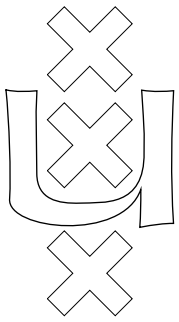
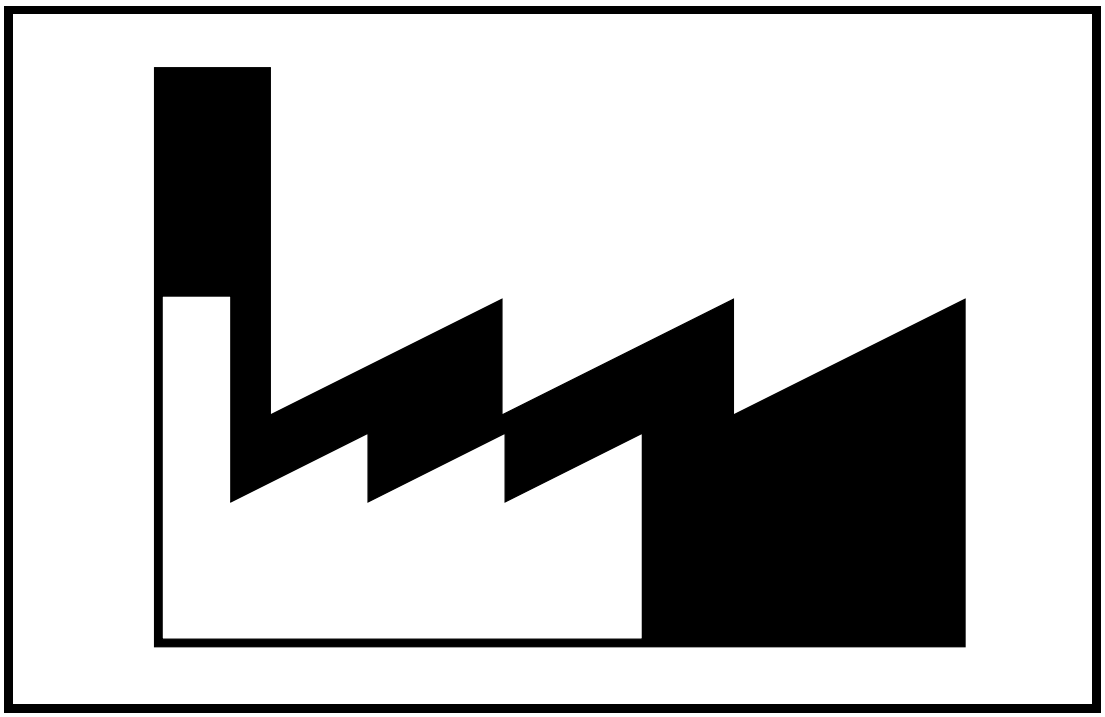


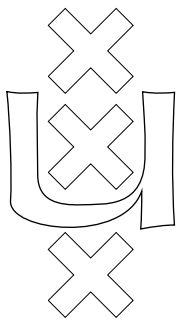
University of Amsterdam
Programming Research Group



Conservative Extension in Positive/Negative Conditional Term Rewriting with Applications to Software Renovation Factories

Wan Fokkink
Chris Verhoef

Report P9802 januari 1998



University of Amsterdam

Department of Computer Science

Programming Research Group

Conservative Extension in Positive/Negative Conditional Term Rewriting with Applications to Software Renovation Factories

Wan Fokkink
Chris Verhoef

Report P9802 januari 1998

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, 1998

Conservative Extension in Positive/Negative Conditional Term Rewriting with Applications to Software Renovation Factories

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 give meaning to conditional term rewriting systems with negative conditions, and to obtain a conservative extension theorem for such systems. These results are useful for development of software renovation factories, and for modular specification of abstract data types.

1 Introduction

There is a strong link between the worlds of conditional term rewriting [29, 7] and of structured operational semantics (SOS) [33]. 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. SOS theorems may apply to the field of conditional term rewriting, and vice versa. In this paper, we will state a theorem for CTRSs that we prove using the above correspondence between TSSs and CTRSs.

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 context rules for all function symbols.

This correspondence was noted, but not exploited, by Groote and Vaandrager [27, 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 [27, 8, 2, 40, 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 a CTRS or TSS with negative conditions, which may contain nonsensical rules such as $a \to b \Leftarrow a \not\twoheadrightarrow b$. We study positive/negative CTRSs [30], which contain negative conditions of the form $s \neg D t$ for relations D, to express that there does not exist a relation sDt. We give meaning to such negative conditions using three-valued stable models [35, 21] from logic programming. Recently, van de Pol [34] used instances of this semantic notion 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 [27, 8, 39, 19, 13]. Groote and Vaandrager [27, 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 [39] proposed more general syntactic criteria, which were later on extended to a setting with inequalities [13]. In [19], 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. Simplifications are that we only treat first-order terms and that we do not allow the possibility that the left-hand side of a rewrite rule in the extension is an original term. We refrain from transposing the conservativity format to CTRSs in full generality, to simplify the presentation, and because it gives us the space to indulge in relevant applications. We refer to [18] where the SOS result has been transposed to CTRSs in the higher-order setting.

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 that each conditional rewrite rule in R_0 is *deterministic* [20]. Furthermore, each conditional rewrite rule in R_1 must contain a fresh function symbol in its left-hand side.

We transpose a conservativity result from the SOS world to the CTRS world. As a consequence, many of the technical notions in this paper have an SOS

counterpart, viz. [19]. As an example the notion of a deterministic CTRS, which stems from [20], was independently discovered in the SOS world [23, 17].

A main aim of this paper is to advocate that the notion of conservative extension is useful not only for TSSs but also for CTRSs. 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 types, 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]. Our result is also applicable in the area of automated software engineering. In this paper we give a formal definition of a software renovation factory that may consist of numerous CTRSs. In order to build such a factory those CTRSs are combined, and then our result comes into play: it gives sufficient conditions so that the functionality of each separate component is not influenced in the presence of other components. This enables reuse of components and a component-based development of such factories. Of course, it cannot be demanded of an operator in a software renovation factory that she remembers our conservativity result upon adding a module to another component. It is, however, possible to implement this check; an automated check on determinism in the SOS world has been incorporated in the tool LATOS [28]. To demonstrate the use of our result, we provide examples from the literature, concerning term rewriting, abstract data types, and software renovation factories.

Related Work The conservativity format for structured operational semantics also has a direct application to term rewriting, as was noticed in [39]. 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. The confluence proof of the TRS consists of about 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, see [39].

In general, studies on modular properties of term rewriting systems 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 often assumed that the signatures of the two rewrite systems are disjoint, and that the variables in a rewrite rule all occur in its left-hand side. CTRSs that satisfy these requirements are automatically within our conservativity format. In this paper it is investigated whether the *full* rewriting relation is preserved for terms over only *one* of the signatures. The signatures of the original CTRS and its extension need not be disjoint.

Toyama [36] showed that confluence is a modular property for TRSs [36], but that in general termination is not [37]. Klop and Barendregt gave a counterexample which shows that completeness is not modular, but Toyama, Klop and Barendregt [38] proved that completeness in combination with left-linearity is modular for TRSs. Ohlebusch [32] 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 [26]). Middeldorp [31] 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 [25] showed that his main results in [26] extend to CTRSs.

2 A Conservative Extension Theorem

2.1 Conditional Rewrite Rules

Definition 1. A *signature* Σ consists of a set of sorts, a countably infinite set $\mathcal{V}$ of variables, and a non-empty set of function symbols f with fixed arities.

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

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

- each variable from $\mathcal{V}$ is a term;
- if function symbol f has arity n, and $t_1, ..., t_n$ are terms over Σ, then $f(t_1, ..., t_n)$ is a term over Σ.

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

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. Following Kaplan [30] we study positive/negative CTRSs, which may contain negative conditions of the form $s \neg D t$, meaning that the relation sDt is not valid.

Definition 3. 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 4. A *(positive/negative) conditional rewrite rule* is of the form $p \Leftarrow C$, where.

- the *conclusion* p is of the form sDt;
- the *conditions* in C are of the form sDt or $s \neg Dt$;

with s and t open terms. A *(positive/negative) conditional term rewriting system* (CTRS) is a set of positive/negative conditional rewrite rules.

We extend the notion of a *deterministic* conditional rewrite rule [20] to the setting with negative conditions.

Definition 5. For a conditional rewrite rule $p \Leftarrow C$, the deterministic variables in this rule are defined inductively as follows.

- All variables in the left-hand side of p are deterministic.
- If sDt is a positive condition in C, and all variables in s are deterministic, then all variables in t are also deterministic.

A conditional rewrite rule is called *deterministic* if all its variables are so.

Definition 6. A *proof* from a CTRS R for a closed rewrite rule $p \Leftarrow C$ (which contains only closed terms) consists of an upwardly branching tree in which all upward paths are finite, where the nodes of the tree are labelled by positive and negative 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.

2.2 Conservative Extension

Definition 7. 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 8. $R_0 \oplus R_1$ is a *conservative extension* of R_0 if for each closed rewrite rule $p \Leftarrow C$ with

- C contains only negative conditions;
- the left-hand side of p is a term over Σ_0;
- there exists a proof from $R_0 \oplus R_1$ for $p \Leftarrow C$;

then there exists a proof from R_0 for $p \Leftarrow C$.

We give an example of an extension that is *not* conservative.

Example 1. 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 $b \to b$. 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.

Note that the CTRS R_0 in Example 1 is not deterministic, because the variable x in the rewrite rule in R_0 is not deterministic (see Definition 5).

Theorem 9. *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. *R_0 is deterministic.*
2. *For each rewrite rule in R_1, the left-hand side of its conclusion contains a function symbol from $\Sigma_1 \setminus \Sigma_0$.*

Proof. We apply induction on the *deterministic distance* of a deterministic variable in a rewrite rule, being the minimal number of steps it takes to deduce that the variable is determinitic in the rewrite rule.

Definition 10. For a deterministic variable x in a rewrite rule $p \Leftarrow C$, its *deterministic distance* $dd(x)$ is defined as follows.

- If x occurs in the left-hand side of p, then $dd(x) \leq n$ for all naturals n.
- If sDt is a condition in C, and $dd(x) \leq n$ for all variables x in s, then $dd(y) \leq n+1$ for all variables y in t.

Finally, $dd(x) = n$ if n is the smallest number such that $dd(x) \leq n$.

Suppose that there exists a proof P from $R_0 \oplus R_1$ for a rewrite rule $\ell \to r \Leftarrow C$, where C consists of negative rewrite steps and ℓ is a term over Σ_0. We show that P is a proof from R_0, by ordinal induction A on the length of P.

Let P have length α, and suppose that we have already proved the case for ordinals smaller than α. The last step in P is constituted by a rewrite rule $\ell' \to r' \Leftarrow C_0$ in $R_0 \oplus R_1$, together with a substitution σ. Since $\sigma(\ell') = \ell$ is a term over Σ_0, ℓ' does not contain a function symbol from $\Sigma_1 \setminus \Sigma_0$. Then Assumption 2 of Theorem 9 implies that $\ell' \to r' \Leftarrow C_0$ is in R_0. According to Assumption 1 of Theorem 9, all variables in $\ell' \to r' \Leftarrow C_0$ are deterministic.

We show that $\sigma(x)$ is a term over Σ_0 for all variables x in $\ell' \to r' \Leftarrow C_0$, by induction B on the deterministic distance of x.

1. $dd(x) = 0$.
   Then x occurs in ℓ'. Since $\sigma(\ell') = \ell$ is a term over Σ_0, clearly $\sigma(x)$ is a term over Σ_0.
2. $dd(x) = k+1$.
   By definition there is a positive rewrite step sDt in C_0 such that x occurs in t, and $dd(y) \leq k$ for all variables y in s. Induction B implies that $\sigma(y)$ is a term over Σ_0 for all variables y in s. Hence, $\sigma(s)$ is a term over Σ_0. Since $\sigma(s)D\sigma(t)$ is proved by a strict sub-proof of P, ordinal induction A implies that R_0 proves this positive rewrite step. In particular, $\sigma(t)$ is a term over Σ_0. Since x occurs in t, clearly $\sigma(x)$ is a term over Σ_0.

Thus, $\sigma(x)$ is a term over Σ_0 for all variables x in $\ell' \to r' \Leftarrow C_0$. In particular, for each condition sDt in C_0, $\sigma(s)$ is a term over Σ_0. Then induction A says that the sub-proof of P for $\sigma(s)D\sigma(t) \Leftarrow C$ is a proof from R_0. Since the last step (with $\ell' \to r' \Leftarrow C_0$ and σ) is in R_0 too, P is a proof from R_0. □

2.3 Three-Valued Stable Models

When there are negative conditions around, it is no longer straightforward to define a sensible rewrite relation. We consider some well-established notions from

logic programming [35, 21]. See [24] for a thorough overview of possibilities to give meaning to negative conditions.

The notion of a three-valued stable model was introduced by Przymusinski [35] in logic programming, based on ideas from Van Emden and Kowalski [15]. It consists of two disjoint collections of positive rewrite steps: intuitively, $\mathtt{T}$ contains the *true* rewrite steps, while $\mathtt{U}$ contains the rewrite steps of which it is *unknown* whether or not they are true. All positive rewrite steps outside $\mathtt{T} \cup \mathtt{U}$ are considered to be *false*. These intuitions are made precise in the definition of a three-valued stable model.

Definition 11. The collections $(\mathtt{T}, \mathtt{U})$ of positive rewrite steps are a *three-valued stable model* for a CTRS R, if the following two requirements hold.

1. The elements of $\mathtt{T}$ 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 $sDt \notin \mathtt{T} \cup \mathtt{U}$.
2. The elements of $\mathtt{T} \cup \mathtt{U}$ 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 $sDt \notin \mathtt{T}$.

Example 2. The CTRS that consists of the rewrite rules $a \to b \Leftarrow a \not\to c$ and $a \to c \Leftarrow a \not\to b$ allows several three-valued stable models: $(\{a \to b\}, \emptyset)$ and $(\{a \to c\}, \emptyset)$ and $(\emptyset, \{a \to b, a \to c\})$.

The following theorem expresses that if $R_0 \oplus R_1$ is a conservative extension of R_0, then each three-valued stable model for R_0 can be obtained by restricting a three-valued stable model for $R_0 \oplus R_1$ to the positive rewrite steps that have a closed original term as left-hand side. The proof is omitted; see [19].

Theorem 12. *Let R_0 be a CTRS over Σ_0. For three-valued stable models $(\mathtt{T}, \mathtt{U})$ for $R_0 \oplus R_1$, we define*

$$\mathtt{T}|\Sigma_0 = \{sDt \in \mathtt{T} \mid s \text{ a closed term over } \Sigma_0\}$$
$$\mathtt{U}|\Sigma_0 = \{sDt \in \mathtt{U} \mid s \text{ a closed term over } \Sigma_0\}.$$

If $R_0 \oplus R_1$ is a conservative extension of R_0, then:

1. *if $(\mathtt{T}, \mathtt{U})$ is a three-valued stable model for $R_0 \oplus R_1$, then $(\mathtt{T}|\Sigma_0, \mathtt{U}|\Sigma_0)$ is a three-valued stable model for R_0;*
2. *each three-valued stable model for R_0 is of the form $(\mathtt{T}|\Sigma_0, \mathtt{U}|\Sigma_0)$, with $(\mathtt{T}, \mathtt{U})$ a three-valued stable model for $R_0 \oplus R_1$.*

According to Przymusinski [35], each CTRS allows a unique three-valued stable model $(\mathtt{T}, \mathtt{U})$ for which the set $\mathtt{U}$ of unknown positive rewrite steps is maximal. Furthermore, Przymusinski showed that this model coincides with the *well-founded* model of Van Gelder, Ross and Schlipf [21]. The next corollary follows from Theorem 12.

Corollary 13. *Let R_0 be a CTRS over Σ_0. If $R_0 \oplus R_1$ is a conservative extension of R_0, and $(\mathtt{T}, \mathtt{U})$ is the well-founded model for $R_0 \oplus R_1$, then $(\mathtt{T}|\Sigma_0, \mathtt{U}|\Sigma_0)$ is the well-founded model for R_0.*

Finally, we mention how two other semantic notions relate to three-valued stable models.

- A *stable model* [22] is a three-valued stable model $(\mathtt{T}, \emptyset)$.
- A CTRS is *complete* [24] if its well-founded model is a stable model.

In our opinion, the well-founded model is a natural model for a positive/negative CTRS. Ideally this model is two-valued, meaning that the CTRS is complete.

Remark. A positive CTRS induces a unique initial model of rewrite steps, which together constitute a minimal model for the CTRS. A rewrite step is in the initial model of a CTRS if and only if there exists a constructive proof for it. This is also the case for the rewrite steps in a three-valued stable model.

For a positive/negative CTRS, Kaplan [30] studied the notion of a *quasi-initial model* of equations (instead of rewrite steps), which are also required to constitute a minimal model for the CTRS. A quasi-initial model is not necessarily unique: the positive/negative CTRS R_0 that consists of the single rule $a \to c \Leftarrow a \neq b$ allows two quasi-initial models, $\{a = c\}$ and $\{a = b\}$. There does not exist a constructive proof for $a = b$; this contrasts with the semantics for positive CTRSs.

Theorem 12 does not hold if we replace "three-valued stable" by "quasi-initial". A counter-example is the extension of R_0 with the constant d and the CTRS R_1 that consists of the rewrite rules $d \to b$ and $d \to c$; this yields a unique quasi-initial model $\{a = b = c = d\}$. However, according to Theorem 9, $R_0 \oplus R_1$ is a conservative extension of R_0. The CTRS R_0 allows a unique three-valued stable model $(\{a \to c\}, \emptyset)$, and $R_0 \oplus R_1$ allows a unique three-valued stable model $(\{a \to c, d \to b, d \to c\}, \emptyset)$.

3 Application to CTRSs of Type III

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

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

. $\twoheadrightarrow$ 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 deterministic. In the second rule, x occurs in the left-hand side of the conclusion, so it is deterministic. Then y is deterministic by condition $x \to y$, so z is deterministic 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. Variables in the right-hand side of p also occur in the left-hand side of p.
C. Variables in C also occur 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 [31, 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 [14], to demonstrate the use of our conservativity result.

Example 3. The CTRS N_0 implements addition on natural numbers. It assumes the constant 0, the unary successor function S, and the binary addition function A. 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 deterministic, because they do not have conditions, and they satisfy criterion B.

The CTRS N_1 implements the Fibonacci numbers. It assumes 0 and S and A, together with the unary Fibonacci function Fib. The rules of N_1 are

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

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 deterministic, Theorem 9 yields that $N_0 \oplus N_1$ is a conservative extension of N_0. The first rule in N_1 is clearly deterministic. In the second rule, x occurs in the left-hand side of the conclusion of the rule, so it is deterministic. Then condition $\mathit{Fib}(x) \twoheadrightarrow (y, z)$ makes that y and z are deterministic. So the second rule in N_1 is also deterministic.

We extend $N_0 \oplus N_1$ with the following two standard module N_2 for a binary equality function eq, which decides whether or not two natural numbers are syntactically equal.

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

Since the left-hand side of the conclusions of both rules contain the fresh function symbol eq, and since $N_0 \oplus N_1$ is deterministic, it follows from Theorem 9 that $N_0 \oplus N_1 \oplus N_2$ is conservative over $N_0 \oplus N_1$.

In a *CTRSs of type III$_n$* (also called 'normal'), 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 deterministic 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 deterministic if all its variables occur in the left-hand side of its conclusion, that is, if it satisfies criteria B and C.

In a *CTRSs of type II* (also called 'join'), conditions are conjuncts of expressions $s \downarrow t$, which denote that s and t reduce to the same term. 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 deterministic variables in a type II rule are the variables that occur in the left-hand side of its conclusion. Hence, a rule of type II is deterministic if it satisfies criteria B and C.

Finally, in a *CTRSs of type I* (also called 'semi-equational'), 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 shows that the syntactic criteria from Theorem 9 are not sufficient to ensure that an extension of a CTRS of type I is conservative.

Example 4. Let $\Sigma_0 = \{a, b\}$ and $\Sigma_1 = \{a, b, c\}$, where a, b, c are constants. Let R_0 consist of the single rule $a \to b \Leftarrow a = b$. Furthermore, let R_1 consist of the two rules $c \to a$ and $c \to b$.

R_0 is deterministic, 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, $a \to b$ is provable from $R_0 \oplus R_1$, but not from R_0. Since a is an original term, $R_0 \oplus R_1$ is not a conservative extension of R_0.

4 Application 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 14. 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 positive/negative CTRS.

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 abstract syntax tree (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 positive/negative 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. The example uses COBOL (Common Business Oriented Language) [1].

Example 5. 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. One of the components that we introduce migrates MicroFocus specific `78` level constant definitions to the `SYMBOLIC CONSTANT` clause (a Fujitsu COBOL specific feature) of the `SPECIAL-NAMES` paragraph. In fact, in the first dialect, the declaration of constants is done in a certain subtree of the AST, and we need to move this information from this subtree to another subtree (the word 'move' is loosely phrased since we also have to modify the syntax of the declarations). An extra problem with the other subtree is that we may need to create it, since it may not yet be present in the original code. The move is implemented in the CTRS R_0 below. It contains five handcrafted rewrite rules that represent the above requirements, plus hundreds of rewrite rules that take care of traversal of the AST, which are generated automatically from the grammar. For detailed information on this generative technology we refer to [10].

Before explaining the rewrite rules, first we focus on notations. The function symbols `f_1` up to and including `f_4` are generated automatically from the COBOL grammar (we renamed them for explanatory reasons). All other expressions that contain numerals are variables; the remaining expressions are terminals (or constant symbols in CTRS terminology). For example, `Ident-div1` is a variable that matches a complete `IDENTIFICATION DIVISION` of a COBOL program; `COMM2*` stands for zero or more COBOL comments; `Sp-name1+` stands for one or more symbolic constants in a `SPECIAL-NAMES` section; and `VALUE` is a terminal representing the COBOL keyword `VALUE`. The five handcrafted rewrite rules are:

```
[1] f_1(COMM1* Ident-div1 Env-div1 DATA DIVISION. COMM2*
    File-sec1 WORKING-STORAGE SECTION. COMM3* Data-desc1* 78 Id1
    Dd-item1* VALUE Id2 Dd-item2*. COMM4* Dd-body1* Data-desc2*
    Link-sec1 Proc-div1)^{ } --->
    COMM1* Ident-div1 f_2(Env-div1)^{Id1 Id2} DATA DIVISION. COMM2*
    File-sec1 WORKING-STORAGE SECTION. COMM3* Data-desc1* Data-desc2*
    Link-sec1 Proc-div1
[2] f_2( )^{Id1 Id2} ---> ENVIRONMENT DIVISION.  f_3( )^{Id1 Id2}
[3] f_3( )^{Id1 Id2} ---> CONFIGURATION SECTION.  f_4( )^{Id1 Id2}
[4] f_4( )^{Id1 Id2} ---> SPECIAL-NAMES. SYMBOLIC CONSTANT Id1 IS Id2.
[5] f_4(Sp-name1+)^{Id1 Id2} ---> Sp-name1+ SYMBOLIC CONSTANT Id1 IS Id2.
```

First notice that R_0 is unconditional: for explanatory reasons we did not impose any conditions on the term rewriting rules of this example. From a software renovation point of view this is unrealistic. However, since we wish to illustrate our conservativity result rather than develop a software renovation factory in this paper, we keep the renovation components as simple as possible. We refer to [10] for a more realistic conditional component, where the condition checks for the presence of the OS/VS COBOL `CURRENT-DATE` special register. If this (in more modern dialects of COBOL obsolete) special register is present, the component renovates the code automatically.

We explain the above rewrite rules in detail. Function symbol `f_1` takes as input a MicroFocus COBOL program and has as output the desired Fujitsu COBOL program, see the end of this paper for a typical example of an original and a rewritten COBOL program. Rule `[1]` defines `f_1`. The argument of `f_1` is a textual representation of an AST containing a pattern that matches a complete COBOL program. It can start with comments, then an `IDENTIFICATION DIVISION` (matched by `Ident-div1`), an `ENVIRONMENT DIVISION` (matched by `Env-div1`), a `DATA DIVISION` (specified in such detail that it matches the `78` level constant definitions that we wish to move to another subtree), and a `PROCEDURE DEFINITION` (matched by the variable `Proc-div1`). The output of `f_1` shows that the first and last divisions are not modified: the parts `COMM1* Ident-div1` and `Proc-div1` are invariant. In the `DATA DIVISION` the `78` level constants are removed. The essential information, residing in variables `Id1` and `Id2`, is stored in memory. This memory is simply a second argument of the function symbols `f_1` – `f_4`, which we call an attribute, denoted using curly braces. The function `f_2` that appears in the output of `f_1` takes care of addition of the constants in the Fujitsu dialect. This is implemented in the rules `[2]`–`[5]`. If there is no `ENVIRONMENT DIVISION` in the COBOL program [1], then the variable `Env-div1` matches an empty subtree, so that rule `[2]` creates a subtree with top-node `ENVIRONMENT DIVISION` and an empty subtree, which is handled by `f_3`. Rule `[2]` passes on the attributes to `f_3`. If the `ENVIRONMENT DIVISION` already exists, then rule `[2]` does not apply; in this case one of the generated rules for `f_2` renames `f_2` into `f_3`, and passes on the attributes to `f_3` as well. We emphasize that whatever the initial situation was, we always end up in the situation that the next function is `f_3`. Rule `[3]` is similar to rule `[2]`; it creates a `CONFIGURATION SECTION` if it is not present. If the `CONFIGURATION SECTION` already exists, then rule `[3]` does not apply; in this case one of the generated rules for `f_3` renames `f_3` into `f_4`, and passes on the attributes to `f_4` as well. Rule `[4]` is also similar to `[2]` and `[3]` in that it creates the `SPECIAL-NAMES` paragraph if it is not present. It also adds the Fujitsu specific `SYMBOLIC CONSTANT` clause and uses the removed variables that reside in the attributes. In the case that there was already such a paragraph, rule `[5]` matches those in the variable `Sp-name1+`, copies them to the output, and adds the `SYMBOLIC CONSTANT` clause.

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 hundreds of 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. We omit these rewrite rules, and refer to [10, p. 150] for an elaborate discussion, explanation and implementation of the four rewrite rules constituting R_1. See [11] for more background on COBOL grammars.

Both R_0 and R_1 serve the purpose of uniformizing the code. It is useful to uniformize code before restructuring, since it decreases the number of possibilities in rewriting the AST in a later phase. For performance reasons we wish to combine both uniformizing components. The question arises whether we can do this safely. This is the case indeed, since R_0 is deterministic, and each rewrite rule in R_1 contains a fresh function symbol at the left-hand side of its conclusion. Such uniformization techniques are common practice in software renovation factories; see [12] for a factory approach where an elimination assembly line for an important class of legacy systems is implemented.

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` and its value `1` are moved from the `DATA DIVISON` to the `ENVIRONMENT DIVISION`. They appear 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'.
```

This transformation can be obtained automatically, using the implementation of the CTRSs R_0 and R_1. The CTRS R_1 has been implemented in [10]; the CTRS R_0 has been defined for the sake of this example, and has been implemented using the same technology.

References

1. ANSI X3.23–1985. Programming Language – COBOL. American National Standards Insitute, Inc, 1985.
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 Working Conference on Reverse Engineering*, pp. 144–155, 1997. 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. Workshops in Computing*, Springer, Available at http://www.springer.co.uk/ewic/workshops/.
12. M.G.J. van den Brand, M.P.A. Sellink, and C. Verhoef. Control flow normalization for COBOL/CICS legacy systems. In *Proceedings 2nd Euromicro Conference on Software Maintenance and Reengineering*. IEEE Computer Society Press, To appear. Available at http://adam.wins.uva.nl/∼x/cfn/cfn.html.
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, M. Okada, and G. Shivkumar. Confluence of conditional rewrite systems. In *Proceedings CTRS'87*, LNCS 308, pp. 31–44. Springer, 1987.
15. M.H. Van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733-742, 1976.
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. Report 95-28, Eindhoven University of Technology, 1995.

18. W.J. Fokkink and C. Verhoef. An SOS message: conservative extension for higher-order positive/negative conditional term rewriting. Report P9715, University of Amsterdam, 1997.
19. W.J. Fokkink and C. Verhoef. A conservative look at operational semantics with variable binding. *Information and Computation*, To appear.
20. H. Ganzinger and U. Waldmann. Termination proofs of well-moded logic programs via conditional rewrite systems. In *Proceedings CTRS'92*, LNCS 656, pp. 430–437. Springer, 1993.
21. A. van Gelder, K. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
22. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings Logic Programming Conference*, pp. 1070–1080. MIT Press, 1988.
23. R.J. van Glabbeek. Full abstraction in structural operational semantics. In *Proceedings AMAST'93*, Workshops in Computing, pp. 77–84. Springer, 1993.
24. 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.
25. B. Gramlich. Sufficient conditions for modular termination of conditional term rewriting systems. In *Proc. CTRS'93*, LNCS 656, pp. 128–142. Springer, 1993.
26. B. Gramlich. Generalized sufficient conditions for modular termination of rewriting. *Applicable Algebra in Engin., Commun. and Comput.*, 5:131–158, 1994.
27. J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.
28. P.H. Hartel. LATOS – a lightweight animation tool for operational semantics. Report DSSE-TR-97-1, University of Southampton, 1997. Available at http://www.dsse.ecs.soton.ac.uk.
29. S. Kaplan. Conditional rewrite rules. *Theoret. Comput. Sc.*, 33(2):175–193, 1984.
30. S. Kaplan. Positive/negative conditional rewriting. In *Proceedings CTRS'87*, LNCS 308, pp. 129–143. Springer, 1987.
31. A. Middeldorp. Modular properties of conditional term rewriting systems. *Information and Computation*, 104(1):110–158, 1993.
32. E. Ohlebusch. On the modularity of termination of term rewriting systems. *Theoretical Computer Science*, 136(2):333–360, 1994.
33. G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University, 1981.
34. J.C. van de Pol. Operational semantics of term rewriting with priorities. *Theoretical Computer Science*, To appear.
35. T.C. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13(4):445–463, 1990.
36. Y. Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.
37. Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25(3):141–143, 1987.
38. 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.
39. C. Verhoef. A general conservative extension theorem in process algebra. In *Proceedings PROCOMET'94*, *IFIP Transactions* A-56, pp. 149–168. Elsevier, 1994.
40. C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nordic Journal of Computing*, 2(2):274–302, 1995.

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

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/`.

[P9801] P. Klint and C. Verhoef. *Evolutionary Software Engineering: A Component-based Approach.*

[P9726] J.A. Bergstra and A. Ponse. *Grid Protocol Specifications.*

[P9725] J.A. Bergstra and A. Ponse. *Process Algebra Primitives for File Transfer.*

[P9724] J.A. Bergstra and A. Ponse. *Process Algebra with Four-Valued Logic.*

[P9723] J.A. Bergstra and A. Ponse. *Kleene's Three-Valued Logic and Process Algebra.*

[P9722] J.A. Bergstra and A. Ponse. *Bochvar-McCarthy Logic and Process Algebra.*

[P9721] M.P.A. Sellink and C. Verhoef. *Reflections on the Evolution of COBOL.*

[P9720] M. van der Graaf. *A Specification of Box to HTML in ASF+SDF.*

[P9719] M. van den Brand, M.P.A. Sellink, and C. Verhoef. *Current Parsing Techniques in Software Renovation Considered Harmful.*

[P9718] M. de Jonge. *Reuse of ASF+SDF Specifications by means of Renaming.*

[P9717] A. Sellink (Ed.). *Participants Proceedings of the 2nd International Workshop on the Theory and Practice of Algebraic Specifications.*

[P9717] E. Visser. *From Context-free Grammars with Priorities to Character Class Grammars.*

[P9716] H.M. Sneed. *Dealing with the Dual Crisis — Year 2000 and Euro — What Reverse Engineering can do to Help.*

[P9715] W. Fokkink and C. Verhoef. *An SOS Message: Conservative Extension in Higher Order Positive/Negative Conditional Term Rewriting.*

[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.*

[P9712] 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.*

[P9708] E. Visser. *Character Classes.*

[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.*