1

Unification in a process algebra with choice and sequential composition.

N.J.Drost
Programming Research Group
University of Amsterdam
The Netherlands

July 29, 1994

[1]This work was partially supported by the Foundation for Computer Science in the Netherlands (SION) with financial support from the Netherlands Organization for Scientific Research (NWO).

Abstract

This paper contains a unification algorithm for a restricted process algebra with as only operators nondeterministic choice $+$ (associative, commutative, idempotent, and with unit element) and sequential composition $\cdot$ (associative, right-distributive over $+$, and with left zero element). The algebra contains only finite processes. The algorithm is based on rules, and is proved sound and complete. As unification in this algebra is infinitary, the algorithm does not terminate in general. Termination is proved for problems consisting of equations where at least one of the two terms is closed. In this case the produced set of solutions is also a minimal set of most general unifiers, and the algorithm has exponential complexity.

Key words and phrases: unification, process algebra.

Contents

Chapter 1

Introduction

This paper gives a unification algorithm for a restricted process algebra with as only operators nondeterministic choice and sequential composition. Process algebras are algebras devised for the description and verification of concurrent processes. It is assumed that processes consist of undivisible atomic actions, which are combined using operators, like nondeterministic choice, sequential composition, and communication. Well known process algebras are CCS [Mil89], CSP [Hoa85], and ACP [BK84], [BW90].

The words "algebra" (or "model") and "theory" (or "equational specification") are used in their model theoretic sense, i.e. an algebra (or model) is a pair consisting of a domain and a set of functions on that domain, a theory (or equational specification) is a pair consisting of a signature and a set of axioms over that signature, an algebra $\mathcal{A}$ is a Σ-algebra if there is a mapping of constant and function symbols in Σ to resp. objects and functions in $\mathcal{A}$, and a Σ-algebra $\mathcal{A}$ is an algebra (a model) of a theory $(\Sigma, \mathcal{E})$ if for all axioms $s = t \in \mathcal{E} : \mathcal{A} \models s = t$. If $(\Sigma, \mathcal{E})$ is a theory, $\mathcal{A}$ is a Σ-algebra, each function in $\mathcal{A}$ corresponds to a function symbol in Σ, and $\mathcal{A} \models s = t$ iff $\mathcal{E} \vdash s = t$, then $(\Sigma, \mathcal{E})$ is called a **complete axiomatization** of $\mathcal{A}$. $\mathcal{A}$ is not necessarily isomorphic to the initial model of $(\Sigma, \mathcal{E})$ because the domain of $\mathcal{A}$ may contain objects that do not correspond to terms in the signature. In the case of process algebras these objects represent infinite processes: processes that may perform an infinite sequence of actions.

Process algebras may be used as a description method in the development of parallel systems. In top-down modular development first an overall description of a system is given, and then the system is divided into a number of modules. The combination of the modules must conform to the overall description of the system. If the description of one or more modules is missing then this corresponds to an equation in a process algebra containing variables. Solving of this equation gives then all possible solutions for the unspecified modules. Parrow [Par89] and Larsen and Liu [LL90] each give a method for solving equation systems in process algebras. Both methods allow a wide range of process algebra operators, but a small subset of equations: only one variable occurrence in each equation. The algorithm of Parrow is not aimed at producing a complete description of all solutions. The method of Larsen and Liu is not easily extendible to other types of equations. So another approach is needed for devising an algorithm for solving wider classes of equations.

Equation solving in abstract algebras is used in many other fields of mathematics and computer science, e.g. deduction systems, term rewriting, logic programming, and expert systems. Syntactic unification is unification in a theory without axioms, so the algebra used is the term algebra. A solution to a set of equations is a substitution that equalizes the left and right side of each equation. The first algorithm for syntactic unification is due to Robinson [Ro65], who was also the first to use the word unification. He defined a partial order on solutions of a unification problem such that one solution is smaller (more general) than a second solution if the second solution is derivable from the first. He also showed that the solution set of a syntactic unification problem can be described by giving one most general unifier. Semantic unification or E-unification is unification in a theory with a set E of axioms. A solution to a problem P is a substitution σ such that $E \vdash s\sigma = t\sigma$ for all equations $s = t$ in P. This is the same as equality in the algebra of open terms modulo E. In semantic unification the solution set (if it exists) of a problem cannot always be described with one most general unifier. The first algorithm for semantic unification was given by Plotkin [Plo72]. He defined the notion of a maximally general set of unifiers, also called a complete set of most general unifiers [JK91].

Siekmann [Sie90] gave a classification of equational theories based on the cardinality of solution sets of problems in the theory. An equational theory is **nullary** if there are equations that have at least one solution but for which there exists no complete set of most general unifiers. It is **infinitary** if there are equations for which there exists only an infinite set of most general unifiers. A **unitary** theory has for each equation that has at least one solution a complete set of most general unifiers of size 1. A **finitary** theory has for each equation that has at least one solution a complete set of most general unifiers of finite size.

Jouannaud and Kirchner [JK91] give an overview of unification algorithms described by rules. A rule transforms an equation set into another equation set with the same set of solutions. An algorithm based on rules should eventually transform the equation set into one or more equation sets in solved form.

E-unification differs from equation solving in a process algebra because the first starts with a given theory and the second with a given algebra. With a process algebra often also a theory is given. If the axioms are a complete axiomatization of the algebra and all objects in the domain of the algebra correspond to terms in the algebra, then equation solving in the process algebra and E-unification coincide. But most process algebras have objects that represent infinite processes and that do not correspond to terms in the signature. This has consequences for the definition of a solution of a unification problem. A solution is then a semantic substitution (a function that replaces variables in processes by process elements) that makes the interpretation of the equations true in the model. A solution is more general than another solution if there is a semantic substitution that makes the first solution equal to the second.

As unification in an algebra with choice and sequential composition that contains infinite processes turned out to be too difficult, here an algebra without infinite processes is considered. Still a solution is defined as a semantic substitution, to make a later addition of infinite processes possible. The operator "nondeterministic choice" has the same behavior as the operator "union" in the algebra of sets. A unification algorithm for sets with union and empty set is given by Baader and Büttner [BB86]. The operator "sequential composition" is associative, so sequential composition is equivalent with concatenation of words. Unification in the algebra of strings with concatenation (also called the word unification problem) is decidable, but a complete set of most general unifiers may be infinite [Mak77]. Abdulrab and Pecuchet [AP89] give an algorithm that produces a description of a set of most general unifiers in the form of a graph. This graph may describe an infinite set of most general unifiers. The graph is not always finite if the set of most general unifiers is finite. Jaffar [Jaf90] gives a unification algorithm that terminates if the complete set of most general unifiers is finite.

A unification algorithm for the process algebra with nondeterministic choice and sequential composition cannot be constructed by combining the unification algorithms for each operator separately, because the algebra has an axiom that contains both operators. Arnborg and Tiden [AT85] analyze algebras with operators $\cdot$ and $+$, and combinations of the axioms

$$
\begin{array}{ll}
x \cdot (y+z) = x \cdot y + x \cdot z & (D_L\text{: left distributivity of } \cdot \text{ over } +), \\
(x+y) \cdot z = x \cdot z + y \cdot z & (D_R\text{: right distributivity of } \cdot \text{ over } +), \\
x \cdot (y \cdot z) = (x \cdot y) \cdot z & (\text{A: associativity of } \cdot), \\
x \cdot 1 = 1 \cdot x = x & (\text{U: unit element of } \cdot).
\end{array}
$$

They give a terminating algorithm for an algebra with only D_L (or D_R) that may be applied in any permutative theory (a theory with all congruence classes finite), and prove that unification is undecidable if A and U are added.

The restricted process algebra in this paper differs from the algebras analyzed by Arnborg and Tiden [AT85]. The operator $\cdot$ (sequential composition) is associative and right-distributive over $+$, but has no unit element. Instead it has a left zero element δ. The operator $+$ (nondeterministic choice) is associative, commutative and idempotent, and has δ as unit element. The idempotence of $+$ makes the theory non-permutative, so the algorithm of Tiden and Arnborg [TA87] is not usable.

The paper is organized as follows: In Chapter 2 the signature, axioms, and model of the algebra are described. Chapter 3 gives some definitions and describes the algorithm, first as a set of rules, and also more detailed as a pseudo-program. Chapter 4 contains the proof that the unification algorithm is sound and complete. Chapter 5 gives a proof that the algorithm terminates on sets of equations with one side closed, Chapter 6 contains a complexity analysis, and Chapter 7 gives conclusions and discussion.

Chapter 2

Algebra and theory

2.1 Introduction

The algebra contains only finite processes. A finite process is a process that may perform only a finite sequence of actions. There are operators for nondeterministic choice and sequential composition. Processes consist of indivisible atomic actions. A distinction is made between successful and unsuccessful termination of a process. The constant δ represents a process that terminates unsuccessfully without performing any actions. There is a finite set of constants. A constant a represents a process that performs action a and then terminates successfully.

2.2 The theory

The signature Σ of the theory contains a finite set $\mathcal{A}$ of constant symbols plus a special constant δ, binary operators $+$ for nondeterministic choice and $\cdot$ for sequential composition, and a countably infinite set $\mathcal{V}$ of variables. A term is a **constant** a, a **variable** x, a **sum** $t_1 + t_2$, a **sequential composition** $t_1 \cdot t_2$, or δ. The set of finite terms over this signature is denoted by $\mathcal{T}$. Notation: $a, b, c, \ldots \in \mathcal{A}$; $A, B, C, \ldots \subseteq \mathcal{A}$, $x, y, z, u, v \in \mathcal{V}$, $X, Y, Z, U, V \subseteq \mathcal{V}$, $s, t, \ldots \in \mathcal{T}$, and $S, T, \ldots \subseteq \mathcal{T}$. The symbol $\cdot$ will often be omitted.
The axiom set $\mathcal{E}$ of the theory is:

$(x + y) + z = x + (y + z)$	(associativity of $+$)
$x + y = y + x$	(commutativity of $+$)
$x + x = x$	(idempotence of $+$)
$(x + y) \cdot z = x \cdot z + y \cdot z$	(right distributivity of $\cdot$ over $+$)
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	(associativity of $\cdot$)
$x + \delta = x$	(unit element of $+$)
$\delta \cdot x = \delta$	(left zero element of .)

Notation: $\equiv$ will be used for syntactic equality between terms and $=_{\mathcal{E}}$ for equality modulo the axioms.

2.3 Syntactic functions

The functions *Var* and *Const* give the variables, resp. the constants of a term. The function *Smd* gives the set of summands of a sum: $Smd(\delta) = \emptyset$, $Smd(t) = \{t\}$ for $t = a, x$, and $t_1 \cdot t_2$, and $Smd(t_1 + t_2) = Smd(t_1) \cup Smd(t_2)$. The function Sub gives the set of subterms of a term, including the term itself: $Sub(t) = \{t\}$ for $t = \delta$, a, and x, and $Sub(t_1 \otimes t_2) = \{ t_1 \otimes t_2 \} \cup Sub(t_1) \cup Sub(t_2)$ for $\otimes = \cdot$ or $\otimes = +$.

For sums also an alternative notation is used. Suppose $I = \{ 1, \ldots, n \}$ and $T = \{ t_1, \ldots, t_n \}$. Then $\sum_{i \in I} t_i$ and ΣT are notations for the term $t_1 + (t_2 + (\ldots + t_n) \ldots)$.

Variables in a term may be replaced by arbitrary terms using a (syntactic) substitution. A **syntactic substitution** is a function from $\mathcal{V}$ to $\mathcal{T}$.It is denoted as $\sigma = \{ x := t_x \mid x \in X \}$, with $x \not\equiv t_x$ for all $x \in X$. Syntactic substitutions are extended in the obvious way to functions from $\mathcal{T}$ to $\mathcal{T}$. Application of a syntactic substitution σ to a term t is denoted as $t\sigma$.

The unification algorithm will work on terms in normal form. First a rewrite system is defined that distributes $\cdot$ over $+$ in terms of the form $(t_1 + t_2) \cdot t_3$, moves brackets in a sequence of sequential composition to the right, and removes terms that occur after a δ in a sequential composition.

$$\begin{array}{ll}
(t_1 + t_2) \cdot t_3 & \longrightarrow t_1 \cdot t_3 + t_2 \cdot t_3, \\
(t_1 \cdot t_2) \cdot t_3 & \longrightarrow t_1 \cdot (t_2 \cdot t_3), \\
\delta \cdot t & \longrightarrow \delta.
\end{array}$$

This rewrite system is terminating and confluent, so it describes a function *PNF* on terms from $\mathcal{T}$. If $PNF(t) \equiv t$ then t is said to be in prenormal form. A term in prenormal form is a constant, a variable, a sum, a **constant product** $a \cdot t$, a **variable product** $x \cdot t$, or δ. The set of terms in prenormal form is called $\mathcal{T}_{PNF}$.

Now a normalizing function on terms from $\mathcal{T}$ is defined. First a sequence of subsets of $\mathcal{T}_{PNF}$ is defined based on the number of occurrences of the symbol "+" of a term in prenormal form, where $\mathcal{T}_n$ denotes the set of all terms t with at most n symbols "+". The function *Nplus* gives the number of occurrences of the operator "+" in a term: $Nplus(t) = 0$ for $t = \delta, a$, and x, $Nplus(t_1 \cdot t_2) = Nplus(t_1) + Nplus(t_2)$, and $Nplus(t_1 + t_2) = Nplus(t_1) + Nplus(t_2) + 1$. For each $\mathcal{T}_n$ a normalizing function NF_n and total order $\leq_n$ are defined. A total order on $\mathcal{A}$ and $\mathcal{V}$ is presupposed.

Definition 2.1 *For each $n \geq 0$ a function $NF_n : \mathcal{T}_n \to \mathcal{T}_n$, and a partial order $\leq_n$ on $\mathcal{T}_n$ are inductively defined as follows:*

For all $n \geq 0$:

$NF_n(\delta) = \delta$, $NF_n(a) = a$, $NF_n(x) = x$, $NF_n(a \cdot t) = a \cdot NF_n(t)$, $NF_n(x \cdot t) = x \cdot NF_n(t)$,
$NF_n(t_1 \cdot t_2) = NF_n(t_1 \cdot t_2)$,
$\delta \leq_n s$, $s \leq_n s$, $a \leq_n x$ for all $s \in \mathcal{T}$, $a \in \mathcal{A}$, and $x \in \mathcal{V}$,
if $s, d \in \mathcal{A} \cup \mathcal{V}$, $t \equiv dt'$ then $s \leq_n t$ iff $s \leq_n d$, and $t \leq_n s$ iff $d \leq_n s$ and $d \not\equiv s$,
if $d, e \in \mathcal{A} \cup \mathcal{V}$, $s \equiv ds'$, $t \equiv et'$ then $s \leq_n t$ iff $d <_n e$, or $d = e$ and $s' \leq_n t'$.

For $n > 0$:
Suppose t is a sum and $Nplus(t) \leq n$.
Define $T \stackrel{\text{def}}{=} \{NF_{n-1}(t') \mid t' \in Smd(t)\}$.
Suppose $T = \{t_1, \ldots, t_m\}$, and $t_i <_{n-1} t_{i+1}$ for all $1 \leq i < m$. Then:
$NF_n(t) = t_1 + \ldots + t_m$.
If s or t is a sum:
Define $T_s \stackrel{\text{def}}{=} \{NF_{n-1}(s') \mid s' \in Smd(s)\}$, $T_t \stackrel{\text{def}}{=} \{NF_{n-1}(t') \mid t' \in Smd(t)\}$,
$T_L = T_s - T_t$, $T_R = T_t - T_s$. Then:
$s \leq_n t$ iff $T_L = \emptyset$, or there is a $t_j \in T_R$ such that for all $s_i \in T_L : s_i \leq_{n-1} t_j$.

Then $\leq_n$ is a total order on $\mathcal{T}_n$ for each $n \geq 0$. Moreover, if $Nplus(t) \geq n$ then $NF_n(t) = NF_{n-1}(t)$, and if $s <_n t$ then $s <_{n-1} t$ for all $n > 0$.
Now the normalizing function $NF : \mathcal{T} \to \mathcal{T}$ is defined as:

Definition 2.2 *$NF(t) = NF_n(PNF(t))$ with $n = Nplus(PNF(t))$.*

Then $NF(s) \equiv NF(t)$ iff $s =_{\mathcal{E}} t$.

2.4 The algebra $\mathcal{P}$

Processes are modeled as well founded Psets. A Pset is a finite set that contains zero or more pairs with as first component an atomic action or a variable, and as second component a Pset or a set containing a special symbol $\surd$.

Definition 2.3 *A Pset is a finite set of zero or more pairs (d, q) with $d \in \mathcal{A} \cup \mathcal{V}$ and q a Pset or $q = \{\surd\}$.*

The operators of $\mathcal{P}$ are $\oplus$ (nondeterministic choice) and $\odot$ (sequential composition).

The model $\mathcal{P}$:
Domain $\mathcal{D}_{\mathcal{P}}$: well-founded Psets.
Operators: $\oplus$ and $\odot$ with:
$p_1 \oplus p_2 = p_1 \cup p_2$,
$p_1 \odot p_2 = \{\, (d, q \odot p_2) \mid (d, q) \in p_1 \,\}$,
where $\{\surd\} \odot p_2$ is defined as p_2.

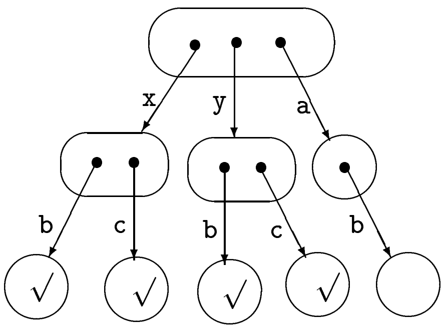
Terms from $\mathcal{T}$ are mapped to Psets as follows:

$$[\![\delta]\!] = \emptyset,\ [\![a]\!] = \{\, (a, \{\surd\}) \,\},\ [\![x]\!] = \{\, (x, \{\surd\}) \,\},\ [\![t_1 + t_2]\!] = [\![t_1]\!] \oplus [\![t_2]\!],\ [\![t_1 \cdot t_2]\!] = [\![t_1]\!] \odot [\![t_2]\!].$$

Sets of terms from $\mathcal{T}$ are mapped to sets of Psets:

$$[\![T]\!] = \{\, [\![t]\!] \mid t \in T \,\}.$$

A Pset may be depicted as a kind of tree, where each set is represented as an oval and each pair as a labeled arrow to an oval:
$[\![\, (x + y) \cdot (b + c) + ab\delta \,]\!]$:



Definition 2.4 *Process p is a* **state** *of process q if $p = q$ or there is a process q' with $(d, q') \in q$ for some $d \in \mathcal{A} \cup \mathcal{V}$, and p is a state of q'.*

Definition 2.5 *Process p* **has a successful termination option** *if $(d, \{\surd\}) \in p$ for some $d \in \mathcal{A} \cup \mathcal{V}$, or there is a pair $(d, p') \in p$ and p' has a successful termination option. A set $\{\surd\}$ is also called a* **successful termination point**.

Definition 2.6 *A* **path** *in a Pset from* p_0 *to* p_n *is a sequence* $(d_1, \ldots, d_n)$ *with* p_0 *a state of* p, $d_i \in \mathcal{A} \cup \mathcal{V}$, *and* $(d_i, p_i) \in p_{i-1}$ *for all* i *with* $1 \leq i \leq n$. *A path from* p_0 *to an occurrence of a variable* x *in* p_n ($(x, q) \in p_n$ *for some* $q \in \mathcal{P} \cup \{\surd\}$) *is the path from* p_0 *to* p_n. *A path from* p_0 *to a successful termination point* $\{\surd\}$ *is a sequence* $(d_1, \ldots d_n, d)$ *for an* $n \geq 0$ *with* $(d, \{\surd\}) \in p_n$ *and* $(d_1, \ldots, d_n)$ *the path from* p_0 *to* p_n.

An **assignment** is a function from $\mathcal{V}$ to $\mathcal{D}_{\mathcal{P}}$. Notation: $\psi = \{ x := q_x \mid x \in X \}$, with $\psi(x) \neq x$ iff $x \in X$. Assignments are used to define semantic substitutions. A semantic substitution is a function from $\mathcal{D}_{\mathcal{P}}$ to $\mathcal{D}_{\mathcal{P}}$, that replaces each occurrence of a variable of a given set of variables by a number of Pset-elements. Application of a semantic substitution α to a process p is denoted as $p\alpha$. The notation for a semantic substitution is equal to the notation of the assignment that is used as a basis: $\alpha = \{ x := q_x \mid x \in X \}$, with $[\![x]\!]\alpha \neq [\![x]\!]$ iff $x \in X$.

Definition 2.7 *Suppose* p *is a Pset and* $\alpha = \{ x := q_x \mid x \in X \}$. *Application of the semantic substitution* α *to process* p *is defined as:*

$$\begin{aligned} p\alpha \;=\; & \{ (d, p'\alpha) \mid (d, p') \in p \textit{ and } d \in \mathcal{A} \cup \mathcal{V} - X \} \\ & \cup \bigcup \{ q_x \odot (p'\alpha) \mid (x, p') \in p \textit{ and } x := q_x \in \alpha \} \end{aligned}$$

where $\{\surd\}\alpha = \{\surd\}$.

Example:
Suppose $p = \{ (a, \emptyset), (x, \{ (y, \{\surd\}) \}) \}$, and $\alpha = \{ x := \{ (b, \{\surd\}) \} \}$,
then $p\alpha = \{ (a, \emptyset), (b, \{ (y, \{\surd\}) \}) \}$.
A semantic substitution will often be denoted as $\alpha = \{ x := [\![t_x]\!] \mid x \in X \}$ with the t_x terms from $\mathcal{T}$. Suppose $\sigma = \{ x := t_x \mid x \in X \}$ is a syntactic substitution. Then $[\![t\sigma]\!] = [\![t]\!]\alpha$. The set X of a semantic substitution $\alpha = \{ x := q_x \mid x \in X \}$ is called the domain of α. The function Dom gives the domain of a semantic substitution. The set $\bigcup \{ y \mid y$ occurs in q_x for some $x \in X \}$ is called the range of α and is produced by the function Ran.

A semantic substitution may be restricted to a subset X of $\mathcal{V}$:

Definition 2.8 *The* **restriction to** $X \in \mathcal{V}$ *of a semantic substitution* α *is defined as:*

$$\alpha_{|X} \stackrel{\text{def}}{=} \{ x := q_x \mid x \in X \textit{ and } x := q_x \in \alpha \}.$$

Chapter 3

The algorithm $\mathbf{UnSeq}_f$

3.1 Introduction and definitions

The unification algorithm presented in this paper is based on rules. A rule transforms a unification problem consisting of a set of equations (possibly existentially quantified) into a number of alternative unification problems. First some general unification definitions are given. These are based on [JK91]. The notation follows the proposal in [DJ91].

Definition 3.1 *A* **(unification) problem (equation solving problem)** *is a possibly existentially quantified set of equations*

$$\exists\,\vec{u}\{s_i =^?_{\mathcal{E}} t_i \mid i \in I\}$$

with $\vec{u}$ a sequence of variables from $\mathcal{V}$ and all s_i and t_i from $\mathcal{T}$.

A solution of a unification problem is a semantic substitution:

Definition 3.2 *Suppose $\exists\,\vec{u}P$ is a problem with $P = \{s_i =^?_{\mathcal{E}} t_i \mid i \in I\}$ and $V = Var(P) - \vec{u}$. A* **solution (unifier)** *of $\exists\,\vec{u}P$ is a semantic substitution α such that there is a semantic substitution β with $Dom(\beta) = \vec{u}$, $Ran(\alpha) \cap \vec{u} = \emptyset$ and*

$$\mathcal{P} \models [\![s_i]\!]\,\alpha_{|V}\,\beta = [\![t_i]\!]\,\alpha_{|V}\,\beta \text{ for all } i \in I.$$

The set of all solutions of a problem $\exists\,\vec{u}P$ is denoted as $Sol(\exists\,\vec{u}P)$.

Definition 3.3 *A problem $\exists\vec{u}_1P_1$ is* **equivalent** *with a problem $\exists\vec{u}_2P_2$ iff $Sol(\exists\vec{u}_1P_1) = Sol(\exists\vec{u}_2P_2)$.*

Definition 3.4 *A* **rule** *is a transition*

$$\exists\,\vec{u}P \Longrightarrow \{\,\exists\vec{u}_1P_1,\ldots,\exists\vec{u}_nP_n\,\}$$

with $\exists\,\vec{u}P$, $\exists\vec{u}_1P_1,\ldots,$ $\exists\vec{u}_nP_n$ unification problems.

Definition 3.5 *A* **solved form** *is a unification problem $\exists\,\vec{u}\,P$, $P = \{\,x_1 =^?_{\mathcal{E}} t_1,\ldots,x_n =^?_{\mathcal{E}} t_n\,\}$, where for all i with $1 \leq i \leq m$: $x_i \notin Var(t_i)$.*
P is **in solved form** *iff P is a solved form. An equation $s =^?_{\mathcal{E}} t$ is* **in solved form** *if $s \equiv x$ and $x \notin Var(t)$.*

Definition 3.6 *A semantic substitution* α *is* **more general** *with respect to a set* $X \subseteq \mathcal{V}$ *than a semantic substitution* β*, notation* $\alpha \leq_X \beta$*, if there is a semantic substitution* γ *such that for all* $x \in X : x\alpha\gamma = x\beta$*. This order is called the* **subsumption order**.

Definition 3.7 *A semantic substitution* α *is* **most general** *in a set of semantic substitutions if it is minimal in this set with respect to the subsumption order.*

Suppose $\exists\,\vec{u}P$ is a unification problem and $V = Var(P) - \vec{u}$.

Definition 3.8 *A set SU of unifiers of* $\exists\,\vec{u}P$ *is* **complete** *if for all* $\alpha \in Sol(\exists\,\vec{u}P)$ *there is a* $\beta \in SU$ *such that* $\beta \leq_V \alpha$*.*

Definition 3.9 *A complete set SU of unifiers of* $\exists\,\vec{u}P$ *is* **minimal** *if for all* $\alpha, \beta \in SU$ *: if* $\alpha \leq_V \beta$*, then* $\alpha = \beta$*.*

3.2 The rules

The rules aim at reducing the size of the equations and the equation set and changing them in the direction of a solved form. This is done by removing superfluous equations, replacing an equation by one or more smaller ones, or determining that no solution exists. Not all necessary rules succeed in reducing the size of the equations. Rule 0.2 (Subs) may increase the size of the equations, and rule 1.4 (Seq_xy) may replace one equation by two equations of equal size. The equations on which the rules are applied must be in normal form, and the new equations must also be in normal form. Rules 0.1 to 0.4 are (modifications of) general unification rules, rules 1.1 to 1.9 are specific for this algebra.
Suppose s and t are terms in normal form, and P is a set of equations.
Rule 0.1 (Eq) removes equations where the left and right hand side are identical.

0.1 (Eq) $\quad \exists\,\vec{u}\,(\,P \cup \{s =^?_{\mathcal{E}} s\}\,) \Longrightarrow \{\ \exists\,\vec{u}\,P\ \}.$

Rule 0.2 (Subs) substitutes an equation in solved form into the other equations.

0.2 (Subs) Suppose $((s \equiv x,\ t \equiv t_x,\ x \in Var(P))$, or $(t \equiv x,\ s \equiv t_x))$, and $x \notin Var(t_x))$.

$$\exists\,\vec{u}\,(\,P \cup \{s =^?_{\mathcal{E}} t\}\,) \Longrightarrow \{\ \exists\,\vec{u}\,(\,\mathrm{Nf}(P\{x := t_x\}) \cup \{x =^?_{\mathcal{E}} t_x\}\,)\ \}.$$

Rule 0.3 (Occur) is a modified version of the occur check in syntactic unification. It is not applied to equations $x =^?_{\mathcal{E}} t_x$ with x or xt a summand of t_x, because these have a solution in this algebra. It also determines all possibilities of removing the occurrence of x in t_x by substituting a δ before it. This is done by determining the set Y of all variables different from x that occur before this occurrence of x. A variable y **occurs before** an occurrence of x in a term t if $\mathrm{Nf}(t)$ has a subterm $y \cdot t'$ and the occurrence of x is in t'. If Y is empty then there is no solution. If Y is not empty then for each $y \in Y$ an alternative is made that contains the original equations plus a new equation $y =^?_{\mathcal{E}} v\delta$ with v a fresh variable. The equation $x =^?_{\mathcal{E}} t_x$ is substituted by $x =^?_{\mathcal{E}} NF(t_x\{y := v\delta\})$.

0.3 (Occur) Suppose $((s \equiv x,\ t \equiv t_x)$, or $(t \equiv x,\ s \equiv t_x))$, there is an occurrence of x in t_x, $x \notin Smd(t_x)$, no term xt' in $Smd(t_x)$, and Y is the set of all variables $\neq x$ that occur before this occurrence of x. Then:

$$\begin{array}{l} \exists\,\vec{u}\,(\,P \cup \{s =^?_{\mathcal{E}} t\}\,) \implies \textbf{if } Y = \emptyset \textbf{ then } \{\mathrm{F}\} \textbf{ else} \\ \quad \{\ \exists\,\vec{u}, v\,(\,P \cup \{\,x =^?_{\mathcal{E}} \mathrm{Nf}(\,t_x\{y_1 := v\delta\}\,),\ y_1 =^?_{\mathcal{E}} v\delta\,\}\,), \\ \quad\quad \ldots \\ \quad\quad \exists\,\vec{u}, v\,(\,P \cup \{\,x =^?_{\mathcal{E}} \mathrm{Nf}(\,t_x\{y_n := v\delta\}\,),\ y_n =^?_{\mathcal{E}} v\delta\,\}\,)\ \} \end{array}$$

where $Y = \{y_1, \ldots, y_n\}$ and v is a fresh variable.

Rule 0.4 (AuxVar) removes an equation of an auxiliary variable that does not occur in the other equations.

0.4 (AuxVar) Suppose $u \notin Var(P)$. Then:
$\exists \vec{u}_1, u, \vec{u}_2\,(\,P \cup \{u =^?_{\mathcal{E}} t_u\}\,) \Longrightarrow \{\ \exists \vec{u}_1, \vec{u}_2\, P\ \}$.

Rule 1.1 (NoSol) detects some cases that have no solution.

1.1 (NoSol) Define $T_s = Smd(s)$, $T_t = Smd(t)$. Suppose
$[\,T_t \cap \mathcal{V} = \emptyset$ and $\{\,\mathcal{A} \cap (T_s - T_t) \neq \emptyset$ or $\exists as' \in T_s$ and no $at' \in T_t\,\}\,]$ or
$[\,T_s \cap \mathcal{V} = \emptyset$ and $\{\,\mathcal{A} \cap (T_t - T_s) \neq \emptyset$ or $\exists at' \in T_t$ and no $as' \in T_s\,\}\,]$. Then:

$$\exists\,\vec{u}\,(\,P \cup \{s =^?_{\mathcal{E}} t\}\,) \Longrightarrow \{F\}.$$

Rules 1.2 to 1.5 work on equations $s =^?_{\mathcal{E}} t$ where both s and t are a product. Rule 1.2 (Seq_aa) is applicable if both s and t are constant products with as left arguments the same constant. It makes a new equation of the right arguments.

1.2 (Seq_aa) $\exists\,\vec{u}\,(\,P \cup \{as =^?_{\mathcal{E}} at\}\,) \Longrightarrow \{\ \exists\,\vec{u}\,(\,P \cup \{s =^?_{\mathcal{E}} t\}\,)\ \}$.

Rule 1.3 (Seq_ax) is used if one term is a constant product and the other is a variable product. The variable can be made equal to the constant, or to the constant followed by something else.

1.3 (Seq_ax) Suppose $s \equiv xs'$ and $t \equiv at'$, or $t \equiv xs'$ and $s \equiv at'$. Then:

$$\exists\,\vec{u}\,(\,P \cup \{s =^?_{\mathcal{E}} t\}\,) \Longrightarrow \left\{ \begin{array}{l} \exists\,\vec{u}\,(\,P \cup \{x =^?_{\mathcal{E}} a,\ s' =^?_{\mathcal{E}} t'\}\,) \\ \exists\,\vec{u}, v\,(\,P \cup \{x =^?_{\mathcal{E}} av,\ vs' =^?_{\mathcal{E}} t'\}\,)\ \end{array} \right\}.$$

with v a fresh variable.

If both terms are variable products and the variables are different, then rule 1.4 (Seq_xy) is applicable. It has an equation $xs =^?_{\mathcal{E}} yt$ as input and produces four alternatives. In the first alternative the variables have equal solutions, and both have no successful termination option. This is expressed by the equations $x =^?_{\mathcal{E}} v\delta$ and $y =^?_{\mathcal{E}} v\delta$. In this case equality of s and t is not required. In the second alternative again x and y must have equal solutions, and $s =_{\mathcal{E}} t$ must also hold. In the other two x overlaps y or y overlaps x.

1.4 (Seq_xy) Suppose $x \not\equiv y$. Then:

$$\exists \vec{u}\,(P \cup \{xs =^?_{\mathcal{E}} yt\}) \Longrightarrow \{ \begin{array}{l} \exists \vec{u}, v\,(P \cup \{x =^?_{\mathcal{E}} v\delta,\ y =^?_{\mathcal{E}} v\delta\}), \\ \exists \vec{u}\,(P \cup \{x =^?_{\mathcal{E}} y,\ s =^?_{\mathcal{E}} t\}), \\ \exists \vec{u}, v\,(P \cup \{x =^?_{\mathcal{E}} yv,\ vs =^?_{\mathcal{E}} t\}), \\ \exists \vec{u}, v\,(P \cup \{y =^?_{\mathcal{E}} xv,\ vt =^?_{\mathcal{E}} s\}) \quad \}. \end{array}$$

with v a fresh variable.

Rule 1.5 (Seq_xx) treats the case where both sides of the equation start with the same variable. It produces two alternatives, one in which solutions of the variable x have no successful termination option, and one in which s and t are equal.

1.5 (Seq_xx) $\exists \vec{u}\,(P \cup \{xs =^?_{\mathcal{E}} xt\}) \Longrightarrow \{ \begin{array}{l} \exists \vec{u}, v\,(P \cup \{x =^?_{\mathcal{E}} v\delta\}), \\ \exists \vec{u}\,(P \cup \{s =^?_{\mathcal{E}} t\}) \quad \}. \end{array}$

with v a fresh variable.

Rule 1.6 (Seq_xδ) is used for equations $\delta =^?_{\mathcal{E}} xs'$ and $xs' =^?_{\mathcal{E}} \delta$, and produces $x =^?_{\mathcal{E}} \delta$.

1.6 (Seq_xδ) Suppose $(s \equiv xs',\ t \equiv \delta)$, or $(t \equiv xs',\ s \equiv \delta)$. Then:

$$\exists \vec{u}\,(P \cup \{s =^?_{\mathcal{E}} t\}) \Longrightarrow \{\ \exists \vec{u}\,(P \cup \{x =^?_{\mathcal{E}} \delta\})\ \}.$$

Rule 1.7 (VarSeq) is meant for equations of the form $x =^?_{\mathcal{E}} xs'$ and $xs' =^?_{\mathcal{E}} x$. Then x cannot have a successful termination option.

1.7 (VarSeq) Suppose $(s \equiv x,\ t \equiv xt')$ or $(t \equiv x,\ s \equiv xt')$. Then:

$$\exists \vec{u}\,(P \cup \{s =^?_{\mathcal{E}} t\}) \Longrightarrow \{\ \exists \vec{u}, v\,(P \cup \{x =^?_{\mathcal{E}} v\delta\})\ \}$$

with v a fresh variable.

Rule 1.8 is usable for equations where one side is a variable, and the other side a sum with a summand starting with this variable.

1.8 (VarSeq$^+$) Suppose $((s \equiv x,\ t \equiv t'),$ or $(t \equiv x,\ s \equiv t')),\ t' =_{\mathcal{E}} t_1 + xt_2,\ xt_2 \notin Smd(t_1)$.

$$\begin{aligned}\exists\,\vec{u}\,(\,P \cup \{s =^?_{\mathcal{E}} t\}\,) \Longrightarrow \{\ \ &\exists\,\vec{u}, v\,(\,P \cup \{x =^?_{\mathcal{E}} t_1\delta + v\delta\}\,),\\ &\exists\,\vec{u}, v_1, v_2\,(\,P \cup \{x =^?_{\mathcal{E}} t_1 + v_1,\ v_1 =^?_{\mathcal{E}} v_2\delta + t_1t_2\delta\}\,)\ \ \}\end{aligned}$$

with v_1 and v_2 fresh variables.

Rule 1.9 (Sum) is applied if one side of the equation is a sum, no other rule is applicable, and the equation is not in solved form. Here only an outline of this rule is given. The full description can be found in the next section.

Rule Sum is based on an algorithm for unification of sets [BB86,Dro92] with adaptations for terms with sequential composition. Each term a or as_i from $Smd(s)$ may be equaled to a term a or at_i from $Smd(t)$ or attributed to some variables from $Smd(t)$.

A term xs_i from $Smd(s)$ could be a sum. Then $x =^?_{\mathcal{E}} u_1 + \ldots + u_n$ for some n and each term u_js_i may be equaled to a term at_k from $Smd(t)$ or to a v_lt_k with $yt_k \in Smd(t)$, or may be attributed to some variables from $Smd(t)$. So procedure SumRule produces equations $x =^?_{\mathcal{E}} u_1 + \ldots + u_n$ (or $x =^?_{\mathcal{E}} \delta$ if $n = 0$) for terms $xs_i \in Smd(s) \cup Smd(t)$, equations $as_i =^?_{\mathcal{E}} at_j$, $as_i =^?_{\mathcal{E}} vt_j$, $us_i =^?_{\mathcal{E}} at_j$, and $us_i =^?_{\mathcal{E}} vt_j$ for $as_i \in Smd(s)$, $at_j \in Smd(t)$, us_i corresponding to an $xs_i \in Smd(s)$, and vt_j corresponding to a $yt_j \in Smd(t)$. It also produces equations $x =^?_{\mathcal{E}} v_1 + \ldots + v_m + t_1 + \ldots + t_n$ (or $x =^?_{\mathcal{E}} \delta$ if $m = n = 0$) for each $x \in Smd(s) \cup Smd(t)$. Here the v_i are fresh variables, and the t_i are constants or terms $at_i{}'$ from $Smd(s) \cup Smd(t)$, or terms $vt_i{}'$ corresponding to a $yt_i{}'$ in $Smd(s) \cup Smd(t)$. The alternatives contain different subsets of these equations.

1.9 (Sum) Suppose s or t is a sum, no other rule is applicable, and neither $s =^?_{\mathcal{E}} t$ nor $t =^?_{\mathcal{E}} s$ is in solved form. Then:

$\exists\,\vec{u}\,(\,P \cup \{s =^?_{\mathcal{E}} t\}\,) \Longrightarrow \{$ all alternatives $\exists\,\vec{u},\,\vec{v}\,(\,P \cup P_1 \cup P_2 \cup P_3 \cup \text{NewEqs}\,)$ produced by procedure SumRule $\}$,
where
$P_1 \subseteq \{\, x =^?_{\mathcal{E}} \delta \mid x \in Smd(s =^?_{\mathcal{E}} t)$ or $\exists xs' \in Smd(s =^?_{\mathcal{E}} t)\,\}$,
$P_2 \subseteq \{\, x =^?_{\mathcal{E}} u_1 + \ldots + u_k \mid \exists xs' \in Smd(s =^?_{\mathcal{E}} t)\,\}$,
$P_3 \subseteq \{\, s_i =^?_{\mathcal{E}} t_j \mid s_i \equiv as_i{}'$ or $us_i{}',\ t_j \equiv at_j{}'$ or $vt_j{}'\,\}$,
NewEqs $\subseteq \{\, x =^?_{\mathcal{E}} v_1 + \ldots + v_m + t_1 + \ldots + t_n \mid x \in Smd(s =^?_{\mathcal{E}} t)\,\}$.

Examples of applications of rule Sum:

$$\{\, x_1 + x_2 =^?_{\mathcal{E}} yt \,\} \ \stackrel{Sum}{\Longrightarrow}\ \{\ \exists u_1, u_2\,\{\, y =^?_{\mathcal{E}} u_1 + u_2,\ x_1 =^?_{\mathcal{E}} u_1t,\ x_2 =^?_{\mathcal{E}} u_2t \,\}\ \},$$

$$\{\, xs =^?_{\mathcal{E}} at_1 + bt_2 \,\} \ \stackrel{Sum}{\Longrightarrow}\ \{\ \exists u_1, u_2\,\{\, x =^?_{\mathcal{E}} u_1 + u_2,\ u_1s =^?_{\mathcal{E}} at_1,\ u_2s =^?_{\mathcal{E}} bt_2 \,\}\ \},$$

$$\begin{aligned}\{\, x + as =^?_{\mathcal{E}} at \,\} \ \stackrel{Sum}{\Longrightarrow}\ \{\ &\{\, x =^?_{\mathcal{E}} \delta,\ as =^?_{\mathcal{E}} at, \,\},\\ &\{\, x =^?_{\mathcal{E}} at,\ as =^?_{\mathcal{E}} at, \,\}\ \},\end{aligned}$$

$$\{\, x_1 + x_2 =^?_{\mathcal{E}} y + at \,\} \ \stackrel{Sum}{\Longrightarrow}\ \{\ \exists u_1, u_2\,\{\, x_1 =^?_{\mathcal{E}} u_1 + at,\ x_2 =^?_{\mathcal{E}} u_2 + at,\ y =^?_{\mathcal{E}} u_1 + u_2 \,\},$$

$$\exists u_1, u_2 \{ x_1 =^?_{\mathcal{E}} u_1,\ x_2 =^?_{\mathcal{E}} u_2 + at,\ y =^?_{\mathcal{E}} u_1 + u_2 \},$$
$$\exists u_1, u_2 \{ x_1 =^?_{\mathcal{E}} u_1 + at,\ x_2 =^?_{\mathcal{E}} u_2,\ y =^?_{\mathcal{E}} u_1 + u_2 \} \}.$$

3.3 The algorithm UnSeq$_f$

The algorithm reads the problem, normalizes the equations, calls recursively procedures for applications of rules, and produces a set of solutions, which may be empty. If rule NoSol is applicable, then this rule is called first. Some alternatives of rules 1.3 (Seq_ax), 1.4 (Seq_xy), and 1.9 (Sum) produce new variables. These variables are placed on a stack called VarStack. The new variables produced by rule 1.8 (VarSeq$^+$) are not placed on this stack. The stack is used to choose the next equation for reduction. As long as the stack is not empty an equation is chosen that contains the variable present on the top of the stack.
The algorithm is given in a kind of pseudo-Pascal. Of most statements the meaning is clear without explanation. The case statement executes the first clause with a condition that is true. The algorithm describes the order in which rules are executed using the stack VarStack containing new variables, and gives a detailed description of rule 1.9 (Sum).

```
prog UnSeq;
var ∃u⃗P :  Problem;
  VarStack :  Stack;
  Sol :  SetOfSolutions;
begin " read equations and put into normal form ";
  " put normalized equations into P ";
  u⃗ := ∅;
  VarStack := empty;
  Sol := ∅;
  CallRules(∃u⃗P,VarStack,Sol)
end.
```

Procedure CallRules chooses an equation from P and calls the procedure corresponding to the applicable rule. The **case** statement chooses the first clause with a condition that is true. If P contains no reducible equations then it is in solved form and it is placed into Sol. VarStack is a stack of new variables. If v is the top stack symbol and there is an equation $vs =^?_{\mathcal{E}} t$ or $v =^?_{\mathcal{E}} t$ then this equation is chosen. If rule 1.3 (Seq_ax), 1.4 (Seq_xy), or 1.9 (Sum) is applied, then in the alternatives that produce new variables these variables are pushed on VarStack.

```
proc CallRules(∃u⃗P,VarStack,Sol);
  case
    "no rule is applicable":
      "put P into Sol"
    "rule 1.1 (NoSol) is applicable":
      skip
    IsEmpty(VarStack):
        if "rule 0.2 is applicable to some equation s =?_E t"
        then Apply(0.2, ∃u⃗P, s =?_E t, VarStack,Sol)
        else
          "choose an equation s =?_E t on which a rule n is applicable";
          Apply(n, ∃u⃗P, s =?_E t, VarStack, Sol)
```

```
      fi
    not IsEmpty(VarStack):
      v := top(VarStack);
      if "an equation s =?_E t is present in P with s ≡ v or s ≡ vs' on which
        rule n is applicable:"
      then Apply(n, ∃ũP, s =?_E t, VarStack, Sol)
      else pop(VarStack)
    fi
  esac
end.
```

Procedure Apply applies a rule, places new variables on VarStack if needed, and continues the recursion.

```
proc Apply(n, ∃ũP, s =?_E t, VarStack, Sol);
var ∃ũ'P' : Problem;
  P := P − {s =?_E t};
  if n = 1.9 then SumRule(n, ∃ũP, s =?_E t, VarStack, Sol)
else
    foreach alternative of rule n do
      P' := P;
      ũ' := ũ;
      "make new variables and equations";
      "add new variables to ũ";
      if n = 1.3 or n = 1.4 then "push new variables on VarStack" fi;
      "add new equations to P";
      CallRules(∃ũP, VarStack, Sol);
    od
  fi
end.
```

In an application of rule Sum to an equation $s =^?_{\mathcal{E}} t$ each summand of s must be equated to some part of t, and vice versa. A variable summand from s may have an overlap with a variable summand from t. For this overlap a new variable is created. Every non-variable summand must be substituted into one or more variables at the other side, or must be made equal to (parts of) non-variable terms at the other side. A term $xs' \in \mathit{Smd}(s)$ could be a sum, and then $x = u_1 + \ldots + u_n$. Each $u_i s'$ must be attributed to a variable, or equated to a term $at' \in \mathit{Smd}(t)$, or to a term $v_j t'$ corresponding to some $yt' \in \mathit{Smd}(t)$. Also $x =^?_{\mathcal{E}} \delta$ is a possible alternative. A term $as' \in \mathit{Smd}(s)$ could be attributed to one or more variables of $\mathit{Smd}(t)$, or could be equated to a term $at' \in \mathit{Smd}(t)$ or to some vt' corresponding to a $yt' \in \mathit{Smd}(t)$.

Procedure SumRule together with all procedures and functions called from SumRule gives the implementation of rule Sum. Procedure Sum, executed on input $s =^?_{\mathcal{E}} t$, first calls function MakeMatchset, which produces sets of matching pairs of terms. In each pair the first term is a summand from s and the second a summand from t. These pairs are of the form (as', at'), (as', xt'), or (xs', yt') and will be used to make new equations. The pair (as', at') becomes the equation $as' =^?_{\mathcal{E}} at'$. In pairs (as', xt') or (xs', yt') the variables may first be changed before an equation is made. If a term xs' occurs in $n > 1$ pairs, then new variables $u_1, \ldots, u_n$ are chosen, a new equation $x =^?_{\mathcal{E}} u_1 + \ldots + u_n$ is made, and in each pair x is replaced by a unique u_i. Function Triple makes triples consisting of the new set of pairs of terms, and two sets that give for each term $xs' \in \mathit{Smd}(s)$ (resp. $yt' \in \mathit{Smd}(t)$) the set of new variables. Then SumRule creates an alternative for each triple, and makes in each alternative a set P_1 of equations corresponding with each pair of terms in the first component of the triple.

```
proc SumRule(∃ūP, VarStack, s =?_E t, Sol);
  P := P − {s =?_E t};
  MatchSet := MakeMatchSet(s =?_E t);
  Triples := MakeTriples( MatchSet, s =?_E t, VarStack );
  foreach Tri ∈ Triples do
    foreach (s', t') ∈ Tri.Pairs do
      " put s' =?_E t' into P1"
    od;
    P' := P ∪ P1;
    SetsAlg(∃ūP', s =?_E t, Tri, Sol)
  od
end.
```

Procedure MakeMatchSet makes all possible sets of pairs (s_i, t_j) with s_i a summand from s, t_j a summand from t, and both starting with the same constant, or one or both with a variable. If one side of the equation $s =^?_{\mathcal{E}} t$ contains no variable summands then each constant product at the other side must be equated to at least one summand at this side. So in this case only those sets of pairs in which all terms at' of the other side occur at least once are retained.

```
function MakeMatchSet(s =?_E t):  SetOfMatchSets;
  Pairs := ∅;
  Ts := Smd(s)-(A ∪ V);
  Tt := Smd(t)-(A ∪ V);
  foreach (si, tj) ∈ Ts × Tt do
    if Match(si, tj) then " put (si, tj) into Pairs " fi
  od;
  Sets := P(pairs);
  PTs := {asi | a ∈ A and asi ∈ Ts};
  PTt := {atj | a ∈ A and atj ∈ Tt};
  if Smd(s) ∩ V = ∅
  then " remove all sets W from Sets with PTt ⊈ second(W) "
  fi;
  if Smd(t) ∩ V = ∅
  then " remove all sets W from Sets with PTs ⊈ first(W) "
  fi;
  MakeMatchSets := Sets
end.
```

Function Match tests whether two terms may be unifiable. Two terms may be unifiable if they start with the same constant, or at least one of the two starts with a variable.

```
function Match(s, t) :  boolean;
  case
    s = as' and t = at' :  Match := true
    s = as' and t = bt' (a ≢ b) :  Match := false
    s = xs' or t = yt' :  Match := true
```

```
  esac
end.
```

Function MakeTriples has as input the sets of pairs produced by procedure MakeMatchSet. It substitutes via procedure ReplVars the x and y in pairs (xs', t) and (s, yt') by fresh variables. It also makes pairs $(xs', (u_1, \ldots, u_m))$ for each term $xs' \in Smd(s)$ where the $u_1, \ldots, u_m$ are the fresh variables substituted for x, and analogous pairs $(yt', (v_1, \ldots, v_n))$ for each $yt' \in Smd(t)$. For each set of pairs in the input MakeTriples produces a triple of sets. The first produced set contains the (possibly modified) pairs from the input set, the second set contains the produced pairs $(xs', (u_1, \ldots, u_m))$ for all terms $xs' \in Smd(s)$, and the third set contains the produced pairs $(yt', (v_1, \ldots, v_n))$ for all terms $yt' \in Smd(t)$.

```
function MakeTriples(MatchSet, s =?_E t, VarStack):  SetOfTriples;
  V_s := Smd(s) ∩ 𝒱;
  V_t := Smd(t) ∩ 𝒱;
  Triples := ∅;
  foreach MS ∈ MatchSet do
    Tri.Pairs := MS;
    Tri.NewVars_s := ∅;
    Tri.NewVars_t := ∅;
    ReplVars(∃ūP, s =?_E t, Tri, VarStack);
    " put Tri into Triples "
  od;
  MakeTriples := Triples
end.
```

Procedure ReplVars has as input a triple containing a set of pairs and two empty sets. For each summand xs' of s it takes all pairs (xs', t') from the first component of the triple . If there are no such pairs then it puts $x =^?_{\mathcal{E}} \delta$ into P_2. Otherwise it replaces in each pair x by a new variable, puts $(xs'$, "new variables") into the second component of the triple, and puts the modified pair back into the first component of the triple. If there is only one pair containing xs', then one new variable is made. This is needed for procedure DistrVarTerms.
Analogous for each summand yt' of t.

```
proc ReplVars(∃ūP, s =?_E t, Tri, VarStack);
  VT_s := {xs' | x ∈ V and xs' ∈ Smd(s)};
  VT_t := {xt' | x ∈ V and xt' ∈ Smd(t)};
  foreach xs' ∈ VT_s do
    n := #{(xs', t') | (xs', t') ∈ Tri.Pairs };
    case
      n = 0:  if Smd(t) ∩ 𝒱 = ∅ then " put x =?_E δ into P_2" fi
      n > 0:  " make a sequence v⃗ = v_1, ..., v_n of fresh variables ";
        for i := 1 to n do
          " replace a pair (xs', t') in Tri.Pairs by (v_i s', t'); "
          push(v_i, VarStack)
        od;
        " add (xs', {v_1, ..., v_n}) to Tri.NewVars_s "
    esac
  od;
  foreach xt' ∈ VT_t do
```

```
    n := #{(s',xt') | (s',xt') ∈ Tri.Pairs };
    case
      n = 0:  if Smd(s) ∩ V = ∅ then " put x =?_E δ into P_2" fi
      n > 0:  " make a sequence w⃗ = w_1,...,w_n of fresh variables ";
        for i := 1 to n do
          " replace a pair (s',xt') in Tri.Pairs by (s',w_i t'); "
          push(w_i,VarStack)
        od;
        " add (xt',{w_1,...,w_n}) to Tri.NewVars_t"
    esac
  od;
  P := P ∪ P_2
end.
```

Procedure SetsAlg makes equations $x =^?_{\mathcal{E}} t_x$ for variables x in $Smd(s =^?_{\mathcal{E}} t)$. $Smd(t_x)$ may contain new variables, constants, terms at' from $Smd(s =^?_{\mathcal{E}} t)$, and terms ut' corresponding to terms yt' from $Smd(s =^?_{\mathcal{E}} t)$, where the u are new variables.

```
proc SetsAlg(∃u⃗P, s =?_E t, Tri, Sol);
  MakeNewVars(u⃗, s =?_E t, NewEqs);
  DistrVarTerms(∃u⃗P, s =?_E t, Tri, NewEqs);
  DistrConstTerms(∃u⃗P, s =?_E t, Tri, NewEqs, Sol)
end.
```

Procedure MakeNewVars makes new variables $u_{\{x,y\}}$ for each pair (x, y) with $x \in Smd(s)$ and $y \in Smd(t)$ and $x \not\equiv y$, and $u_{\{z\}}$ for each $z \in Smd(s) \cup Smd(t)$. Each new variable $u_{\{x,y\}}$ stands for the intersection of summands substituted into x and into y (see [Dro92]). Each new variable $u_{\{x,y\}}$is attributed to x and to y, and each new variable $u_{\{z\}}$ to z.

```
proc MakeNewVars(s =?_E t, NewEqs);
  V_s := Smd(s) ∩ V; V_t := Smd(t) ∩ V;
  X := V_s − V_t; Y := V_t − V_s; Z := V_s ∩ V_t;

  foreach (x,y) ∈ (X × Y) ∪ (X × Z) ∪ (Y × Z) do
    " choose fresh variables v_{x,y} and add them to u⃗";
  od;
  foreach z ∈ Z do
    " choose fresh variables v_{z} and add them to u⃗";
  od;

  foreach x ∈ X do
    " put x =?_E Nf(Σ_{y∈Y} v_{x,y} + Σ_{z∈Z} v_{x,z}) into NewEqs "
  od;
  foreach y ∈ Y do
    " put y =?_E Nf(Σ_{x∈X} v_{x,y} + Σ_{z∈Z} v_{y,z}) into NewEqs "
  od;
  foreach z ∈ Z do
    " put z =?_E Nf(v_{z} + Σ_{x∈X} v_{x,z} + Σ_{y∈Y} v_{y,z}) into NewEqs "
  od
end.
```

Procedure DistrVarTerms makes for each term xs' in $Smd(t)$ another set of new variables, one for each variable $y \in Smd(s)$. (The first set of new variables was made by procedure MakeTriples.) The idea is again that x is a sum, and to each $y_i \in Smd(t)$ a term $v_i s$ is attributed. (v_i may be δ.) To P_3 an equation $x =^?_{\mathcal{E}} t_x$ is added, where t_x is the sum of all new variables created for x in xs', both here and in procedure MakeTriples.
Analogous for terms yt' in $Smd(s)$.

```
proc DistrVarTerms(∃u⃗P, s =?_E t, Tri, NewEqs);
  V_s := Smd(s) ∩ 𝒱; V_t := Smd(t) ∩ 𝒱;
  X := V_s − V_t; Y := V_t − V_s; Z := V_s ∩ V_t;
  VT_s := {vs' | v ∈ 𝒱, vs' ∈ Smd(s)};
  VT_t := {vt' | v ∈ 𝒱, vt' ∈ Smd(t)};
  n_s := #V_s; n_t := #V_t;
  foreach vs' ∈ VT_s do
    if (vs', V) ∈ Tri.NewVars_s then NV := V else NV := ∅ fi;
    if n_t > 0 then
    " choose a set FV = {v_1, ..., v_{n_t}} of fresh variables ";
      foreach y_i ∈ V_t do
        " replace y_i =?_E t_i in NewEqs by y_i =?_E Nf(t_i + v_i s') "
      od
    else FV := ∅
    fi;
    if NV ∪ FV ≠ ∅ then
      " add v =?_E Nf(Σ(NV ∪ FV)) to P_3";
      " add NV ∪ FV to u⃗"
    fi
  od;

  foreach wt' ∈ VT_t do
    if (wt', V) ∈ Tri.NewVars_t then NV := V else NV := ∅ fi;
    if n_s > 0 then
    " choose a set FV = {w_1, ..., w_{n_s}} of fresh variables ";
      foreach x_i ∈ V_s do
        " replace x_i =?_E t_i in NewEqs by x_i =?_E Nf(t_i + w_i t') "
      od
    else FV := ∅
    fi;
    if NV ∪ FV ≠ ∅ then
      " add w =?_E Nf(Σ(NV ∪ FV)) to P_3";
      " add NV ∪ FV to u⃗"
    fi
  od;
  P := P ∪ P_3
end.
```

Procedure DistrConstTerms may make a number of alternatives. In each alternative the constants and constant products from $Smd(s)$ are attributed to a different set of variables from $Smd(t)$, and vice versa. This is done in a different way for single and double constants and single and double constant products. A double constant is a constant that occurs in both $Smd(s)$ and $Smd(t)$, a single constant occurs in either $Smd(s)$ or $Smd(t)$. A double constant product occurs in a pair of Tri.Pairs, a single constant product occurs in $Smd(s) \cup Smd(t)$ but not in a pair of Tri.pairs. Each single constant or single constant product from $Smd(s)$ is attributed to a nonempty subset of variables from $Smd(t) - Smd(s)$, or to one variable from $Smd(s) \cap Smd(t)$, and analogous for single constants and single constant products from $Smd(t)$. Each double constant or double constant product is attributed to a subset of variables from $Smd(s)$ or $Smd(t)$.

proc `DistrConstTerms(`$\exists\vec{u}P$`, `$s =^?_{\mathcal{E}} t$`, Tri, NewEqs, Sol);`
  V_s `:=` $Smd(s) \cap \mathcal{V}$; V_t `:=` $Smd(t) \cap \mathcal{V}$;
  X `:=` $V_s - V_t$; Y `:=` $V_t - V_s$; Z `:=` $V_s \cap V_t$;
  C_D `:=` $Smd(s) \cap Smd(t) \cap \mathcal{A}$;
  C_L `:=` $(Smd(s) - Smd(t)) \cap \mathcal{A}$;
  C_R `:=` $(Smd(t) - Smd(s)) \cap \mathcal{A}$;
  T_{D_L} `:=` $\{ as' \mid (as', t') \in$ `Tri.Pairs for some` $t' \}$;
  T_{D_R} `:=` $\{ at' \mid (s', at') \in$ `Tri.Pairs for some` $s' \}$;
  T_D `:=` $(T_{D_L} \cup T_{D_R}) - \{ at' \mid (as', at') \in$ `Tri.Pairs and` $at' \notin Smd(s) \}$;
  T_L `:=` $\{ as' \in Smd(s) \mid as' \notin T_{D_L}) \}$;
  T_R `:=` $\{ at' \in Smd(t) \mid at' \notin T_{D_R}) \}$;
  $\mathtt{Sets}_L$ `:=` $(\mathcal{P}(Y) - \emptyset) \cup \{ \{z\} \mid z \in Z \}$;
  $\mathtt{Sets}_R$ `:=` $(\mathcal{P}(X) - \emptyset) \cup \{ \{z\} \mid z \in Z \}$;
  $\mathtt{Sets}_D$ `:=` $\mathcal{P}(X) \cup \mathcal{P}(Y)$;

  `Suppose` $T_L \cup C_L = \{ t_1, \ldots, t_k \}$, $T_R \cup C_R = \{ t_{k+1}, \ldots, t_m \}$, $T_D \cup C_D = \{ t_{m+1}, \ldots, t_n \}$.
  foreach `sequence` $(S_1, \ldots, S_k, S_{k+1}, \ldots, S_m, S_{m+1}, \ldots, S_n)$
    with $S_1, \ldots, S_k \in \mathtt{Sets}_L$, $S_{k+1}, \ldots, S_m \in \mathtt{Sets}_R$, $S_{m+1}, \ldots, S_n \in \mathtt{Sets}_D$
  do
    for `i:=1` **to** n **do**
      foreach $x \in S_i$ **do**
        `" replace` $x =^?_{\mathcal{E}} t_x$ `in NewEqs by` $x =^?_{\mathcal{E}} \mathtt{Nf}(t_x + t_i)$ `"`
      od
    od;
    P `:=` $P \cup$ `NewEqs;`
    `CallRules(`$\exists\vec{u}P$`, Sol)`
  od
end.

The algorithm produces a set Sol, containing sets $\{ x =^?_{\mathcal{E}} t_x \mid x \in X \}$ that are in solved form. The quantification of these sets is omitted, because these variables do not occur any more as the left side of an equation. A set $\{ x =^?_{\mathcal{E}} t_x \mid x \in X \}$ is easily converted to a solution by replacing each $x =^?_{\mathcal{E}} t_x$ by $x := [\![t_x]\!]$.

Chapter 4

Properties of UnSeq_f

4.1 Correctness of UnSeq_f

The algorithm is correct if the set of rules is complete and each rule is sound.

Definition 4.1 *A set of unification rules is* **complete** *iff for all problems $\exists\,\vec{u}\,P$: if P is not in solved form then at least one rule is applicable.*

Definition 4.2 *A rule $\exists\,\vec{u}\,P \Longrightarrow \{\exists\,\vec{u_i}\,P_i \mid i \in I\}$ is* **sound** *iff $Sol(\,\exists\,\vec{u}\,P\,) = \bigcup_{i\in I} Sol(\,\exists\,\vec{u_i}\,P_i\,)$.*

First it is proved that the algorithm UnSeq_f is complete.

Proposition 4.1 *If the algorithm $UnSeq_f$ produces an intermediate result $\exists\,\vec{u}\,P$ that is not a solved form, then at least one rule is applicable to $\exists\,\vec{u}\,P$.*

Proof:
Suppose $P = \{s_i =^?_{\mathcal{E}} t_i \mid i \in I\,\}$ and $\exists\vec{u}P$ is not in solved form. Several cases are distinguished:
1) Each s_i is a variable x_i.
1.1) There are i, j such that $x_i \equiv x_j$ (and $t_i \not\equiv t_j$).
If $x_i \notin Var(t_i)$ then rule 0.2 (Subs) is applicable. Otherwise see 1.2.
1.2) There are i, j such that x_i occurs in t_j.
If $i \neq j$ and $x_i \notin Var(t_i)$ then rule 0.2 (Subs) is applicable. If x_i occurs in t_i then x_i may occur in $Smd(t_i)$, in a term $x_i t'$ in $Smd(t_i)$, or somewhere else in t_i. If there is a term $x_i t'$ in $Smd(t_i)$ then rule 1.7 (VarSeq) or 1.8 (VarSeq^+) is applicable. If $x_i \in Smd(t_i)$ and there is no term $x_i t'$ in $Smd(t_i)$ then rule 0.1 (Eq) or 1.9 (Sum) is applicable. If x_i only occurs somewhere else in t_i then rule 1.3 (Occur) is applicable.
2) There is at least one equation $s_i =^?_{\mathcal{E}} t_i$ with s_i not a variable. Check all possible cases for s_i and t_i. □

Theorem 4.2 *The rules 0.1 to 0.4 and 1.1 to 1.8 are sound.*

Proof:
Suppose α is a solution of $\exists\,\vec{u}\,P$. Define $V = Var(P) - \vec{u}$, $\alpha' = \alpha_{|V}$. Then there is an assignment β with $\text{Dom}(\beta) \subseteq \vec{u}$ such that $[\![s]\!]\alpha'\beta = [\![t]\!]\alpha'\beta$ for all $s =^?_{\mathcal{E}} t \in P$.
0.1 (Eq), 0.2 (Subs), 0.4 (AuxVar), 1.1 (NoSol), 1.2 (Seq_aa), 1.5 (Seq_xx), and 1.6 (Seq_xδ): easy.
0.3 (Occur).
Suppose rule 0.3 is applied to $\exists\,\vec{u}\,(\,P \cup \{x =^?_{\mathcal{E}} t_x\}\,)$ and t_x has no summand x or xt'. Define

$Y = \{ y_1, \ldots, y_n \}$ as the set of all variables occurring before the occurrence of x in t_x, and define $n = \#Y$.
Suppose $n = 0$.
Then it must be proved that $\exists\, \vec{u}\, (P \cup \{s =^?_{\mathcal{E}} t\})$ has no solutions.
So suppose α is a solution of $\exists\, \vec{u}\, (P \cup \{s =^?_{\mathcal{E}} t\})$ and α' and β are as above. Then $[\![x]\!]\alpha'\beta = [\![t_x]\!]\alpha'\beta$ and there is a path $a_1, \ldots, a_m$ from the root of $[\![t_x]\!]$ to the occurrence of x in $[\![t_x]\!]$, with $m > 0$. So there is an infinite path in $[\![t_x]\!]\alpha'\beta$. But $\mathcal{P}$ only contains processes with finite paths. Contradiction. So $\exists\, \vec{u}\, P$ has no solution.
Suppose $n > 0$:
Then rule 0.3 produces n alternatives $\exists\, \vec{u}, v\, (P \cup \{ x =^?_{\mathcal{E}} \mathrm{Nf}(t_x\{y := v\delta\}),\ y =^?_{\mathcal{E}} v\delta \})$. Now it must be proved that:
1) If α is a solution of $\exists\, \vec{u_i}\, P_i$ for some i with $1 \leq i \leq n$, then α is a solution of $\exists\, \vec{u}\, P$.
Proof: easy.
2) If α is a solution of $\exists\, \vec{u}\, P$ then there is an i with $1 \leq i \leq n$ such that α is a solution of $\exists\, \vec{u_i}\ P_i$.
Proof:
Suppose there is no such i. Then for all $y \in Y$ α is not a solution of $\exists v\, \{y =^?_{\mathcal{E}} v\delta\}$. So for each $y \in Y$ there is no β such that $[\![y]\!]\alpha'\beta = [\![v\delta]\!]\alpha'\beta$. But α is a solution of $\exists\, \vec{u}\, P$, so there is a β with $[\![x]\!]\alpha'\beta = [\![t_x]\!]\alpha'\beta$, and for all $y \in Y : [\![y]\!]\alpha'\beta \neq [\![v\delta]\!]\alpha'\beta$.
Take some γ that substitutes constants for all variables in $[\![x]\!]\alpha'\beta$ and $[\![t_x]\!]\alpha'\beta$. Then $[\![x]\!]\alpha'\beta\gamma = [\![t_x]\!]\alpha'\beta\gamma$. And also for all $y \in Y : [\![y]\!]\alpha'\beta\gamma \neq [\![v\delta]\!]\alpha'\beta\gamma$.
Now look at the path in $[\![t_x]\!]$ from the root to the occurrence of x. Say this path is $d_1, \ldots, d_m$ with all $d_j \in \mathcal{A} \cup \mathcal{V}$. Then for each d_j: $[\![d_j]\!]\alpha'\beta\gamma$ has length > 0, contains no variables, and terminates successfully. So $[\![t_x]\!]\alpha'\beta\gamma$ has an infinite path. Contradiction. So there is an i such that α is a solution of $\exists\, \vec{u_i}\ P_i$.
<u>1.3 (Seq_ax)</u>:
Let rule 1.3 be applied to $\exists\, \vec{u}\, (P \cup \{xs =^?_{\mathcal{E}} at\})$ producing $\exists\, \vec{u}\, (P \cup \{x =^?_{\mathcal{E}} a, s =^?_{\mathcal{E}} t\})$ and $\exists\, \vec{u}, v\, \{ P \cup \{ x =^?_{\mathcal{E}} av,\ vs =^?_{\mathcal{E}} t \})$. Suppose α is a solution of $\exists\, \vec{u}\, (P \cup \{ x =^?_{\mathcal{E}} a,\ s =^?_{\mathcal{E}} t \})$, and α' and β are defined as before. Then $[\![x]\!]\alpha'\beta = [\![a]\!]$ and $[\![s]\!]\alpha'\beta = [\![t]\!]\alpha'\beta$, so $[\![xs]\!]\alpha'\beta = [\![at]\!]\alpha'\beta$.
Suppose α is a solution of $\exists\, \vec{u}, v\, (P \cup \{ x =^?_{\mathcal{E}} av,\ vs =^?_{\mathcal{E}} t \})$, and α' and β are defined as before. Then $[\![x]\!]\alpha'\beta = [\![av]\!]\alpha'\beta$ and $[\![vs]\!]\alpha'\beta = [\![t]\!]\alpha'\beta$, so $[\![xs]\!]\alpha'\beta = [\![x]\!]\alpha'\beta \odot [\![s]\!]\alpha'\beta = [\![av]\!]\alpha'\beta \odot [\![s]\!]\alpha'\beta = [\![a]\!]\alpha'\beta \odot [\![vs]\!]\alpha'\beta = [\![a]\!]\alpha'\beta \odot [\![t]\!]\alpha'\beta = [\![at]\!]\alpha'\beta$. Define $\beta' = \beta_{|\vec{u}}$. Then also $[\![xs]\!]\alpha'\beta' = [\![at]\!]\alpha'\beta'$.
Suppose α is a solution of $\exists\, \vec{u}\, (P \cup \{ xs =^?_{\mathcal{E}} at \})$, and α' and β are defined as before. Then $[\![xs]\!]\alpha'\beta = [\![at]\!]\alpha'\beta = \{ (a, [\![t]\!]\alpha'\beta) \}$. There are two possibilities:
1) $[\![x]\!]\alpha'\beta = \{ (a, \{\surd\}) \}$ and $[\![s]\!]\alpha'\beta = [\![t]\!]\alpha'\beta$. Then α is a solution of $\exists\, \vec{u}\, (P \cup \{x =^?_{\mathcal{E}} a,\ s =^?_{\mathcal{E}} t\})$.
2) $[\![x]\!]\alpha'\beta = \{ (a, p) \}$ for some process p, and $p \odot [\![s]\!]\alpha'\beta = [\![t]\!]\alpha'\beta$.
Take a fresh variable v and define β' as $\beta \cup \{v := p\}$. Then $[\![x]\!]\alpha'\beta' = \{ (a, p) \} = \{ (a, [\![v]\!]\alpha'\beta') \} = [\![av]\!]\alpha'\beta$, and $[\![vs]\!]\alpha'\beta' = [\![v]\!]\alpha'\beta' \odot [\![s]\!]\alpha'\beta = p \odot [\![s]\!]\alpha'\beta = [\![t]\!]\alpha'\beta$. So α is a solution of $\exists\, \vec{u}, v\, (P \cup \{ x =^?_{\mathcal{E}} av,\ vs =^?_{\mathcal{E}} t \})$.
<u>1.4 (Seq_xy)</u>:
Let rule 1.4 be applied to $\exists\, \vec{u}\, (P \cup \{xs =^?_{\mathcal{E}} yt\})$. Suppose α is a solution of one of the alternatives: similar as in 1.3.
Suppose α is a solution of $\exists\, \vec{u}(P \cup \{xs =^?_{\mathcal{E}} yt\})$, and α' and β are defined as before. Then $[\![xs]\!]\alpha'\beta = [\![yt]\!]\alpha'\beta$. So $[\![x]\!]\alpha'\beta \odot [\![s]\!]\alpha'\beta = [\![y]\!]\alpha'\beta \odot [\![t]\!]\alpha'\beta$, and there are four possibilities:
1) $[\![x]\!]\alpha'\beta = [\![y]\!]\alpha'\beta$, and $[\![x]\!]\alpha'\beta$ and $[\![y]\!]\alpha'\beta$ have both no successful termination option,
2) $[\![x]\!]\alpha'\beta = [\![y]\!]\alpha'\beta$, and $[\![s]\!]\alpha'\beta = [\![t]\!]\alpha'\beta$,
3) $[\![x]\!]\alpha'\beta = [\![y]\!]\alpha'\beta \odot p$ for some process p, and $p \odot [\![s]\!]\alpha'\beta = [\![t]\!]\alpha'\beta$,

4) $[\![x]\!]\alpha'\beta \odot p = [\![y]\!]\alpha'\beta$ for some process p, and $[\![s]\!]\alpha'\beta = p \odot [\![t]\!]\alpha'\beta$.
1.7 (VarSeq):
Let rule 1.7 be applied to $\exists\,\vec{u}\,(\,P \cup \{\, x =^?_{\mathcal{E}} xt \,\})$ producing $\exists\,\vec{u}, v\,(\,P \cup \{\, x =^?_{\mathcal{E}} v\delta \,\})$.
Suppose α is a solution of $\exists\,\vec{u}, v\,(\,P \cup \{\, x =^?_{\mathcal{E}} v\delta \,\})$. Then it must be proved that α is a solution of $\exists\,\vec{u}\,(\,P \cup \{\, s =^?_{\mathcal{E}} t \,\})$. Proof: easy.
Suppose α is a solution of $\exists\,\vec{u}\,(\,P \cup \{\, s =^?_{\mathcal{E}} t \,\})$. Then it must be proved that α is a solution of $\exists\,\vec{u}, v\,(\,P \cup \{\, x =^?_{\mathcal{E}} v\delta \,\})$. Proof:
Suppose α' and β are defined as before. Then $[\![x]\!]\alpha'\beta = [\![xt]\!]\alpha'\beta$.
Suppose $[\![x]\!]\alpha'\beta$ has a successful termination option.
Suppose $[\![t]\!]\alpha'\beta$ has no successful termination option. Then $[\![x]\!]\alpha'\beta \neq [\![xt]\!]\alpha'\beta$. So suppose $[\![t]\!]\alpha'\beta$ has a successful termination option. Suppose the maximum length of a path from the root to a successful termination point in $[\![x]\!]\alpha'\beta$ is m and in $[\![t]\!]\alpha'\beta$ is n. Then $n > 0$, and the maximum length of a path to a successful termination point in $[\![xs']\!]\alpha'\beta$ is $m + n \neq m$, so $[\![x]\!]\alpha'\beta \neq [\![xt]\!]\alpha'\beta$. So $[\![x]\!]\alpha'\beta$ has no successful termination option, and $[\![x]\!]\alpha'\beta = [\![x]\!]\alpha'\beta \odot [\![\delta]\!]$. Define $\beta' = \beta \cup \{v := [\![x]\!]\alpha'\beta\}$. Then $[\![x]\!]\alpha'\beta' = [\![v\delta]\!]\alpha'\beta'$, so α is a solution of $\exists\,\vec{u}, v\,(P \cup \{x =^?_{\mathcal{E}} v\delta\})$.
1.8 (VarSeq$^+$):
Let rule 1.8 be applied to $\exists\,\vec{u}\,(\,P \cup \{\, x =^?_{\mathcal{E}} t_1 + x \cdot t_2 \,\})$, producing $\exists\,\vec{u}, v\,(\,P \cup \{\, x =^?_{\mathcal{E}} t_1\delta + v\delta \,\})$ and $\exists\,\vec{u}, v_1, v_2\,(\,P \cup \{\, x =^?_{\mathcal{E}} t_1 + v_1,\ v_1 =^?_{\mathcal{E}} v_2\delta + t_1t_2\delta \,\})$.
Suppose α is a solution of $\exists\,\vec{u}, v\,(\,P \cup \{\, x =^?_{\mathcal{E}} t_1\delta + v\delta \,\})$. Then α must be a solution of $\exists\,\vec{u}\,(\,P \cup \{s =^?_{\mathcal{E}} t\})$. Proof: easy.
Suppose α is a solution of $\exists\,\vec{u}, v_1, v_2\,(\,P \cup \{\, x =^?_{\mathcal{E}} t_1 + v_1,\ v_1 =^?_{\mathcal{E}} v_2\delta + t_1t_2\delta \,\})$. Then α must be a solution of $\exists\,\vec{u}\,(\,P \cup \{\, s =^?_{\mathcal{E}} t \,\})$. Proof: easy.
Suppose α is a solution of $\exists\,\vec{u}\,(\,P \cup \{\, x =^?_{\mathcal{E}} t_1 + xt_2 \,\})$. Then there must be an alternative $\exists\,\vec{u_i}\,P_i$ such that α is a solution of $\exists\,\vec{u_i}\,P_i$. Proof:
Suppose α' and β are defined as before. Then $[\![x]\!]\alpha'\beta = [\![t_1 + xt_2]\!]\alpha'\beta$, so $[\![t_1]\!]\alpha'\beta \subseteq [\![x]\!]\alpha'\beta$.
Suppose $[\![x]\!]\alpha'\beta$ has no successful termination option. Then there is a process p such that $[\![x]\!]\alpha'\beta = ([\![t_1]\!]\alpha'\beta \oplus p) \odot [\![\delta]\!]$. Define $\beta' = \beta \cup \{\, v := p \,\}$. Then $[\![x]\!]\alpha'\beta' = [\![(t_1 + v)\delta]\!]\alpha'\beta'$. So α is a solution of $\exists\,\vec{u}, v\,(\,P \cup \{\, x =^?_{\mathcal{E}} t_1\delta + v\delta \,\})$.
Suppose $[\![x]\!]\alpha'\beta$ has a successful termination option. If $[\![t_2]\!]\alpha'\beta$ has a successful termination option, then $[\![t_2]\!]$ has a path of length ≥ 1 to a successful termination point, and $[\![xt_2]\!]\alpha'\beta \subseteq [\![x]\!]\alpha'\beta$, so $[\![x]\!]\alpha'\beta$ has a path of infinite length. So $[\![t_2]\!]\alpha'\beta$ has no successful termination option. Now define $p = [\![xt_2]\!]\alpha'\beta$, and $\beta' = \beta \cup \{v_1 := p,\ v_2 := p\}$. Then $[\![t_1 + v_1]\!]\alpha'\beta' = [\![t_1]\!]\alpha'\beta' \oplus [\![xt_2]\!]\alpha'\beta' = [\![x]\!]\alpha'\beta'$, $[\![v_2\delta + t_1t_2\delta]\!]\alpha'\beta' = ([\![xt_2]\!]\alpha'\beta' \oplus [\![t_1t_2]\!]\alpha'\beta') \odot [\![\delta]\!] = [\![(x + t_1)t_2]\!]\alpha'\beta' \odot [\![\delta]\!] = [\![xt_2]\!]\alpha'\beta' = [\![v_1]\!]\alpha'\beta'$. So α is a solution of $\exists\,\vec{u}, v_1, v_2\,(\,P \cup \{\, x =^?_{\mathcal{E}} t_1 + v_1,\ v_1 =^?_{\mathcal{E}} v_2\delta + t_1t_2\delta \,\})$. $\square$

Theorem 4.3 *Rule 1.9 is sound.*

Proof:
Let rule 1.9 be applied to $\exists\,\vec{u}\,(\,P \cup \{s =^?_{\mathcal{E}} t\}\,)$. Then it must be proved that the set of solutions of the original problem is equal to the union of the sets of solutions of the produced alternatives.
In I) it is proved that a solution of an alternative is also a solution of the original problem.
In II) it is proved that for each solution of the original problem there is an alternative such that this solution is also a solution of the alternative.
I) Suppose α is a solution of an alternative $\exists\,\vec{u},\,\vec{v}\,(\,P \cup P_1 \cup P_2 \cup P_3 \cup \text{NewEqs}\,)$ produced by rule 1.9, where
P is the original set of equations minus $s =^?_{\mathcal{E}} t$,
P_1 contains the equations $s_i =^?_{\mathcal{E}} t_j$ produced by procedure SumRule,

P_2 contains the equations $x =^?_\mathcal{E} \delta$ produced by procedure ReplVars,
P_3 contains the equations $x =^?_\mathcal{E} x_1 + \ldots + x_n$ produced by procedure DistrVarTerms, and
NewEqs contains the equations $x =^?_\mathcal{E} t_x$ produced by procedure SetsAlg.
Define $V = \mathit{Var}(P \cup \{s =^?_\mathcal{E} t\}) - \vec{u}$, $\alpha' = \alpha_{|V}$. Then there is an assignment β with $\mathrm{Dom}(\beta) \subseteq \vec{u} \cup \vec{v}$ such that for all $s' =^?_\mathcal{E} t' \in P \cup P_1 \cup P_2 \cup P_3 \cup$ NewEqs : $[\![s']\!]\alpha'\beta = [\![t']\!]\alpha'\beta$.
Now it must be proved that $[\![s]\!]\alpha'\beta = [\![t]\!]\alpha'\beta$.
First I will prove that $[\![s]\!]\alpha'\beta \subseteq [\![t]\!]\alpha'\beta$. Consider all possibilities for $s_i \in \mathit{Smd}(s)$:

1) s_i is a constant a.
   Then $a \in \mathit{Smd}(t)$, or there is an $y \in \mathit{Smd}(t)$ with $y =^?_\mathcal{E} t_y \in$ NewEqs and $a \in \mathit{Smd}(t_y)$ (see procedure DistrConstTerms). So $[\![a]\!] \subseteq [\![t]\!]\alpha'\beta$.
2) s_i is a term as_i'. Three cases:
   a) There is an equation $as_i' =^?_\mathcal{E} at_j'$ in P_1.
      Then $at_j' \in \mathit{Smd}(t)$ (see procedures SumRule and MakeMatchSet, functions Match and MakeTriples, and procedure ReplVars) and $[\![as_i']\!]\alpha'\beta = [\![at_j']\!]\alpha'\beta$, so $[\![s_i']\!]\alpha'\beta = [\![t_j']\!]\alpha'\beta$.
   b) There is an equation $as_i' =^?_\mathcal{E} v_k t_j$ in P_1.
      Then there is an equation $u =^?_\mathcal{E} v_1 + \ldots + v_k + \ldots + v_n$ in P_3 for some $n \geq 0$ (see procedure ReplVars), and $ut_j \in \mathit{Smd}(t)$ (see functions MakeMatchSet and MakeTriples). So $[\![as_i']\!]\alpha'\beta \subseteq [\![t]\!]\alpha'\beta$.
   c) There is no equation $as_i' =^?_\mathcal{E} t_j$ in P_1.
      Then $as_i \in T_L$ (see procedure DistrConstTerms), so there is an $y \in \mathit{Smd}(t)$ with $y =^?_\mathcal{E} t_y \in$ NewEqs and $as_i \in \mathit{Smd}(t_y)$. Then $[\![as_i']\!]\alpha'\beta \subseteq [\![t_y]\!]\alpha'\beta = [\![y]\!]\alpha'\beta \subseteq [\![t]\!]\alpha'\beta$.
3) s_i is a term xs_i'. Two cases:
   a) There is an equation $x =^?_\mathcal{E} \delta$ in P_2.
      Then $[\![xs_i']\!]\alpha'\beta = [\![\delta]\!] \subseteq [\![t]\!]\alpha'\beta$.
   b) There is an equation $x =^?_\mathcal{E} x_1 + \ldots + x_n$ in P_3 for an $n > 0$.
      Then for each x_j there is an equation $x_j s_i' =^?_\mathcal{E} t_k$ in P_1 (see procedure ReplVars) or there is an $y \in \mathit{Smd}(t)$ with $y =^?_\mathcal{E} t_y \in$ NewEqs and $x_j s_i' \in \mathit{Smd}(t_y)$ (see procedure DistrVarTerms). In the last case $[\![x_j s_i']\!]\alpha'\beta \subseteq [\![t]\!]\alpha'\beta$. If $x_j s_i' =^?_\mathcal{E} t_k \in P_1$, then $t_k = at_k'$ and $at_k' \in \mathit{Smd}(t)$ (see functions MakeMatchSet and MakeTriples) or $t_k = v_l t_k'$, there is an equation $u =^?_\mathcal{E} v_1 + \ldots + v_l + \ldots + v_m$ in P_3 for some $m \geq 0$, and $ut_k \in \mathit{Smd}(t)$. Then also $[\![x_j s_i']\!]\alpha'\beta \subseteq [\![t]\!]\alpha'\beta$. So $[\![xs_i']\!]\alpha'\beta \subseteq [\![t]\!]\alpha'\beta$.
4) s_i is a variable x.
   Then NewEqs contains an equation $x =^?_\mathcal{E} t_x$ and $[\![x]\!]\alpha'\beta = [\![t_x]\!]\alpha'\beta$.
   Now I prove that $[\![t_x]\!]\alpha'\beta \subseteq [\![t]\!]\alpha'\beta$.
   Suppose $t_i \in \mathit{Smd}(t_x)$. Four cases:
   a) t_i is a variable v.
      Then there is a variable y in $\mathit{Smd}(t)$ with $y =^?_\mathcal{E} t_y \in$ NewEqs and $v \in \mathit{Smd}(t_y)$ (see procedure MakeNewVars). So $[\![t_i]\!]\alpha'\beta \subseteq [\![t_y]\!]\alpha'\beta = [\![y]\!]\alpha'\beta \subseteq [\![t]\!]\alpha'\beta$.
   b) t_i is a constant a.
      Then $a \in \mathit{Smd}(t)$, or $a \in \mathit{Smd}(s)$ and x occurs also in $\mathit{Smd}(t)$. Then $[\![t_i]\!]\alpha'\beta \subseteq [\![t]\!]\alpha'\beta$.
   c) t_i is a term at_i'. Two cases:
      i) $at_i' \in T_R$ (see procedure DistrConstTerms).
         Then $at_i' \in \mathit{Smd}(t)$, so $[\![at_i']\!]\alpha'\beta \subseteq [\![t]\!]\alpha'\beta$.
      ii) $at_i' \in T_D$ (see procedure DistrConstTerms).
         Then there is an equation $at_i' =^?_\mathcal{E} t_j$ in P_1. If $t_j \equiv at_j'$ then $t_j \in \mathit{Smd}(t)$ (see functions MakeMatchPairs and MakeTriples). If $t_j \equiv v_k t_j'$ then there is an equation $u =^?_\mathcal{E} v_1 + \ldots + v_k + \ldots + v_n$ in P_3 for some $n \geq 0$, and $ut_j' \in \mathit{Smd}(t)$ (see procedures ReplVars and DistrVarTerms). So $[\![at_i']\!]\alpha'\beta \subseteq [\![t]\!]\alpha'\beta$.

d) t_i is a term $v_k t_i{}'$.

Then there is an equation $u =^?_{\mathcal{E}} v_1 + \ldots + v_k + \ldots + v_n$ in P_3 for some $n \geq 1$, and $ut_i{}' \in \mathit{Smd}(t)$ (see procedure DistrVarTerms). So $[\![v_k t_i{}']\!]\alpha'\beta \subseteq [\![ut_j{}']\!]\alpha'\beta \subseteq [\![t]\!]\alpha'\beta$.

So $[\![t_x]\!]\alpha'\beta \subseteq [\![t]\!]\alpha'\beta$.

So $[\![s]\!]\alpha'\beta \subseteq [\![t]\!]\alpha'\beta$. In the same way: $[\![t]\!]\alpha'\beta \subseteq [\![s]\!]\alpha'\beta$, so $[\![s]\!]\alpha'\beta = [\![t]\!]\alpha'\beta$. Define $\beta' = \beta_{|\vec{u}}$. Then also $[\![s]\!]\alpha'\beta' = [\![t]\!]\alpha'\beta'$, so α is a solution of $\exists\, \vec{u}\, (\, P \cup \{ s =^?_{\mathcal{E}} t \}\,)$.

II) Suppose α is a solution of $\exists\, \vec{u}\, (\, P \cup \{ s =^?_{\mathcal{E}} t \}\,)$. Then it must be proved that there is an alternative P_i such that α is a solution of P_i. Now P_i will be constructed as $P_i = \exists\, \vec{u}, \vec{v}\, (\, P \cup P_1 \cup P_2 \cup P_3 \cup \text{NewEqs}\,)$, where $\vec{v}$, P_1, P_2, P_3, and NewEqs will be defined in the following.

First P_2 is defined:

$$P_2 = \{\, x =^?_{\mathcal{E}} \delta \mid [\![x]\!]\alpha'\beta = [\![\delta]\!] \text{ and } xt' \in \mathit{Smd}(s =^?_{\mathcal{E}} t) \text{ for some } t' \,\}.$$

Now NewEqs will be defined and triples will be made that will later be used for making equations for P_3. Define $V_s = \mathit{Smd}(s) \cap \mathcal{V}$, $V_t = \mathit{Smd}(t) \cap \mathcal{V}$, $C_s = \mathit{Smd}(s) \cap \mathcal{A}$, $C_t = \mathit{Smd}(t) \cap \mathcal{A}$, $T_s = \mathit{Smd}(s) - (\mathcal{A} \cup \mathcal{V})$, $T_t = \mathit{Smd}(t) - (\mathcal{A} \cup \mathcal{V})$, $X = V_s - V_t$, $Y = V_t - V_s$, $Z = V_s \cap V_t$, $C_L = C_s - C_t$, $C_R = C_t - C_s$, $C_D = C_s \cap C_t$. Choose sets $\vec{v}_1$ and $\vec{v}_2$ of fresh variables with

$$\vec{v}_1 = \{\, v_{\{x,y\}} \mid x \in X,\ y \in Y \cup Z \,\} \cup \{\, v_{\{y,z\}} \mid y \in Y,\ z \in Z \,\} \cup \{\, v_{\{z\}} \mid z \in Z \,\}, \text{ and}$$

$$\vec{v}_2 = \{\, v_{xs',y} \mid xs' \in T_s,\, y \in V_t \,\} \cup \{\, v_{yt',x} \mid yt' \in T_t,\, x \in V_s \,\}.$$

Add the fresh variables in $\vec{v}_1$ and $\vec{v}_2$ to $\vec{v}$. Now define:

$$\begin{array}{rcl}
T_{D_L} & = & \{\, as' \in T_s \mid \exists at' \in T_t : [\![as']\!]\alpha'\beta = [\![at']\!]\alpha'\beta \,\} \\
& \cup & \{\, as' \in T_s \mid \exists yt' \in T_t : [\![as']\!]\alpha'\beta \subseteq [\![yt']\!]\alpha'\beta \,\}, \\
T_{D_R} & = & \{\, at' \in T_t \mid \exists as' \in T_s : [\![at']\!]\alpha'\beta = [\![as']\!]\alpha'\beta \,\} \\
& \cup & \{\, at' \in T_t \mid \exists xs' \in T_s : [\![at']\!]\alpha'\beta \subseteq [\![xs']\!]\alpha'\beta \,\},
\end{array}$$

$$\begin{array}{rcl}
T_L & = & \{\, as' \in T_s \mid as' \notin T_{D_L} \,\}, \\
T_R & = & \{\, at' \in T_t \mid at' \notin T_{D_R} \,\}, \\
T_D & = & (T_{D_L} \cup T_{D_R}) - \{\, at' \in T_t \mid at' \notin T_s,\, \exists as' \in T_s : [\![at']\!]\alpha'\beta = [\![as']\!]\alpha'\beta \,\},
\end{array}$$

$$\begin{array}{rcl}
T_x & = & \{\, a \mid a \in C_R,\ [\![a]\!] \subseteq [\![x]\!]\alpha'\beta \,\} \cup \{\, a \mid a \in C_D,\ [\![a]\!] \subseteq ([\![x]\!]\alpha'\beta - \bigcup[\![V_t]\!]\alpha'\beta) \,\} \cup \\
& & \{\, at' \in T_R \mid [\![at']\!]\alpha'\beta \subseteq [\![x]\!]\alpha'\beta \,\} \cup \{\, as' \in T_D \mid [\![as']\!]\alpha'\beta \subseteq ([\![x]\!]\alpha'\beta - \bigcup[\![V_t]\!]\alpha'\beta) \,\} \cup \\
& & \{\, v_{yt',x} \cdot t' \mid yt' \in T_t \,\}, \\
T_y & = & \{\, a \mid a \in C_L,\ [\![a]\!] \subseteq [\![y]\!]\alpha'\beta \,\} \cup \{\, a \mid a \in C_D,\ [\![a]\!] \subseteq ([\![y]\!]\alpha'\beta - \bigcup[\![V_s]\!]\alpha'\beta) \,\} \cup \\
& & \{\, as' \in T_L \mid [\![as']\!]\alpha'\beta \subseteq [\![y]\!]\alpha'\beta \,\} \cup \{\, as' \in T_D \mid [\![as']\!]\alpha'\beta \subseteq ([\![y]\!]\alpha'\beta - \bigcup[\![V_s]\!]\alpha'\beta) \,\} \cup \\
& & \{\, v_{xs',y} s' \mid xs' \in T_s \,\}.
\end{array}$$

Compute the sets T_z for all $z \in Z$ as follows:

foreach $z \in Z$ **do** $T_z := \{\, v_{xs',z} \cdot s' \mid xs' \in T_s \,\} \cup \{\, v_{yt',z} \cdot t' \mid yt' \in T_t \,\}$ **od;**

foreach $a \in C_L \cup C_R$ `with` $[\![a]\!]\alpha'\beta \subseteq \bigcup[\![Z]\!]\alpha'\beta - \bigcup[\![X \cup Y]\!]\alpha'\beta$ **do**

  `"choose a` z `with` $[\![a]\!] \in [\![z]\!]\alpha'\beta$ `and add` a `to` T_z `"`

od;

foreach $at' \in T_L \cup T_R$ `with` $[\![at']\!]\alpha'\beta \subseteq \bigcup[\![Z]\!]\alpha'\beta - \bigcup[\![X \cup Y]\!]\alpha'\beta$ **do**

  `"choose a` z `with` $[\![at']\!]\alpha'\beta \subseteq [\![z]\!]\alpha'\beta$ `and add` at' `to` T_z `"`

od;

Now define NewEqs:

$$\begin{aligned}
\text{NewEqs} \;=\; & \{\, x =^?_{\mathcal{E}} \textstyle\sum_{y\in Y} v_{\{x,y\}} + \sum_{z\in Z} v_{\{x,z\}} + \Sigma T_x \mid x \in X \,\} \\
& \cup \{\, y =^?_{\mathcal{E}} \textstyle\sum_{x\in X} v_{\{x,y\}} + \sum_{z\in Z} v_{\{y,z\}} + \Sigma T_y \mid y \in Y \,\} \\
& \cup \{\, z =^?_{\mathcal{E}} v_{\{z\}} + \textstyle\sum_{x\in X} v_{\{x,z\}} + \sum_{y\in Y} v_{\{y,z\}} + \Sigma T_z \mid z \in Z \,\},
\end{aligned}$$

Then rule 1.9 produces alternatives with this version of NewEqs, because
- the right sets of new variables for pairs (x,y) and for variables $z \in Z$ are attributed to the old variables,
- single constants ($a \in C_R$) and single constant terms ($at_i \in T_R$) from $Smd(t)$ are only attributed to a nonempty set of single variables ($x \in X$) from $Smd(s)$ or one double variable ($z \in Z$), and analogous for constants from $Smd(s)$ and variables from $Smd(t)$.
- double constants ($a \in C_D$) and double constant terms ($at_i \in T_D$) are only attributed to a set of single variables from $Smd(s)$ or a set of single variables from $Smd(t)$,
- for each term $ut' \in Smd(t)$ and each $x \in Smd(s)$ a term $v_{yt',x} \cdot t'$ is present in t_x, and analogous for $xs' \in Smd(s)$ and $y \in Smd(t)$.
Define

$$\begin{aligned}
\gamma \;=\; & \{\, v_{\{x,y\}} := [\![x]\!]\alpha'\beta \cap [\![y]\!]\alpha'\beta \mid x \in X,\; y \in Y \,\} \cup \\
& \{\, v_{\{x,z\}} := [\![x]\!]\alpha'\beta \cap [\![z]\!]\alpha'\beta \mid x \in X \cup Y,\; z \in Z \,\} \cup \\
& \{\, v_{\{z\}} := [\![z]\!]\alpha'\beta \mid z \in Z \,\} \cup \\
& \{\, v_{yt',x} := q_{yt',x} \mid yt' \in T_t,\; x \in V_s,\; q_{yt',x} \odot [\![t']\!]\alpha'\beta = [\![yt']\!]\alpha'\beta \cap [\![x]\!]\alpha'\beta \,\} \cup \\
& \{\, v_{xs',y} := q_{xs',y} \mid xs' \in T_s,\; y \in V_t,\; q_{xs',y} \odot [\![s']\!]\alpha'\beta = [\![xs']\!]\alpha'\beta \cap [\![y]\!]\alpha'\beta \,\}.
\end{aligned}$$

Then $\alpha'\beta\gamma$ makes all equations of NewEqs true. Suppose $x \in X,\; x =^?_{\mathcal{E}} t_x \in \text{NewEqs}$. Then:

$$\begin{aligned}
[\![t_x]\!]\alpha'\beta\gamma \;&=\; \textstyle\bigcup_{y\in Y\cup Z} [\![v_{\{x,y\}}]\!]\alpha'\beta\gamma \cup \bigcup [\![T_x]\!]\alpha'\beta\gamma \\
&=\; ([\![x]\!]\alpha'\beta \cap \textstyle\bigcup [\![V_t]\!]\alpha'\beta) \cup \\
&\quad\; ([\![x]\!]\alpha'\beta \cap \textstyle\bigcup [\![C_L]\!]\alpha'\beta) \cup ([\![x]\!]\alpha'\beta \cap (\bigcup [\![C_D]\!]\alpha'\beta - \bigcup [\![V_t]\!]\alpha'\beta)) \cup \\
&\quad\; ([\![x]\!]\alpha'\beta \cap \textstyle\bigcup [\![T_L]\!]\alpha'\beta) \cup ([\![x]\!]\alpha'\beta \cap (\bigcup [\![T_{D_L}]\!]\alpha'\beta - \bigcup [\![V_t]\!]\alpha'\beta)) \cup \\
&\quad\; ([\![x]\!]\alpha'\beta \cap \textstyle\bigcup_{y_t \in T_t} [\![yt']\!]\alpha'\beta) \\
&=\; [\![x]\!]\alpha'\beta \cap [\![t]\!]\alpha'\beta\gamma = [\![x]\!]\alpha'\beta = [\![x]\!]\alpha'\beta\gamma.
\end{aligned}$$

In the same way: $[\![y]\!]\alpha'\beta\gamma = [\![t_y]\!]\alpha'\beta\gamma$ for all $y \in Y$ and $y =^?_{\mathcal{E}} t_y \in \text{NewEqs}$.
Suppose $z \in Z$ and $z =^?_{\mathcal{E}} t_z \in \text{NewEqs}$. Then:

$$\begin{aligned}
[\![t_z]\!]\alpha'\beta\gamma \;&=\; [\![v_{\{z\}}]\!]\alpha'\beta\gamma \cup \textstyle\bigcup_{x\in X\cup Y} [\![v_{\{x,z\}}]\!]\alpha'\beta\gamma \cup \bigcup [\![T_z]\!]\alpha'\beta\gamma \\
&=\; [\![z]\!]\alpha'\beta \cup ([\![z]\!]\alpha'\beta \cap \ldots) \\
&=\; [\![z]\!]\alpha'\beta = [\![z]\!]\alpha'\beta\gamma.
\end{aligned}$$

So α is a solution of NewEqs.

Now add equations to P_1 and P_3 and extend γ for fresh variables as follows:
1) Take all pairs (as_i, at_j) with $as_i \in Smd(s)$, $at_j \in Smd(t)$, and $[\![as_i]\!]\alpha'\beta = [\![at_j]\!]\alpha'\beta$, and put $as_i =^?_{\mathcal{E}} at_j$ into P_1. Then also $[\![as_i]\!]\alpha'\beta\gamma = [\![at_j]\!]\alpha'\beta\gamma$.
2) Consider each term $s_i \in Smd(s)$ with $s_i \equiv xs_i'$. For each s_i make intersections $[\![s_i]\!]\alpha'\beta \cap [\![t_j]\!]\alpha'\beta$ for all $t_j \in Smd(t)$ of the form at_j' or yt_j'. Suppose $t_1, \ldots, t_{n_t}$ ($n_t \geq 0$) are the terms from $Smd(t)$ with a nonempty intersection, and suppose $Smd(t) \cap \mathcal{V} = \{\, y_1, \ldots, y_{m_t} \,\}$ ($m_t \geq 0$). If $m_t + n_t > 0$ then make fresh variables $v_1, \ldots, v_{n_t}$, put these into $\vec{v}$, and put $x =^?_{\mathcal{E}} v_1 + \ldots + v_{n_t} + \sum_{y\in V_t} v_{xs_i',y}$ into P_3. (Variables $v_{xs_i',y}$ were chosen when NewEqs was defined.)

Also extend γ: Define processes p_j such that $p_j \odot [\![s_i']\!]\alpha'\beta = [\![t_j]\!]\alpha'\beta \cap [\![s_i]\!]\alpha'\beta$, and put $v_j := p_j$ into γ for all j with $1 \leq j \leq n_t$. Then:

$$\begin{aligned}
&[\![v_1 + \ldots + v_{n_t} + \textstyle\sum_{y \in V_t} v_{xs_i',y}]\!]\alpha'\beta\gamma \odot [\![s_i']\!]\alpha'\beta\gamma = \\
&\quad [\![v_1 + \ldots + v_{n_t} + \textstyle\sum_{y \in V_t} v_{xs_i',y}]\!]\gamma \odot [\![s_i']\!]\alpha'\beta = \\
&\quad (p_1 \oplus \ldots \oplus p_{n_t} \oplus p_{xs_i',y_1} \oplus \ldots \oplus p_{xs_i',y_{m_t}}) \odot [\![s_i']\!]\alpha'\beta = \\
&\quad ([\![t_1 + \ldots + t_{n_t}]\!]\alpha'\beta \cup [\![y_1 + \ldots + y_{m_t}]\!]\alpha'\beta) \cap [\![s_i]\!]\alpha'\beta = \\
&\quad [\![s_i]\!]\alpha'\beta = [\![xs_i']\!]\alpha'\beta\gamma = [\![x]\!]\alpha'\beta\gamma \odot [\![s_i']\!]\alpha'\beta\gamma,
\end{aligned}$$

so also $[\![v_1 + \ldots + v_{n_t} + \sum_{y \in V_t} v_{xs_i',y}]\!]\alpha'\beta\gamma = [\![x]\!]\alpha'\beta\gamma$, and α is a solution of $x =^?_{\mathcal{E}} v_1 + \ldots + v_{n_t} + \sum_{y \in V_t} v_{xs_i',y}$.
Now for each pair $(xs_i', t_1), \ldots, (xs_i', t_{n_t})$ add an equation to P_1 or make a triple. If $t_j \equiv at_j'$, then put $v_j s_i' =^?_{\mathcal{E}} t_j$ into P_1. Then $[\![t_j]\!]\alpha'\beta$ contains only one pair $(a, [\![t_j']\!]\alpha'\beta)$, so $[\![t_j]\!]\alpha'\beta \cap [\![s_i]\!]\alpha'\beta = [\![t_j]\!]\alpha'\beta$, and $[\![v_j s_j']\!]\alpha'\beta\gamma = p_j \odot [\![s_i']\!]\alpha'\beta = [\![t_j]\!]\alpha'\beta\gamma$. If $t_j \equiv yt_j'$, then make a triple (v_j, xs_i', yt_j').
3) Consider each term t_j in $Smd(t)$ with $t_j \equiv yt_j'$. Here also for each t_j all intersections of $[\![t_j]\!]\alpha'\beta$ with $[\![s_i]\!]\alpha'\beta$ are made for all $s_i \in (Smd(s) - (\mathcal{A} \cup \mathcal{V}))$. Suppose $s_1, \ldots, s_{n_s}$ $(n > 0)$ are all terms from $Smd(s) - (\mathcal{A} \cup \mathcal{V})$ with a nonempty intersection, and $Smd(s) \cap \mathcal{V} = \{x_1, \ldots, x_{m_s}\}$. Then in the same way as in 2) make fresh variables $w_1, \ldots, w_{n_s}$. Add these to $\vec{v}$ and add to P_3:

$$y =^?_{\mathcal{E}} w_1 + \ldots + w_{n_s} + \textstyle\sum_{x \in V_s} v_{yt_j',x}.$$

Also extend γ with $w_i := q_i$ for processes q_i with $q_i \odot [\![t_j']\!]\alpha'\beta = [\![s_i]\!]\alpha'\beta \cap [\![t_j]\!]\alpha'\beta$ for $1 \leq i \leq n_s$. Then again $[\![w_1 + \ldots + w_{n_s} + \sum_{x \in V_s} v_{yt_j',x}]\!]\alpha'\beta\gamma = [\![y]\!]\alpha'\beta\gamma$ (proof analogous to proof in 2)), so α is a solution of all equations $y =^?_{\mathcal{E}} w_1 + \ldots + w_{n_s} + \sum_{x \in V_s} v_{yt_j',x}$ in P_3.
Also add equations to P_1 for pairs $(s_1, yt_j'), \ldots, (s_{n_s}, yt_j')$. If $s_i \equiv as_i'$, then put $w_i t_j' =^?_{\mathcal{E}} s_i$ into P_1. Again $[\![w_i t_j']\!]\alpha'\beta\gamma = [\![s_i]\!]\alpha'\beta\gamma$.
If $s_i \equiv xs_i'$, then there is a triple (v_j, xs_i', yt_j'). Then put $v_j s_i' =^?_{\mathcal{E}} w_i t_j'$ into P_1. Now $[\![v_j s_i']\!]\alpha'\beta\gamma = [\![v_j]\!]\gamma \odot [\![s_i]\!]\alpha'\beta = p_j \odot [\![s_i']\!]\alpha'\beta = [\![t_j]\!]\alpha'\beta \cap [\![s_i]\!]\alpha'\beta = [\![s_i]\!]\alpha'\beta \cap [\![t_j]\!]\alpha'\beta = q_i \odot [\![t_j']\!]\alpha'\beta = [\![w_i]\!]\gamma \odot [\![t_j']\!]\alpha'\beta = [\![w_i t_j']\!]\alpha'\beta\gamma$. So α is a solution of all equations $v_j s_i' =^?_{\mathcal{E}} w_i t_j'$ in P_1.
Now $\exists\, \vec{u}, \vec{v}\, (P \cup P_1 \cup P_2 \cup P_3 \cup \text{NewEqs})$ is an alternative produced by procedure SumRule, and α is a solution of $\exists\, \vec{u}, \vec{v}\, (P \cup P_1 \cup P_2 \cup P_3 \cup \text{NewEqs})$. □

4.2 Termination and minimality of UnSeq$_f$ for sets of semiclosed equations

The sequential composition operator of process algebra strongly resembles the concatenation operator on strings. Unification in the algebra of strings with a nonempty set of symbols, concatenation, and empty string, also called word unification, is infinitary [Mak77]. Also unification in a string algebra without a unit element is infinitary. Example: the problem $\{ xa =^?_{\mathcal{E}} ax \}$ has solutions $x = a,\ x = aa,\ x = aaa,\ \ldots$. If this problem is seen as a problem in the process algebra used here, then it has the same minimal and complete set of most general unifiers, so unification in this algebra is also infinitary. So termination of the algorithm may only be proved for restricted classes of problems.

If process algebra is used in system specification, the most commonly used equations have one side without variables representing the specification, and one side with variables representing an incomplete implementation. Such equations will be called semiclosed equations. In the following it is proved that the algorithm terminates on problems consisting of semiclosed equations, and that the produced set of solutions is minimal.

An execution of UnSeq_f on input P may be considered as a tree with in each node a problem. The root node contains the initial problem, the leaves contain problems in solved form or $\{F\}$, and all other nodes contain intermediate results. On each non-leaf node a rule is applied, and the children of the node correspond to all alternatives produced by the rule.

First it is proved that the equations in all nodes of an execution tree of UnSeq_f with as input a set of semiclosed equations have a number of properties.

Proposition 4.4
Suppose $UnSeq_f$ is executed with input P_0 of equations having no variables in one side, and $\exists\, \vec{u_i}\, P_i$ is an intermediate result of some execution path. Then:

1. *if $s =^?_{\mathcal{E}} t \in P_i$ and neither s nor t is a variable, then $s =^?_{\mathcal{E}} t$ is semiclosed,*
2. *if $s =^?_{\mathcal{E}} t \in P_i$ and $((s \equiv x, t \equiv t_x)$ or $(t \equiv x, s \equiv t_x))$, then:*
   (t_x is closed or $t_x \equiv au_1$ or $t_x =_{\mathcal{E}} u_1 + \ldots + u_m + t'$ with t' closed),
   all u_j are different from x, all u_j are in VarStack, and if $x \in$ Varstack then all u_j are nearer to the top than x,
3. *if $v \in$ VarStack, then the only occurrences of v in P_i are:*
   (a) *one equation $v =^?_{\mathcal{E}} t_v$ or $vs' =^?_{\mathcal{E}} t'$,*
   (b) *at most one equation $u =^?_{\mathcal{E}} av$ or $u =^?_{\mathcal{E}} t_u$ with $t_u =_{\mathcal{E}} v + v_1 + \ldots + v_m + t'$.*

Proof:
Induction on the number n of rule applications in the path from the root of the execution tree to this node.
$n = 0$: assumption, and VarStack = empty.
$n > 0$: check all rules.
0.1 (Eq): trivial.
0.2 (Subs):
Suppose rule Subs substitutes $x =^?_{\mathcal{E}} t_x$ into the other equations of P_i. Equations $s =^?_{\mathcal{E}} t$ with $s, t \notin \mathcal{V}$ still are semiclosed after substitution. So consider equations $y =^?_{\mathcal{E}} t_y$ with $x \in Var(t_y)$. If VarStack $= \emptyset$ then t_y is closed (IH ii). So suppose that top(VarStack)$= x$. Then t_x is closed (IH ii), and there is no other equation $x =^?_{\mathcal{E}} t_x{}'$ in P_i (IH iii). Then there are two possibilities:
1) $t_y =_{\mathcal{E}} x + u_1 + \ldots + u_n + t'$, t' closed. Then $t_y\{x := t_x\} =_{\mathcal{E}} u_1 + \ldots + u_n + t_x + t'$, and $t_x + t'$ is closed.
2) $t_y \equiv ax$. Then $t_y\{x := t_x\}$ is closed.
0.3 (Occur): never applicable (IH).
0.4 (AuxVar): easy.
1.1 (NoSol): trivial.
1.2 (Seq_aa): easy.
1.3 (Seq_ax):
Suppose rule 1.3 is applied on $xs' =^?_{\mathcal{E}} at' \in P_i$. The first alternative produces equations $x =^?_{\mathcal{E}} a$ and $s =^?_{\mathcal{E}} t$. These have the right properties. The second alternative produces equations $x =^?_{\mathcal{E}} av$ and $vs' =^?_{\mathcal{E}} t'$ with t closed, and pushes v on VarStack. So i), ii), and iii) hold.
Suppose $x \in$ VarStack. Then $xs' =^?_{\mathcal{E}} at'$ is replaced by $x =^?_{\mathcal{E}} a$ or $x =^?_{\mathcal{E}} av$, so iii) still holds for x.
1.4 (Seq_xy), 1.5 (Seq_xx), 1.7 (VarSeq), and 1.8 (VarSeq$^+$) are never applicable (IH).

1.6 (Seq_xδ): easy.

1.9 (Sum):

Suppose rule Sum is executed on an equation $s =^?_{\mathcal{E}} t$, and $\{ at' \in Smd(t) \mid a \in \mathcal{A} \} = \{ t_1, \ldots, t_n \}$. Suppose top(VarStack) $= v$. Then $s \equiv vs'$, and t is closed. Then Sum produces an equation $v =^?_{\mathcal{E}} v_1 + \ldots + v_n$ with fresh variables $v_1, \ldots, v_n$, pushes $v_1, \ldots, v_n$ on VarStack, and produces equations $v_1 s' =^?_{\mathcal{E}} t_1, \ldots, v_n s' =^?_{\mathcal{E}} t_n$.

Suppose VarStack $= \emptyset$, and suppose t is the closed term. Then Sum may produce equations of the form:

- $as' =^?_{\mathcal{E}} at'$ for terms $as' \in Smd(s)$ and $at' \in Smd(t)$,
- $x =^?_{\mathcal{E}} \delta$,
- $x =^?_{\mathcal{E}} a_1 + \ldots + a_k + t_1 + \ldots + t_l$ with all $a_i \in Smd(t)$ and $t_i \in \{ t_1, \ldots, t_n \}$ for variables $x \in Smd(s)$,
- $x =^?_{\mathcal{E}} u_1 + \ldots + u_n$ for terms $xs' \in Smd(s)$, and for each u_i an equation $u_i s' =^?_{\mathcal{E}} t_i$ with $t_i \in \{ t_1, \ldots, t_n \}$, and u_1 to u_n are pushed onto VarStack.

Then i), ii), and iii) hold. □

Theorem 4.5 *Suppose P_0 is a set of semiclosed equations. Then the algorithm $UnSeq_f$ terminates on input P_0.*

Proof:

Define the **length** of a term as the number of symbols in a term. Consider sets of semiclosed equations, and all possible intermediate results produced by UnSeq$_f$ with such sets as input. Take only the equations $s =^?_{\mathcal{E}} t \in P$ with neither s nor t a variable. Partition this set of equations: $s =^?_{\mathcal{E}} t \in \mathrm{Sum}_n$ if the closed term has length n and rule Sum is applicable to $s =^?_{\mathcal{E}} t$, $s =^?_{\mathcal{E}} t \in \mathrm{Other}_n$ if the closed term has length n and another rule than Sum is applicable to $s =^?_{\mathcal{E}} t$. Now define a metric $M(P)$ on sets P of equations as follows:

$M(P) = (S_k, O_k, \ldots, S_1, O_1)$, where $S_i = \#\mathrm{Sum}_i$, $O_i = \#\mathrm{Other}_i$, and k is the length of the longest closed term in P. On this metric a total order is defined: $(m_k, \ldots, m_1) < (n_l, \ldots, n_1)$ iff $k < l$, or $k = l$ and there is an $i \leq k$ with $m_k = n_k, \ldots, m_{i+1} = n_{i+1}$, and $m_i < n_i$.

Consider an execution path of UnSeq$_f$ on input P_0, with intermediate results $\exists u_i P_i$, and consider the sequence $M(P_0), \ldots, M(P_i), \ldots$. Conjecture: this sequence is decreasing.

Proof: check that all rules decrease $M(P)$ or keep $M(P)$ constant.

0.1 (Eq): removes an equation $s =^?_{\mathcal{E}} s$. If s is a variable, then $M(P_i) = M(P_{i+1})$, else $M(P_i) > M(P_{i+1})$.

0.2 (Subs):

Suppose top(VarStack) $= v$. Then Subs uses an equation $v =^?_{\mathcal{E}} t_v$ for substitution, with t_v closed, and it only substitutes v in an equation $u =^?_{\mathcal{E}} av$ or $u =^?_{\mathcal{E}} t_u$ with $t_u =_{\mathcal{E}} v + v_1 + \ldots + v_n + t'$ with t' closed. Then $M(P_i) = M(P_{i+1})$.

Suppose VarStack $= \emptyset$. Then Subs uses an equation $x =^?_{\mathcal{E}} t_x$ for substitution with t_x closed. If $x =^?_{\mathcal{E}} t_x$ is substituted in an equation $s =^?_{\mathcal{E}} t$ with t closed and $s \not\equiv xs'$ then this does not change $M(P)$. If $x =^?_{\mathcal{E}} t_x$ is applied to an equation $xs' =^?_{\mathcal{E}} t$ then the resulting equation may belong in Sum_n, and $M(P)$ increases. But in that case $x \in Var(P_0)$ (Proposition 4.4), so this may only occur a finite number of times in an execution path.

0.3 (Occur): never applicable.

0.4 (AuxVar): Application of AuxVar does not change $M(P)$.

1.1 (NoSol): The set of equations becomes empty, so $M(P)$ decreases.

1.2 (Seq_aa), 1.3 (Seq_ax), and 1.6 (Seq_xδ): The new equation has a smaller closed term, so $M(P)$ decreases.

1.4 (Seq_xy), 1.5 (Seq_xx), 1.7 (VarSeq), and 1.8 (VarSeq$^+$): never applied.

1.9 (Sum):
Suppose rule 1.9 is applied to $s =^?_{\mathcal{E}} t \in P$ and t is closed.
Suppose t is a sum. Then $s =^?_{\mathcal{E}} t$ is replaced by equations $x =^?_{\mathcal{E}} t_x$, $as_i =^?_{\mathcal{E}} at_j$ and $us_i =^?_{\mathcal{E}} at_j$ with $at_j \in \mathit{Smd}(t)$, so $M(P_i) > M(P_{i+1})$.
Suppose t is not a sum. Then s is a sum, and $t \equiv \delta$ or $t \equiv at'$. Now $s =^?_{\mathcal{E}} t$ is replaced by equations of the form $x =^?_{\mathcal{E}} \delta$, $x =^?_{\mathcal{E}} at'$, $x =^?_{\mathcal{E}} x_1$, and $ds' =^?_{\mathcal{E}} at'$ $(d \in \mathcal{A} \cup \mathcal{V})$, so $M(P_i) > M(P_{i+1})$.

So only rules 0.2 (Subs) and 0.4 (AuxVar), and possibly 0.1 (Eq) do not decrease $M(P)$. And a sequence of applications of 0.1, 0.2, and 0.4 without applications of other rules is always finite. So all execution paths of an execution tree of UnSeq_f are finite, and each rule produces a finite number of alternatives, so the execution terminates. □

Now it is proved that UnSeq_f with as input a set of semiclosed equations produces a minimal set of solutions.

Theorem 4.6 *Suppose $\exists \vec{u_n} P_n$ is a solved form produced by $UnSeq_f$ with input a set of semiclosed equations, and $x =^?_{\mathcal{E}} t_x \in P_n$.*
Then t_x contains no variables.

Proof:
Suppose t_x contains a variable u. Then P_n also contains an equation $us =^?_{\mathcal{E}} t$ or $u =^?_{\mathcal{E}} t$ (Proposition 4.4), so P_n is not a solved form. Contradiction. So t_x contains no variables. □

Corollary 4.7 *Suppose P_0 is a set of semiclosed equations. Then the set of solutions produced by $UnSeq_f$ on input P_0 is minimal.*

4.3 Complexity of UnSeq_f for sets of semiclosed equations

As termination of UnSeq_f is only proved for sets of semiclosed equations, the complexity of UnSeq_f can also only be computed for sets of semiclosed equations. The worst-case complexity of UnSeq_f for these equation sets is determined by the maximum depth and the maximum branching degree of all possible execution trees. The depth and branching degree are dependent on the number of summands in sum subterms of the equations. Suppose t is a term and T is a set of terms. Define $n(t)$ as the maximum number of summands in any subterm of term t, and $n(T)$ as the maximum number of summands in any subterm of terms $t \in T$. If UnSeq_f has as input a set of semiclosed equations, then the sum subterms of the closed terms of the intermediate equations have no more summands than the sum subterms of the closed terms of the original equations:

Proposition 4.8 *Suppose $UnSeq_f$ is executed with as input a set of semiclosed equations, $y =^?_{\mathcal{E}} t_y$ occurs in an intermediate result $\exists \vec{u_n} P_n$, and $x \in \mathit{Smd}(t_y)$. Then P_n contains an equation $xs =^?_{\mathcal{E}} at$ or $x =^?_{\mathcal{E}} a$ or $x =^?_{\mathcal{E}} at$.*

Proof:
Induction on the number n of rule applications along the path from the root to this node.
$n = 0$: There is no equation $y =^?_{\mathcal{E}} t_y$ in P_0 with $x \in \mathit{Smd}(t_y)$.
$n > 0$: Check all rules.
The only relevant rules are 1.3 (Seq_ax) and 1.9 (Sum).
1.3 (Seq_ax): Equation $xs =^?_{\mathcal{E}} at$ is replaced by $x =^?_{\mathcal{E}} a$ or $x =^?_{\mathcal{E}} av$.
1.9 (Sum): Equations $y =^?_{\mathcal{E}} u_1, \ldots + u_k$ are made, and also equations $u_i s' =^?_{\mathcal{E}} at_j'$ for each u_i. □

Proposition 4.9 *Suppose UnSeq$_f$ is executed with as input a set* $\{ s_1 =^?_{\mathcal{E}} t_1, \ldots, s_k =^?_{\mathcal{E}} t_k \}$ *with all* t_i *closed, and in an intermediate result of the execution tree an equation* $s =^?_{\mathcal{E}} t$ *is present. Then* $n(t) \leq n(T)$, *where* $T = \{t_1, \ldots, t_k\}$.

Proof:
Induction on the number n of rule applications along the path from the root of the execution tree to this intermediate result.
$n = 0$: trivial.
$n > 0$: check all rules.
<u>0.1 (Eq)</u>: trivial.
<u>0.2 (Subs)</u>:
Suppose Subs uses an equation $x =^?_{\mathcal{E}} t_x$ for substitution.
If $x =^?_{\mathcal{E}} t_x$ is substituted into some $s =^?_{\mathcal{E}} t$ with t closed, then t does not change, so $n(t)$ also does not change.
If there is an equation $y =^?_{\mathcal{E}} t_y$ with x occurring in t_y, then $t_y \equiv xt'$ or $t_y =_{\mathcal{E}} x + u_1 + \ldots + u_k + t'$ for some $k \geq 0$, and top(VarStack) $= x$ (Proposition 4.8). If $t_y = x + u_1 + \ldots + u_k + t'$ for some $k \geq 0$ then $t_x \equiv a$ or $t_x \equiv at'$ (Proposition ...). And (IH) $n(t_x) \leq n(T)$ and $n(t_y) \leq n(T)$, so $n(t_y[x := t_x]) \leq n(T)$.
<u>0.4 (AuxVar)</u>, <u>1.1 (NoSol)</u>, <u>1.2 (Seq_aa)</u>, <u>1.3 (Seq_ax)</u>, <u>1.6 (Seq_xδ)</u>: easy.
<u>0.3 (Occur)</u>, <u>1.4 (Seq_xy)</u>, <u>1.5 (Seq_xx)</u>, <u>1.7 (VarSeq)</u>, <u>1.8 (VarSeq$^+$)</u>: never applicable.
<u>1.9 (Sum)</u>:
Rule Sum, executed on an equation $s =^?_{\mathcal{E}} t$, may produce equations $x =^?_{\mathcal{E}} \delta$ or $x =^?_{\mathcal{E}} u_1 + \ldots + u_k$ for variables x with $xs' \in \mathit{Smd}(s)$, equations $as' =^?_{\mathcal{E}} at'$ for $as' \in \mathit{Smd}(s)$ and $at' \in \mathit{Smd}(t)$, equations $vs' =^?_{\mathcal{E}} at'$ for some $xs' \in \mathit{Smd}(s)$ and $at' \in \mathit{Smd}(t)$, and equations $x =^?_{\mathcal{E}} t_1{}' + \ldots + t_k{}'$ for $x \in \mathit{Smd}(s)$ and $t_1{}', \ldots, t_k{}' \in \mathit{Smd}(t)$. In equations $x =^?_{\mathcal{E}} u_1 + \ldots + u_k$ each u_i corresponds to a different summand t' from t, so for all new equations $x =^?_{\mathcal{E}} t_x$: $n(t_x) \leq n(T)$. $\square$

Suppose $P = \{s_1 =^?_{\mathcal{E}} t_1, \ldots, s_n =^?_{\mathcal{E}} t_n\}$ is input to UnSeq$_f$ and all t_i are closed. Define $S = \{s_1, \ldots, s_n\}$ and $T = \{t_1, \ldots, t_n\}$. Then the depth of the execution tree is dependent on the depth of the closed terms. The **depth $D(t)$ of a term** t is defined as:

$$D(\delta) = 0, \ \ D(a) = D(x) = 1,$$
$$D(t_1 + t_2) = \max(D(t_1), D(t_2)),$$
$$D(t_1 \cdot t_2) = D(t_1) + D(t_2).$$

The function D is easily extended to sets T of terms:

$$D(T) = \textstyle\sum_{t \in T} D(t).$$

After substitution of all equations $x =^?_{\mathcal{E}} t_x$ that are initially present there are never two or more equations $x =^?_{\mathcal{E}} t_1$ and $x =^?_{\mathcal{E}} t_2$ except equations $x =^?_{\mathcal{E}} u_1 + \ldots + u_k$ and $x =^?_{\mathcal{E}} v_1 + \ldots + v_l$, produced by an application of rule Sum to an equation with summands xs_1 and xs_2. So the length of a path in the execution tree is determined by the number of rule applications needed to reduce an equation to an equation in solved form, and the number of new equations that are produced during rule applications. Rule 1.1 (NoSol) terminates an execution path. Rules 0.1 (Eq) and 0.4 (AuxVar) remove an equation. Rule 1.6 (Seq_xδ) produces an equation in solved form. Rules 0.3 (Occur), 1.4 (Seq_xy), 1.5 (Seq_xx), 1.7 (VarSeq), and 1.8 (VarSeq$^+$) are never applied if UnSeq$_f$ starts with a set of semiclosed equations (Proposition 4.4). So the interesting rules are 0.2 (Subs), 1.2 (Seq_aa), 1.3 (Seq_ax), and 1.9 (Sum). Rules 1.2 and 1.3 both replace an equation by a new one with the depth of the closed term decreased by one.

Rule 1.3 also produces an equation in solved form. Application of 0.2 (Subs) does not change the length of closed terms. So an execution path without applications of rule 1.9 (Sum) is proportional to $D(T)$.

Rule 1.9 may produce a great number of new equations that are not in solved form. Suppose rule Sum is applied to an equation $s =^?_{\mathcal{E}} t$ with t closed and $D(t) = D$. Then it may produce equations $as' =^?_{\mathcal{E}} at'$ for $as' \in Smd(s)$ and $at' \in Smd(t)$, equations $v_j s' =^?_{\mathcal{E}} at'$ for $xs' \in Smd(s)$ and $at' \in Smd(t)$, and equations $x =^?_{\mathcal{E}} u_1 + \ldots + u_k$ for $xs' \in Smd(s)$. In an execution path variables x in s may be replaced by terms t_x. If t_x is a sum, it has at most $n(T)$ summands (Proposition 4.9). Define:

$$
\begin{aligned}
n_S &= n(S),\\
n_T &= n(T),\\
n_{vp} &= \text{the maximum number of variable summands in a subterm of a term in } S,\\
n_v &= \text{the maximum number of variable product summands in a subterm of a term in } S,\\
n_{cpS} &= \text{the maximum number of constant product summands in a subterm of a term in } S,\\
n_{cpT} &= \text{the maximum number of constant product summands in a subterm of a term in } T,\\
n_{do} &= \text{the maximum number of double occurrences in a sum subterm of a term in } S,\\
&\quad\ \text{where a summand } xs_1 \text{ is a double occurrence if there is also another summand } xs_2\\
&\quad\ \text{in the same sum},\\
n_{pS} &= n_{vp} + n_{cpS},\\
n_{v(p)} &= n_v + n_{vp},\\
D_T &= D(T).
\end{aligned}
$$

Then s contains at most $n_{v(p)} \cdot n_{cpT} + n_{cpS}$ summands of the form as'. Now Sum creates at most $(n_{v(p)} \cdot n_{cpT} + n_{cpS}) \cdot n_{cpT}$ equations of the form $as' =^?_{\mathcal{E}} at'$. Also Sum creates at most $n_{vp} \cdot n_{cpT}$ equations of the form $vs' =^?_{\mathcal{E}} at'$. Rule Sum also produces at most n_{do} equations $x =^?_{\mathcal{E}} u_1 + \ldots + u_k$, $x =^?_{\mathcal{E}} v_1 + \ldots + v_m$ $(k, m \leq n_{cpT})$. First equations $u_i =^?_{\mathcal{E}} t_{u_i}, \ldots, v_j =^?_{\mathcal{E}} t_{v_j}$ will be produced and substituted, and then at most $n_{do} - 1$ equations $t_{u_1} + \ldots + t_{u_l} = t_{v_1} + \ldots + t_{v_m}$ are created. Each produces at most n^2_{cpT} new equations $at_i =^?_{\mathcal{E}} at_j$. So via equations $x =^?_{\mathcal{E}} u_1 + \ldots + u_k$, $x =^?_{\mathcal{E}} v_1 + \ldots + v_m$ rule Sum produces at most $(n_{do} - 1) \cdot n_{cpT}$ new equations $at_i =^?_{\mathcal{E}} at_j$ with $D(at_i) \leq D$ and $D(at_j) \leq D$. So Sum , applied to $s =^?_{\mathcal{E}} t$, creates at most $(n_{v(p)} \cdot n_{cpT} + n_{cpS}) \cdot n_{cpT} + n_{vp} \cdot n_{cpT} + (n_{do} - 1) \cdot n^2_{cpT}$ new equations of the form $as' =^?_{\mathcal{E}} at'$ or $vs' =^?_{\mathcal{E}} at'$ with $D(at') \leq D(t)$. On these equations first rule 1.2 (Seq_aa) or 1.3 (Seq_ax) must be applied before a new application of rule Sum may become possible. So an application of rule Sum on an equation with a closed term of depth D leads to at most $(n_{v(p)} \cdot n_{cpT} + n_{pS} + (n_{do} - 1) \cdot n_{cpT}) \cdot n_{cpT}$ new applications of rule Sum to equations with a depth of the closed term $\leq D - 1$. Then the length of an execution path is at most

$$((n_{v(p)} \cdot n_{cpT} + n_{pS} + (n_{do} - 1) \cdot n_{cpT}) \cdot n_{cpT})^{D_T}.$$

The branching degree of an application of rule Sum to an equation $s =^?_{\mathcal{E}} t$ is determined by the number of possible subsets of the set of equations $ds' =^?_{\mathcal{E}} at'$ $(d \in \mathcal{A} \cup \mathcal{V})$, and the number of possibilities of attributing terms a and at' from $Smd(t)$ to variables in $Smd(s)$. The number of possible sets with equations $ds' =^?_{\mathcal{E}} at'$ has complexity $O(2^{(n_{v(p)} \cdot n_{cpT} + n_{pS}) \cdot n_{cpT}})$. Procedure DistrConstTerms attributes summands a and at' from $Smd(t)$ to variables in $Smd(s)$. For each summand all alternatives are made in which the summand is attributed to a different (nonempty) subset of variables from $Smd(s)$. This produces at most $2^{n_v \cdot n_T}$ alternatives. So the branching degree of an application of rule Sum is $O(2^{n_v \cdot n_T + (n_{v(p)} \cdot n_{cpT} + n_{pS}) \cdot n_{cpT}})$, and the worst case complexity of UnSeq_f with as input a set of semiclosed equations is:

$$O(2^{n_v \cdot n_T + (n_{v(p)} \cdot n_{cpT} + n_{pS}) \cdot n_{cpT} \cdot ((n_{v(p)} \cdot n_{cpT} + n_{pS} + n_{do} \cdot n_{cpT}) \cdot n_{cpT})^{D_T}}).$$

So the worst case complexity of UnSeq_f with as input a set of semiclosed equations is doubly exponential in the sum of the depths of the closed terms, and exponential in the maximal numbers of different types of summands in sum subterms. A less complicated formula results if n_v, n_{vp}, and n_{pS} are replaced by n_S, and n_{cpT} by n_T:

$$O(2^{n_S \cdot n_T (n_T^2 + n_T + 1) \cdot (n_T \cdot (n_S + n_T)^2)^{D_T}}).$$

The worst case complexity of UnSeq_f may be reduced by imposing more restrictions on the input equations.
1) If no sum subterm contains variables or terms xs' then the worst case complexity of UnSeq_f becomes:

$$O(2^{(n_{cpS} \cdot n_{cpT})^{D_T+1}}).$$

2) If no sum subterm of the set P of input equations contains variables or summands xs' and also no sum subterm in P contains different summands at' and at'', then rule Sum produces only one alternative, and the branching degree of the execution tree is two, determined by rule 1.3 (Seq_ax). The maximum length of an execution path is now determined by the sum of the lengths of all closed terms. Define $L(P)$ as the sum of the lengths of the closed terms in all equations in P that are not in solved form. If both terms of an equation are closed, then use only the smallest one. Rules 1.2 (Seq_aa), 1.3 (Seq_ax), and 1.9 (Sum) in this case all decrease $L(P)$, so the maximum length of an execution path is $L(P)$. Then the complexity of UnSeq_f becomes:

$$O(2^{L(P)}).$$

So in this case the complexity of UnSeq_f is exponential in the sum of the lengths of the closed terms.

Chapter 5

Conclusions and discussion

A unification algorithm was given for a process algebra with only finite processes and as operators nondeterministic choice (associative, commutative, idempotent, and unit element), and sequential composition (associative, right-distributive over choice, and left zero element). The algorithm was proved sound and complete. As the unification problem in this algebra is infinitary, the algorithm does not terminate in general. Termination was proved for sets of semiclosed equations This corresponds to the most probable use of equations in system development, namely equations representing incomplete implementations that must conform to complete specifications. In this case the algorithm has a complexity that is doubly exponential in the sum of the sizes of the closed terms and exponential in the maximal number of summands in sums.

The algorithms of Parrow [Par89] and Larsen & Liu [LL90] also permit only semiclosed equations. Their equations contain more process algebra operators, but less variable occurrences than the equations for which termination of the algorithm devised in this paper is proved. Also their models contain infinite processes, which are not present in my model.

If only closed terms are considered, there is a striking resemblance between process algebras with action prefix and process algebras with sequential composition. Each term from an algebra with choice and sequential composition is equivalent to exactly one term of the form $a_1 \cdot t_1 + \ldots + a_m \cdot t_m + b_1 + \ldots + b_n$ or δ, which easily maps to a term with choice and action prefix. But if unification is considered, this resemblance is completely absent. The unification problem in an algebra with choice and action prefix is finitary, even in the presence of infinite processes [Dro93], but in the algebra with choice and sequential composition it is infinitary.

It is still an open question whether there are more classes of equation sets for which the unification problem is finitary, and whether the algorithm terminates on inputs that have a finite minimal and complete set of most general unifiers. This is a topic for further research. Other possible topics are extensions of the algebra with infinite processes and other operators, e.g. parallel composition and communication.

Bibliography

[AP89] Abdulrab, H. and J.-P. Pécuchet: Solving word equations. Journal of Symbolic Computation 8, pp. 499-521. 1989.

[AT85] Arnborg, S. and E. Tiden: Unification problems with one-sided distributivity. In: Proceedings of the 1st conference on Rewriting Techniques and Applications, LNCS 202, pp. 398-406. 1985.

[BB88] Baader, F., and W. Büttner: Unification in Commutative Idempotent Monoids. TCS 56, p. 345-352. 1988.

[BW90] Baeten, J.C.M., and W.P. Weijland: Process Algebra. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, Cambridge. 1990.

[BK84] Bergstra, J.A., and J.W. Klop: Process Algebra for Synchronous Communication. Inf. and Control 60, pp. 109-137. 1984.

[DJ91] Dershowitz, N., and J.-P. Jouannaud: Notations for rewriting. Bull. of the Eur. Ass. for Theor. Comp. Sci. 43, pp. 162-173. 1991.

[Dro93] Drost, N.J.: Unification in an Algebra with Choice and Action Prefix. Report P93xx, Programming Research Group, University of Amsterdam. 1993.

[Hoa85] Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall, London. 1985.

[Jaf90] Jaffar, J.: Minimal and Complete Word Unification. JACM37,1, pp. 47-85. 1990.

[JL91] Jonsson, B., and K.G. Larsen: On the Complexity of Equation Solving in Process Algebra. In: Abramsky, S., and T.S.E. Maibaum (Eds): TAPSOFT '91. LNCS 493, pp. 381-396. 1991.

[JK91] Jouannaud, J.-P., and C. Kirchner: Solving Equations in Abstract Algebras: A Rule-Based Survey of Unification. In: Lassez, J.-L. and G. Plotkin (eds): Computational Logic. Essays in honor of Alan Robinson. pp. 257-321. MIT Press, Cambridge, MA. 1991.

[LL90] Larsen, K.G. and X. Liu: Equation Solving Using Modal Transition Systems. In: Proceedings on Logic In Computer Science, pp. 108-117. 1990.

[Mak77] Makanin, G.S. The problem of solvability of equations in a free semigroup. Math. USSR Sbornik 32:2, pp. 129-198. 1977.

[Mil89] Milner, R.: Communication and Concurrency. Prentice Hall, London. 1989.

[Par89] Parrow, J.: Submodule construction as equation solving in CCS. TCS 68, pp.175-202. 1989.

[Ro65] Robinson, J.A.: A Machine Oriented Logic Based on the Resolution Principle. JACM 12, pp. 23-41. 1965.

[Sie90] Siekmann, J.H.: Unification theory. Decision Support Systems, pp. 315-337. 1990.

[TA87] Tiden, E. and S. Arnborg: Unification Problems with One-Sided Distributivity. J. Symbolic Computation 3, pp. 183-202. 1987.