in the left-hand side of its conclusion. Nevertheless, since the left-hand sides of the conclusions of the two rules in N_1 contain the fresh function symbol Fib, and since N_0 is source-dependent, Theorem 12 yields that $N_0 \oplus N_1$ is a conservative extension of N_0.

The first rule in N_1 is clearly source-dependent. In the second rule, x occurs free in the left-hand side of the conclusion of the rule, so it is source-dependent. Then condition $Fib(x) \twoheadrightarrow (y, z)$ makes that y and z are source-dependent. So the second rule in N_1 is also source-dependent. Hence, $N_0 \oplus N_1$ is source-dependent, so we can extend this CTRS in a conservative way, as long as we make sure that the rules in the extension have a fresh function symbol in their left-hand sides.

In a *CTRSs of type III$_n$*, conditions are conjuncts of expressions $s \twoheadrightarrow t$ where t is a closed normal form. In particular, terms at the right-hand sides of conditions are closed, so it follows that the only source-dependent variables in a type III$_n$ rule are the ones that occur in the left-hand side of its conclusion. Hence, a rule of type III$_n$ is source-dependent if it satisfies criteria B and C, which together ensure that all free variables in the rule occur in the left-hand side of its conclusion. Hence, we obtain the following corollary.

Corollary 13. *Assume two type III$_n$ CTRSs (Σ_0, R_0) and (Σ_1, R_1). If for each rule in R_0, its free variables all occur free in the left-hand side of its conclusion, and if for each rule in R_1, the left-hand side of its conclusion contains a function symbol from $\Sigma_1 \backslash \Sigma_0$, then $R_0 \oplus R_1$ is a conservative extension of R_0.*

In a *CTRSs of type II*, conditions are conjuncts of expressions $s \downarrow t$, which denote that both s and t reduce to a term u. This can be formulated in type III style: $s \twoheadrightarrow y$ and $t \twoheadrightarrow y$, where y is a fresh variable. Since the y is fresh, again the only source-dependent variables in a type II rule are the variables that occur free in the left-hand side of its conclusion. Hence, a rule of type II is source-dependent if it satisfies criteria B and C.

Corollary 14. *Assume two type II CTRSs (Σ_0, R_0) and (Σ_1, R_1). If for each rule in R_0, its free variables all occur free in the left-hand side of the rule, and if for each rule in R_1, the left-hand side of its conclusion contains a function symbol from $\Sigma_1 \backslash \Sigma_0$, then $R_0 \oplus R_1$ is a conservative extension of R_0.*

Finally, in a *CTRSs of type I*, conditions are conjuncts of expressions $s = t$, which denote that s rewrites to t if the rewrite rules may be applied both from left to right and from right to left. The following example, due to Aart Middeldorp [27], shows that the syntactic criteria from Theorem 12 alone are not sufficient to ensure that an extension of a CTRS of type I is conservative.

Example 8. Let $\Sigma_0 = \{a, b, f\}$ and $\Sigma_1 = \{a, b, c\}$, where a, b, c are constants and f is of arity one. Let R_0 consist of the following rule:

$$f(x) \to x \Leftarrow x = a.$$

Furthermore, let R_1 consist of the following two rules:

$$c \to a$$
$$c \to b.$$

R_0 is source-dependent, and even satisfies criteria A, B, and C. Also, the left-hand side of the conclusions of the rules in R_1 contain the fresh function symbol c. However, $f(b) \to b$ is provable from $R_0 \oplus R_1$, but not from R_0. Since $f(b)$ is an original term, $R_0 \oplus R_1$ is not a conservative extension of R_0.

The conservativity results do extend to type I CTRSs under the assumption that R_0 and $R_0 \oplus R_1$ are *confluent*, i.e., if R_0 (or $R_0 \oplus R_1$) proves $s \twoheadrightarrow t$ and $s \twoheadrightarrow t'$, then there is a u such that R_0 (or $R_0 \oplus R_1$) proves $t \twoheadrightarrow u$ and $t' \twoheadrightarrow u$. Namely, under this assumption, a condition $s = t$ is equivalent to a condition $s \downarrow t$.

Corollary 15. *Assume type I CTRSs (Σ_0, R_0) and (Σ_1, R_1). If for each rule in R_0, its free variables occur free in the left-hand side of its conclusion, and if rules in R_1 have a function symbol from $\Sigma_1 \setminus \Sigma_0$ in the left-hand side of its conclusion, and if R_0 and $R_0 \oplus R_1$ are confluent, then $R_0 \oplus R_1$ is conservative over R_0.*

Example 9. The following modular specification of abstract data types stems from [5]. It starts with a standard semantics for the Booleans, which are built from the constants *true* and *false*, and the conjunction $\wedge$, which is considered modulo commutativity and associativity.

$$\begin{aligned} x \wedge \mathit{true} &\to x \\ x \wedge \mathit{false} &\to \mathit{false} \\ x \wedge x &\to x \\ x \wedge y &\to y \wedge x \\ (x \wedge y) \wedge z &\to x \wedge (y \wedge z) \\ x \wedge (y \wedge z) &\to (x \wedge y) \wedge z \end{aligned}$$

Next, we introduce a collection of names, built from the letters a-z and A-Z. An inductive definition of a binary equality function on names, which decides whether or not two names are equal, would require 53^2 rules. However, by the use of a negative condition we can limit its definition to only two rules.

$$\begin{aligned} \mathit{eq}(v, v) &\to \mathit{true} \\ \mathit{eq}(v, w) &\to \mathit{false} \;\Leftarrow\; \mathit{eq}(v, w) \not\twoheadrightarrow \mathit{true} \end{aligned}$$

Since the left-hand side of the conclusions of both rules contain the fresh function symbol *eq*, and since the rules for the Booleans satisfy criterion B, it follows from Corollary 13 that the combination of the modules of Booleans and names are conservative over the module of Booleans.

Finally, we define the data type of sets of names. They are built from the empty set $\emptyset$, a mapping i from names to singleton sets, and the union $\cup$, which is considered modulo commutativity and associativity. Sets are also considered modulo union with $\emptyset$. The function $\mathit{in}(n, s)$ evaluates to *true* if the name n is in the set s, and to *false* otherwise. The function $x \subseteq y$ evaluates to *true* if x is a subset of y, and to *false* otherwise. The semantics of sets is as follows.

$$\begin{aligned} x \cup \emptyset &\to x \\ x &\to x \cup \emptyset \\ x \cup x &\to x \\ x \cup y &\to y \cup x \\ (x \cup y) \cup z &\to x \cup (y \cup z) \end{aligned}$$

$$
\begin{aligned}
x \cup (y \cup z) &\to (x \cup y) \cup z \\
in(v, \emptyset) &\to false \\
in(v, i(v) \cup x) &\to true \\
in(v, i(w) \cup x) &\to in(v, x) \;\Leftarrow\; eq(v, w) \twoheadrightarrow false \\
x \subseteq x \cup y &\to true \\
x \subseteq y &\to false \;\Leftarrow\; in(v, x) \twoheadrightarrow false \wedge in(v, y) \twoheadrightarrow true
\end{aligned}
$$

The rules for sets all contain fresh function symbols in the left-hand side of their conclusions, and the rules for Booleans and names satisfy criteria B and C. So according to Corollary 13, the combination of the modules of Booleans, names and sets is conservative over the modules of Booleans and names.

5 Applications to Software Renovation Factories

One way of looking at renovating a software system is to consider it as an annotated abstract syntax tree (AST) that needs to be manipulated. This manipulation can be rewriting. This idea underlies the following definition of a software renovation factory (this definition is implicitly assumed in [10] where it is shown how to generate useful rewrite systems from a context-free grammar).

Definition 16. A *software renovation factory* is a set of software renovation assembly lines. A *software renovation assembly line* is an ordered set of (renovation) components. A *(renovation) component* is a higher-order conditional term rewriting system with negative conditions.

We explain what our theorem and software renovation factories have in common (we refer to [9] for a quick introduction to the field of reverse engineering and system renovation). Code that needs to be renovated is first parsed, resulting in an AST. Renovation of code amounts to conditionally rewriting the AST to a desired normal form. Then the AST is unparsed, resulting in renovated code. To renovate code, it is customary to combine existing renovation components. This can be done sequentially in an assembly line by applying components in a fixed order, or simultaneously by taking the sum of components, or as a combination of these two. Our theorem is important for the simultaneous combination of components, which amounts to taking the sum of CTRSs. The question that arises is whether the sum is conservative over the separate components, i.e., is the functionality of an extended component the same as before?

We give an example, and apply our theorem to it, to ensure that the combination of components does not influence the behaviour of the separate components. In the example to follow we use COBOL (this stands for *Common Business Oriented Language*). It is not necessary to have specific COBOL knowledge in order to understand the example, because we look at it from a rewriting perspective.

Example 10. Suppose that a company moves to Japan and that they wish to migrate their mission-critical business software from MicroFocus COBOL to Fujitsu COBOL, since the local programmers are familiar with the latter dialect. To

that end we define a component that migrates MicroFocus specific `78` level constant definitions to the `SYMBOLIC CONSTANT` clause (a Fujitsu COBOL specific feature) of the `SPECIAL-NAMES` paragraph. If there is no `SPECIAL-NAMES` paragraph in the original code, it needs to be created, if there is no `CONFIGURATION SECTION` in the original code, we have to create it, and if there is no `ENVIRONMENT DIVISION` in the original code we have to create it.

The above requirements can be implemented as a CTRS R_0 with five handcrafted rewrite rules (plus hundreds of generated rewrite rules that take care of traversal of the AST, see [10] for details). We define one rule to remove the `78` level constant definitions from the `DATA DIVISION` and to remember the values of the constants. This rewrite rule removes certain nodes from the AST. We have three rewrite rules to create the `SPECIAL-NAMES` paragraph, the `CONFIGURATION SECTION`, and the `ENVIRONMENT DIVISION`, if they were not present. These three rewrite rules add certain nodes to the AST. Finally, we define a rewrite rule to add the constant values to the Fujitsu specific `SYMBOLIC CONSTANT` clause. This rewrite rule adds certain nodes to the AST.

Now suppose that the MicroFocus COBOL code is written by a COBOL programmer who learned COBOL before 1985. Then it is likely that the code will contain implicit scope terminators for `IF` statements instead of explicit ones (before 1985 the explicit scope terminator `END-IF` did not exist). These requirements can be implemented in a CTRS R_1 with four handwritten rewrite rules (plus the generated ones). We have three rewrite rules that handle the three possible ways to implicitly terminate a COBOL `IF` phrase, and one additional rewrite rule to remove separator periods. Further discussion of these rewrite rules is out of scope, since they require a thorough knowledge of the COBOL grammar. We refer to [11] for details on COBOL grammars and to [10] for an elaborate discussion and explanation of the four rewrite rules constituting R_1.

Both rewrite systems serve the purpose of uniformizing the code. It is useful to uniformize code before restructuring, since it will decrease the number of possibilities in rewriting the AST in a later phase. For performance reasons we wish to combine both uniformizing components. Now the question arises whether we can do this safely. Since the conditions of our theorem hold, there is no problem: R_0 is source-dependent, and the rewrite rules in R_1 contain a fresh function symbol at the left-side of their conclusions.

Below we provide an original coding and its rewritten code, to give some idea of the result of rewriting code using combined components. These COBOL programs print the word `HAIKU`. We explain the code fragments, and show where the rewrite rules changed the original code at the left-hand side. The `78` level constant `CON` is removed and put in the `SYMBOLIC CONSTANT` clause of the paragraph called `SPECIAL-NAMES`. Indeed, the appropriate paragraph, section and division are created. Note that the syntax of constants in MicroFocus COBOL and Fujitsu COBOL differs; of course, the rewrite rule takes care of that. The `IF` is terminated by the separator period after the first print statement `DISPLAY 'HAI'`. The rewrite system adds an explicit scope terminator `END-IF` in the code at the right-hand side.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HAIKU.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 VAR-1.
  02 SUB-1 PIC X COMP-X.
78 CON VALUE 1.
  02 SUB-2 PIC X COMP-X VALUE CON.
PROCEDURE DIVISION.
  IF SUB-2 = 1
    DISPLAY 'HAI'.
  DISPLAY  'KU'.
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HAIKU.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
  SYMBOLIC CONSTANT
    CON IS 1.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 VAR-1.
  02 SUB-1 PIC X COMP-X.
  02 SUB-2 PIC X COMP-X VALUE CON.
PROCEDURE DIVISION.
  IF SUB-2 = 1
    DISPLAY 'HAI'
  END-IF.
  DISPLAY  'KU'.
```

References

1. S.F. Allen, R.L. Constable, D.J. Howe, and W.E. Aitken. The semantics of reflected proof. In *Proceedings LICS'90*, pp. 95–197. IEEE Computer Society Press, 1990.
2. J.C.M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In *Proceedings CONCUR'93*, LNCS 715, pp. 477–492. Springer, 1993.
3. J.C.M. Baeten, J.A. Bergstra, J.W. Klop, and W.P. Weijland. Term-rewriting systems with rule priorities. *Theoretical Computer Science*, 67(2/3):283–301, 1989.
4. J.A. Bergstra, J. Heering, and P. Klint, editors. *Algebraic Specification*. ACM Press in cooperation with Addison Wesley, 1989.
5. J.A. Bergstra, J. Heering, and P. Klint. Module algebra. *Journal of the ACM*, 37(2):335–372, 1990.
6. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3), 1984.
7. J.A. Bergstra and J.W. Klop. Conditional rewrite rules: confluence and termination. *Journal of Computer and System Sciences*, 32(3):323–362, 1986.
8. R.N. Bol and J.F. Groote. The meaning of negative premises in transition system specifications. *Journal of the ACM*, 43(5):863–914, 1996.
9. M.G.J. van den Brand, P. Klint, and C. Verhoef. Reverse engineering and system renovation – an annotated bibliography. *ACM Software Engineering Notes*, 22(1):57–68, 1997. http://adam.wins.uva.nl/∼x/reeng/REanno.html.
10. M.G.J. van den Brand, M.P.A. Sellink, and C. Verhoef. Generation of components for software renovation factories from context-free grammars. In *Proceedings 4th Conference on Reverse Engineering*, 1997. To appear. http://adam.wins.uva.nl/∼x/trans/trans.html.
11. M.G.J. van den Brand, M.P.A. Sellink, and C. Verhoef. Obtaining a COBOL grammar from legacy code for reengineering purposes. In *Proceedings 2nd Workshop on the Theory and Practice of Algebraic Specifications*, 1997. To appear. http://adam.wins.uva.nl/∼x/coboldef/coboldef.html.

12. R.L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice Hall, 1986.
13. P.R. D'Argenio and C. Verhoef. A general conservative extension theorem in process algebras with inequalities. *Theoretical Computer Science*, 177:351–380, 1997.
14. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Volume B*, pp. 243–320. Elsevier, 1990.
15. N. Dershowitz, M. Okada, and G. Shivkumar. Confluence of conditional rewrite systems. In *Proceedings CTRS'87*, LNCS 308, pp. 31–44. Springer, 1987.
16. W.J. Fokkink and R.J. van Glabbeek. Ntyft/ntyxt rules reduce to ntree rules. *Information and Computation*, 126(1):1–10, 1996.
17. W.J. Fokkink and C. Verhoef. A conservative look at term deduction systems with variable binding. Logic Group Preprint Series 140, Utrecht University, 1995. http://www.phil.ruu.nl/preprints.html.
18. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings Logic Programming Conference*, pp. 1070–1080. MIT Press, 1988.
19. R.J. van Glabbeek. The meaning of negative premises in transition system specifications II. In *Proceedings ICALP'96*, LNCS 1099, pp. 502–513. Springer, 1996.
20. B. Gramlich. Sufficient conditions for modular termination of conditional term rewriting systems. In *Proc. CTRS'93*, LNCS 656, pp. 128–142. Springer, 1993.
21. B. Gramlich. Generalized sufficient conditions for modular termination of rewriting. *Applicable Algebra in Engin., Comm. and Comput.*, 5:131–158, 1994.
22. J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.
23. S. Kaplan. Conditional rewrite rules. *TCS*, 33(2):175–193, 1984.
24. S. Kaplan. Positive/negative conditional rewriting. In *Proceedings CTRS'87*, LNCS 308, pp. 129–143. Springer, 1987.
25. J.W. Klop. Term rewriting systems. In *Handbook of Logic in Computer Science, Volume I*, pp. 1–116. Oxford University Press, 1992.
26. A. Middeldorp. Modular properties of conditional term rewriting systems. *Information and Computation*, 104(1):110–158, 1993.
27. A. Middeldorp. Personal communication, October 1995.
28. E. Ohlebusch. On the modularity of termination of term rewriting systems. *Theoretical Computer Science*, 136(2):333–360, 1994.
29. G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University, 1981.
30. J.C. van de Pol. Operational semantics of term rewriting with priorities. Logic Group Preprint Series 162, Utrecht University, 1996. To appear in *TCS*.
31. Y. Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.
32. Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25(3):141–143, 1987.
33. Y. Toyama, J.W. Klop, and H.P. Barendregt. Termination for direct sums of left-linear complete term rewriting systems. *JACM*, 34(1):128–143, 1987.
34. C. Verhoef. A general conservative extension theorem in process algebra. In *Proceedings PROCOMET'94*, *IFIP Transactions* A-56, pp. 149–168. Elsevier, 1994.
35. C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nordic Journal of Computing*, 2(2):274–302, 1995.

Technical Reports of the Programming Research Group

Note: These reports can be obtained using the technical reports overview on our WWW site
(`http://www.wins.uva.nl/research/prog/reports/`) or by anonymous ftp to `ftp.wins.uva.nl`,
directory `pub/programming-research/reports/`.

[P9714] M. van den Brand, M.P.A. Sellink, and C. Verhoef. *Control Flow Normalization for COBOL/CICS Legacy Systems.*

[P9713] B. Diertens. *Simulation and Animation of Process Algebra Specifications.*

[P9714] V. Partington. *Implementation of an Imperative Programming Language with Backtracking.*

[P9711] L. Moonen. *A Generic Architecture for Data Flow Analysis to Support Reverse Engineering.*

[P9710] B. Luttik and E. Visser. *Specification ofRewriting Strategies.*

[P9709] J.A. Bergstra and M.P.A. Sellink. *An Arithmetical Module for Rationals and Reals.*

[P9707] E. Visser. *Scannerless Generalized-LR Parsing.*

[P9706] E. Visser. *A Family of Syntax Definition Formalisms.*

[P9705] M.G.J. van den Brand, M.P.A. Sellink, and C. Verhoef. *Generation of Components for Software Renovation Factories from Context-free Grammars.*

[P9704] P.A. Olivier. *Debugging Distributed Applications Using a Coordination Architecture.*

[P9703] H.P. Korver and M.P.A. Sellink. *A Formal Axiomatization for Alphabet Reasoning with Parametrized Processes.*

[P9702] M.G.J. van den Brand, M.P.A. Sellink, and C. Verhoef. *Reengineering COBOL Software Implies Specification of the Underlying Dialects.*

[P9701] E. Visser. *Polymorphic Syntax Definition.*

[P9618] M.G.J. van den Brand, P. Klint, and C. verhoef. *Re-engineering needs Generic Programming Language Technology.*

[P9617] P.I. Manuel. *ANSI Cobol III in SDF + an ASF Definition of a Y2K Tool.*

[P9616] P.H. Rodenburg. *A Complete System of Four-valued Logic.*

[P9615] S.P. Luttik and P.H. Rodenburg. *Transformations of Reduction Systems.*

[P9614] M.G.J. van den Brand, P. Klint, and C. Verhoef. *Core Technologies for System Renovation.*

[P9613] L. Moonen. *Data Flow Analysis for Reverse Engineering.*

[P9612] J.A. Hillebrand. *Transforming an ASF+SDF Specification into a ToolBus Application.*

[P9611] M.P.A. Sellink. *On the conservativity of Leibniz Equality.*

[P9610] T.B. Dinesh and S.M. Üsküdarlı. *Specifying input and output of visual languages.*
[P9609] T.B. Dinesh and S.M. Üsküdarlı. *The VAS formalism in VASE.*
[P9608] J.A. Hillebrand. *A small language for the specification of Grid Protocols.*
[P9607] J.J. Brunekreef. *A transformation tool for pure Prolog programs: the algebraic specification.*
[P9606] E. Visser. *Solving type equations in multi-level specifications (preliminary version).*
[P9605] P.R. D'Argenio and C. Verhoef. *A general conservative extension theorem in process algebras with inequalities.*
[P9602b] J.A. Bergstra and M.P.A. Sellink. *Sequential data algebra primitives (revised version of P9602).*
[P9604] E. Visser. *Multi-level specifications.*
[P9603] M.G.J. van den Brand, P. Klint, and C. Verhoef. *Reverse engineering and system renovation: an annotated bibliography.*
[P9602] J.A. Bergstra and M.P.A. Sellink. *Sequential data algebra primitives.*
[P9601] P.A. Olivier. *Embedded system simulation: testdriving the ToolBus.*
[P9512] J.J. Brunekreef. *TransLog, an interactive tool for transformation of logic programs.*
[P9511] J.A. Bergstra, J.A. Hillebrand, and A. Ponse. *Grid protocols based on synchronous communication: specification and correctness.*
[P9510] P.H. Rodenburg. *Termination and confluence in infinitary term rewriting.*

This article was processed using the LaTeX macro package with LLNCS style