Structured Contextual Rewriting[1]

Qian, Zhenyu

FB 3 Mathematik/Informatik, University of Bremen
D-2800 Bremen 33, Fed. Rep. of Germany
UUCP: unido!ubrinf!qian

Abstract

In this paper, we develop a mechanism, which we call *structured contextual system*, (SCS for short,) to deal with some non-finitely-based algebraic specifications. The sufficient condition for confluence and termination of this kind of systems is also considered, based on a generalization of the appoach by O'Donell.

1. Introduction

Up to now much of work has been done on the specification of abstract data types in the algebraic approach ([ADJ 78]) and on its implementation by term rewrite systems (short: TRS) ([De 85] [HO 80]). This work provides a sound foundation for the definition of semantics of specification languages, (e.g. [BG 79]), for theorem-proving ([De 83]), for program development ([CIP-L] [De 83]).

One restriction of the up to now research is that most of the work, especially that discussing operational properties of TRS's, has assumed that the set of rewrite rules with variables is finite, i.e. the TRS's are finitely based, although this is not necessary from the viewpoint of the algebraic and operational semantics. In fact, some abstract data types can only be specified by an infinite set of equations with variables, i.e. they are non-finitely-based (cf. [Ta 79], [DMT 85] and the following sections). We define a mechanism, which we call *structured contextual system*, to describe such non-finitely-based specifications. The description is an extension of the conventional TRS's. In addition, we can use this method to express conditional rewrite systems (see [Ka 83]), and the expressions suggest a simple implementation in terms of their syntactical structure.

In order to consider the operational properties of an SCS, we generalize the sufficient condition for confluence and termination proposed by [Ro 73] and [O'D 77]. To be able to deal with more real conventional TRS's, first we extend the notion of non-overlapping by a new notion called *non-interfering*, which forms the central part of a weaker sufficient condition for confluence and termination of conventional TRS's. Then we apply the fundamental idea to SCS. We feel that the extension is intuitively clear and understandable.

[1]This work is supported by the Commission of the European Community under the ESPRIT Programme in the PROSPECTRA Project. ref#390

This paper is a compact form of [Qi 86a], [Qi 86b], [Qi 86c], where proofs for the new theorems and more motivating examples can be found. The work coincides with the approach of Program Development by Specification and Transformation (cf. [PROSPECTRA]), and can be used to help in designing a transformation calculus and a transformation language (cf. [K-B 86]). In Section 2, some widely used basic notations and results are repeated briefly. Section 3 clarifies the definition of SCS. In Section 4 we consider the generalization of sufficient condition for confluence and termination of conventional TRS's, and use the idea to explain such a condition for SCS. In Section 5, 6, we consider related work and further research.

2. Notations and Some Known Results

In this section we summarize some of the notions and known results developed in [ADJ 78] [De 85] [O'D 77] [HO 80], which are necessary in our paper.

We use $<S,\Sigma>$ (or short: Σ) to denote a signature with sort set S and finite function family Σ: $<\Sigma_{w,s}>_{w\in S^*,s\in S}$. An S-indexed family of denumerable variable sets will be denoted as X: $<X_s>_{s\in S}$. T_Σ denotes $<T_{\Sigma,s}>_{s\in S}$, where $T_{\Sigma,s}$ denotes the set of those terms constructed only by functions in Σ and has an outermost function of the arity w->s. $T_\Sigma(X)$ denotes the family of sets of terms possibly with variables. For a term t, Var(t) denotes the set of all variables in t. If Var(t)=ϕ, t is a ground term. A specification $<S,\Sigma,E>$ contains a signature and a denumerable set E of equations in $T_\Sigma(X)$. An equation is expressed as t=t'. Note that the syntactical identity is also expressed as t=t', but no ambiguity is possible because of the different contexts.

The ancestor relation $\leq$ (ancestor of) is an ordering on $\mathbf{N}^*$ satisfying $\omega_1 \leq \omega_2$ iff $\exists\, \omega'$, $\omega_1\omega' = \omega_2$. If neither of them is an ancestor of the other, then $\omega_1 \perp \omega_2$ (independence). $\perp\Omega$ means $\forall\, \omega,\omega' \in \Omega$, $\omega \perp \omega'$ or $\omega=\omega'$.

For an occurrence $\omega\in\Omega$ of a term t, t/ω denotes the subterm cut from t at ω. $t{:}\omega$ denotes the outermost function of t/ω.

If we consider that the equations of a specification are oriented in a proof system, then we call it a term rewrite system (short: TRS). In this case we call each element of E a rule. t=t' is called *steady* iff Var(t)$\supseteq$Var(t'), and *left-linear* iff each variable ocurrs at most once in t. Two rules $u_i=\bar{u}_i$, i=1,2, are called *non-overlapping* iff for their instances $t_i=\bar{t}_i$, i=1,2, if $\exists\, \omega$, $t_1/\omega=t_2$, then $\exists\, \omega'$, $\omega' \leq \omega$ and $u_1{:}\omega'$ is a variable.

Given a term t. Assume $\perp\Omega$, $\Omega\subseteq$Occ(t). Parallel substitution in term t at Ω by t' is denoted as t[Ω<-t']. For short we write t[c<-t'] instead of t[t^{-1}(c)<-t'] for $c\in X\cup\Sigma$. An assignment σ replaces each variable with some term. We define a quasi-ordering $\leq$ in $T_\Sigma(X)$ in terms of assignment: $u\leq t$ iff $\exists\, \sigma$, $\sigma u = t$. We define: $t\equiv t'$ (*variable-renaming*) iff $t\leq t'$ and $t'\leq t$.

Let $\rightarrow$ be the reduction relation on terms in the common sense. A normal form of a term t is a term t' such that $t \rightarrow^* t'$ and there is no t'' such that $t' \rightarrow t''$. A TRS is called *confluent* iff for terms t, t_1, t_2, if $t \rightarrow^* t_1$ and $t \rightarrow^* t_2$, then $\exists$ t', $t_1 \rightarrow^* t'$ and $t_2 \rightarrow^* t'$. A TRS is called *terminating* iff any sequence of its reductions will terminate. Confluence or termination w.r.t. ground terms means that we consider the TRS only having ground rules. We can assume that a TRS contains only ground rules even if it has rules with variables. In fact, we can imagine that a rule with variables represents all its possible ground instances if the TRS is generated.

We discuss in this paper only the following *full replacement strategy*: Reduce a term at all its redexes from innermost to outermost. Repeat the above process as a whole until a normal form is found.

A TRS is called *relatively terminating* if any term will reach a normal form after a finite number of reductions, whenever this term has a normal form. Since we shall mainly discuss reductions of ground terms under the full replacement strategy, in this paper confluence means confluence w.r.t. ground terms, termination means relative termination under the full replacement strategy w.r.t. ground terms.

3. Structured Contextual System

The notion of structured contextual system (SCS for short) is developed to describe some non-finitely-based equational systems or TRS's, without the need to introduce any auxiliary symbols. It can also be used to specify the algebras given by some conditional equational systems.

3.1. Motivations

Example 1 ([DMT 85])

Let f, g and h be 1-ary functions and x be a variable. Equation set $\{fg^ihx = fg^jhx,\ i,j \in \mathbf{N}\}$ is non-finitely-based.

Example 2

Let assume a function set {f(_,_),h(_),a,b,c}. An algebra is specified by the equations { h(t) = h(t') }, where t' corresponds to t such that all occurrences of symbol a in t are replaced by symbol b in t'. E.g. h(f(f(c,a),a)) = h(f(f(c,b),b)) is an equation in this set. This specification is also non-finitely-based. A real example of this algebra is the transformation of all function declarations and their calls in a program fragment into procedure ones. (See [BaWö 82] and [Qi 86b].)

3.2. Informal Explanation of SCS

Before we make any formal definitions, we try first to explain the notions by an example of SCS specifying the algebra given in Example 1.

Example 3

DMT'
sorts: s
subsorts: $s', \bar{s}' \subseteq s$
opns: $f, g, h : s \rightarrow s$
eqns: **for all** $x{:}s;\ (v, \bar{v}){:}(s', \bar{s}')$

$$f(g(v)) = f(\bar{v})$$

where $s{:}h(x) = \bar{s}'{:}h(x)$

$$\begin{aligned} g(v) &= \quad g\bar{v} \\ g(v) &= \quad \bar{v} \end{aligned}$$

fig.1

Explanations: Compared with the traditional algebraic specification, we have the following extensions:

1. s' and $\bar{s}'$ are two sorts related to the original sort s.
2. v and $\bar{v}$ are two associated variables of two associated sorts s' and $\bar{s}'$, resp. x is a variable in the conventional sense.
3. The legal terms of s' is defined by the left column following s' in the where-part, and those of $\bar{s}'$ by the right column following $\bar{s}'$.
4. The mechanism for computation is unification/co-generation. E.g. if we are going to prove DMT' $\vdash f(g(g(h(a)))) = f(g(h(a)))$ (*), we do the following:
   a) We attempt to use $f(g(v)) = f(\bar{v})$ to unify (*), and find out that (*) is true in DMT' if we can find associated pair of terms (g(h(a)), g(h(a))) for $(v, \bar{v})$ in S' and $\bar{S}'$.
   b) For this purpose, we use $g(v) = g(\bar{v})$ in where-part to confirm (g(h(a)), g(h(a))), and find out that the above is true if (h(a), h(a)) is a possible instantiated pair for $(v, \bar{v})$ in S' and $\bar{S}'$.
   c) Furthermore, by h(x) = h(x) in where-part, we know that we will succeed if we instantiate x with a.

What we have achieved by SCS is that some contextual properties appear syntactically.

3.3. Definition of SCS

Definition 1 (TGS)

A term generation system (short: TGS) on a given signature SIG:$<S_b,\Sigma>$, which is called base signature of this TGS, is a triple $<S,V,P>$ such that

1. each $s \in S$ corresponds to a base sort, denoted as $base(s) \in S_b$;
2. $V = \langle V_s \rangle_{s \in S}$ is a family of denumerable sets of distinct variables. We denote $v \in V_s$ as $v{:}s$;
3. $P \subset S x T_\Sigma(V)$ is a finite set of productions, which are denoted as $\langle s{:}u \rangle$ or $s{:}u$ with $u \in T_{\Sigma, base(s)}(V)$.

A TGS in the SCS of Example 3 contains the production set $\{S'{:}h(x), S'{:}g(v), S'{:}g(v')\}$ over the base signature $(\{S\}, \{f,g,h,a\})$ with variable family $(\{x,...\}_S, \{v,v'...\}_{S'}, \{\bar{v},...\}_{\bar{S}'})$. The other TGS contains the production set $\{S'{:}h(x), S'{:}g(\bar{v})\}$.

Remark: Two productions differ, if they are not identical. That is, not identical $u,u' \in T_\Sigma(V)$ constitute different productions $\langle s{:}u \rangle$, $\langle s{:}u' \rangle$, even when they are equal under a variable-renaming.

Note that the base signature can be considered as a TGS over itself. We denote this TGS as SIG(V) for a base signature SIG. From now on we assume all TGS's in a common context are based on the same signature.

Definition 2 (production)

Let $A = \langle S,V,P \rangle$ be a TGS on $\langle S_b, \Sigma \rangle$ and $t \in T_\Sigma(V)$. Let $v \in Var(t)$ and $v{:}s$ for some $s \in S$. Let $p = \langle s{:}t_p \rangle \in P$. Then

(i) $t \twoheadrightarrow_{v,p} t[v \leftarrow t']$ (t produces $t[v \leftarrow t']$ at v by p) iff $\exists\, t' \in T_{\Sigma, base(s)}(V)$ such that
 a) $t' \equiv t_p$
 b) $Var(t) \cap Var(t') = \phi$

(ii) $\twoheadrightarrow_A$ denotes $\twoheadrightarrow_{v,p}$ for some v and p in A. $\twoheadrightarrow_A^*$ denotes the reflexive and transitive closure of $\twoheadrightarrow_A$.

Remark: Point a) and b) say that a new term is generated without variable conflicts.

Definition 3

Let $A = \langle S,V,P \rangle$ be a TGS. Its language is defined as $T_A = \{t \in T_\Sigma \mid \exists\, v \in V, v \twoheadrightarrow_A^* t\}$. For $t' \in T_\Sigma(V)$, we define the language of $A(t')$ as $T_A(t') = \{t \in T_\Sigma \mid t' \twoheadrightarrow_A^* t\}$. Note that variables only serve as a place-holder for the terms of a certain sort. If $t' = v$ with $v{:}s$, we can write $A(s)$ for $A(v)$, $T_A(s)$ for $T_A(v)$. We assume here that TGS's have no productions that can never be used in the language generation. Note that we always have $T_A(s) \subset T_{\Sigma, base(s)}$ for TGS A on $\langle S_b, \Sigma \rangle$.

It is obvious that the TGS of any column in the equation part of Example 3 produces the language $\{fg^ihx \mid x \in T_{\{f,g,h,a\}}, i \in \mathbf{N}\}$.

Definition 4

Let V be a variable family indexed by sort sets S. A relation m on V is said to be sort-preserving iff for $(v_0, v'_0), (v_1, v'_1) \in m$, $v_0, v_1{:}s \in S$ iff $v'_0, v'_1{:}s' \in S$.
For a sort-preserving relation m, an induced relation im on S is defined

by: $(s,s')\in im$ iff $\exists$ $(v,v')\in m$ with $v{:}s$, $v'{:}s'$.

Note that the renaming of terms is sort-preserving.

Definition 5 (ATGS)

Let $A=<S,V,P>$, $\bar{A}=<S,V,\bar{P}>$ be two TGS's. Assume that we have a sort-preserving bijection mv on V, called <u>variable association</u>, and a bijection mp: $P\mathrm{x}\bar{P}$, called <u>production association</u>. An <u>associated TGS</u> (ATGS for short) is a 4-tuple $<A,\bar{A},mv,mp>$ such that

1. if $(<s{:}t>,<\bar{s}{:}\bar{t}>)\in mp$ for some t and $\bar{t}$, then for the induced map imv of mv, $(s,\bar{s})\in imv$, and
2. not both $(<s{:}u>,p)\in mp$ and $(p',<s{:}t>)\in mp$ for any s, u, t, p and p'.

<u>Remarks</u>:

1. The introduction of the above two associations is based on the remark made after Definition 1 and the fact that V is a family of denumerable sets. Otherwise the notion of bag (cf. [MW 85]) should be introduced.
2. The Point.2 means that terms of a sort can not be produced in both TGS's.

In Example 3, the mp is expressed by = The occurrences of x on both sides of $(<s'{:}hx>,<\bar{s}'{:}hx>)$ (abbreviated as hx=hx there) should be understood as two variables of two sorts with equivalent sets of terms, which are well-formed terms in the sense of conventional algebraic specification. The sorts for $fgv=f\bar{v}$ are anonymous because no variables are declared explicitly for them. All these notations will be formally clarified later on.

Definition 6 (associated productions)

Let $<A,\bar{A},mv,mp>$ be an ATGS with $A=<S,V,P>$, $\bar{A}=<S,V,\bar{P}>$. Let $(<s{:}t'>,<\bar{s}{:}\bar{t}'>)\in mp$. Let u, $\bar{u}\in T_\Sigma(V)$. For $(v,\bar{v})\in mv$, $v{:}s$, $\bar{v}{:}\bar{s}$, let $t=u[v<-i(t')]$, $\bar{t}=\bar{u}[\bar{v}<-\bar{i}(\bar{t}')]$ under two variable-renamings i and $\bar{i}$. We say that $u\text{-}\!\gg_{v,<s:t'>}t$ and $\bar{u}\text{-}\!\gg_{\bar{v},<\bar{s}:\bar{t}'>}\bar{t}$ are <u>associated</u> iff for any $v'\in Var(t')$ $\bar{v}'\in Var(\bar{t}')$, if $(v',\bar{v}')\in mv$, then $(i(v'),\bar{i}(\bar{v}'))\in mv$.

<u>Remark</u>: The above definition requires that i and $\bar{i}$ are so determined that mv between t and $\bar{t}$ respects the original mv between variables in t' and $\bar{t}'$. Note that the productions may still be associated even if $v\notin Var(u)$ or $\bar{v}\notin Var(\bar{u})$.

Examples of associated production steps are those production steps given in the Point. a, b, c, of the explanations made after Example 3.

Definition 7

(i) Let $AA=<A,\bar{A},mv,mp>$ be an ATGS. Assume that at least one of $v\in Var(t_1)$, $\bar{v}\in Var(\bar{t}_1)$ is true. Let $(v,\bar{v})\in mv$ and $(p,\bar{p})\in mp$. $<t_1,\bar{t}_1>\text{-}\!\gg_{v,p,\bar{v},\bar{p}}<t_2,\bar{t}_2>$, or $<t_1,\bar{t}_1>\text{-}\!\gg_{AA}<t_2,\bar{t}_2>$, pronounced as <u>co-produces</u>, iff $t_1\text{-}\!\gg_{v,p}t_2$ and $\bar{t}_1\text{-}\!\gg_{\bar{v},\bar{p}}\bar{t}_2$ are associated.
We use $\text{-}\!\gg^{*}_{AA}$ to denote the reflexive and transitive closure of $\text{-}\!\gg_{AA}$.

(ii) The language of AA is $G_{AA}=\{<t,\bar{t}> \mid t,\bar{t}\in T_\Sigma, \exists\ (v,\bar{v})\in mv: <v,\bar{v}> \twoheadrightarrow^*_{AA} <t,\bar{t}>\}$.
We call elements of G_{AA} ground equations. We denote these equations as $<t,\bar{t}>$.

In Example 3, we have $<v,\bar{v}> \twoheadrightarrow^* <g^ihx,g^jhx>$, where $i\geq j$ and $x\in T_{\{f,g,h,a\}}$.

Remark: In (ii) we have in fact defined all the possible ground instances represented by an ATGS. That is why we call them ground equations.

If we want to get the set of the equations of certain form, we can indicate the main equation (or mother equation in [K-B 86]) in an ATGS as a start point for the production of ground equations. Note that from now on we define the notions in the context of TRS. We can also do the same for equational system.

Definition 8 (SCR)
Let AA be an ATGS. Assume $(v,\bar{v})\in mv$ for $v{:}s$, $\bar{v}{:}\bar{s}$ with $base(s)=base(\bar{s})$. A structured contextual rule (short: SCR) w.r.t. $(s,\bar{s})$ is denoted as $AA(s,\bar{s})$ or $AA(v,\bar{v})$. Its language is defined by: $G_{AA}(s,\bar{s})=G_{AA}(v,\bar{v})=\{<t,\bar{t}> \mid t,\bar{t}\in T_\Sigma, <v,\bar{v}> \twoheadrightarrow^*_{AA} <t,\bar{t}>\}$.

In the above case, if $<s{:}t'>$, $<\bar{s}{:}\bar{t}'>$ are the only productions for sorts s and $\bar{s}$, resp., and they will never be used after the first application, we can denote $AA(s,\bar{s})$ as $AA(t',\bar{t}')$. In Example 3, $fgv = f\bar{v}$ correspond to t' and $\bar{t}'$, resp. Its language $T_{DMT'} = \{<fg^ihx,fg^jhx> \mid i>j,\ x\in T_{\{f,g,h,a\}}\}$. The notion of SCR simplifies the form of the transformational language suggested in [K-B 86].

We assume that $AA(v,\bar{v})$ contains no production associations which are never used in the generation of its language.

Definition 9 (SCS)
A structured contextual TRS (short: SCS) is a triple $<S_b,\Sigma,E>$ such that E is a finite set of SCR's.

The language of the above E can be denoted as $G_E=\bigcup_{R\in E} G_R$, which is an enumerable set of ground rules. The semantics of $<S_b,\Sigma,E>$ is the semantics of $<S_b,\Sigma,G_E>$.

3.4. SCS as Generalization of Conventional TRS

Assume that two ATGS AA, BB have the same sorts, functions and variable set. BB is called a base ATGS of AA if $G_{BB}\subseteq G_{AA}$. Consider two ATGS's AA_i, $i=1,2$, which have a common base ATGS BB. Then we may hope that we do not have to describe BB two times.

In an SCS E, all SCR's have a common base, which specifies all legal terms on the base signature. We assign a special S_b-induced family of variable sets X such that the following holds:

1. If $x \in X$ and $x{:}s$, then $x'{:}s$ implies $x' \in X$.
2. For $(x,\bar{x}) \in mv$, $x \in X$ iff $\bar{x} \in X$. Then we can let mv_X denote the part of mv that is on X.
3. Let $base(s)=base(\bar{s})=s_b$. Let $(x_i,\bar{x}_i) \in mv_X$, $x_i{:}s_i$ and $\bar{x}_i{:}\bar{s}_i$ with $base(s_i)=s_{ib}=base(\bar{s}_i)$, for $1 \le i \le n$. For each $f \in \Sigma$ with arity $s_{1b}...s_{nb} \to s_b$, there is $(<s{:}f(x_1,...,x_n)>,<\bar{s}{:}f(\bar{x}_1,...,\bar{x}_n)>) \in mp$, and there are $x,\bar{x} \in X$ with $x{:}s$, $\bar{x}{:}\bar{s}$.

Due to the above properties of X, we can simplify the notations by using only half of X, half of the sort set for X and considering mv_X as identical relation. Then for any sorts with variables in X, $(<s{:}f(x_1,...,x_n)>,<\bar{s}{:}f(\bar{x}_1,...,\bar{x}_n)>)$ can be denoted as $(<s{:}f(x_1,...,x_n)>,<s{:}f(x_1,...,x_n)>)$. $<SIG(X),SIG(X),1_X,1_P>$ is a base ATGS for all SCS's on the base signature SIG and denotes nothing else than a conventional TRS which has no rewrite rules. In general, we do not have to specify the possible co-productions of variables in X. One of such variable x is given in Example 3. In this case, the SCR's of the form $AA(t,\bar{t})$ with $Var(t) \subseteq X$, $Var(\bar{t}) \subseteq X$ are conventional rewrite rules. This means that SCS contains the conventional TRS as a special case.

If we not only allow the ground rules as the language of an SCS, but also rules with variables in X, using G(X) to denote this language, then we constitute a conventional TRS $<S_b,\Sigma,G(X)>$ with an infinite (but enumerable!) rule set. In this sense, we have achieved a description of some non-finitely-based TRS. In addition, all the discussions for SCS are valid for conventional TRS's.

The variables in X are called terminal, others, non-terminal.

4. Sufficient Condition for Confluence and Termination

4.1. Generalization of the Classical Sufficient Condition

Later on in this paper, "classical approach" means the approach of [O'D 77], "conventional rule" means rewrite rule without non-terminal variables.

The main part in the classical sufficient condition for confluence and termination by O'Donell (cf. [O'D 77]) is non-overlapping of the rewrite rules. The following example shows that some TRS's are intuitively confluent, although the condition of non-overlapping fails.

Example 4

Let a TRS contain rules $h(f(x,a))=h(f(x,b))$, $f(c,x)=f(d,x)$ on functions { h(_), f(_,_), a, b, c, d}. It does not satisfy the non-overlapping property because for the instances $h(f(c,a))=h(f(c,b))$, $f(c,a)=f(d,a)$, $h(f(x,a)){:}1=f(\_,\_) \in \Sigma$. However, these two rules will not destroy confluence, as illustrated by the following diagram:

$$\begin{tikzcd}
& h(f(c,a)) \arrow[dl] \arrow[dr] & \\
h(f(d,a)) \arrow[dr] & & h(f(c,b)) \arrow[dl] \\
& h(f(d,b)) &
\end{tikzcd}$$

fig.2

This example represents a large class of rewritings, where real subterm changes of two rules happen only within a common context, but these changes are independent. The TRS is overlapping because of the common context. In the above example the function f is a context, within which a and c can be rewritten into b and d, resp.

In order to develop a suficient condition considering the above example, we generalize the classical condition by replacing the notion of non-overlapping with non-interfering. Including non-overlapping as a special case, it allows two rules to overlap, but forbid them to change dependent subterms. In the following, we explain this notion informally by examples. We claim that any non-interfering TRS, which is, in addition, left-linear, steady, consistent (for these notions cf. [O'D 77]), is confluent and terminating. For more details one should refer to [Qi 86c].

More exactly speaking, two rules $t_1=\bar{t}_1$, $t_2=\bar{t}_2$ are said to be non-interfering iff if they are both applicable to (i.e. unifiable with) a term t, then the following holds:

1. They do not change dependent subterms of t.
2. Each of them should be still applicable to the term obtained after application of the other one to t.
3. Since both rules are applicable to one term, we can assume that u is in t_1 the subterm that is unifiable with t_2. We require that if a variable x in u corresponding to a by $t_2=\bar{t}_2$ changed subterm in t_2, then each occurrence of x in $\bar{t}_1$ should be in a subterm unifiable with t_2.

Example 5

a) The rule set {h(f(x,a))=g(f(x,b),f(x,c)), f(d,x)=f(e,x)} is non-interfering. E.g. the rewriting of term h(f(d,a)) will terminate and the unique normal form is g(f(e,b),f(e,c)), no matter in which order the rules are applied.

b) The rule set {f(x,c)=f(x,f(x,d)), f(a,x)=f(b,x)} is non-interfering.

c) The rule set {h(f(x,a),y)=g(f(x,g(y))), f(c,x)=f(d,x), b=e} is non-interfering.

d) The rule set { h(f(c,a))=g(f(c,b),f(c,b)), f(c,x)=f(d,x) } is not non-interfering, since the application of the second rule to term h(f(c,a)), to which both rules are applicable, yields term h(f(d,a)), to which the first rule is no longer applicable.

e) The rule set {h(f(x,a))=g(f(x,b),x), f(c,x)=f(d,x)} is not non-interfering, since the second occurrence of x on the right-hand side of the first rule is not in a subterm that is unifiable with the second rule. (Compare with the Example b.)

f) The rule set {h(f(x,a))=g(f(x,g(x))), f(c,x)=f(d,x)} is not non-interfering for the same reason as above. (Compare with the Example c.)

4.2. Sufficient Condition for Confluence and Termination of SCS

We first claim that an SCS is confluent and terminating if it is left-linear, steady, consistent and non-interfering. We omit the definitions of the other notions in the context of SCS and give a very brief informal explanation of the notion of non-interfering. For more details about non-interfering and other notions, one may refer to [Qi 86c].

Compared with conventional TRS's, we have in an SCS a where-part, which defines the possible pairs of terms instantiated for associated variables on both sides of a rule. Since the structures of possible instances are given by an SCS, we do not feel much more difficult than in the conventional case to formulate the notion of non-interfering (and the whole sufficient condition). However, for technical reasons, we have to require that the overlapping parts of two SCS's have the same sort- and subsort-structure (up to renaming). Then we can localize the definition by considering the corresponding production associations of the overlapping parts.

5. Other Approaches to Non-Finitely-Based TRS

Some other investigations in this area have been done. Taylor in [Ta 79] has analysed the non-finitely-based specifications. [DMT 85] discussed the possibility of describing non-finitely-based specifications by auxiliary symbols. For example, for the specification in Example 1, [DMT 85] has given a solution using an auxiliary symbol g': {fgx=fg'x, g'gx=gg'x, g'hx=hx}. In [BMR 86] a transformation system in the above style based on the transformation language OPTRAN (see [MWW 86]) is discussed. However, these auxiliary symbols have no original semantics. This is undesirable, especially in an user-interactive system.

Another approach to cope with the non-finitely-based specification is conditional equational system. By describing the following conditional specification using SCS, we can see the essential relationship between them.

```
DIV2
BOOL + NAT +
sorts:
opns: div2: Nat→Nat
      even: Nat→Bool
eqns: for all n : Nat
   even(succ(succ(n))) = even(n)
   even(succ(0)) = false
   even(0) = true
   even(n) = true ⇒
        div2(succ(succ(n))) = succ(div2(n))
   div2(0) = 0
```

```
DIV2'
   BOOL + NAT +
   sorts:
   subsorts: Even,Half⊆Nat
   opns: div2: Nat→Nat
   eqns: for all (v,v̄):(Even,Half)
      div2(v) = v̄
      where
         Even: succ(succ(v)) = Half: succ(v̄)
                    0         =        0
```

In the notations of SCS, we do not need the auxiliary boolean function "even" to define the partiality of div2. Instead, we give the structure of the possible terms directly. Note that the SCS suggests a more efficient implementation.

6. Further Researches

[O'D 85] discusses the topic of designing a programming language based on the equational logic. Our approach could be a basis for a transformation language, an extension of the language of [O'D 85]. A draft discussion has been made in [K-B 86]. For open questions see [Qi 86c].

From the theoretical point of view, we feel that the SCS style of describing an equational specification might be of some use in the study of error algebra. (See [Kre 86] for another interesting approach.)

Another problem, which is of interest from the practical as well as theoretical point of view, is how to simplify the description of SCS.

We have introduced a sufficient condition for non-interfering of an SCS. How to implement this checking mechanically requires a lot of implementation work. But we feel that the designer of an SCS should write the system in a way to ease this checking.

If we consider a calculus of program development or transformation, (see e.g. [Pe 84], [JHW 86],) it is natural to ask whether SCS could contribute to the framework of a calculus, and which kind of influences it might have on the calculus. Among others, there would be the question of composition of transformation operations and validation of a transformation against the original semantics of the source language.

7. Acknowledgment

I would like to thank B.Krieg-Brückner for the encouragement and B.Krieg-Brückner, B.Hoffmann, Wei Li for the valuable discussions. Thanks are also due to B. Gersdorf and St. Kahrs for valuable comments on the earlier version of this paper.

8. References

[ADJ 78] Goguen,J.A., Thatcher,J.W., Wagner,E.G.: "An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types" in: Current Trends in Programming Methodology, Vol. IV (ed. Yeh,R.T.), Prentice Hall, 1978.

[BG 79] Burstall,R.M., Goguen,J.A., "The Semantics of Clear, a Specification Language" In: Proc. of the 1979 Copenhagen Winter School on Abstract Software Specification, LNCS Vol. 86, Springer-Verlag, 1980.

[BaWö 82] Bauer,F.L.,Wössner,H., "Algorithmic Language and Program Development" Springer-Verlag 1982

[BMR 86] Badt,P., Möncke,U. and Raber,P., "Specification of Recursive Patterns" PROSPECTRA Research Report S.1.6_SN_4.0, March, 1986

[CIP-L] Bauer,F.L., Broy,M., Dosch,W., Geiselbrechtinger,F., Hesse,W., Gratz,R., Krieg-Brückner,B., Laut,A., Matzner,T., Möller,B., Nickl,F., Partsch,H., Pepper,P.,Samelson,K., Wirsing,M., Wössner,H., "The Munich Project CIP. Volume I: The Wide Spectrum Language CIP-L" LNCS 183, Springer 1985

[De 83] Dershowitz,N., "Applications for the Knuth-Bendix Completion Procedure", Aerospace Report No. atr-83(8478)-2, Lab. Operations, The Aerospace Corporation,. 1983

[De 85] Dershowitz,N., "Computing with Rewrite Systems", in: Information and Control Vol.65, No.2/3, May/June 1985

[DMT 85] Dershowitz,N., Marcus,L. and Tarlecki,A.,"Existence, Uniqueness, and Construction of Rewrite Systems" Aerospace Report No. atr-85(8354)-7, Lab. Operations, The Aerospace Corporation, 1985

[HO 80] Huet,G., Oppen,D.C., "Equations and Rewrite Rules : A Survey" in: Formal Language Theory: Perspectives and Open Problems (ed. Book,R.), Academic Press, New York, 1980

[JHW 86] Jähnichen,S., Hussain,F.A., Weber,M., "Program Development Using a Design Calculus", Proceeding of the ESPRIT Technical Week, 1986.

[Ka 83] Kaplan,S. "Conditional Term Rewrite Systems" Report No.150, LRI-Orsay, France 1983. Revised Version in: Theo. Comp. Sci. Vol.33, 1984

[K-B 86] Krieg-Brückner,B., "Informal Specification of TrafoLa" PROSPECTRA Research Report M.1.1.S1-SN-10.0, Bremen, March, 1986.

[Kre 86] Kreowski, H.-J., "Partial Algebras Flow From Algebraic Specification", draft version, FB Mathematik/Informatik, Univ. of Bremen, Feb. 1986.

[MW 85] Manna,Z., Waldinger,R.: "The Logic Basis for Computer Programming Vol. 1: Deductive Reasoning", Addison-Wesley Publishing Company, Inc., 1985

[O'D 77] O'Donell,M.J., "Computing in Systems Described by Equations" LNCS 58, Springer-Verlag 1977

[O'D 85] O'Donell,M.J., "Equational Logic as a Programming Language", MIT Press Series in the Foundations of Computing, Cambridge,Massachusetts, London,England, 1985

[Pe 84] Pepper,P., "A Simple Calculus for Program Transformations (Inclusive of Induction)", Research Report TUM-I8409, Technical Univ. of Munich, July 1984

[PROSPECTRA]
Krieg-Brückner,B., Ganzinger,H., Broy,M., Wilhelm,R., McGettrick,A.M., Campbell,I.G., Winterstein,G., "PROgram development by SPECification and TRAnsformation, Project Summary." Univ. of Bremen, 1985

[Qi 86a] Qian,Zh., "Recursive Presentation of Program Transformation" PROSPECTRA Study Notes M.1.1.S1-SN-17.1, Univ. of Bremen, Sept. 1986

[Qi 86b] Qian,Zh., "An Example of Recursively Presented Transformation" PROSPECTRA Study Notes M.1.1.S1-SN-21.0, Univ. of Bremen, Nov. 1986

[Qi 86c] Qian,Zh., "Sufficient Condition for Confluent and Terminating Term Rewrite System and its Extension to Recursively Presented Term Rewriting" PROSPECTRA Study Notes M.1.1.S1-SN-16.2, Univ. of Bremen, Nov. 1986

[Ta 79] Talor,W., "Equational Logic" in: Universal Algebra" (ed. Gratzer,G.), second edition, Springer-Verlag 1979