# The Critical Pair Lemma: A Case Study for Induction Proofs With Partial Functions*

Jürgen Giesl

TU Darmstadt, Alexanderstr. 10, 64283 Darmstadt, Germany,
E-mail: giesl@informatik.tu-darmstadt.de

**Abstract**

In [9] we presented a calculus for automated induction proofs about partial functions. In contrast to previous work, our approach also allows us to derive induction schemes from the recursions of partial (and in particular, non-terminating) algorithms. In this way, existing induction theorem provers can be directly extended to partial functions without changing their logical framework.

This report contains a large collection of theorems from the area of term rewriting systems which were proved with our calculus (including Knuth and Bendix' well-known critical pair lemma). These examples demonstrate the power of our approach and they show that induction schemes based on partial functions are indeed needed frequently.

## 1 Introduction

*Induction* is the essential proof method for the verification of functional programs. For that reason, several techniques[1] have been developed to compute suitable induction relations and to perform induction proofs automatically, cf. e.g. [2, 5, 12, 20, 21]. However, most of these techniques are only sound if all occurring functions are total.

In [9] we showed that by slightly restricting the prerequisites of these techniques it is nevertheless possible to use them for partial functions, too. In particular, the well-known proof technique of performing *inductions w.r.t. algorithms* can also be applied for partial functions, i.e. (under certain conditions) one may even perform inductions w.r.t. non-terminating algorithms. In this way, this successful method for finding appropriate induction relations automatically can be used for partial functions as well. Hence, with our approach the well-known techniques for automated induction proofs can be *directly* applied to partial functions.

To show that the calculus developed in [9] can be used to prove relevant theorems about (possibly) partial functions, in the following case study we apply our calculus to prove prove more than 400 conjectures from the area of term rewriting systems (TRSs).

The (possibly) partial functions occurring in these conjectures can be divided into several classes, cf. [9, Section 7]. For example, there are partial functions like $\mathsf{first}$, which returns the first element of a list of terms, but which is undefined if the termlist is empty. Of course, such functions could easily be transformed into total ones, but this would change their semantics and could result in non-intuitive theorems. Moreover, for an automatic transformation of such partial functions into total ones, in general reasoning about partial functions would still be required (cf. the problem with exactness proofs of domain predicates in [9]).

But for many interesting algorithms their exact domain cannot be determined automatically at all. In particular, as the halting problem is undecidable (and as totality is not even semi-decidable), there are even many important *total* algorithms where totality cannot be verified automatically. For example, the well-known unification algorithm $\mathsf{unifies}$ by *J. A. Robinson* [18] is total, but its termination is a "deep theorem" [17]

*Technical Report IBN 98/49, Darmstadt University of Technology.

[1] There are two research paradigms for the automation of induction proofs, viz. *explicit* and *implicit* induction (e.g. [1, 11]), where we only focus on the first one.

and none of the current methods for automated termination analysis succeeds with this example. Hence, such functions cannot be handled by (fully) automated theorem provers without the ability of reasoning about *possibly* partial functions. In contrast to previous correctness proofs of the unification algorithm (e.g. [15, 17], our calculus can prove its partial correctness by induction w.r.t. $\mathsf{unifies}$ without having to verify its termination.

But even worse, there are numerous practically relevant partial algorithms whose domain is *undecidable*, i.e. there does not *exist* any exact domain predicate. For instance, our collection contains an algorithm $\mathsf{rewrites*}(s, t, R)$ which returns $\mathsf{true}$ iff the term $s$ rewrites to the term $t$ w.r.t. the TRS $R$ in arbitrary many steps. The domain of this algorithm is obviously undecidable. Hence, if one wants to prove any conjecture about such algorithms, one definitely needs a method to deal with partial functions.

In particular, with our calculus we can also prove partial truth of a variant of *D. E. Knuth* and *P. B. Bendix*' critical pair lemma [14] which states that if all critical pairs of a term rewriting system are joinable, then the system is locally confluent. As stressed in [10], no assumption of termination is necessary for this conjecture. The proof of this fundamental theorem is the last one in our collection and most of the preceding conjectures are needed as lemmata for this proof. Our verification of the critical pair lemma required several inductions w.r.t. functions like $\mathsf{rewrites*}$ whose domains are undecidable. Thus, our proof differs substantially from other case studies in related areas (e.g. the proofs of the Church-Rosser theorem for the $\lambda$-calculus in [16, 19]).

In Section 2 we introduce the data types and in Section 3 we give the definitions of all algorithms used. The remaining sections contain theorems proved with our calculus. For a detailed overview the reader is referred to the table of contents at the end of the report.

Several of these theorems are *partial correctness theorems*. Those theorems have a "(pc)" in their title. To ease readability, for such a conjecture $\varphi$ with the top-level terms $t_1, \ldots, t_n$ instead of $\mathsf{def}(t_1, \ldots, t_n) = \mathsf{true} \Rightarrow \varphi$ we just wrote $\varphi$ as an abbreviation. For the other theorems we stated the $\mathsf{def}$-terms explicitly. For the sake of brevity we only sketched the proofs of the theorems mentioned. In particular, we omitted most of the proofs about definedness terms and only mentioned them at points where they are especially interesting. In general, definedness conditions have to be checked in all following cases:

- Whenever one performs an induction w.r.t. a possibly partial function $f$ during the proof of $\varphi$, one has to prove the permissibility conjecture $\neg\mathsf{def}(f(x^*)) = \mathsf{true} \Rightarrow \varphi$. (If $f(x^*)$ occurs in the conjecture $\varphi$ and if $\varphi$ is a partial correctness statement, then this condition is always fulfilled.) Similar permissibility conjectures also have to be checked for structural induction, symbolic evaluation, and case analysis.

- If a conjecture containing partial functions is proved by induction, then one has to ensure that definedness of the induction conclusion implies definedness of the induction hypothesis.

- If a lemma $\psi$ containing partial functions is used for the proof of $\varphi$, then one has to check that definedness of $\varphi$ implies definedness of (the corresponding instantiation of) $\psi$.

We only regard universally closed formulas where we omitted all quantifiers to ease readability and moreover, instead of $t = \mathsf{true}$ we usually just wrote $t$.

## 2 Data Types

**Booleans** (bool)

$\mathsf{true}$ : $\mathsf{bool}$
$\mathsf{false}$ : $\mathsf{bool}$

**Natural Numbers** (nat)

$0$ : $\mathsf{nat}$
$\mathsf{s}$ : $\mathsf{nat} \times \mathsf{nat} \to \mathsf{nat}$

The following data type represents both terms and lists of terms.

**Termlists** (term)

e : term
var : nat × term → term
func : nat × term × term → term

We use naturals as names for the variables and function symbols. Then var$(n, v)$ represents the list beginning with the variable $n$ followed by the remainder list $v$. Analogously, func$(n, u, v)$ denotes the list starting with a term where the function symbol $n$ is applied to the argument list $u$ (and the tail of the list is $v$). So for example, the list $[x_1,\ f_1(x_2)]$ is denoted as var(1, func(1, var(2, e), e)). The reason for using just one data type of termlists (instead of two separate *mutually recursive* types for terms and termlists) is that our formalization simplifies the proofs considerably. Techniques for automated reasoning about mutually recursive data types and algorithms can for instance be found in [1, 4, 8, 13].

The data type tll is used to represent lists of lists of terms.

**Lists of Termlists** (tll)

empty : tll
add : term × tll → tll

For example, this data type is necessary for an algorithm like rewrite_rule$(t, l, r)$ which generates the list of all termlists that can be obtained by rewriting the termlist $t$ with the rule $l \to r$. (Note that an algorithm like rewrite_rule cannot only operate on terms instead of termlists, because to rewrite the *term* $f(t^*)$ one has to rewrite the *termlist* of its arguments $t^*$.)

To simplify the presentation in [9], we omitted the data type tll there. So the algorithm rewrites* from [9] corresponds to the algorithm rewrites_list*_exists in this report, the algorithm joinable from [9] corresponds to the algorithm joinable_list in this report, and the formulation of the critical pair lemma is also slightly different.

# 3 Algorithms

## 3.1 Basic Algorithms on bool, nat, and term

### 3.1.1 Negation on bool

*function* not : bool → bool
not(true) = false
not(false) = true

### 3.1.2 Predecessor on nat

*function* p : nat → nat
p(0) = 0
p(s($x$)) = $x$

### 3.1.3 Equality on nat

*function* eq : nat × nat → bool
eq(0, 0) = true
eq(0, s($y$)) = false
eq(s($x$), 0) = false
eq(s($x$), s($y$)) = eq($x$, $y$)

#### 3.1.4 Greater-Equal on nat

$$
\begin{array}{l}
\textit{function}\ \mathsf{ge} : \mathsf{nat} \times \mathsf{nat} \to \mathsf{bool} \\
\quad \mathsf{ge}(x, \mathsf{0}) \quad\quad\;\; = \mathsf{true} \\
\quad \mathsf{ge}(\mathsf{0}, \mathsf{s}(y)) \quad\; = \mathsf{false} \\
\quad \mathsf{ge}(\mathsf{s}(x), \mathsf{s}(y)) = \mathsf{ge}(x, y)
\end{array}
$$

#### 3.1.5 Greater on nat

$$
\begin{array}{l}
\textit{function}\ \mathsf{gt} : \mathsf{nat} \times \mathsf{nat} \to \mathsf{bool} \\
\quad \mathsf{gt}(\mathsf{0}, y) \quad\quad\;\; = \mathsf{false} \\
\quad \mathsf{gt}(\mathsf{s}(x), \mathsf{0}) \quad\; = \mathsf{true} \\
\quad \mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) = \mathsf{gt}(x, y)
\end{array}
$$

#### 3.1.6 Addition on nat

$$
\begin{array}{l}
\textit{function}\ \mathsf{plus} : \mathsf{nat} \times \mathsf{nat} \to \mathsf{nat} \\
\quad \mathsf{plus}(\mathsf{0}, y) \quad\;\; = y \\
\quad \mathsf{plus}(\mathsf{s}(x), y) \;\; = \mathsf{s}(\mathsf{plus}(x, y))
\end{array}
$$

#### 3.1.7 Equality on term

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textit{function}\ \mathsf{eqterm} : \mathsf{term} \times \mathsf{term} \to \mathsf{bool}} \\
\quad \mathsf{eqterm}(\mathsf{e}, \mathsf{e}) & = \mathsf{true} \\
\quad \mathsf{eqterm}(\mathsf{e}, \mathsf{var}(n_2, r_2)) & = \mathsf{false} \\
\quad \mathsf{eqterm}(\mathsf{e}, \mathsf{func}(n_2, s_2, r_2)) & = \mathsf{false} \\
\quad \mathsf{eqterm}(\mathsf{var}(n_1, r_1), \mathsf{e}) & = \mathsf{false} \\
\quad \mathsf{eqterm}(\mathsf{var}(n_1, r_1), \mathsf{var}(n_2, r_2)) & = \mathsf{eq}(n_1, n_2) \mathbin{\wedge} \mathsf{eqterm}(r_1, r_2) \\
\quad \mathsf{eqterm}(\mathsf{var}(n_1, r_1), \mathsf{func}(n_2, s_2, r_2)) & = \mathsf{false} \\
\quad \mathsf{eqterm}(\mathsf{func}(n_1, s_1, r_1), \mathsf{e}) & = \mathsf{false} \\
\quad \mathsf{eqterm}(\mathsf{func}(n_1, s_1, r_1), \mathsf{var}(n_2, r_2)) & = \mathsf{false} \\
\quad \mathsf{eqterm}(\mathsf{func}(n_1, s_1, r_1), \mathsf{func}(n_2, s_2, r_2)) & = \mathsf{eq}(n_1, n_2) \mathbin{\wedge} \mathsf{eqterm}(s_1, s_1) \mathbin{\wedge} \mathsf{eqterm}(r_1, r_2)
\end{array}
$$

As in [9], "$t_1 \wedge t_2$" abbreviates "$\mathsf{if}(t_1, t_2, \mathsf{false})$" and "$t_1 \wedge t_2 \wedge \ldots$" abbreviates "$t_1 \wedge (t_2 \wedge \ldots)$" to ease readability.

#### 3.1.8 First Element of term

$$
\begin{array}{l}
\textit{function}\ \mathsf{first} : \mathsf{term} \to \mathsf{term} \\
\quad \mathsf{first}(\mathsf{var}(n, r)) \quad\;\; = \mathsf{var}(n, \mathsf{e}) \\
\quad \mathsf{first}(\mathsf{func}(n, s, r)) = \mathsf{func}(n, s, \mathsf{e})
\end{array}
$$

This function is partial ($\mathsf{first}(\mathsf{e})$ is not defined).

#### 3.1.9 Tail of term

$$
\begin{array}{l}
\textit{function}\ \mathsf{tail} : \mathsf{term} \to \mathsf{term} \\
\quad \mathsf{tail}(\mathsf{var}(n, r)) \quad\;\; = r \\
\quad \mathsf{tail}(\mathsf{func}(n, s, r)) = r
\end{array}
$$

This function is partial ($\mathsf{tail}(\mathsf{e})$ is not defined).

#### 3.1.10 Second Element of term

$$
\begin{array}{l}
\textit{function}\ \mathsf{second} : \mathsf{term} \to \mathsf{term} \\
\quad \mathsf{second}(t) = \mathsf{first}(\mathsf{tail}(t))
\end{array}
$$

This function is partial (it is only defined for terms of at least length two).

#### 3.1.11 Tail of Tail of term

*function* ttail : term → term
$\mathsf{ttail}(t) = \mathsf{tail}(\mathsf{tail}(t))$

This function is partial (it is only defined for termlists of length two or more).

#### 3.1.12 Length of a Termlist

*function* length : term → nat

$$
\begin{aligned}
&\mathsf{length}(\mathsf{e}) &&= 0\\
&\mathsf{length}(\mathsf{var}(n,r)) &&= \mathsf{s}(\mathsf{length}(r))\\
&\mathsf{length}(\mathsf{func}(n,s,r)) &&= \mathsf{s}(\mathsf{length}(r))
\end{aligned}
$$

#### 3.1.13 Number of Symbols in a Termlist

*function* symbols : term → nat

$$
\begin{aligned}
&\mathsf{symbols}(\mathsf{e}) &&= 0\\
&\mathsf{symbols}(\mathsf{var}(n,r)) &&= \mathsf{s}(\mathsf{symbols}(r))\\
&\mathsf{symbols}(\mathsf{func}(n,s,r)) &&= \mathsf{s}(\mathsf{plus}(\mathsf{symbols}(s),\mathsf{symbols}(r)))
\end{aligned}
$$

#### 3.1.14 Adding a Term to a Lists of Terms

*function* addterm : term × term → term

$$
\begin{aligned}
&\mathsf{addterm}(\mathsf{var}(n,\mathsf{e}),t) &&= \mathsf{var}(n,t)\\
&\mathsf{addterm}(\mathsf{func}(n,s,\mathsf{e}),t) &&= \mathsf{func}(n,s,t)
\end{aligned}
$$

This function is partial (it is not defined if the first argument has a length different from 1).

#### 3.1.15 Appending two Termlists

*function* appendterm : term × term → term

$$
\begin{aligned}
&\mathsf{appendterm}(\mathsf{e},t) &&= t\\
&\mathsf{appendterm}(\mathsf{var}(n,r),t) &&= \mathsf{var}(n,\mathsf{appendterm}(r,t))\\
&\mathsf{appendterm}(\mathsf{func}(n,s,r),t) &&= \mathsf{func}(n,s,\mathsf{appendterm}(r,t))
\end{aligned}
$$

#### 3.1.16 Test Whether a term is Built With a Function

*function* first_is_func : term → bool

$$
\begin{aligned}
&\mathsf{first\_is\_func}(\mathsf{e}) &&= \mathsf{false}\\
&\mathsf{first\_is\_func}(\mathsf{var}(n,r)) &&= \mathsf{false}\\
&\mathsf{first\_is\_func}(\mathsf{func}(n,s,r)) &&= \mathsf{true}
\end{aligned}
$$

#### 3.1.17 Leading Function of a term

*function* func_name : term → nat
$\mathsf{func\_name}(\mathsf{func}(n,s,r)) = n$

This function is partial.

#### 3.1.18 Arguments of the Leading Function Symbol

*function* func_args : term → term
$\mathsf{func\_args}(\mathsf{func}(n,s,r)) = s$

This function is partial.

### 3.1.19 Leading Variable of a term

*function* var_name : term $\to$ nat
  $\mathsf{var\_name}(\mathsf{var}(n, r)) = n$

This function is partial.

### 3.1.20 Test Whether a Variable Occurs in a Termlist

*function* occurs : nat $\times$ term $\to$ bool

$$
\begin{aligned}
&\mathsf{occurs}(n, \mathsf{e}) &&= \mathsf{false}\\
&\mathsf{occurs}(n, \mathsf{var}(m, r)) &&= \mathsf{if}(\mathsf{eq}(n, m), \mathsf{true}, \mathsf{occurs}(n, r))\\
&\mathsf{occurs}(n, \mathsf{func}(m, s, r)) &&= \mathsf{occurs}(n, \mathsf{appendterm}(s, r))
\end{aligned}
$$

### 3.1.21 Compute the List of Variables in a Termlist

*function* vars : term $\to$ term

$$
\begin{aligned}
&\mathsf{vars}(\mathsf{e}) &&= \mathsf{e}\\
&\mathsf{vars}(\mathsf{var}(n, s)) &&= \mathsf{var}(n, \mathsf{vars}(s))\\
&\mathsf{vars}(\mathsf{func}(n, s, r)) &&= \mathsf{appendterm}(\mathsf{vars}(s), \mathsf{vars}(r))
\end{aligned}
$$

### 3.1.22 Test Whether Two Lists of Variables are Disjoint

*function* disjoint : term $\times$ term $\to$ bool

$$
\begin{aligned}
&\mathsf{disjoint}(\mathsf{e}, t) &&= \mathsf{true}\\
&\mathsf{disjoint}(\mathsf{var}(n, r), t) &&= \mathsf{if}(\mathsf{occurs}(n, t), \mathsf{false}, \mathsf{disjoint}(r, t)
\end{aligned}
$$

This function is partial (it is only defined if the first argument is a list of variables (of the form $\mathsf{var}(n_1, \mathsf{var}(n_2, \mathsf{var}(\ldots, \mathsf{e})))$) resp. if one of the variables in the first argument does not occur in the second and before that variable there were only variables in the first argument).

### 3.1.23 Compute the Maximum of a List of Variables

*function* max : term $\to$ nat

$$
\begin{aligned}
&\mathsf{max}(\mathsf{e}) &&= 0\\
&\mathsf{max}(\mathsf{var}(n, \mathsf{e})) &&= n\\
&\mathsf{max}(\mathsf{var}(n, \mathsf{var}(m, t))) &&= \mathsf{if}(\mathsf{ge}(n, m), \mathsf{max}(\mathsf{var}(n, t)), \mathsf{max}(\mathsf{var}(m, t)))
\end{aligned}
$$

Again, this function is partial.

### 3.1.24 Rename all Variables in a Termlist

$\mathsf{rename}(n, t)$ adds $n$ to all variables in the termlist $t$.

*function* rename : term $\times$ nat $\to$ term

$$
\begin{aligned}
&\mathsf{rename}(\mathsf{e}, n) &&= \mathsf{e}\\
&\mathsf{rename}(\mathsf{var}(m, t), n) &&= \mathsf{var}(\mathsf{plus}(m, n), \mathsf{rename}(t, n))\\
&\mathsf{rename}(\mathsf{func}(m, s, t), n)) &&= \mathsf{func}(m, \mathsf{rename}(s, n), \mathsf{rename}(t, n))
\end{aligned}
$$

### 3.1.25 Test Whether the Variables in one Termlist are a Subset of Another

*function* subseteq : term $\times$ term $\to$ bool

$$
\begin{aligned}
&\mathsf{subseteq}(\mathsf{e}, t) &&= \mathsf{true}\\
&\mathsf{subseteq}(\mathsf{var}(n, r), t) &&= \mathsf{if}(\mathsf{occurs}(n, t), \mathsf{subseteq}(r, t), \mathsf{false})\\
&\mathsf{subseteq}(\mathsf{func}(n, s, r), t) &&= \mathsf{subseteq}(\mathsf{appendterm}(s, r), t)
\end{aligned}
$$

### 3.1.26 Disjoint Union of Two Lists of Variables

$\mathit{function}$ $\mathsf{disjoint\_union} : \mathsf{term} \times \mathsf{term} \to \mathsf{term}$

$$\begin{aligned}
&\mathsf{disjoint_union}(v_1, \mathsf{e}) &&= v_1\\
&\mathsf{disjoint_union}(v_1, \mathsf{var}(n, v_2)) &&= \mathsf{if}(\mathsf{occurs}(n, v_1),\\
&&&\qquad \mathsf{disjoint_union}(v_1, v_2),\\
&&&\qquad \mathsf{disjoint_union}(\mathsf{appendterm}(v_1, \mathsf{var}(n, \mathsf{e})), v_2))
\end{aligned}$$

This function is partial (it is only defined if the first argument is a list of variables resp. if one of the variables in the first argument does not occur in the second and before that variable there were only variables in the first argument).

### 3.1.27 Test whether two Terms Occur Consecutive in a Termlist

The following function test whether two certain terms are on positions $2n-1$ and $2n$ in a termlist. Similar to $\wedge$, "$t_1 \vee t_2$" abbreviates "$\mathsf{if}(t_1, \mathsf{true}, t_2)$" and "$t_1 \vee t_2 \vee \ldots$" abbreviates "$t_1 \vee (t_2 \vee \ldots)$" to ease readability. Moreover, $t_1 \wedge t_2 \vee t_3$ stands for $(t_1 \wedge t_2) \vee t_3$ etc.

$\mathit{function}$ $\mathsf{in} : \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{bool}$

$$\mathsf{in}(s, t, r) = \mathsf{ge}(\mathsf{length}(r), \mathsf{s}(\mathsf{s}(0))) \wedge \big(\mathsf{eqterm}(s, \mathsf{first}(r)) \wedge \mathsf{eqterm}(t, \mathsf{second}(r)) \vee \mathsf{in}(s, t, \mathsf{ttail}(r))\big)$$

### 3.1.28 Test whether a Pair of Terms Occurs on Even Position in a Termlist

Similar to $\mathsf{in}$, the function $\mathsf{membereven}([t_1, t_2], r)$ tests whether the terms $t_1$ and $t_2$ occur consecutive in $r$ (starting on an even position).

$\mathit{function}$ $\mathsf{membereven} : \mathsf{term} \times \mathsf{term} \to \mathsf{bool}$

$$\begin{aligned}
\mathsf{membereven}(t, r) = \mathsf{ge}(\mathsf{length}(r), \mathsf{s}(\mathsf{s}(0))) \wedge \big(&\mathsf{eqterm}(\mathsf{first}(t), \mathsf{first}(r)) \wedge \mathsf{eqterm}(\mathsf{tail}(t), \mathsf{second}(r)) \vee\\
&\mathsf{membereven}(t, \mathsf{ttail}(r))\big)
\end{aligned}$$

### 3.1.29 Check Whether a List of Terms is a TRS

For variables denoting term rewriting systems we will use capital letters (e.g. $R$) to conform with the term rewriting literature. Then the function $\mathsf{trs}(R)$ checks whether $R$ is a proper term rewriting system (in particular, right hand sides of rules may only contain variables from their corresponding left hand sides).

$\mathit{function}$ $\mathsf{trs} : \mathsf{term} \to \mathsf{bool}$

$$\begin{aligned}
&\mathsf{trs}(\mathsf{e}) &&= \mathsf{true}\\
&\mathsf{trs}(\mathsf{var}(n, q)) &&= \mathsf{false}\\
&\mathsf{trs}(\mathsf{func}(n, s, q)) &&= \mathsf{ge}(\mathsf{length}(q), \mathsf{s}(0)) \wedge \mathsf{subseteq}(\mathsf{vars}(\mathsf{first}(q)), \mathsf{vars}(s)) \wedge \mathsf{trs}(\mathsf{tail}(q))
\end{aligned}$$

## 3.2 Algorithms for Substitutions

Substitutions are modelled by termlists of the form [variable, term, variable, term, . . .]. Intuitively, the first element is to be substituted by the second, the third is to be substituted by the fourth etc. We use $\sigma, \tau$, etc. for variables which are intended to denote substitutions.

### 3.2.1 Check Whether a Termlist Denotes a Substitution

$\mathit{function}$ $\mathsf{is\_subst} : \mathsf{term} \to \mathsf{bool}$

$$\begin{aligned}
&\mathsf{is_subst}(\mathsf{e}) &&= \mathsf{true}\\
&\mathsf{is_subst}(\mathsf{var}(n, t)) &&= \mathsf{if}(\mathsf{eqterm}(t, \mathsf{e}), \mathsf{false}, \mathsf{is_subst}(\mathsf{tail}(t)))\\
&\mathsf{is_subst}(\mathsf{func}(n, s, t)) &&= \mathsf{false}
\end{aligned}$$

#### 3.2.2 Applying Substitutions to Variables

*function* $\mathsf{apply\_subst\_var} : \mathsf{term} \times \mathsf{nat} \rightarrow \mathsf{term}$

$$\begin{aligned}
&\mathsf{apply_subst_var}(\mathsf{e}, n) &&= \mathsf{var}(n, \mathsf{e})\\
&\mathsf{apply_subst_var}(\mathsf{var}(m, t), n) &&= \mathsf{if}(\mathsf{eq}(m, n), \mathsf{first}(t), \mathsf{apply_subst_var}(\mathsf{tail}(t), n))
\end{aligned}$$

This function is partial (it is not defined if the first argument is not a proper substitution).

#### 3.2.3 Applying Substitutions to Termlists

*function* $\mathsf{apply\_subst} : \mathsf{term} \times \mathsf{term} \rightarrow \mathsf{term}$

$$\begin{aligned}
&\mathsf{apply_subst}(\sigma, \mathsf{e}) &&= \mathsf{e}\\
&\mathsf{apply_subst}(\sigma, \mathsf{var}(n, r)) &&= \mathsf{addterm}(\mathsf{apply_subst_var}(\sigma, n), \mathsf{apply_subst}(\sigma, r))\\
&\mathsf{apply_subst}(\sigma, \mathsf{func}(n, s, r)) &&= \mathsf{func}(n, \mathsf{apply_subst}(\sigma, s), \mathsf{apply_subst}(\sigma, r))
\end{aligned}$$

#### 3.2.4 Applying Lists of Substitutions to Termlists

*function* $\mathsf{apply\_subst\_list} : \mathsf{tll} \times \mathsf{term} \rightarrow \mathsf{term}$

$$\begin{aligned}
&\mathsf{apply_subst_list}(\mathsf{empty}, t) &&= \mathsf{empty}\\
&\mathsf{apply_subst_list}(\mathsf{add}(\sigma, l), t) &&= \mathsf{add}(\mathsf{apply_subst}(\sigma, t), \mathsf{apply_subst_list}(l, t))
\end{aligned}$$

Hence, we have $\mathsf{apply\_subst\_list}(\langle \sigma_1, \ldots, \sigma_n \rangle, t) = \langle \sigma_1(t), \ldots, \sigma_n(t) \rangle$ (Here $\langle \sigma_1, \ldots, \sigma_n \rangle$ is a $\mathsf{tll}$.)

#### 3.2.5 Applying Substitutions to tll's

*function* $\mathsf{apply\_subst\_tll} : \mathsf{term} \times \mathsf{tll} \rightarrow \mathsf{tll}$

$$\begin{aligned}
&\mathsf{apply_subst_tll}(\sigma, \mathsf{empty}) &&= \mathsf{empty}\\
&\mathsf{apply_subst_tll}(\sigma, \mathsf{add}(t, l)) &&= \mathsf{add}(\mathsf{apply_subst}(\sigma, t), \mathsf{apply_subst_tll}(\sigma, l))
\end{aligned}$$

#### 3.2.6 Domain of a Substitution

*function* $\mathsf{dom} : \mathsf{term} \rightarrow \mathsf{term}$

$$\begin{aligned}
&\mathsf{dom}(\mathsf{e}) &&= \mathsf{e}\\
&\mathsf{dom}(\mathsf{var}(n, r)) &&= \mathsf{var}(n, \mathsf{dom}(\mathsf{tail}(r)))
\end{aligned}$$

This function is partial.

#### 3.2.7 Renaming the Domain of a Substitution

*function* $\mathsf{rename\_dom} : \mathsf{term} \times \mathsf{nat} \rightarrow \mathsf{term}$

$$\begin{aligned}
&\mathsf{rename_dom}(\mathsf{e}, n) &&= \mathsf{e}\\
&\mathsf{rename_dom}(\mathsf{var}(m, r), n) &&= \mathsf{appendterm}(\mathsf{var}(\mathsf{plus}(m, n), \mathsf{first}(r)), \mathsf{rename_dom}(\mathsf{tail}(r), n))
\end{aligned}$$

This function is also partial.

#### 3.2.8 Matching Algorithm (Tests Whether a Termlist Matches Another One)

The algorithm matches calls an auxiliary algorithm matches_aux where the third argument is used to store the parts of the matcher already computed. Hence, $\mathsf{matches\_aux}(s, t, \sigma)$ is true iff $\sigma(s)$ matches $t$.

*function* $\mathsf{matches} : \mathsf{term} \times \mathsf{term} \rightarrow \mathsf{bool}$

$$\mathsf{matches}(s, t) = \mathsf{matches\_aux}(s, t, \mathsf{e})$$

$$
\begin{array}{ll}
\textit{function}\ \mathsf{matches\_aux} : \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{bool} & \\
\quad \mathsf{matches\_aux}(\mathsf{e}, t, \sigma) & = \mathsf{eqterm}(\mathsf{e}, t) \\
\quad \mathsf{matches\_aux}(\mathsf{var}(n, r), t, \sigma) & = \mathsf{not}(\mathsf{eqterm}(\mathsf{e}, t)) \wedge \\
& \quad \mathsf{if}(\mathsf{occurs}(n, \mathsf{dom}(\sigma)), \\
& \qquad \mathsf{eqterm}(\mathsf{first}(t), \mathsf{apply\_subst\_var}(\sigma, n)) \wedge \mathsf{matches\_aux}(r, \mathsf{tail}(t), \sigma), \\
& \qquad \mathsf{matches\_aux}(r, \mathsf{tail}(t), \mathsf{appendterm}(\mathsf{var}(n, \mathsf{first}(t)), \sigma))) \\
\quad \mathsf{matches\_aux}(\mathsf{func}(n, s, r), t, \sigma) & = \mathsf{first\_is\_func}(t) \wedge \mathsf{eq}(n, \mathsf{func\_name}(t)) \wedge \mathsf{eq}(\mathsf{length}(\mathsf{func\_args}(t)), \mathsf{length}(s))) \wedge \\
& \quad \mathsf{matches\_aux}(\mathsf{appendterm}(s, r), \mathsf{appendterm}(\mathsf{func\_args}(t), \mathsf{tail}(t)), \sigma)
\end{array}
$$

#### 3.2.9 Matching Algorithm (Computes the Matcher of Two Termlists)

$$
\begin{array}{l}
\textit{function}\ \mathsf{matcher} : \mathsf{term} \times \mathsf{term} \to \mathsf{term} \\
\quad \mathsf{matcher}(s, t) = \mathsf{matcher\_aux}(s, t, \mathsf{e})
\end{array}
$$

$$
\begin{array}{ll}
\textit{function}\ \mathsf{matcher\_aux} : \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{term} & \\
\quad \mathsf{matcher\_aux}(\mathsf{e}, \mathsf{e}, \sigma) & = \sigma \\
\quad \mathsf{matcher\_aux}(\mathsf{var}(n, r), t, \sigma) & = \mathsf{if}(\mathsf{occurs}(n, \mathsf{dom}(\sigma)), \\
& \qquad \mathsf{matcher\_aux}(r, \mathsf{tail}(t), \sigma), \\
& \qquad \mathsf{matcher\_aux}(r, \mathsf{tail}(t), \mathsf{appendterm}(\mathsf{var}(n, \mathsf{first}(t)), \sigma))) \\
\quad \mathsf{matcher\_aux}(\mathsf{func}(n, s, r), t, \sigma) & = \mathsf{matcher\_aux}(\mathsf{appendterm}(s, r), \mathsf{appendterm}(\mathsf{func\_args}(t), \mathsf{tail}(t)), \sigma)
\end{array}
$$

This function is partial (it is only defined if the first argument matches the second).

#### 3.2.10 Unification Algorithm (Tests Whether two Termlists are Unifiable)

$$
\begin{array}{ll}
\textit{function}\ \mathsf{unifies} : \mathsf{term} \times \mathsf{term} \to \mathsf{bool} & \\
\quad \mathsf{unifies}(\mathsf{e}, t) & = \mathsf{eqterm}(\mathsf{e}, t) \\
\quad \mathsf{unifies}(\mathsf{var}(n_1, r_1), \mathsf{e}) & = \mathsf{false} \\
\quad \mathsf{unifies}(\mathsf{var}(n_1, r_1), \mathsf{var}(n_2, r_2)) & = \mathsf{unifies}(\mathsf{apply\_subst}(\mathsf{var}(n_1, \mathsf{var}(n_2, \mathsf{e})), r_1), \\
& \qquad\qquad \mathsf{apply\_subst}(\mathsf{var}(n_1, \mathsf{var}(n_2, \mathsf{e})), r_2)) \\
\quad \mathsf{unifies}(\mathsf{var}(n_1, r_1), \mathsf{func}(n_2, s_2, r_2)) & = \mathsf{not}(\mathsf{occurs}(n_1, s_2)) \wedge \\
& \quad \mathsf{unifies}(\mathsf{apply\_subst}(\mathsf{var}(n_1, \mathsf{func}(n_2, s_2, \mathsf{e})), r_1), \\
& \qquad\qquad \mathsf{apply\_subst}(\mathsf{var}(n_1, \mathsf{func}(n_2, s_2, \mathsf{e})), r_2)) \\
\quad \mathsf{unifies}(\mathsf{func}(n_1, s_1, r_1), \mathsf{e}) & = \mathsf{false} \\
\quad \mathsf{unifies}(\mathsf{func}(n_1, s_1, r_1), \mathsf{var}(n_2, r_2)) & = \mathsf{unifies}(\mathsf{var}(n_2, r_2), \mathsf{func}(n_1, s_1, r_1)) \\
\quad \mathsf{unifies}(\mathsf{func}(n_1, s_1, r_1), \mathsf{func}(n_2, s_2, r_2)) & = \mathsf{eq}(n_1, n_2) \wedge \mathsf{eq}(\mathsf{length}(s_1), \mathsf{length}(s_2)) \wedge \\
& \quad \mathsf{unifies}(\mathsf{appendterm}(s_1, r_1), \mathsf{appendterm}(s_2, r_2))
\end{array}
$$

#### 3.2.11 Unification Algorithm (Computes the Most General Unifier of two Termlists)

$$
\begin{array}{ll}
\textit{function}\ \mathsf{mgu} : \mathsf{term} \times \mathsf{term} \to \mathsf{bool} & \\
\quad \mathsf{mgu}(\mathsf{e}, \mathsf{e}) & = \mathsf{e} \\
\quad \mathsf{mgu}(\mathsf{var}(n_1, r_1), t) & = \mathsf{if}(\mathsf{eqterm}(\mathsf{var}(n_1, \mathsf{e}), \mathsf{first}(t)), \\
& \qquad \mathsf{mgu}(r_1, \mathsf{tail}(t)), \\
& \qquad \mathsf{appendterm}(\mathsf{var}(n_1, \mathsf{first}(t)), \\
& \qquad\qquad \mathsf{mgu}(\mathsf{apply\_subst}(\mathsf{var}(n_1, \mathsf{first}(t)), r_1), \\
& \qquad\qquad\quad \mathsf{apply\_subst}(\mathsf{var}(n_1, \mathsf{first}(t)), \mathsf{tail}(t)))) \\
\quad \mathsf{mgu}(\mathsf{func}(n_1, s_1, r_1), \mathsf{var}(n_2, r_2)) & = \mathsf{mgu}(\mathsf{var}(n_2, r_2), \mathsf{func}(n_1, s_1, r_1)) \\
\quad \mathsf{mgu}(\mathsf{func}(n_1, s_1, r_1), \mathsf{func}(n_2, s_2, r_2)) & = \mathsf{mgu}(\mathsf{appendterm}(s_1, r_1), \mathsf{appendterm}(s_2, r_2))
\end{array}
$$

This function is partial (it is only defined if the arguments are unifiable).

#### 3.2.12 Test Whether One Substitution is A Specialization of Another

The following algorithm tests whether one substitution is more special than another, i.e. $\mathsf{special\_subst}(\sigma, \tau)$ is true iff $\sigma = \sigma \circ \tau$. Note that $\sigma = \sigma \circ \tau$ only has to be tested for all elements of the domain of $\tau$.

*function* special_subst : term × term → bool

$$\begin{aligned}
&\mathsf{special_subst}(\sigma, \mathsf{e}) && = \mathsf{true}\\
&\mathsf{special_subst}(\sigma, \mathsf{var}(n,t)) && = \mathsf{eqterm}(\mathsf{apply_subst_var}(\sigma,n), \mathsf{apply_subst}(\sigma, \mathsf{first}(t))) \wedge \mathsf{special_subst}(\sigma, \mathsf{tail}(t))
\end{aligned}$$

#### 3.2.13 Check Whether a Substitution Contains no Duplicates

no_duplicates checks whether a substitution contains two variable-term pairs with the same variable.

*function* no_duplicates : term → bool

$$\begin{aligned}
&\mathsf{no_duplicates}(\mathsf{e}) && = \mathsf{true}\\
&\mathsf{no_duplicates}(\mathsf{var}(n,t)) && = \mathsf{if}(\mathsf{occurs}(n, \mathsf{dom}(\mathsf{tail}(t))), \mathsf{false}, \mathsf{no_duplicates}(\mathsf{tail}(t)))
\end{aligned}$$

#### 3.2.14 Composition of Substitutions

The next algorithm composes substitutions where $\mathsf{compose}(\tau, \sigma)$ computes $\sigma \circ \tau$, i.e. $\tau$ is applied first. Note that the order of the variables in $\mathsf{compose}(\tau, \sigma)$ is the one in $\sigma$ followed by those variables occurring only in the domain of $\tau$. This is necessary in order to guarantee that $\sigma = \sigma \circ \mathsf{mgu}(s,t)$ holds for every unifier $\sigma$ of $s$ and $t$. For that purpose, the algorithm compose uses an auxiliary algorithm compose_aux.

*function* compose : term × term → term

$$\mathsf{compose}(\tau, \sigma) = \mathsf{compose\_aux}(\tau, \sigma, \mathsf{disjoint\_union}(\mathsf{dom}(\sigma), \mathsf{dom}(\tau)))$$

#### 3.2.15 Composition of Substitutions on a Certain Domain

The algorithm $\mathsf{compose\_aux}(\tau, \sigma, v)$ computes the restriction of $\sigma \circ \tau$ on the domain $v$.

*function* compose_aux : term × term × term → term

$$\begin{aligned}
&\mathsf{compose_aux}(\tau, \sigma, \mathsf{e}) && = \mathsf{e}\\
&\mathsf{compose_aux}(\tau, \sigma, \mathsf{var}(n,t)) && = \mathsf{if}(\mathsf{eqterm}(\mathsf{apply_subst}(\sigma, \mathsf{apply_subst_var}(\tau,n)), \mathsf{var}(n,\mathsf{e})) \wedge\\
& && \qquad \mathsf{not}(\mathsf{occurs}(n, \mathsf{dom}(\sigma))),\\
& && \qquad \mathsf{compose_aux}(\tau, \sigma, t),\\
& && \qquad \mathsf{var}(n, \mathsf{addterm}(\mathsf{apply_subst}(\sigma, \mathsf{apply_subst_var}(\tau, n)),\\
& && \qquad\qquad \mathsf{compose_aux}(\tau, \sigma, t)))
\end{aligned}$$

#### 3.2.16 Changing a Substitution in One Argument

The result of $\mathsf{replace}(\sigma, n, s)$ is like $\sigma$, but if $\sigma$ already contained a value for the variable $n$, then this value is now changed to $s$.

*function* replace : term × nat × term → term

$$\begin{aligned}
&\mathsf{replace}(\mathsf{e}, n, s) && = \mathsf{e}\\
&\mathsf{replace}(\mathsf{var}(m,t), n, s) && = \mathsf{if}(\mathsf{eq}(m,n),\\
& && \qquad \mathsf{var}(n, \mathsf{addterm}(s, \mathsf{tail}(t))),\\
& && \qquad \mathsf{appendterm}(\mathsf{var}(m, \mathsf{first}(t)), \mathsf{replace}(\mathsf{tail}(t), n, s)))
\end{aligned}$$

### 3.3 Algorithms on tll

#### 3.3.1 Appending two Lists of Termlists

*function* append : tll × tll → tll

$$\begin{aligned}
&\mathsf{append}(\mathsf{empty}, l) && = l\\
&\mathsf{append}(\mathsf{add}(t, l_1), l_2) && = \mathsf{add}(t, \mathsf{append}(l_1, l_2))
\end{aligned}$$

### 3.3.2 Member on tll

*function* member : term × tll → bool

$\mathsf{member}(t, \mathsf{empty}) = \mathsf{false}$

$\mathsf{member}(t, \mathsf{add}(r, l)) = \mathsf{if}(\mathsf{eqterm}(t, r), \mathsf{true}, \mathsf{member}(t, l))$

### 3.3.3 Test Whether One tll is a Subset of Another

*function* subseteq_list : tll × tll → bool

$\mathsf{subseteq\_list}(\mathsf{empty}, k) = \mathsf{true}$

$\mathsf{subseteq\_list}(\mathsf{add}(t, l), k) = \mathsf{if}(\mathsf{member}(t, k), \mathsf{subseteq\_list}(l, k), \mathsf{false})$

### 3.3.4 Remove all Occurrences of an Element from a tll

*function* remove : term × tll → tll

$\mathsf{remove}(t, \mathsf{empty}) = \mathsf{empty}$

$\mathsf{remove}(t, \mathsf{add}(s, k)) = \mathsf{if}(\mathsf{eqterm}(t, s), \mathsf{remove}(t, k), \mathsf{add}(s, \mathsf{remove}(t, k)))$

### 3.3.5 Compute the Number of Elements Contained in One tll but not in the Other

*function* setdiff : tll × tll → nat

$\mathsf{setdiff}(\mathsf{empty}, k) = 0$

$\mathsf{setdiff}(\mathsf{add}(t, l), k) = \mathsf{if}(\mathsf{member}(t, k), \mathsf{setdiff}(\mathsf{remove}(t, l), \mathsf{remove}(t, k)), \mathsf{s}(\mathsf{setdiff}(\mathsf{remove}(t, l), k)))$

### 3.3.6 Test Whether Two tll's are Disjoint

*function* disjoint_list : tll × tll → bool

$\mathsf{disjoint\_list}(\mathsf{empty}, l) = \mathsf{true}$

$\mathsf{disjoint\_list}(\mathsf{add}(t, k), l) = \mathsf{if}(\mathsf{member}(t, l), \mathsf{false}, \mathsf{disjoint\_list}(k, l))$

### 3.3.7 Test Whether a tll is Empty

*function* is_empty : tll → bool

$\mathsf{is\_empty}(\mathsf{empty})) = \mathsf{true}$

$\mathsf{is\_empty}(\mathsf{add}(t, k)) = \mathsf{false}$

### 3.3.8 Test Whether the Length of a tll is Even

*function* hasevenlength : tll → bool

$\mathsf{hasevenlength}(\mathsf{empty}) = \mathsf{true}$

$\mathsf{hasevenlength}(\mathsf{add}(t, \mathsf{empty})) = \mathsf{false}$

$\mathsf{hasevenlength}(\mathsf{add}(t_1, \mathsf{add}(t_2, l))) = \mathsf{hasevenlength}(l)$

### 3.3.9 List of all First Elements of a tll

*function* first_list : tll → bool

$\mathsf{first\_list}(\mathsf{empty}) = \mathsf{empty}$

$\mathsf{first\_list}(\mathsf{add}(r, l)) = \mathsf{add}(\mathsf{first}(r), \mathsf{first\_list}(l))$

### 3.3.10 List of all Tails of a tll

*function* tail_list : tll → bool

$\mathsf{tail\_list}(\mathsf{empty}) = \mathsf{empty}$

$\mathsf{tail\_list}(\mathsf{add}(r, l)) = \mathsf{add}(\mathsf{tail}(r), \mathsf{tail\_list}(l))$

### 3.3.11 Applying a Function to all Termlists in a $\mathsf{tll}$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textit{function}\ \mathsf{apply} : \mathsf{nat} \times \mathsf{tll} \to \mathsf{tll}} \\
\ \mathsf{apply}(n, \mathsf{empty}) & = \mathsf{empty} \\
\ \mathsf{apply}(n, \mathsf{add}(s, l)) & = \mathsf{add}(\mathsf{func}(n, s, \mathsf{e}), \mathsf{apply}(n, l))
\end{array}
$$

Hence, $\mathsf{apply}(n, \langle s_1, \ldots, s_n \rangle) = \langle \mathsf{func}(n, s_1, \mathsf{e}), \ldots, \mathsf{func}(n, s_n, \mathsf{e}) \rangle$. This function is total.

### 3.3.12 Applying a Function to Every Second Termlist in a $\mathsf{tll}$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textit{function}\ \mathsf{apply\_narrowlist} : \mathsf{nat} \times \mathsf{tll} \to \mathsf{tll}} \\
\ \mathsf{apply\_narrowlist}(n, \mathsf{empty}) & = \mathsf{empty} \\
\ \mathsf{apply\_narrowlist}(n, \mathsf{add}(\sigma, \mathsf{add}(s, l))) & = \mathsf{add}(\sigma, \mathsf{add}(\mathsf{func}(n, s, \mathsf{e}), \mathsf{apply\_narrowlist}(n, l)))
\end{array}
$$

Hence, $\mathsf{apply\_narrowlist}(n, \langle \sigma_1, s_1, \ldots, \sigma_n, s_n \rangle) = \langle \sigma_1, \mathsf{func}(n, s_1, \mathsf{e}), \ldots, \sigma_n, \mathsf{func}(n, s_n, \mathsf{e}) \rangle$. (This function is partial, i.e. it is only defined on lists of even length.)

### 3.3.13 Appending a Termlist to Every Term in a $\mathsf{tll}$ (in the back)

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textit{function}\ \mathsf{addtail} : \mathsf{tll} \times \mathsf{term} \to \mathsf{tll}} \\
\ \mathsf{addtail}(\mathsf{empty}, t) & = \mathsf{empty} \\
\ \mathsf{addtail}(\mathsf{add}(s, l), t) & = \mathsf{add}(\mathsf{addterm}(s, t), \mathsf{addtail}(l, t))
\end{array}
$$

Hence, $\mathsf{addtail}(\langle s_1, \ldots, s_n \rangle, [t_1 \ldots t_m]) = \langle [s_1\, t^*], \ldots, [s_n\, t^*] \rangle$.

### 3.3.14 Adding a Term to Every Termlist in a $\mathsf{tll}$ (in the front)

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textit{function}\ \mathsf{addfirst} : \mathsf{term} \times \mathsf{tll} \to \mathsf{tll}} \\
\ \mathsf{addfirst}(t, \mathsf{empty}) & = \mathsf{empty} \\
\ \mathsf{addfirst}(t, \mathsf{add}(s, l)) & = \mathsf{add}(\mathsf{addterm}(t, s), \mathsf{addfirst}(t, l))
\end{array}
$$

Hence, $\mathsf{addfirst}(t, \langle s_1, \ldots, s_n \rangle) = \langle [t\, s_1], \ldots, [t\, s_n] \rangle$.

### 3.3.15 Computing all Combinations of two $\mathsf{tll}$'s

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textit{function}\ \mathsf{all\_combinations} : \mathsf{tll} \times \mathsf{tll} \to \mathsf{tll}} \\
\ \mathsf{all\_combinations}(k, \mathsf{empty}) & = \mathsf{empty} \\
\ \mathsf{all\_combinations}(k, \mathsf{add}(s, l)) & = \mathsf{append}(\mathsf{addtail}(k, s), \mathsf{all\_combinations}(k, l))
\end{array}
$$

Hence, $\mathsf{all\_combinations}(\langle k_1 \ldots k_n \rangle, \langle s_1 \ldots s_m \rangle) = \langle [k_1\, s_1], [k_2\, s_1], \ldots, [k_n\, s_1], \ldots, [k_1\, s_m], [k_2\, s_m], \ldots, [k_n\, s_m] \rangle$.

### 3.3.16 Appending an Instantiated Termlist to Every Second Term in a $\mathsf{tll}$ (in the back)

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textit{function}\ \mathsf{back\_narrowlist} : \mathsf{tll} \times \mathsf{term} \to \mathsf{tll}} \\
\ \mathsf{back\_narrowlist}(\mathsf{empty}, t) & = \mathsf{empty} \\
\ \mathsf{back\_narrowlist}(\mathsf{add}(\sigma, \mathsf{add}(s, l)), t) & = \mathsf{add}(\sigma, \mathsf{add}(\mathsf{addterm}(s, \mathsf{apply\_subst}(\sigma, t)), \mathsf{back\_narrowlist}(l, t)))
\end{array}
$$

Hence, $\mathsf{back\_narrowlist}(\langle \sigma_1, s_1, \ldots, \sigma_n, s_n \rangle, [t_1 \ldots t_m]) = \langle \sigma_1, [s_1,\, \sigma_1(t^*)], \ldots, \sigma_n, [s_n,\, \sigma_n(t^*)] \rangle$.

### 3.3.17 Appending a Termlist to Every Termlist in a $\mathsf{tll}$ (in the front)

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textit{function}\ \mathsf{append\_list} : \mathsf{term} \times \mathsf{tll} \to \mathsf{tll}} \\
\ \mathsf{append\_list}(t, \mathsf{empty}) & = \mathsf{empty} \\
\ \mathsf{append\_list}(t, \mathsf{add}(s, l)) & = \mathsf{add}(\mathsf{appendterm}(t, s), \mathsf{append\_list}(t, l))
\end{array}
$$

Hence, $\mathsf{append\_list}(t^*, \langle [s_{1,1} \ldots s_{1,m_1}], \ldots, [s_{n,1} \ldots s_{n,m_n}] \rangle) = \langle [t^*\, s_{1,1} \ldots s_{1,m_1}], \ldots, [t^*\, s_{n,1} \ldots s_{n,m_n}] \rangle$.

### 3.3.18 Adding an Instantiated Term to Every Second Termlist in a $\mathsf{tll}$ (in the front)

$$\begin{array}{ll}
\textit{function}\ \mathsf{add_narrowlist} : \mathsf{term} \times \mathsf{tll} \to \mathsf{tll} & \\
\quad \mathsf{add_narrowlist}(t, \mathsf{empty}) & = \mathsf{empty} \\
\quad \mathsf{add_narrowlist}(t, \mathsf{add}(\sigma, \mathsf{add}(s, l))) & = \mathsf{add}(\sigma, \mathsf{add}(\mathsf{addterm}(\mathsf{apply_subst}(\sigma, t), s), \mathsf{add_narrowlist}(t, l)))
\end{array}$$

Hence, $\mathsf{add\_narrowlist}(t, \langle \sigma_1, [s_{1,1} \dots s_{1,m_1}], \dots, \sigma_n, [s_{n,1} \dots s_{n,m_n}] \rangle) = \langle \sigma_1, [\sigma_1(t), s_{1,1}, \dots, s_{1,m_1}], \dots, \sigma_n, [\sigma_n(t), s_{n,1}, \dots, s_{n,m_n}] \rangle$. (This function is again partial.)

### 3.3.19 Removing the Odd Elements from a $\mathsf{tll}$

$$\begin{array}{ll}
\textit{function}\ \mathsf{remove_subst} : \mathsf{tll} \to \mathsf{term} & \\
\quad \mathsf{remove_subst}(\mathsf{empty}) & = \mathsf{e} \\
\quad \mathsf{remove_subst}(\mathsf{add}(\sigma, \mathsf{add}(s, l))) & = \mathsf{appendterm}(s, \mathsf{remove_subst}(l))
\end{array}$$

Hence, $\mathsf{remove\_subst}(\langle \sigma_1, s_1^*, \dots, \sigma_n, s_n^* \rangle) = [s_1^* \dots s_n^*]$.

### 3.3.20 Check Whether a Pair is a Specialization of a Pair in a Narrowlist

The function $\mathsf{special}(\sigma, s, l)$ (where $l$ is a $\mathsf{tll}$) says whether there are consecutive elements (starting on even position) $\tau, q$ in $l$ such that $\sigma, s$ is a special case of $\tau, q$, i.e. such that $\sigma = \sigma \circ \tau$ and $s = \sigma(q)$.

$$\begin{array}{ll}
\textit{function}\ \mathsf{special} : \mathsf{term} \times \mathsf{term} \times \mathsf{tll} \to \mathsf{tll} & \\
\quad \mathsf{special}(\sigma, s, \mathsf{empty}) & = \mathsf{false} \\
\quad \mathsf{special}(\sigma, s, \mathsf{add}(\tau, \mathsf{add}(q, l))) & = \ \ \mathsf{special_subst}(\sigma, \tau) \wedge \mathsf{eqterm}(s, \mathsf{apply_subst}(\sigma, q)) \\
& \quad\ \vee \mathsf{special}(\sigma, s, l)
\end{array}$$

### 3.3.21 Adding Terms from Two $\mathsf{tll}$'s

$$\begin{array}{ll}
\textit{function}\ \mathsf{addtermtwice} : \mathsf{tll} \times \mathsf{tll} \to \mathsf{tll} & \\
\quad \mathsf{addtermtwice}(\mathsf{empty}, \mathsf{empty}) & = \mathsf{empty} \\
\quad \mathsf{addtermtwice}(\mathsf{add}(s, l_1), \mathsf{add}(t, l_2)) & = \mathsf{add}(\mathsf{addterm}(s, t), \mathsf{addtermtwice}(l_1, l_2))
\end{array}$$

### 3.3.22 Check Whether a $\mathsf{tll}$ only Consists of one Element

$$\begin{array}{ll}
\textit{function}\ \mathsf{onlyconsistsof} : \mathsf{tll} \times \mathsf{term} \to \mathsf{bool} & \\
\quad \mathsf{onlyconsistsof}(\mathsf{empty}, t) & = \mathsf{true} \\
\quad \mathsf{onlyconsistsof}(\mathsf{add}(s, l), t) & = \mathsf{eqterm}(s, t) \wedge \mathsf{onlyconsistsof}(l, t)
\end{array}$$

### 3.3.23 Applying a Function to Two $\mathsf{tll}$'s

$$\begin{array}{ll}
\textit{function}\ \mathsf{applytwice} : \mathsf{nat} \times \mathsf{tll} \times \mathsf{tll} \to \mathsf{tll} & \\
\quad \mathsf{applytwice}(n, \mathsf{empty}, \mathsf{empty}) & = \mathsf{empty} \\
\quad \mathsf{applytwice}(n, \mathsf{add}(s, l_1), \mathsf{add}(t, l_2)) & = \mathsf{add}(\mathsf{func}(n, s, t), \mathsf{applytwice}(n, l_1, l_2))
\end{array}$$

## 3.4 Algorithms for Rewriting

### 3.4.1 Check Whether One Termlist Rewrites to Another With a Certain Rule in One Step

The following algorithm $\mathsf{rewrites\_rule}(t, s, l, r)$ returns $\mathsf{true}$ iff $t$ can be rewritten to $s$ (in one step) using the rule $l \to r$.

$$
\begin{array}{lll}
\textit{function}\ \mathsf{rewrites\_rule} : \mathsf{term} \times \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{bool} \\
\quad \mathsf{rewrites\_rule}(\mathsf{e}, s, l, r) & = & \mathsf{false} \\
\quad \mathsf{rewrites\_rule}(\mathsf{var}(n, t), s, l, r) & = & \mathsf{eqterm}(\mathsf{first}(s), \mathsf{var}(n, \mathsf{e})) \wedge \mathsf{rewrites\_rule}(t, \mathsf{tail}(s), l, r) \\
\quad \mathsf{rewrites\_rule}(\mathsf{func}(n, u, t), s, l, r) & = & \quad \mathsf{eqterm}(\mathsf{first}(s), \mathsf{func}(n, u, \mathsf{e})) \wedge \mathsf{rewrites\_rule}(t, \mathsf{tail}(s), l, r) \\
& & \vee\ \mathsf{first\_is\_func}(s) \wedge \mathsf{eq}(\mathsf{func\_name}(s), n) \wedge \\
& & \quad \mathsf{eqterm}(\mathsf{tail}(s), t) \wedge \mathsf{rewrites\_rule}(u, \mathsf{func\_args}(s), l, r) \\
& & \vee\ \mathsf{matches}(l, \mathsf{func}(n, u, \mathsf{e})) \wedge \\
& & \quad \mathsf{eqterm}(\mathsf{first}(s), \mathsf{apply\_subst}(\mathsf{matcher}(l, \mathsf{func}(n, u, \mathsf{e})), r)) \wedge \\
& & \quad \mathsf{eqterm}(\mathsf{tail}(s), t)
\end{array}
$$

### 3.4.2 Compute the Matcher Used in a Reduction

The next algorithm returns the matcher used in the reduction. Note that this algorithm is partial.

$$
\begin{array}{lll}
\textit{function}\ \mathsf{rewrites\_matcher} : \mathsf{term} \times \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{term} \\
\quad \mathsf{rewrites\_matcher}(\mathsf{var}(n, t), s, l, r) & = & \mathsf{rewrites\_matcher}(t, \mathsf{tail}(s), l, r) \\
\quad \mathsf{rewrites\_matcher}(\mathsf{func}(n, u, t), s, l, r) & = & \mathsf{if}(\mathsf{eqterm}(\mathsf{first}(s), \mathsf{func}(n, u, \mathsf{e})), \\
& & \quad \mathsf{rewrites\_matcher}(t, \mathsf{tail}(s), l, r), \\
& & \quad \mathsf{if}(\mathsf{rewrites\_rule}(u, \mathsf{func\_args}(s), l, r), \\
& & \qquad \mathsf{rewrites\_matcher}(u, \mathsf{func\_args}(s), l, r), \\
& & \qquad \mathsf{matcher}(l, \mathsf{func}(n, u, \mathsf{e}))))
\end{array}
$$

### 3.4.3 Check Whether One Termlist Rewrites to Another w.r.t. a TRS in One Step

In the next algorithm, $\mathsf{rewrites}(t, s, R)$ is $\mathsf{true}$ iff $t$ can be reduced to $s$ in one step using a rule of the TRS $R$. Again this algorithm is partial.

$$
\begin{array}{lll}
\textit{function}\ \mathsf{rewrites} : \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{bool} \\
\quad \mathsf{rewrites}(t, s, \mathsf{e}) & = & \mathsf{false} \\
\quad \mathsf{rewrites}(t, s, \mathsf{func}(n, u, r)) & = & \mathsf{if}(\mathsf{rewrites\_rule}(t, s, \mathsf{func}(n, u, \mathsf{e}), \mathsf{first}(r)), \mathsf{true}, \mathsf{rewrites}(t, s, \mathsf{tail}(r)))
\end{array}
$$

### 3.4.4 Compute the Rule Used in a Reduction

The following algorithm $\mathsf{rule}(t, s, R)$ returns the rule used in the reduction of $t$ to $s$. The algorithm is only defined if $t$ indeed rewrites to $s$ in one step.

$$
\begin{array}{l}
\textit{function}\ \mathsf{rule} : \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{term} \\
\quad \mathsf{rule}(t, s, \mathsf{func}(n, u, r)) = \mathsf{if}(\mathsf{rewrites\_rule}(t, s, \mathsf{func}(n, u, \mathsf{e}), \mathsf{first}(r)), \mathsf{func}(n, u, \mathsf{first}(r)), \mathsf{rule}(t, s, \mathsf{tail}(r)))
\end{array}
$$

### 3.4.5 Generate all Termlists Obtained in One Rewrite Step

The following algorithm $\mathsf{rewrite\_rule}(t, l, r)$ generates all termlists that can be obtained from $t$ by applying one rewrite step with the rule $l \to r$. More precisely, $[s_1, \ldots, s_n]$ is in $\mathsf{rewrite\_rule}([t_1, \ldots, t_n], l, r)$ iff there exists an $i$ such that $t_i \to_{l \to r} s_i$ and $t_j = s_j$ for all $j \neq i$. This algorithm is total.

$$
\begin{array}{lll}
\textit{function}\ \mathsf{rewrite\_rule} : \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{tll} \\
\quad \mathsf{rewrite\_rule}(\mathsf{e}, l, r) & = & \mathsf{empty} \\
\quad \mathsf{rewrite\_rule}(\mathsf{var}(n, t), l, r) & = & \mathsf{append\_list}(\mathsf{var}(n, \mathsf{e}), \mathsf{rewrite\_rule}(t, l, r)) \\
\quad \mathsf{rewrite\_rule}(\mathsf{func}(n, u, t), l, r) & = & \mathsf{append}(\mathsf{addtail}(\mathsf{if}(\mathsf{matches}(l, \mathsf{func}(n, u, \mathsf{e})), \\
& & \qquad\qquad \mathsf{add}(\mathsf{apply\_subst}(\mathsf{matcher}(l, \mathsf{func}(n, u, \mathsf{e})), r), \\
& & \qquad\qquad\quad \mathsf{apply}(n, \mathsf{rewrite\_rule}(u, l, r))), \\
& & \qquad\qquad \mathsf{apply}(n, \mathsf{rewrite\_rule}(u, l, r))), \\
& & \qquad\quad t), \\
& & \quad \mathsf{append\_list}(\mathsf{func}(n, u, \mathsf{e}), \mathsf{rewrite\_rule}(t, l, r)))
\end{array}
$$

### 3.4.6 Compute All Substitutions Obtainable by One Rewrite Step

The next algorithm $\mathsf{all\_reductions}(\sigma, l, r)$ returns the list of all substitutions (as a $\mathsf{tll}$) which result from $\sigma$ by applying the rule $l \to r$ once to one term in $\sigma$'s domain.

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textit{function}\ \mathsf{all\_reductions} : \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{tll}} \\
\quad \mathsf{all\_reductions}(\mathsf{e}, l, r) & = \mathsf{empty} \\
\quad \mathsf{all\_reductions}(\mathsf{var}(n,t), l, r) & = \mathsf{append}(\mathsf{append\_list}(\mathsf{var}(n,\mathsf{e}), \mathsf{addtail}(\mathsf{rewrite\_rule}(\mathsf{first}(t), l, r), \mathsf{tail}(t))), \\
& \qquad\qquad\quad \mathsf{append\_list}(\mathsf{var}(n, \mathsf{first}(t)), \mathsf{all\_reductions}(\mathsf{tail}(t), l, r))
\end{array}
$$

### 3.4.7 Generate all Termlists Obtained in One Rewrite Step (by a Certain Rule) from a tll

The following algorithm $\mathsf{rewrite\_rule\_list}(k, l, r)$ (which is similar to $\mathsf{rewrite\_rule}$) generates the list of all termlists $t'$, such that $t'$ results from one $t \in k$ by one rewrite step with the rule $l \to r$.

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textit{function}\ \mathsf{rewrite\_rule\_list} : \mathsf{tll} \times \mathsf{term} \times \mathsf{term} \to \mathsf{tll}} \\
\quad \mathsf{rewrite\_rule\_list}(\mathsf{empty}, l, r) & = \mathsf{empty} \\
\quad \mathsf{rewrite\_rule\_list}(\mathsf{add}(t,k), l, r) & = \mathsf{append}(\mathsf{rewrite\_rule}(t, l, r), \mathsf{rewrite\_rule\_list}(k, l, r))
\end{array}
$$

### 3.4.8 Check Whether a tll Rewrites To a Termlist in Arbitrary Many Steps

The next algorithm checks whether one of the termlists in the $\mathsf{tll}$ $k$ rewrites to $s$ using an arbitrary number of reductions with the rule $l \to r$. This algorithm is *inherently* partial, i.e. it may be non-terminating. Moreover, its domain is *undecidable*, since its termination corresponds to the termination of a one-rule term rewriting system [6].

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textit{function}\ \mathsf{rewrites\_rule\_list*} : \mathsf{tll} \times \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{bool}} \\
\quad \mathsf{rewrites\_rule\_list*}(\mathsf{empty}, s, l, r) & = \mathsf{false} \\
\quad \mathsf{rewrites\_rule\_list*}(\mathsf{add}(t,k), s, l, r) & = \mathsf{if}(\mathsf{member}(s, \mathsf{add}(t,k)), \\
& \qquad \mathsf{true}, \\
& \qquad \mathsf{if}(\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(\mathsf{add}(t,k), l, r), \mathsf{add}(t,k)), \\
& \qquad\quad \mathsf{false}, \\
& \qquad\quad \mathsf{rewrites\_rule\_list*}(\mathsf{append}(\mathsf{add}(t,k), \mathsf{rewrite\_rule\_list}(\mathsf{add}(t,k), l, r)), \\
& \qquad\qquad\qquad\qquad\qquad s, l, r)))
\end{array}
$$

In this algorithm (and also in the corresponding following ones) we could have used a recursive call of the form $\mathsf{rewrites\_rule\_list*}(\mathsf{rewrite\_rule\_list}(\mathsf{add}(t,k), l, r))$ instead. But the reason for choosing the above formulation is that it simplifies the subsequent proofs on the correspondence between rewriting and joinability.

### 3.4.9 Check Whether a Termlist Rewrites To Another in Arbitrary Many Steps

Similarly, $\mathsf{rewrites\_rule*}(t, s, l, r)$ checks whether $t$ rewrites to $s$ using an arbitrary number of reductions with the rule $l \to r$.

$$
\begin{array}{l}
\textit{function}\ \mathsf{rewrites\_rule*} : \mathsf{term} \times \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{bool} \\
\quad \mathsf{rewrites\_rule*}(t, s, l, r) = \mathsf{rewrites\_rule\_list*}(\mathsf{add}(t, \mathsf{empty}), s, l, r)
\end{array}
$$

### 3.4.10 Check Whether a Termlist Rewrites To All Termlists from a tll in Arbitrary Many Steps

The following algorithm $\mathsf{rewrite^*\_all}(s, \langle t_1 \ldots t_n \rangle, l, r)$ checks whether $\mathsf{rewrites\_rule*}(s, t_i, l, r)$ holds for all $t_i$. (Here, $\langle t_1 \ldots t_n \rangle$ is a $\mathsf{tll}$.)

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textit{function}\ \mathsf{rewrite^*\_all} : \mathsf{term} \times \mathsf{tll} \times \mathsf{term} \times \mathsf{term} \to \mathsf{bool}} \\
\quad \mathsf{rewrite^*\_all}(s, \mathsf{empty}, l, r) & = \mathsf{true} \\
\quad \mathsf{rewrite^*\_all}(s, \mathsf{add}(t,k), l, r) & = \mathsf{rewrites\_rule*}(s, t, l, r) \wedge \mathsf{rewrite^*\_all}(s, k, l, r)
\end{array}
$$

### 3.4.11 Check Whether Every Termlist of a tll is Reachable From Another tll

The following algorithm rewrites_list*_all$(k_1, k_2, l, r)$ checks if every termlist of $k_2$ can be reached by rewriting a termlist from $k_1$.

*function* rewrites_list*_all : tll × tll × term × term → bool
rewrites_list*_all$(k,$ empty$, l, r)$ = true
rewrites_list*_all$(k,$ add$(s, k'), l, r)$ = rewrites_rule_list*$(k, s, l, r)$ ∧ rewrites_list*_all$(k, k', l, r)$

### 3.4.12 Check Whether a tll Rewrites To a Termlist from Another tll in Arbitrary Many Steps

The following variant of the above algorithm checks whether one of the termlists in the tll $k$ rewrites to one of the termlists in the tll $k'$ using an arbitrary number of reductions with the rule $l \to r$. Again, its domain is undecidable.

```
function rewrites_rule_list*_exists : tll × tll × term × term → bool
rewrites_rule_list*_exists(empty, k', l, r)              = false
rewrites_rule_list*_exists(add(t, k), empty, l, r)       = false
rewrites_rule_list*_exists(add(t, k), add(s, k'), l, r)  =

      if(disjoint_list(add(s, k'), add(t, k)),
         if(subseteq_list(rewrite_rule_list(add(t, k), l, r), add(t, k)),
            false,
            rewrites_rule_list*_exists(append(add(t, k), rewrite_rule_list(add(t, k), l, r)),
                                       add(s, k'),
                                       l,
                                       r)),
         true)
```

### 3.4.13 Check Whether a Termlist Rewrites To a Termlist from a tll in Arbitrary Many Steps

The next algorithm rewrite*_exists$(s, \langle t_1 \ldots t_n \rangle, l, r)$ checks whether rewrites_rule*$(s, t_i, l, r)$ holds for one $t_i$. (Here, $\langle t_1 \ldots t_n \rangle$ is a tll.)

*function* rewrite*_exists : term × tll × term × term → bool
rewrite*_exists$(s, k, l, r)$ = rewrites_rule_list*_exists(add$(s,$ empty$), k, l, r)$

### 3.4.14 Check Whether a tll Rewrites To a Termlist From Another tll in Arbitrary Many Steps via a TRS

This algorithm is like rewrites_rule_list*_exists, but it works with a TRS $R$ instead of just a rule. Hence, rewrites_list*_exists$(k_1, k_2, R)$ checks whether one of the termlists of $k_1$ reduces to one of the termlists in $k_2$ w.r.t. the TRS $R$.

```
function rewrites_list*_exists : tll × tll × term → bool
rewrites_list*_exists(k1, k2, R) = if(disjoint_list(k1, k2),
                                      if(subseteq_list(rewrite_list(k1, R), k1),
                                         false,
                                         rewrites_list*_exists(append(k1, rewrite_list(k1, R)), k2, R)),
                                      true)
```

A modification of this algorithm (which was called rewrites*) was discussed in [9, Section 7]. To ease the presentation there, we simplified the arguments in the recursive call (cf. the remarks for the algorithm rewrites_rule_list*) and only presented a version of the algorithm operating on term's instead of tll's.

#### 3.4.15 Generate all Termlists Obtained in One Rewrite Step from a tll

The following algorithm rewrite_list$(k, R)$ is similar to rewrite_rule_list, but it computes the list of all termlists obtainable from any termlist of $k$ in one step by *any* rule of $R$.

*function* rewrite_list : tll × term → tll
  rewrite_list$(k,$ e$)$ = empty
  rewrite_list$(k,$ func$(n,s,t))$ = append(rewrite_rule_list$(k,$ func$(n,s,$ e$),$ first$(t))$, rewrite_list$(k,$ tail$(t)))$

#### 3.4.16 Check Whether a tll Rewrites To a Termlist w.r.t. a TRS in Arbitrary Many Steps

This function again has an undecidable domain.

*function* rewrites_list* : tll × term × term → bool
```
rewrites_list*(k, t, R) = if(member(t, k)
                             true,
                             if(subseteq_list(rewrite_list(k, R), k),
                               false,
                               rewrites_list*(append(k, rewrite_list(k, R)), t, R)))
```

#### 3.4.17 Check Whether One Termlist Rewrites To Another w.r.t. a TRS in Arbitrary Many Steps

This function also has an undecidable domain.

*function* rewrites* : term × term × term → bool
  rewrites*$(s,t,R)$ = rewrites_list*(add$(s,$ empty$)$$, t, R)$

### 3.5 Algorithms for Narrowing and Critical Pairs

#### 3.5.1 Check Whether a tll is a Narrowlist

We use a special kind of tll's for narrowing, viz. tll's of the form $\langle \sigma_1, s_1, \sigma_2, s_2, \ldots \rangle$, where $\sigma_i$ are substitutions and $s_i$ are terms of length 1. This algorithm checks if a tll is such a narrowlist.

*function* is_narrowlist : tll → bool
  is_narrowlist(empty) = true
  is_narrowlist(add$(t,$ empty$)$) = false
  is_narrowlist(add$(t,$ add$(s,k)))$ = is_subst$(t)$ ∧ eq(length$(s)$, s(0)) ∧ is_narrowlist$(k)$

#### 3.5.2 Computing Narrowings

The following algorithm narrow$(t,l,r)$ computes all narrowings of $t$ using the rule $l \to r$. More precisely, the result of narrow$([t_1,\ldots,t_n],l,r)$ is a tll where $\sigma$ and $[s_1,\ldots,s_n]$ are on positions $2j$ and $2j+1$ in narrow$([t_1,\ldots,t_n],l,r)$ iff there exists an $i \in \{1,\ldots,n\}$ such that $t_i$ narrows to $s_i$ via the mgu $\sigma$ using the rule $l \to r$ and $s_j = \sigma(t_j)$ for all $j \neq i$.

*function* narrow : term × term × term → tll
```
narrow(e, l, r)             = empty
narrow(var(n, t), l, r)     = add_narrowlist(var(n, e), narrow(t, l, r))
narrow(func(n, s, t), l, r) = append(back_narrowlist(if(unifies(l, func(n, s, e)),
                                                         add(mgu(l, func(n, s, e)),
                                                             add(apply_subst(mgu(l, func(n, s, e)), r),
                                                                 apply_narrowlist(n, narrow(s, l, r)))),
                                                         apply_narrowlist(n, narrow(s, l, r)))
                                                      t),
                                     add_narrowlist(func(n, s, e), narrow(t, l, r)))
```

#### 3.5.3 Critical Pairs of Two Rules

This function computes all critical pairs that can be built with two rules. More precisely, $\mathsf{cp\_rule}(l, r, l', r')$ is a list of terms were the first and the second one form a critical pair, the third and the fourth form a critical pair etc. A critical pair of $l \to r$ and $l' \to r'$ is a pair of terms $\sigma(r)$ and $l[\sigma(r')]_\pi$, if $l|_\pi$ is not a variable and $l|_\pi$ unifies with $l'$ using the mgu $\sigma$.

*function* $\mathsf{cp\_rule} : \mathsf{term} \times \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{term}$
$\mathsf{cp\_rule}(l, r, l', r') = \mathsf{remove\_subst}(\mathsf{add\_narrowlist}(r, \mathsf{narrow}(l, \mathsf{rename}(l', \mathsf{s}(\mathsf{max}(l))), \mathsf{rename}(r', \mathsf{s}(\mathsf{max}(l))))))$

### 3.6 Algorithms for Joinability

#### 3.6.1 Check Whether Two tll's Are Joinable

The algorithm $\mathsf{joinable\_list}(k_1, k_2, R)$ returns true iff there are termlists $s \in k_1$, $t \in k_2$ such that $s$ and $t$ are joinable. If this is not the case, then however it may be that joinable_list is non-terminating. Thus, this function is partial and its domain is undecidable.

*function* $\mathsf{joinable\_list} : \mathsf{tll} \times \mathsf{tll} \times \mathsf{term} \to \mathsf{bool}$

$$
\begin{aligned}
\mathsf{joinable\_list}(k_1, k_2, R) = \mathsf{if}(&\mathsf{disjoint\_list}(k_1, k_2),\\
&\mathsf{if}(\mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k_1, R), k_1) \wedge \mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k_2, R), k_2),\\
&\quad \mathsf{false},\\
&\quad \mathsf{joinable\_list}(\mathsf{append}(k_1, \mathsf{rewrite\_list}(k_1, R)), \mathsf{append}(k_2, \mathsf{rewrite\_list}(k_2, R)), R)),\\
&\mathsf{true})
\end{aligned}
$$

#### 3.6.2 Check Whether Two Termlists Are Joinable

This algorithm tests whether two termlists are joinable with the rules of a given TRS. Again, its domain is undecidable.

*function* $\mathsf{joinable} : \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{bool}$
$\mathsf{joinable}(s, t, R) = \mathsf{joinable\_list}(\mathsf{add}(s, \mathsf{empty}), \mathsf{add}(t, \mathsf{empty}), R)$

#### 3.6.3 Test Whether Elements in a List are Joinable

The algorithm $\mathsf{joinable\_pairs}([t_1, t_2, t_3, t_4, \ldots], R)$ checks whether $t_1$ and $t_2$ are joinable, whether $t_3$ and $t_4$ are joinable, etc.

*function* $\mathsf{joinable\_pairs} : \mathsf{term} \times \mathsf{term} \to \mathsf{bool}$

$$
\begin{aligned}
\mathsf{joinable\_pairs}(\mathsf{e}, R) &= \mathsf{true}\\
\mathsf{joinable\_pairs}(\mathsf{var}(n, t), R) &= \mathsf{rewrites^*}(\mathsf{first}(t), \mathsf{var}(n, \mathsf{e}), R) \wedge \mathsf{joinable\_pairs}(\mathsf{tail}(t), R)\\
\mathsf{joinable\_pairs}(\mathsf{func}(n, s, t), R) &= \mathsf{joinable}(\mathsf{func}(n, s, \mathsf{e}), \mathsf{first}(t)) \wedge \mathsf{joinable\_pairs}(\mathsf{tail}(t), R)
\end{aligned}
$$

#### 3.6.4 Check Whether all Critical Pairs of a TRS are Joinable

*function* $\mathsf{jcp} : \mathsf{term} \to \mathsf{bool}$
$\mathsf{jcp}(R) = \mathsf{jcp\_aux1}(R, R, R)$

#### 3.6.5 Check Whether all Critical Pairs of a TRS with Another One are Joinable

The algorithm $\mathsf{jcp\_aux1}(R_1, R_2, R_3)$ checks whether all critical pairs built by overlapping a rule of $R_1$ with a rule of $R_2$ are joinable using $R_3$.

*function* $\mathsf{jcp\_aux1} : \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{bool}$

$$
\begin{aligned}
\mathsf{jcp\_aux1}(\mathsf{e}, R, R') &= \mathsf{true}\\
\mathsf{jcp\_aux1}(\mathsf{func}(n, s, t), R, R') &= \mathsf{jcp\_aux2}(\mathsf{func}(n, s, \mathsf{e}), \mathsf{first}(t), R, R') \wedge \mathsf{jcp\_aux1}(\mathsf{tail}(t), R, R')
\end{aligned}
$$

### 3.6.6 Check Whether all Critical Pairs of a Rule with a TRS are Joinable

The algorithm $\mathsf{jcp\_aux2}(l, r, R, R')$ checks whether all critical pairs built by overlapping the rule $l \to r$ with a rule of $R$ are joinable using $R'$.

$$
\begin{array}{ll}
\textit{function}\ \mathsf{jcp\_aux2} : \mathsf{term} \times \mathsf{term} \times \mathsf{term} \times \mathsf{term} \to \mathsf{bool} & \\
\ \mathsf{jcp\_aux2}(l, r, \mathsf{e}, R') & = \mathsf{true} \\
\ \mathsf{jcp\_aux2}(l, r, \mathsf{func}(n, s, t), R') & = \mathsf{joinable\_pairs}(\mathsf{cp\_rule}(l, r, \mathsf{func}(n, s, \mathsf{e}), \mathsf{first}(t)), R') \land \\
 & \quad\ \mathsf{jcp\_aux2}(l, r, \mathsf{tail}(t), R')
\end{array}
$$

# 4 Theorems about Booleans, Naturals, Termlists, and tll's

## 4.1 Totality of not, eq, ge, gt, plus, eqterm, length, appendterm, first_is_func, vars, rename, remove, setdiff, trs

By structural induction (Rule 2″) the following theorems can easily be proved:

$$\mathsf{def}(x) \Rightarrow \mathsf{def}(\mathsf{not}(x)) \tag{1}$$
$$\mathsf{def}(x, y) \Rightarrow \mathsf{def}(\mathsf{eq}(x, y)) \tag{2}$$
$$\mathsf{def}(x, y) \Rightarrow \mathsf{def}(\mathsf{ge}(x, y)) \tag{3}$$
$$\mathsf{def}(x, y) \Rightarrow \mathsf{def}(\mathsf{gt}(x, y)) \tag{4}$$
$$\mathsf{def}(x, y) \Rightarrow \mathsf{def}(\mathsf{plus}(x, y)) \tag{5}$$
$$\mathsf{def}(s, t) \Rightarrow \mathsf{def}(\mathsf{eqterm}(s, t)) \tag{6}$$
$$\mathsf{def}(t) \Rightarrow \mathsf{def}(\mathsf{length}(t)) \tag{7}$$
$$\mathsf{def}(s, t) \Rightarrow \mathsf{def}(\mathsf{appendterm}(s, t)) \tag{8}$$
$$\mathsf{def}(s, t) \Rightarrow \mathsf{def}(\mathsf{first\_is\_func}(s, t)) \tag{9}$$
$$\mathsf{def}(t) \Rightarrow \mathsf{def}(\mathsf{vars}(t)) \tag{10}$$
$$\mathsf{def}(t, n) \Rightarrow \mathsf{def}(\mathsf{rename}(t, n)) \tag{11}$$
$$\mathsf{def}(t, k) \Rightarrow \mathsf{def}(\mathsf{remove}(t, k)) \tag{12}$$
$$\mathsf{def}(k_1, k_2) \Rightarrow \mathsf{def}(\mathsf{setdiff}(k_1, k_2)) \tag{13}$$
$$\mathsf{def}(R) \Rightarrow \mathsf{def}(\mathsf{trs}(R)) \tag{14}$$

## 4.2 Definedness of first, tail, second, ttail, addterm, func_name, func_args

The following theorems state that first and tail are defined if their argument is not e.

$$\mathsf{def}(t) \land \neg t = \mathsf{e} \Rightarrow \mathsf{def}(\mathsf{first}(t)) \tag{15}$$
$$\mathsf{def}(t) \land \neg t = \mathsf{e} \Rightarrow \mathsf{def}(\mathsf{tail}(t)) \tag{16}$$

These theorems can easily be proved by structural induction (where the induction hypothesis is not used). In a similar way one can also prove the following conjectures.

$$\mathsf{def}(t) \land \neg t = \mathsf{e} \land \neg \mathsf{tail}(t) = \mathsf{e} \Rightarrow \mathsf{def}(\mathsf{second}(t)) \tag{17}$$
$$\mathsf{def}(t) \land \neg t = \mathsf{e} \land \neg \mathsf{tail}(t) = \mathsf{e} \Rightarrow \mathsf{def}(\mathsf{ttail}(t)) \tag{18}$$
$$\mathsf{def}(s) \land \mathsf{length}(t) = \mathsf{s}(0) \Rightarrow \mathsf{def}(\mathsf{addterm}(t, s)) \tag{19}$$
$$\mathsf{first\_is\_func}(t) \Rightarrow \mathsf{def}(\mathsf{func\_name}(t)) \tag{20}$$
$$\mathsf{first\_is\_func}(t) \Rightarrow \mathsf{def}(\mathsf{func\_args}(t)) \tag{21}$$

### 4.3 Totality of in

The function in has partial auxiliary functions, but it itself is total.

$$\mathsf{def}(s,t,r) \Rightarrow \mathsf{def}(\mathsf{in}(s,t,r)). \tag{22}$$

This can easily be proved by two nested structural inductions.
In a similar way one can prove

$$\mathsf{def}(t,r) \wedge \neg t = \mathsf{e} \Rightarrow \mathsf{def}(\mathsf{membereven}(t,r)). \tag{23}$$

### 4.4 Totality of occurs and subseteq

To prove the totality of occurs

$$\mathsf{def}(n,t) \Rightarrow \mathsf{def}(\mathsf{occurs}(n,t)) \tag{24}$$

we cannot directly use structural induction. Instead, we generate the corresponding domain predicate $\theta_{\mathsf{occurs}}$. To compare the input arguments with the arguments in the recursive call we use the ordering $\succ$, where a pair of data objects $\langle n,t\rangle$ is $\succ$-greater than another one $\langle n',t'\rangle$ iff the number of var- and func-occurrences in $t$ is greater than in $t'$. Then the difference equivalents $\Delta_{\succ}(\langle n,\mathsf{var}(m,r)\rangle,\langle n,r\rangle)$ and $\Delta_{\succ}(\langle n,\mathsf{func}(m,s,r)\rangle,\langle n,\mathsf{appendterm}(s,r)\rangle)$ are equivalent to true. (This ordering $\succ$ and the difference equivalents can easily be generated automatically using the techniques in [3, 7].) Hence, we obtain the following domain predicate

*function* $\theta_{\mathsf{occurs}} : \mathsf{nat} \times \mathsf{term} \to \mathsf{bool}$
$\theta_{\mathsf{occurs}}(n,\mathsf{e}) = \mathsf{true}$
$\theta_{\mathsf{occurs}}(n,\mathsf{var}(m,r)) = \mathsf{if}(\mathsf{eq}(n,m),\mathsf{true},\theta_{\mathsf{occurs}}(n,r))$
$\theta_{\mathsf{occurs}}(n,\mathsf{func}(m,s,r)) = \theta_{\mathsf{occurs}}(n,\mathsf{appendterm}(s,r))$.

Now $\theta_{\mathsf{occurs}}(n,t) = \mathsf{true}$ can easily be proved by induction w.r.t. $\theta_{\mathsf{occurs}}$. (In fact, the simplification techniques of [3] can directly simplify the above algorithm to true resp. the method of [7] can directly prove termination of the total algorithm occurs.) In a similar way one can also prove

$$\mathsf{def}(s,t) \Rightarrow \mathsf{def}(\mathsf{subseteq}(s,t)). \tag{25}$$

### 4.5 Totality of append, member, subseteq_list, disjoint_list, is_empty, hasevenlength, apply, onlyconsistsof, applytwice

The following conjectures can again be proved by an easy structural induction.

$$\mathsf{def}(l_1,l_2) \Rightarrow \mathsf{def}(\mathsf{append}(l_1,l_2)) \tag{26}$$
$$\mathsf{def}(t,l) \Rightarrow \mathsf{def}(\mathsf{member}(t,l)) \tag{27}$$
$$\mathsf{def}(l_1,l_2) \Rightarrow \mathsf{def}(\mathsf{subseteq\_list}(l_1,l_2)) \tag{28}$$
$$\mathsf{def}(l_1,l_2) \Rightarrow \mathsf{def}(\mathsf{disjoint\_list}(l_1,l_2)) \tag{29}$$
$$\mathsf{def}(l) \Rightarrow \mathsf{def}(\mathsf{is\_empty}(l)) \tag{30}$$
$$\mathsf{def}(l) \Rightarrow \mathsf{def}(\mathsf{hasevenlength}(l)) \tag{31}$$
$$\mathsf{def}(n,l) \Rightarrow \mathsf{def}(\mathsf{apply}(n,l)) \tag{32}$$
$$\mathsf{def}(t,l) \Rightarrow \mathsf{def}(\mathsf{append\_list}(t,l)) \tag{33}$$
$$\mathsf{def}(l,t) \Rightarrow \mathsf{def}(\mathsf{onlyconsistsof}(l,t)) \tag{34}$$
$$\mathsf{def}(n,l_1,l_2) \Rightarrow \mathsf{def}(\mathsf{applytwice}(n,l_1,l_2)) \tag{35}$$

### 4.6 Transitivity of ge (pc)

The next conjecture states that ge is transitive.

$$\mathsf{ge}(x,y) \wedge \mathsf{ge}(y,z) \Rightarrow \mathsf{ge}(x,z) \tag{36}$$

This example is used in [2] to demonstrate the need for merging induction relations. As described in [9] we can model this technique by using appropriate instantiations of non-induction variables in the induction hypotheses. Hence, the conjecture can easily be proved using (the extension of) Rule 1″. For that purpose we use an induction w.r.t. $\mathsf{ge}(x,z)$ and instantiate $y$ with $\mathsf{p}(y)$ in the induction hypothesis. Then after symbolic evaluation the induction conclusion is

$$\mathsf{ge}(\mathsf{s}(x),y) \wedge \mathsf{ge}(y,\mathsf{s}(z)) \Rightarrow \mathsf{ge}(x,z)$$

and the induction hypothesis is

$$\mathsf{ge}(x,\mathsf{p}(y)) \wedge \mathsf{ge}(\mathsf{p}(y),z) \Rightarrow \mathsf{ge}(x,z).$$

Now the induction step formula is proved by induction (resp. case analysis) w.r.t. p. In both cases, symbolic evaluation results in a tautology. In a similar way one can also prove

$$\mathsf{gt}(x,y) \wedge \mathsf{ge}(y,z) \quad \Rightarrow \quad \mathsf{gt}(x,z), \tag{37}$$

$$\mathsf{ge}(x,y) \wedge \mathsf{gt}(y,z) \quad \Rightarrow \quad \mathsf{gt}(x,z). \tag{38}$$

### 4.7 Reflexivity of ge (pc)

The following conjecture is easily proved by structural induction on $x$.

$$\mathsf{ge}(x,x) \tag{39}$$

In a similar way one can also prove

$$\neg\mathsf{gt}(x,x), \tag{40}$$

$$\mathsf{gt}(\mathsf{s}(x),x) \tag{41}$$

### 4.8 ge is a Total Relation (pc)

The next conjecture states that for every pair $x, y$ of numbers we have $\mathsf{ge}(x,y)$ or $\mathsf{ge}(y,x)$.

$$\neg\mathsf{ge}(x,y) \Rightarrow \mathsf{ge}(y,x) \tag{42}$$

It can be proved by induction w.r.t. ge. In a similar way one can also prove

$$\mathsf{ge}(x,\mathsf{s}(y)) \Rightarrow \neg\mathsf{ge}(x,y), \tag{43}$$

$$\mathsf{gt}(x,y) \Rightarrow \mathsf{ge}(x,y) \tag{44}$$

### 4.9 Associativity of plus (pc)

The associativity of plus can be proved by a straightforward induction w.r.t. plus using $x, y$ as induction variables.

$$\mathsf{plus}(x,\mathsf{plus}(y,z)) = \mathsf{plus}(\mathsf{plus}(x,y),z) \tag{45}$$

### 4.10 Commutativity of plus (pc)

The following theorem states that plus is commutative.

$$\mathsf{plus}(n, m) = \mathsf{plus}(m, n) \tag{46}$$

By induction w.r.t. plus (using the induction variables $n, m$), the conjecture is transformed into

$$m = \mathsf{plus}(m, 0)$$

(which can be proved by structural induction on $m$), (5), and

$$\mathsf{s}(\mathsf{plus}(m, x)) = \mathsf{plus}(m, \mathsf{s}(x))$$

(which can be proved by induction w.r.t. plus using $m, x$ as induction variables).

### 4.11 plus is Injective For Fixed Second Argument (pc)

The next conjecture says that if one argument of plus is fixed, then plus is injective.

$$\mathsf{plus}(m, n) = \mathsf{plus}(k, n) \Rightarrow m = k \tag{47}$$

Using (46), the conjecture can be transformed into

$$\mathsf{plus}(n, m) = \mathsf{plus}(n, k) \Rightarrow m = k$$

which can be proved by induction w.r.t. plus.

### 4.12 Additions are Greater Than or Equal To Arguments (pc)

The next conjecture states that $\mathsf{plus}(x, y)$ is greater than or equal to $y$.

$$\mathsf{ge}(\mathsf{plus}(x, y), y) \tag{48}$$

It can be proved by induction w.r.t. plus. In the base case, we need lemma (39) and in the step case, the induction formula is a consequence of (41), (37), and (44). Using (48) and (46), we can also prove

$$\mathsf{ge}(\mathsf{plus}(x, y), x). \tag{49}$$

Moreover, using these lemmata one can prove

$$\neg x = \mathsf{0} \Rightarrow \mathsf{gt}(\mathsf{plus}(x, y), y), \tag{50}$$

$$\neg y = \mathsf{0} \Rightarrow \mathsf{gt}(\mathsf{plus}(x, y), x) \tag{51}$$

by induction resp. case analysis w.r.t. plus.

### 4.13 $\dot\wedge$ is Conjunction

The next conjecture states that $\dot\wedge$ returns true iff both its arguments are true.

$$t \mathbin{\dot\wedge} s \Leftrightarrow t \wedge s \tag{52}$$

Without using abbreviations this theorem reads

$$\mathsf{if}(t, s, \mathsf{false}) = \mathsf{true} \Leftrightarrow t = \mathsf{true} \wedge s = \mathsf{true}.$$

Rule 4$''$ transforms this conjecture into the two lemmata stating both directions of the conjecture. These lemmata can easily be proved using Rule 6$''$. In a similar way one can also prove

$$t \mathbin{\dot\vee} s \Leftrightarrow \mathsf{def}(t) \wedge (t \vee s). \tag{53}$$

Note however, that $t \vee s \Rightarrow t \mathbin{\dot\vee} s$ is only partially true, but not true.

## 4.14 $\mathsf{eqterm}$ Computes Equality (pc)

This theorem states that for defined terms, $\mathsf{eq}$ computes the equality.

$$\mathsf{eq}(n, m) \Leftrightarrow n = m. \tag{54}$$

It can be proved by induction w.r.t. $\mathsf{eq}$. In a similar way one can also prove partial truth of

$$\mathsf{eqterm}(s, t) \Leftrightarrow s = t \tag{55}$$

using (54) and (52).

## 4.15 $\mathsf{first}$ is Idempotent (pc)

The next conjecture states that $\mathsf{first}$ is idempotent.

$$\mathsf{first}(\mathsf{first}(t)) = \mathsf{first}(t) \tag{56}$$

It can be proved by induction resp. case analysis w.r.t. $\mathsf{first}$. In this way one can also prove

$$\mathsf{def}(\mathsf{first}(t)) = \mathsf{true} \Rightarrow \mathsf{def}(\mathsf{first}(\mathsf{first}(t))) = \mathsf{true}. \tag{57}$$

## 4.16 Tail of First Element is Empty (pc)

This conjecture states that the $\mathsf{tail}$ of $\mathsf{first}(t)$ is empty.

$$\mathsf{tail}(\mathsf{first}(t)) = \mathsf{e} \tag{58}$$

It can easily be proved by induction (resp. case analysis) w.r.t. $\mathsf{first}$. In a similar way one can also prove

$$\mathsf{eqterm}(\mathsf{first}(t), \mathsf{e}) = \mathsf{false}, \tag{59}$$
$$\mathsf{length}(\mathsf{first}(s)) = \mathsf{s}(0). \tag{60}$$

## 4.17 Correctness of $\mathsf{addterm}$, $\mathsf{tail}$, and $\mathsf{first}$ (pc)

This theorem states that $\mathsf{addterm}$, $\mathsf{tail}$, and $\mathsf{first}$ are correct.

$$\mathsf{addterm}(\mathsf{first}(t), \mathsf{tail}(t)) = t \tag{61}$$

It can be directly proved by "induction" w.r.t. $\mathsf{first}$.

## 4.18 Definedness of $\mathsf{addterm}$ and Length

This conjecture states that $\mathsf{addterm}$ is only defined if its first argument has length 1.

$$\mathsf{def}(\mathsf{addterm}(s, t)) \Rightarrow \mathsf{length}(s) = \mathsf{s}(0) \tag{62}$$

It can be proved by induction (resp. case analysis) w.r.t. $\mathsf{addterm}$.

## 4.19 $\mathsf{first}$ and $\mathsf{tail}$ for Length 1 (pc)

The following theorem describes the behaviour of $\mathsf{first}$ and $\mathsf{tail}$ for terms of length 1.

$$\mathsf{length}(t) = \mathsf{s}(0) \Rightarrow \mathsf{first}(t) = t \wedge \mathsf{tail}(t) = \mathsf{e} \tag{63}$$

It can be proved by induction (resp. case analysis) w.r.t. $\mathsf{length}$ using the lemma $\mathsf{length}(r) = 0 \Rightarrow r = \mathsf{e}$ (which is also provable by induction w.r.t. $\mathsf{length}$).

## 4.20 Properties of Added Terms (pc)

The following conjectures are immediately proved by induction (resp. case analysis) w.r.t. addterm. In particular, (66) states that appendterm and addterm are the same (provided that addterm is defined) and (67) says that to prove the equality of two terms built with addterm one can look at the arguments.

$$\mathsf{tail}(\mathsf{addterm}(s,t)) = t \tag{64}$$
$$\mathsf{eqterm}(\mathsf{addterm}(s,t),\mathsf{e}) = \mathsf{false} \tag{65}$$
$$\mathsf{addterm}(t,s) = \mathsf{appendterm}(t,s) \tag{66}$$
$$\mathsf{addterm}(s,t) = \mathsf{addterm}(u,v) \Leftrightarrow s = u \wedge t = v \tag{67}$$

## 4.21 Correctness of func_args (pc)

This conjecture states that func_args is correct.

$$\mathsf{first}(t) = \mathsf{func}(n, \mathsf{func\_args}(t), \mathsf{e}). \tag{68}$$

It can be directly proved by "induction" w.r.t. func_args.

## 4.22 Terms in a Termlist Have Length 1 (pc)

The next theorem states that the truth of in implies that the first two arguments have length 1 (this will be used later to prove that left- and right-hand sides of rules have length 1).

$$\mathsf{in}(s,t,r) \Rightarrow \mathsf{length}(s) = \mathsf{s}(0) \wedge \mathsf{length}(t) = \mathsf{s}(0) \tag{69}$$

It is a consequence of (55) and (60).

## 4.23 Connection Between in and membereven (pc)

The following theorem states the (obvious) connection between in and membereven.

$$\mathsf{membereven}(\mathsf{addterm}(s,t),r) = \mathsf{in}(s,t,r) \tag{70}$$

It can be proved by induction w.r.t. in using $\mathsf{first}(\mathsf{addterm}(s,t)) = s$ and $\mathsf{tail}(\mathsf{addterm}(s,t)) = t$ (which can be proved by induction resp. case analysis w.r.t addterm).

## 4.24 Associativity of appendterm (pc)

This is the associativity theorem for appendterm.

$$\mathsf{appendterm}(\mathsf{appendterm}(r,s),t) = \mathsf{appendterm}(r,\mathsf{appendterm}(s,t)) \tag{71}$$

It can be proved by a straightforward induction w.r.t. appendterm and in a similar way one can also prove associativity of append.

$$\mathsf{append}(k_1,\mathsf{append}(k_2,k_3)) = \mathsf{append}(\mathsf{append}(k_1,k_2),k_3) \tag{72}$$

## 4.25 Appending Empty Lists (pc)

The following conjectures say that appending empty lists resp. tll's does not change the result. Both theorems can easily be proved by structural induction.

$$\mathsf{appendterm}(t,\mathsf{e}) = t \tag{73}$$
$$\mathsf{append}(k,\mathsf{empty}) = k \tag{74}$$

## 4.26 First and Second Element of Appended Lists (pc)

The following conjectures says that if $\mathsf{first}(s)$ resp. $\mathsf{second}(s)$ is defined, then to compute the first resp. the second element of $\mathsf{appendterm}(s,t)$ one only has to look at $s$.

$$\mathsf{first}(\mathsf{appendterm}(s,t)) = \mathsf{first}(s) \tag{75}$$
$$\mathsf{second}(\mathsf{appendterm}(s,t)) = \mathsf{second}(s) \tag{76}$$

Conjecture (75) is proved by induction (resp. case analysis) w.r.t. $\mathsf{first}$. In conjecture (76) we perform a symbolic evaluation (evaluating $\mathsf{second}$) and then it can be proved by induction w.r.t. $\mathsf{tail}$ using (75).

## 4.27 Length of Appended Lists (pc)

The next theorem says that the length of two appended terms is greater than or equal to the length of the first component term.

$$\mathsf{ge}(\mathsf{length}(\mathsf{appendterm}(s,t)), \mathsf{length}(s)) \tag{77}$$

It can be proved by a straightforward induction w.r.t. $\mathsf{appendterm}$.

## 4.28 Decomposing Appended Lists With Equal Length (pc)

The next theorem states that if $\mathsf{appendterm}(u_1,r_1)$ and $\mathsf{appendterm}(u_2,r_2)$ are equal and if $u_1$ and $u_2$ have equal length, then the corresponding component lists are equal, too.

$$\mathsf{length}(u_1) = \mathsf{length}(u_2) \wedge \mathsf{appendterm}(u_1,r_1) = \mathsf{appendterm}(u_2,r_2) \Rightarrow u_1 = u_2 \wedge r_1 = r_2 \tag{78}$$

The conjecture is proved by a straightforward induction w.r.t. $\mathsf{eqterm}(u_1,u_2)$. (This can be done although $\mathsf{eqterm}$ does not occur in the conjecture, because $\mathsf{def}(\mathsf{eqterm}(u_1,u_2))$ is always $\mathsf{true}$ for defined arguments, cf. (6).)

## 4.29 Empty Number of Symbols (pc)

The next conjecture states that if the number of symbols in a term is 0, then it is the empty term. It can be immediately proved by induction (resp. case analysis) w.r.t. $\mathsf{symbols}$.

$$\mathsf{symbols}(t) = 0 \Rightarrow t = \mathsf{e} \tag{79}$$

## 4.30 Number of Symbols in Appended Lists (pc)

This conjecture states that the number of symbols in an appended list is the sum of the number of symbols in both component lists.

$$\mathsf{symbols}(\mathsf{appendterm}(s,t)) = \mathsf{plus}(\mathsf{symbols}(s), \mathsf{symbols}(t)) \tag{80}$$

The conjecture can be proved by induction w.r.t. $\mathsf{appendterm}$ using (45).

## 4.31 Distributivity of vars over appendterm (pc)

The next conjecture states that the variables in an appended termlist are the ones obtained by appending the variables from both sublists.

$$\mathsf{vars}(\mathsf{appendterm}(s,t)) = \mathsf{appendterm}(\mathsf{vars}(s), \mathsf{vars}(t)) \tag{81}$$

The conjecture is proved by induction w.r.t. $\mathsf{appendterm}$. The case $s = \mathsf{e}$ is trivial and in the case $s = \mathsf{var}(n,r)$ the induction conclusion is a consequence of the induction hypothesis. In the case $s = \mathsf{func}(n,u,r)$, the induction conclusion can be transformed into the induction hypothesis and (71).

## 4.32 $\mathsf{vars}$ is Idempotent (pc)

This theorem says that applying $\mathsf{vars}$ twice is the same as applying it once.

$$\mathsf{vars}(\mathsf{vars}(t)) = \mathsf{vars}(t) \tag{82}$$

The conjecture is proved by induction w.r.t. $\mathsf{vars}$. The base case is easy and in the case $t = \mathsf{var}(n, s)$ the induction conclusion can be directly reduced to the induction hypothesis. In the case $t = \mathsf{func}(n, s, r)$, after symbolic evaluation the induction conclusion is

$$\mathsf{vars}(\mathsf{appendterm}(\mathsf{vars}(s), \mathsf{vars}(r))) = \mathsf{appendterm}(\mathsf{vars}(s), \mathsf{vars}(r)).$$

This can be transformed into an instantiation of (81) and

$$\mathsf{appendterm}(\mathsf{vars}(\mathsf{vars}(s)), \mathsf{vars}(\mathsf{vars}(r))) = \mathsf{vars}(\mathsf{appendterm}(\mathsf{vars}(s), \mathsf{vars}(r)))$$

which follows from the two induction hypotheses.

## 4.33 $\mathsf{vars}$ on Appended Variable Lists (pc)

The next conjecture states that the variables in an appended variable list consist of just this variable list.

$$\mathsf{vars}(\mathsf{appendterm}(\mathsf{vars}(s), \mathsf{vars}(t))) = \mathsf{appendterm}(\mathsf{vars}(s), \mathsf{vars}(t)) \tag{83}$$

The conjecture can be transformed into an instantiation of (81) and

$$\mathsf{vars}(\mathsf{vars}(\mathsf{appendterm}(s, t))) = \mathsf{appendterm}(\mathsf{vars}(s), \mathsf{vars}(t)).$$

This in turn can be transformed into an instantiation of (82) and into (81).

## 4.34 Subsets of Empty Lists are Empty (pc)

The next conjecture says that if $k$ is a sublist of the empty list, then $k$ is empty.

$$\mathsf{subseteq\_list}(k, \mathsf{empty}) \Rightarrow k = \mathsf{empty} \tag{84}$$

This can be proved by structural induction (resp. case analysis) on $k$ (as the induction hypothesis is not used).

## 4.35 Appending the Left Arguments of $\mathsf{subseteq}$ (pc)

This conjecture says that if both $t_1$ and $t_2$ are sublists of $s$, then so is $\mathsf{appendterm}(t_1, t_2)$.

$$\mathsf{subseteq}(t_1, s) \wedge \mathsf{subseteq}(t_2, s) \Rightarrow \mathsf{subseteq}(\mathsf{appendterm}(t_1, t_2), s) \tag{85}$$

The conjecture can be proved by induction w.r.t. $\mathsf{appendterm}$, where in the last step case one needs (71). In a similar way (using (72)) one can prove the corresponding statement for $\mathsf{tll}$'s.

$$\mathsf{subseteq\_list}(l_1, l) \wedge \mathsf{subseteq\_list}(l_2, l) \Rightarrow \mathsf{subseteq\_list}(\mathsf{append}(l_1, l_2), l) \tag{86}$$

## 4.36 Stability of $\mathsf{subseteq}$ under $\mathsf{var}$ (pc)

This theorem says that if $v$ is a sublist of $w$, then this also holds for $\mathsf{var}(n, w)$.

$$\mathsf{subseteq}(v, w) \Rightarrow \mathsf{subseteq}(v, \mathsf{var}(n, w)) \tag{87}$$

The conjecture is proved by induction w.r.t. $\mathsf{subseteq}$. The base case is trivial. If $v = \mathsf{var}(m, r)$ then the induction conclusion is transformed into the induction hypothesis and

$$\mathsf{occurs}(m, w) \Rightarrow \mathsf{occurs}(m, \mathsf{var}(n, w))$$

which can be proved by symbolic evaluation. In the case $v = \mathsf{func}(m,s,r)$, the induction conclusion is directly implied by the induction hypothesis.
In an analogous way, one can prove the corresponding theorem about $\mathsf{tll}$'s

$$\mathsf{subseteq\_list}(k_1,k_2) \Rightarrow \mathsf{subseteq\_list}(k_1,\mathsf{add}(t,k_2)) \tag{88}$$

using the conjecture

$$\mathsf{member}(s,k_2) \Rightarrow \mathsf{member}(s,\mathsf{add}(t,k_2))$$

which can be proved by symbolic evaluation.

### 4.37 Stability of $\mathsf{subseteq}$ under $\mathsf{func}$ on Arguments (pc)

This conjecture states that if $v$ is a sublist of $w$, then this also holds for $\mathsf{func}(n,w,q)$.

$$\mathsf{subseteq}(v,w) \Rightarrow \mathsf{subseteq}(v,\mathsf{func}(n,w,q)) \tag{89}$$

The conjecture is again proved by induction w.r.t. $\mathsf{subseteq}$. The base case is trivial and in the case $v = \mathsf{func}(m,s,r)$, the induction conclusion is directly implied by the induction hypothesis. If $v = \mathsf{var}(m,r)$ then the induction conclusion is transformed into the induction hypothesis and

$$\mathsf{occurs}(m,w) \Rightarrow \mathsf{occurs}(m,\mathsf{appendterm}(w,r)).$$

This can be proved by induction w.r.t. $\mathsf{appendterm}$ where in the case $w = \mathsf{func}(m',s',r')$ one needs (71).

### 4.38 Stability of $\mathsf{subseteq}$ under $\mathsf{func}$ on Tail (pc)

This conjecture states that if $v$ is a sublist of $w$, then this also holds for $\mathsf{func}(n,q,w)$.

$$\mathsf{subseteq}(v,w) \Rightarrow \mathsf{subseteq}(v,\mathsf{func}(n,q,w)) \tag{90}$$

The conjecture is proved by induction w.r.t. $\mathsf{subseteq}$ where again the base case is trivial and in the case $v = \mathsf{func}(m,s,r)$, the induction conclusion is directly implied by the induction hypothesis. If $v = \mathsf{var}(m,r)$ then the induction conclusion is transformed into the induction hypothesis and

$$\mathsf{occurs}(m,w) \Rightarrow \mathsf{occurs}(m,\mathsf{appendterm}(r,w))$$

This is a consequence of $\mathsf{occurs}(m,w) \Rightarrow \mathsf{occurs}(m,\mathsf{appendterm}(w,r))$ (which was verified during the proof of (89)) and

$$\mathsf{occurs}(m,\mathsf{appendterm}(w,r)) \Rightarrow \mathsf{occurs}(m,\mathsf{appendterm}(r,w)).$$

This conjecture is proved by induction w.r.t. $\mathsf{appendterm}$. The base case is trivial. If $w = \mathsf{var}(k,s)$ and $\mathsf{eq}(m,k)$, then by (54) one has to prove

$$\mathsf{occurs}(m,\mathsf{appendterm}(r,\mathsf{var}(m,s)))$$

which can be done by structural induction on $r$. If $w = \mathsf{var}(k,s)$ and $\neg\mathsf{eq}(m,k)$, then by structural induction on $r$ one can prove

$$\mathsf{occurs}(m,\mathsf{appendterm}(r,s)) \Rightarrow \mathsf{occurs}(m,\mathsf{appendterm}(r,\mathsf{var}(k,s))).$$

Then the induction hypothesis implies the induction conclusion.
Finally, in the last case one has to prove the lemma

$$\mathsf{occurs}(m,\mathsf{appendterm}(r,\mathsf{appendterm}(s_1,s_2))) \Rightarrow \mathsf{occurs}(m,\mathsf{appendterm}(r,\mathsf{func}(k,s_1,s_2)))$$

to apply the induction hypothesis. Again this can be done by induction on $r$.

### 4.39 Reflexivity of $\mathsf{subseteq}$ (pc)

This theorem states that $\mathsf{subseteq}$ is reflexive.

$$\mathsf{subseteq}(v, v) \tag{91}$$

We prove the conjecture by structural induction on $v$. If $v = \mathsf{e}$, then the proof is trivial. If $v = \mathsf{var}(m, v')$, then the induction conclusion follows from the induction hypothesis and (87). If $v = \mathsf{func}(m, v_1, v_2)$, then the induction hypotheses, (89), (90), and (85) apply the induction conclusion.
In a similar way one can also prove the corresponding statement for $\mathsf{subseteq\_list}$

$$\mathsf{subseteq\_list}(k, k) \tag{92}$$

using the lemma (88).

### 4.40 Stability of $\mathsf{occurs}$ under Subsets (pc)

The next conjecture states that if $n$ occurs in a list of terms, then this also holds for every superlist.

$$\mathsf{occurs}(n, v_1) \land \mathsf{subseteq}(v_1, v_2) \Rightarrow \mathsf{occurs}(n, v_2) \tag{93}$$

The proof is done by induction w.r.t. $\mathsf{subseteq}$. In a similar way one can also prove the corresponding theorem for $\mathsf{tll}$'s.

$$\mathsf{subseteq\_list}(k_1, k_2) \land \mathsf{member}(t, k_1) \Rightarrow \mathsf{member}(t, k_2) \tag{94}$$

### 4.41 Transitivity of $\mathsf{subseteq}$ (pc)

The following conjecture is the transitivity of $\mathsf{subseteq}$.

$$\mathsf{subseteq}(u, v) \land \mathsf{subseteq}(v, w) \Rightarrow \mathsf{subseteq}(u, w) \tag{95}$$

The conjecture is proved by induction w.r.t. $\mathsf{subseteq}(u, v)$ using (93).
In a similar way (using (94)) one can also prove

$$\mathsf{subseteq\_list}(l_1, l_2) \land \mathsf{subseteq\_list}(l_2, l_3) \Rightarrow \mathsf{subseteq\_list}(l_1, l_3). \tag{96}$$

Moreover, by using a case analysis, (92) and (96) also imply

$$\mathsf{subseteq\_list}(k, \mathsf{if}(b, \mathsf{add}(t, k), k)). \tag{97}$$

### 4.42 Appending the Right Arguments of $\mathsf{subseteq}$ (Version 1) (pc)

This theorem says that the variables of every termlist $v_1$ are contained in $\mathsf{appendterm}(v_1, v_2)$.

$$\mathsf{subseteq}(v_1, \mathsf{appendterm}(v_1, v_2)) \tag{98}$$

It can be proved by a straightforward induction w.r.t. $\mathsf{appendterm}$. In a similar way one can also prove

$$\mathsf{subseteq\_list}(l, \mathsf{append}(l, l')). \tag{99}$$

### 4.43 Appending the Right Arguments of $\mathsf{subseteq}$ (Version 2) (pc)

This is the converse of the above theorem relating the second argument of $\mathsf{appendterm}$.

$$\mathsf{subseteq}(v_2, \mathsf{appendterm}(v_1, v_2)) \tag{100}$$

It can be proved by induction w.r.t. $\mathsf{appendterm}$ using (91) and (87). In a similar way one can also prove

$$\mathsf{subseteq\_list}(l, \mathsf{append}(l', l)). \tag{101}$$

## 4.44 Appending Both Arguments of subseteq (pc)

The following theorems gives more facts about the relation of subseteq and appendterm.

$$\mathsf{subseteq}(t_1, t_2) \wedge \mathsf{subseteq}(s_1, s_2) \Rightarrow \mathsf{subseteq}(\mathsf{appendterm}(t_1, s_1), \mathsf{appendterm}(t_2, s_2)) \tag{102}$$

It is a consequence of (85), (98), and (100).
In an analogous way one can prove

$$\mathsf{subseteq\_list}(k_1, k_2) \wedge \mathsf{subseteq\_list}(l_1, l_2) \Rightarrow \mathsf{subseteq\_list}(\mathsf{append}(l_1, k_1), \mathsf{append}(l_2, k_2)). \tag{103}$$

## 4.45 Arguments and Tails are Subsets of Function Applications (pc)

The next conjecture says that all variables occurring in the arguments of a function and in the tail of the function also occur in the whole termlist.

$$\mathsf{subseteq}(\mathsf{appendterm}(s, r), \mathsf{func}(n, s, r)) \tag{104}$$

This is an easy consequence of $\mathsf{subseteq}(\mathsf{func}(n, s, r), \mathsf{func}(n, s, r)) = \mathsf{subseteq}(\mathsf{appendterm}(s, r), \mathsf{func}(n, s, r))$ (which can be proved by symbolic evaluation) and of (91).

## 4.46 Variables in Arguments also Occur in the Termlist (pc)

The following theorem is a consequence of (104), (98), and (95).

$$\mathsf{subseteq}(s, \mathsf{func}(n, s, r)) \tag{105}$$

## 4.47 Variables in Heads also Occur in the Termlist (pc)

The next conjecture is a consequence of (98), (66), (61) and (91).

$$\mathsf{subseteq}(\mathsf{first}(t), t) \tag{106}$$

In a similar way one can also prove

$$\mathsf{subseteq}(\mathsf{tail}(t), t). \tag{107}$$

## 4.48 Removing the Head of a Superlist (pc)

This conjecture says that if $v_1$ is a sublist of $\mathsf{var}(n, v_2)$, then $v_1$ is also a sublist of $v_2$, provided that whenever $n$ occurs in $v_1$ then $n$ also occurs in $v_2$.

$$\mathsf{subseteq}(v_1, \mathsf{var}(n, v_2)) \wedge (\mathsf{occurs}(n, v_1) \Rightarrow \mathsf{occurs}(n, v_2)) \Rightarrow \mathsf{subseteq}(v_1, v_2) \tag{108}$$

The conjecture is a consequence of

$$\neg\mathsf{occurs}(n, v_1) \wedge \mathsf{subseteq}(v_1, \mathsf{var}(n, v_2)) \Rightarrow \mathsf{subseteq}(v_1, v_2),$$

(which can be proved by induction w.r.t. subseteq) and

$$\mathsf{occurs}(n, v_2) \Rightarrow \mathsf{subseteq}(\mathsf{var}(n, v_2), v_2),$$

(which is a consequence of (91), and (95)).

## 4.49 Lists are Subsets of Disjoint Unions (Version 1) (pc)

The next theorem says that every list of variables $v_1$ is a subset of the disjoint union of $v_1$ and $v_2$.

$$\mathsf{subseteq}(v_1, \mathsf{disjoint\_union}(v_1, v_2)) \tag{109}$$

Using (91) and (25), we transform the conjecture into

$$\mathsf{subseteq}(v_1, w) \Rightarrow \mathsf{subseteq}(v_1, \mathsf{disjoint\_union}(w, v_2)).$$

This conjecture is now proved by induction w.r.t. disjoint_union. The base case $v_2 = \mathsf{e}$ is easy. In the step case we have $v_2 = \mathsf{var}(n, q)$. If $\mathsf{occurs}(n, w)$, then the induction conclusion can be reduced to the induction hypothesis. If $\neg\mathsf{occurs}(n, w)$, then the induction conclusion is

$$\mathsf{subseteq}(v_1, w) \Rightarrow \mathsf{subseteq}(v_1, \mathsf{disjoint\_union}(\mathsf{appendterm}(w, \mathsf{var}(n, \mathsf{e})), q))$$

and the induction hypothesis is

$$\mathsf{subseteq}(v_1, \mathsf{appendterm}(w, \mathsf{var}(n, \mathsf{e}))) \Rightarrow \mathsf{subseteq}(v_1, \mathsf{disjoint\_union}(\mathsf{appendterm}(w, \mathsf{var}(n, \mathsf{e})), q)).$$

Hence, the theorem is proved by Rule 4$''$ using (100) and (95).

## 4.50 Lists are Subsets of Disjoint Unions (Version 2) (pc)

The following theorem is similar to the one above, but works with the second argument of disjoint_union.

$$\mathsf{subseteq}(v_2, \mathsf{disjoint\_union}(v_1, v_2)) \tag{110}$$

The conjecture can be proved using Rule 1$''$ by induction w.r.t. disjoint_union. In the case $v_2 = \mathsf{e}$ it is trivial. If $v_2 = \mathsf{var}(n, v)$ and $\mathsf{occurs}(n, v_1)$, then the induction conclusion can be transformed into the induction hypothesis. Otherwise (if $\neg\mathsf{occurs}(n, v_1)$) the induction conclusion is

$$\mathsf{subseteq}(\mathsf{var}(n, v), \mathsf{disjoint\_union}(\mathsf{appendterm}(v_1, \mathsf{var}(n, \mathsf{e})), v_2))$$

which can be transformed into the induction hypothesis and into

$$\mathsf{occurs}(n, \mathsf{disjoint\_union}(\mathsf{appendterm}(v_1, \mathsf{var}(n, \mathsf{e})), v_2)).$$

This in turn can be transformed into

$$\mathsf{subseteq}(\mathsf{var}(n, \mathsf{e}), \mathsf{disjoint\_union}(\mathsf{appendterm}(v_1, \mathsf{var}(n, \mathsf{e})), v_2)).$$

This can be proved by (100) and (109).

## 4.51 Occurrence of Variables in Unions of Lists (pc)

The following theorem states that if $n$ occurs in $v_1$ or $v_2$, then it also occurs in the disjoint union of of $v_1$ and $v_2$.

$$\mathsf{occurs}(n, v_1) \vee \mathsf{occurs}(n, v_2) \Rightarrow \mathsf{occurs}(n, \mathsf{disjoint\_union}(v_1, v_2)). \tag{111}$$

The theorem is a consequence of (109), (110), and (93).

## 4.52 Occurrence of Variables in Appended Termlists (pc)

This conjecture states that if $n$ is a member of an appended termlist, then it is a member of one of the argument lists.

$$\mathsf{occurs}(n, \mathsf{appendterm}(s, t)) \Rightarrow \mathsf{occurs}(n, s) \vee \mathsf{occurs}(n, t) \tag{112}$$

The conjecture is proved by induction w.r.t. appendterm. In a similar way one can also prove

$$\mathsf{member}(t, \mathsf{append}(k_1, k_2)) \Rightarrow \mathsf{member}(t, k_1) \vee \mathsf{member}(t, k_2). \tag{113}$$

## 4.53 Commutation of appendterm (pc)

This conjecture states that if one commutes the arguments of appendterm then the resulting termlist has the same arguments.

$$\mathsf{subseteq}(\mathsf{appendterm}(s,t),\mathsf{appendterm}(t,s)) \tag{114}$$

The conjecture is a consequence of (85), (98), and (100). In a similar way one can also prove the corresponding theorem for tll's

$$\mathsf{subseteq\_list}(\mathsf{append}(k_1,k_2),\mathsf{append}(k_2,k_1)). \tag{115}$$

## 4.54 Application of disjoint to Empty Termlist (pc)

This conjecture says that every term is disjoint with the empty termlist.

$$\mathsf{disjoint}(t,\mathsf{e}) \tag{116}$$

We transform the conjecture into

$$s = \mathsf{e} \Rightarrow \mathsf{disjoint}(t,s)$$

and prove it by induction w.r.t. disjoint.

## 4.55 Application of disjoint_list to Empty tll (pc)

This is the corresponding conjecture for disjoint_list.

$$\mathsf{disjoint\_list}(k,\mathsf{empty}). \tag{117}$$

It can easily be proved by structural induction on $k$.

## 4.56 Lists with Equal Elements are Not Disjoint (pc)

The next theorem states that if $k_1$ and $k_2$ have a common element, then they are not disjoint.

$$\mathsf{member}(s,k_1) \wedge \mathsf{member}(s,k_2) \Rightarrow \mathsf{disjoint\_list}(k_1,k_2) = \mathsf{false} \tag{118}$$

The conjecture is proved by induction w.r.t. disjoint_list. In the case $k_1 = \mathsf{add}(t,l)$ and $\mathsf{member}(t,k_2) = \mathsf{false}$ one needs (55) in order to prove that the induction hypothesis entails the induction conclusion.

## 4.57 Disjointness of Appended Lists (pc)

This theorem says that if both $s$ and $t$ are disjoint with $r$, then so is the appended list of $s$ and $t$.

$$\mathsf{disjoint}(s,r) \wedge \mathsf{disjoint}(t,r) \Rightarrow \mathsf{disjoint}(\mathsf{append}(s,t),r) \tag{119}$$

It can be proved by a straightforward induction w.r.t. appendterm. In a similar way one can prove

$$\mathsf{disjoint\_list}(k_1,k) \wedge \mathsf{disjoint\_list}(k_2,k) \Rightarrow \mathsf{disjoint\_list}(\mathsf{append}(k_1,k_2),k). \tag{120}$$

## 4.58 Stability of disjoint under Subsets (pc)

The following theorem states that subsets of disjoint variable lists are also disjoint.

$$\mathsf{subseteq}(v_1,v_2) \wedge \mathsf{subseteq}(v_3,v_4) \wedge \mathsf{disjoint}(v_2,v_4) \Rightarrow \mathsf{disjoint}(v_1,v_3) \tag{121}$$

The conjecture is proved by induction w.r.t. $\mathsf{disjoint}(v_1,v_3)$. The base case is trivial. If $v_1 = \mathsf{var}(n,r)$ then we have $\neg\mathsf{occurs}(n,v_3)$ (due to (93)) and to

$$\mathsf{occurs}(n,v_2) \wedge \mathsf{disjoint}(v_2,v_4) \Rightarrow \neg\mathsf{occurs}(n,v_4)$$

which can be proved by induction w.r.t. $\mathsf{disjoint}(v_2, v_4)$). Hence, the induction conclusion follows from the induction hypothesis.
In a similar way one can prove the corresponding theorem for $\mathsf{tll}$'s.

$$\mathsf{subseteq\_list}(k_1, k_2) \wedge \mathsf{subseteq\_list}(k_3, k_4) \wedge \mathsf{disjoint\_list}(k_2, k_4) \Rightarrow \mathsf{disjoint\_list}(k_1, k_3) \tag{122}$$

using (94).

## 4.59 Commutativity of disjoint (pc)

This is the commutativity theorem for $\mathsf{disjoint}$.

$$\mathsf{disjoint}(s, t) = \mathsf{disjoint}(t, s) \tag{123}$$

We prove the theorem by induction w.r.t. $\mathsf{disjoint}(s, t)$. If $s = \mathsf{e}$, then the theorem follows from (116). If $s = \mathsf{var}(n, r)$ and $\mathsf{occurs}(n, t)$, then we have to prove

$$\mathsf{occurs}(n, t) \Rightarrow \neg\mathsf{disjoint}(t, \mathsf{var}(n, r)).$$

This lemma can be proved by induction w.r.t. $\mathsf{occurs}$. The base case is trivial and in the case $t = \mathsf{var}(m, s)$ and $\mathsf{eq}(n, m)$ the conjecture follows from (54). If $\mathsf{eq}(n, m) = \mathsf{false}$, then the induction conclusion follows from the induction hypothesis, (121), (91), and (100). If $t = \mathsf{func}(m, s_1, s_2)$, then the induction conclusion is a consequence of the induction hypothesis and

$$\mathsf{disjoint}(\mathsf{func}(m, s_1, s_2), r) \Rightarrow \mathsf{disjoint}(\mathsf{appendterm}(s_1, s_2), r).$$

This is a consequence of (121) and (104).

Finally, we consider the case $s = \mathsf{var}(n, r)$ and $\mathsf{occurs}(n, t) = \mathsf{false}$. Now the induction conclusion is implied by the induction hypothesis, (121), (91), and (100).

## 4.60 Commutativity of disjoint_list (pc)

This is the corresponding theorem for $\mathsf{disjoint\_list}$.

$$\mathsf{disjoint\_list}(l, k) = \mathsf{disjoint\_list}(k, l) \tag{124}$$

The proof is similar to the proof of (123). We use an induction w.r.t. $\mathsf{disjoint}(l, k)$. If $l = \mathsf{e}$, then the theorem follows from (117). If $l = \mathsf{add}(t, k')$ and $\mathsf{member}(t, l)$, then we have to prove

$$\mathsf{member}(t, l) \Rightarrow \neg\mathsf{disjoint\_list}(k_2, \mathsf{add}(y, k')).$$

This lemma can be proved by induction w.r.t. $\mathsf{member}$. The base case is trivial and in the case $l = \mathsf{add}(r, l')$ and $\mathsf{eqterm}(t, r)$ the conjecture follows from (55). If $\mathsf{eqterm}(t, r) = \mathsf{false}$, then the induction conclusion follows from the induction hypothesis, (122), (92), and (101).

Finally, we consider the case $l = \mathsf{add}(t, k')$ and $\mathsf{member}(t, l) = \mathsf{false}$. Now the induction conclusion is implied by the induction hypothesis, (122), (92), and (101).

## 4.61 Reflexivity of disjoint_list (pc)

This theorem proves that $\mathsf{disjoint\_list}$ is reflexive.

$$\neg k = \mathsf{empty} \Rightarrow \mathsf{disjoint\_list}(k, k) = \mathsf{false} \tag{125}$$

This can easily be proved by structural induction (resp. case analysis) on $k$ (as the induction hypothesis is not used).

## 4.62 Maximal Variable is Greater than or Equal to the Head (pc)

This conjecture states that the maximal variable of a variable list is at least as great as the first element.

$$\mathsf{ge}(\mathsf{max}(\mathsf{var}(n,t)),n) \tag{126}$$

We transform the conjecture into

$$s = \mathsf{e} \vee \mathsf{ge}(\mathsf{max}(s), \mathsf{var\_name}(s))$$

and prove it by induction w.r.t. max using (39), (42), and (36).

## 4.63 Variables that do not Occur in Termlists (pc)

This conjecture states that a variable with higher index than all variables of $r$ does not occur in $r$.

$$\neg\mathsf{occurs}(\mathsf{plus}(m,\mathsf{s}(\mathsf{max}(v))),v) \tag{127}$$

We prove the conjecture by induction w.r.t. max. If $v = \mathsf{e}$, then the proof is trivial. If $v = \mathsf{var}(n,\mathsf{e})$, then by (54) it suffices to prove

$$\neg\mathsf{plus}(m,\mathsf{s}(n)) = n.$$

This is a consequence of (48), (41), (38), and (40).

If $v = \mathsf{var}(n,\mathsf{var}(k,t))$ and $\mathsf{ge}(n,k)$, then the induction conclusion follows from the induction hypothesis, (54) and

$$\neg\mathsf{plus}(m,\mathsf{s}(\mathsf{max}(n,t))) = k$$

(which is due to (48), (41), (38), (37), and (126)). The other step case can be proved in an analogous way.

## 4.64 Exchanging tail and rename (pc)

The following conjecture says that one may exchange tail and rename.

$$\mathsf{tail}(\mathsf{rename}(t,n)) = \mathsf{rename}(\mathsf{tail}(t),n) \tag{128}$$

It can easily be proved by induction (resp. case analysis) w.r.t. tail.

## 4.65 Distributivity of rename over appendterm (pc)

This conjecture states that instead of renaming an appended list one may rename the arguments.

$$\mathsf{rename}(\mathsf{appendterm}(s,t),n) = \mathsf{appendterm}(\mathsf{rename}(s,n),\mathsf{rename}(t,n)) \tag{129}$$

It can be proved by a straightforward induction w.r.t. appendterm.

## 4.66 Length of Renamed Termlists (pc)

This conjecture states that the length does not change by renaming.

$$\mathsf{length}(t) = \mathsf{length}(\mathsf{rename}(t,n)) \tag{130}$$

It is easily proved by induction w.r.t. rename.

### 4.67 Stability of subseteq under rename (pc)

This conjecture says that if all variables of $r$ are contained in $l$, then this is also true after renaming.

$$\mathsf{subseteq}(\mathsf{vars}(r), \mathsf{vars}(l)) \Rightarrow \mathsf{subseteq}(\mathsf{vars}(\mathsf{rename}(r, n)), \mathsf{vars}(\mathsf{rename}(l, n))) \tag{131}$$

The conjecture is proved by induction w.r.t. $\mathsf{rename}(r, n)$. The base case is trivial. If $r = \mathsf{var}(m, t)$, then the induction conclusion follows from the induction hypothesis and

$$\mathsf{occurs}(m, \mathsf{vars}(l)) \Rightarrow \mathsf{occurs}(\mathsf{plus}(m, n), \mathsf{vars}(\mathsf{rename}(l, n))).$$

This can be proved by induction w.r.t. rename using (93) and (112).

In the last case, the induction conclusion is implied by the induction hypothesis, (98), (100), (95), and (85).

### 4.68 Renamed Termlists Have Disjoint Variables (pc)

This conjecture states that if all variables in a termlist are renamed appropriately, then the new termlist and the original termlist are variable disjoint.

$$\mathsf{disjoint}(v, \mathsf{vars}(\mathsf{rename}(t, \mathsf{s}(\mathsf{max}(v))))) \tag{132}$$

We first commute the arguments of disjoint (using (123)). The conjecture is proved by structural induction on $t$. The case $t = \mathsf{e}$ is easy. If $t = \mathsf{var}(m, q)$, then the induction conclusion follows from the induction hypothesis and (127). In the last step case, to transform the induction conclusion into the induction hypothesis one needs lemma (119).

### 4.69 Stability of subseteq_list under remove (pc)

This theorem states that if $k$ is a sublist of $l$, then this also holds if an element is removed from both lists.

$$\mathsf{subseteq\_list}(k, l) \Rightarrow \mathsf{subseteq\_list}(\mathsf{remove}(t, k), \mathsf{remove}(t, l)) \tag{133}$$

This can be proved by induction w.r.t. subseteq_list.

### 4.70 Stability of ¬subseteq_list under remove (pc)

This is the converse to the above theorem. If $k$ is no sublist of $l$, then this also holds if an element is removed from both lists, provided it occurred at least in $l$.

$$\mathsf{member}(t, l) \wedge \neg\mathsf{subseteq\_list}(k, l) \Rightarrow \neg\mathsf{subseteq\_list}(\mathsf{remove}(t, k), \mathsf{remove}(t, l)) \tag{134}$$

We prove the conjecture by induction w.r.t. subseteq_list. The case $k = \mathsf{empty}$ is trivial. If $k = \mathsf{add}(s, k')$ and $\neg\mathsf{member}(s, l)$ then this implies $\neg s = t$. Hence, the conjecture follows from (94) and

$$\mathsf{subseteq\_list}(\mathsf{remove}(t, l), l)$$

(which is proved by induction w.r.t. remove using (101)).

Finally, if $k = \mathsf{add}(s, k')$ and $\mathsf{member}(s, l)$, then the premises imply $\neg\mathsf{subseteq\_list}(k', l)$. If $s = t$, then the induction conclusion can be directly transformed into the induction hypothesis and otherwise one needs the lemma

$$\mathsf{member}(s, l) \wedge \neg s = t \Rightarrow \mathsf{member}(s, \mathsf{remove}(t, l))$$

which can be proved by induction w.r.t. member.

## 4.71 Removing Non-Contained Elements From Lists (pc)

The nest conjecture states that if $l$ does not contain $t$, then removing $t$ does not change $l$.

$$\neg\mathsf{member}(t,l) \Rightarrow \mathsf{remove}(t,l) = l \tag{135}$$

It can be proved by a straightforward induction w.r.t. member.

## 4.72 Lists with the Same Elements (Version 1) (pc)

This conjecture says that if $k_1$ contains no arguments that are not also contained in $k_2$, then $k_1$ is a subset of $k_2$.

$$\mathsf{setdiff}(k_1,k_2) = \mathsf{0} \Rightarrow \mathsf{subseteq\_list}(k_1,k_2) \tag{136}$$

It can be proved by induction w.r.t. setdiff using (134).

## 4.73 Lists with the Same Elements (Version 2) (pc)

This is the other direction of the above conjecture.

$$\mathsf{subseteq\_list}(k_1,k_2) \Rightarrow \mathsf{setdiff}(k_1,k_2) = \mathsf{0} \tag{137}$$

It can be proved by induction w.r.t. setdiff using (133).

## 4.74 Connection Between subseteq_list and setdiff (pc)

This theorem says that if $k_1 \subset k_2 \subseteq k_3$ then $\mathsf{setdiff}(k_3,k_1) > \mathsf{setdiff}(k_3,k_2)$.

$$\begin{aligned}&\mathsf{subseteq\_list}(k_1,k_2) \wedge \mathsf{not}(\mathsf{subseteq\_list}(k_2,k_1)) \wedge \mathsf{subseteq\_list}(k_2,k_3) \Rightarrow\\ &\quad \mathsf{ge}(\mathsf{setdiff}(k_3,k_1),\mathsf{setdiff}(k_3,k_2)) \wedge \mathsf{not}(\mathsf{ge}(\mathsf{setdiff}(k_3,k_2),\mathsf{setdiff}(k_3,k_1)))\end{aligned} \tag{138}$$

The theorem is transformed into

$$\mathsf{subseteq\_list}(k_1,k_2) \wedge \mathsf{subseteq\_list}(k_2,k_3) \Rightarrow \mathsf{ge}(\mathsf{setdiff}(k_3,k_1),\mathsf{setdiff}(k_3,k_2)) \tag{139}$$

and

$$\begin{aligned}&\mathsf{subseteq\_list}(k_1,k_2) \wedge \mathsf{not}(\mathsf{subseteq\_list}(k_2,k_1)) \wedge \mathsf{subseteq\_list}(k_2,k_3) \Rightarrow\\ &\quad \mathsf{not}(\mathsf{ge}(\mathsf{setdiff}(k_3,k_2),\mathsf{setdiff}(k_3,k_1))).\end{aligned} \tag{140}$$

We first sketch the proof of (139). For that purpose we use an induction w.r.t. both $\mathsf{setdiff}(k_3,k_2)$ and $\mathsf{setdiff}(k_3,k_1)$ (i.e. we perform an induction w.r.t. $\mathsf{setdiff}(k_3,k_2)$ and change the non-induction variable $k_1$ appropriately, cf. the merging technique of [2, 12]). In the base case ($k_3 = \mathsf{empty}$) the proof is trivial. If $k_3 = \mathsf{add}(t,l)$ then we have to regard the different cases. If $\mathsf{member}(t,k_1)$, then (94) implies $\mathsf{member}(t,k_2)$. Hence, the induction conclusion can be transformed into the induction hypothesis and (133). Otherwise, if $\neg\mathsf{member}(t,k_1)$ and $\mathsf{member}(t,k_2)$, then the induction conclusion follows from the induction hypothesis, (133), (135), (41), and (44). Finally, if $\neg\mathsf{member}(t,k_1)$ and $\neg\mathsf{member}(t,k_2)$, then instead of (41) and (44) one can use symbolic evaluation to transform the induction conclusion into the induction hypothesis.

Now we prove (140) applying the same induction relation. In the base case ($k_3 = \mathsf{empty}$) we use (84). If $k_3 = \mathsf{add}(t,l)$ and $\mathsf{member}(t,k_1)$, then (94) again implies $\mathsf{member}(t,k_2)$. So the induction conclusion follows from the induction hypothesis, (133), and (134). Otherwise, if $\neg\mathsf{member}(t,k_1)$ and $\mathsf{member}(t,k_2)$, then the induction conclusion is implied by (133), (134), (139), (43), and the induction hypothesis. Finally, if $\neg\mathsf{member}(t,k_1)$ and $\neg\mathsf{member}(t,k_2)$, then the induction conclusion can be transformed into the induction hypothesis, (133), and (134) by symbolic evaluation.

## 4.75 Distributivity of tail_list over append (pc)

The next theorem says that computing the tail-list of an appended tll is the same as computing the tail-lists of both arguments and appending them afterwards.

$$\mathsf{append}(\mathsf{tail\_list}(k_1), \mathsf{tail\_list}(k_2)) = \mathsf{tail\_list}(\mathsf{append}(k_1, k_2)) \tag{141}$$

The conjecture can be proved by an easy induction w.r.t. append and in this way one can also prove

$$\mathsf{append}(\mathsf{first\_list}(k_1), \mathsf{first\_list}(k_2)) = \mathsf{first\_list}(\mathsf{append}(k_1, k_2)). \tag{142}$$

## 4.76 Stability of member under tail_list(pc)

This conjecture states that if $t$ is a member of the tll $k$ then the tail of $t$ is a member of the tail-list of $k$.

$$\mathsf{member}(t, k) \Rightarrow \mathsf{member}(\mathsf{tail}(t), \mathsf{tail\_list}(k)) \tag{143}$$

This can easily be proved by induction w.r.t. member using (55). In this way one can also prove

$$\mathsf{member}(t, k) \Rightarrow \mathsf{member}(\mathsf{first}(t), \mathsf{first\_list}(k)). \tag{144}$$

## 4.77 Stability of subseteq_list under tail_list (pc)

This conjecture says that is $k_1$ is a subset of $k_2$, then this also holds for the tail-lists.

$$\mathsf{subseteq\_list}(k_1, k_2) \Rightarrow \mathsf{subseteq\_list}(\mathsf{tail\_list}(k_1), \mathsf{tail\_list}(k_2)) \tag{145}$$

It can be proved by induction w.r.t. subseteq_list using (143).

## 4.78 Disjointness of tll's from Disjointness of Their Tails or Heads (pc)

The next theorem states that is the lists of heads or the list of tails of two tll's are disjoint, then the two tll's are also disjoint.

$$\mathsf{disjoint\_list}(\mathsf{first\_list}(k_1), \mathsf{first\_list}(k_2)) \vee \mathsf{disjoint\_list}(\mathsf{tail\_list}(k_1), \mathsf{tail\_list}(k_2)) \Rightarrow \mathsf{disjoint\_list}(k_1, k_2) \tag{146}$$

The conjecture is proved by induction w.r.t. disjoint_list using (143) and (144).

## 4.79 Adding Empty Termlists (pc)

The following theorem states that if one adds an empty termlist to every term in a tll, then the tll does not change.

$$k = \mathsf{addtail}(k, \mathsf{e}) \tag{147}$$

It is easily proved by structural induction on $k$ using $\mathsf{addterm}(s, \mathsf{e}) = s$ which can be proved by induction (resp. case analysis) w.r.t. addterm.

## 4.80 Application of first_list to addtail (pc)

The following conjecture says that (if addtail is defined), then first_list is the inverse to addtail.

$$\mathsf{first\_list}(\mathsf{addtail}(k, s)) = k \tag{148}$$

The conjecture is proved by induction w.r.t. addtail. The base case is obvious and the step case follows using $\mathsf{first}(\mathsf{addterm}(s, t)) = s$ (which can be proved by induction resp. case analysis w.r.t. addterm).

### 4.81 member and apply (pc)

The following conjecture states that one may drop function contexts when regarding member.

$$\mathsf{member}(\mathsf{func}(n, s, \mathsf{e}), \mathsf{apply}(n, k)) \Rightarrow \mathsf{member}(s, k) \tag{149}$$

It can easily be proved by induction w.r.t. member. In this way one can also prove

$$\mathsf{member}(\mathsf{addterm}(s, t), \mathsf{addtail}(k, t)) \Rightarrow \mathsf{member}(s, k) \tag{150}$$

and

$$\mathsf{member}(u, k) \Rightarrow \mathsf{member}(\mathsf{func}(n, u, t), \mathsf{addtail}(\mathsf{apply}(n, k), t)). \tag{151}$$

### 4.82 Stability of subseteq_list Under apply (pc)

The following conjecture states that if $\mathsf{apply}(n, k_1)$ is a subset of $\mathsf{apply}(n, k_2)$, then this also holds for $k_1$ and $k_2$.

$$\mathsf{subseteq\_list}(\mathsf{apply}(n, k_1), \mathsf{apply}(n, k_2)) \Rightarrow \mathsf{subseteq\_list}(k_1, k_2) \tag{152}$$

The conjecture can be proved by induction w.r.t. apply. In the step case one also needs (149).
In a similar way one can also prove

$$\mathsf{subseteq\_list}(\mathsf{addtail}(k_1, t), \mathsf{addtail}(k_2, t)) \Rightarrow \mathsf{subseteq\_list}(k_1, k_2) \tag{153}$$

and a corresponding statement for addfirst.

### 4.83 Stability of disjoint_list under apply (pc)

This conjecture states that if two tll's $k_1$ and $k_2$ are disjoint, then this also holds if one applies a function to them.

$$\mathsf{disjoint\_list}(k_1, k_2) \Rightarrow \mathsf{disjoint\_list}(\mathsf{apply}(n, k_1), \mathsf{apply}(n, k_2)) \tag{154}$$

The conjecture is proved by induction w.r.t. disjoint_list where in the step case one needs the lemma (149).
In a similar way one can also prove

$$\mathsf{disjoint\_list}(k_1, k_2) \Rightarrow \mathsf{disjoint\_list}(\mathsf{addtail}(k_1, t), \mathsf{addtail}(k_2, t)) \tag{155}$$

using the lemma (150).

### 4.84 Distributivity of addtail over append (pc)

The next theorem says that when adding a new tail to every term in an appended tll, then one may instead perform this adding on each of the arguments of append.

$$\mathsf{addtail}(\mathsf{append}(k_1, k_2), t) = \mathsf{append}(\mathsf{addtail}(k_1, t), \mathsf{addtail}(k_2, t)) \tag{156}$$

The conjecture is proved by induction w.r.t. append. In this way one can also prove

$$\mathsf{addfirst}(t, \mathsf{append}(k_1, k_2)) = \mathsf{append}(\mathsf{addfirst}(t, k_1), \mathsf{addfirst}(t, k_2)) \tag{157}$$

and

$$\mathsf{apply}(n, \mathsf{append}(k_1, k_2)) = \mathsf{append}(\mathsf{apply}(n, k_1), \mathsf{apply}(n, k_2)). \tag{158}$$

### 4.85 member for addtail (pc)

The next conjecture states that if $t$ is contained in the tll $k$, then one may also add a new tail to $t$ and $k$.

$$\mathsf{member}(t, k) \Rightarrow \mathsf{member}(\mathsf{addterm}(t, s), \mathsf{addtail}(k, s)) \tag{159}$$

The conjecture is proved by induction w.r.t. member using (55). In a similar way one can also prove

$$\mathsf{member}(t, k) \Rightarrow \mathsf{member}(\mathsf{addterm}(s, t), \mathsf{addfirst}(s, k)) \tag{160}$$

and

$$\mathsf{member}(s, k) \Rightarrow \mathsf{member}(\mathsf{func}(n, s, \mathsf{e}), \mathsf{apply}(n, k)). \tag{161}$$

### 4.86 Stability of $\mathsf{subseteq\_list}$ under $\mathsf{addtail}$ (pc)

The next conjecture says that if $k_1$ is a subset of $k_2$, then this also holds if a termlist is added to each element of these lists.

$$\mathsf{subseteq\_list}(k_1, k_2) \Rightarrow \mathsf{subseteq\_list}(\mathsf{addtail}(k_1, t), \mathsf{addtail}(k_2, t)) \tag{162}$$

The conjecture is proved by induction w.r.t. $\mathsf{subseteq\_list}$ using (159).

### 4.87 $\mathsf{back\_narrowlist}$ has Even Length (pc)

The next conjecture states that every $\mathsf{tll}$ built with $\mathsf{back\_narrowlist}$ has even length.

$$\mathsf{hasevenlength}(\mathsf{back\_narrowlist}(l, t)) \tag{163}$$

The conjecture can be proved by a straightforward induction w.r.t. $\mathsf{back\_narrowlist}$.

### 4.88 $\mathsf{append\_list}$ lifts $\mathsf{appendterm}$ to $\mathsf{tll}$'s (pc)

This is the correctness theorem for $\mathsf{append\_list}$.

$$\mathsf{member}(t, k) \Rightarrow \mathsf{member}(\mathsf{appendterm}(s, t), \mathsf{append\_list}(s, k)) \tag{164}$$

It is easily proved by induction w.r.t. $\mathsf{append\_list}$.

### 4.89 Stability of $\mathsf{subseteq\_list}$ under $\mathsf{append\_list}$ (pc)

The next conjecture relates $\mathsf{subseteq\_list}$ and $\mathsf{append\_list}$.

$$\mathsf{subseteq\_list}(l_1, l_2) \Rightarrow \mathsf{subseteq\_list}(\mathsf{append\_list}(t, l_1), \mathsf{append\_list}(t, l_2)) \tag{165}$$

It is proved by induction w.r.t. $\mathsf{append\_list}$ using (164).

### 4.90 $\mathsf{tail\_list}$ when Appending Terms of Length 1 (pc)

This conjecture states that if one appends a term of length 1, then $\mathsf{tail\_list}$ is the inverse operation to $\mathsf{append\_list}$.

$$\mathsf{length}(t) = \mathsf{s}(0) \Rightarrow \mathsf{tail\_list}(\mathsf{append\_list}(t, k)) = k. \tag{166}$$

This can be proved by induction w.r.t. $\mathsf{append\_list}$ using $\mathsf{length}(t) = \mathsf{s}(0) \Rightarrow \mathsf{tail}(\mathsf{appendterm}(t, s)) = s$ (which can be proved by structural induction (resp. case analysis) on $t$).

### 4.91 Elements of $\mathsf{append\_list}$ (pc)

The next conjecture says that if $\mathsf{tail}(s)$ is a member of $k$, then appending $\mathsf{first}(s)$ to every element of $k$ generates a list containing $s$.

$$\mathsf{member}(\mathsf{tail}(s), k) \Rightarrow \mathsf{member}(s, \mathsf{append\_list}(\mathsf{first}(s), k)) \tag{167}$$

The conjecture can be proved by structural induction on $k$, where in the step case one uses (61) and (66).

### 4.92 $\mathsf{tail\_list}$ of $\mathsf{addtail}$ (pc)

The next conjecture states an obvious connection for $\mathsf{tail\_list}$ and $\mathsf{addtail}$.

$$\mathsf{onlyconsistsof}(\mathsf{tail\_list}(\mathsf{addtail}(k, t)), t) \tag{168}$$

It can easily be proved by induction w.r.t. $\mathsf{addtail}$.

### 4.93 Variables in Rules of TRSs (pc)

The following theorem says that for a TRS, the right-hand sides of rules only contain variables from the left-hand side.

$$\mathsf{trs}(R) \wedge \mathsf{in}(l, r, R) \Rightarrow \mathsf{subseteq}(\mathsf{vars}(r), \mathsf{vars}(l)) \tag{169}$$

It can be proved by a straightforward induction w.r.t. trs.

### 4.94 Rules of TRSs are Built With Functions (pc)

The next conjecture says that in a TRS all left-hand sides are built with a function symbol.

$$\mathsf{trs}(R) \wedge \mathsf{in}(l, r, R) \Rightarrow \mathsf{first\_is\_func}(l) \tag{170}$$

It can be proved by induction (resp. case analysis) w.r.t. trs, as the induction hypothesis is not used.

## 5 Theorems about Substitutions

This section consists of theorems about algorithms dealing with substitutions.

### 5.1 Totality of is_subst

The following theorem is easily proved by structural induction using (55) and (16).

$$\mathsf{def}(t) \Rightarrow \mathsf{def}(\mathsf{is\_subst}(t)) \tag{171}$$

### 5.2 Definedness of apply_subst_var, apply_subst, dom, apply_subst_tll, special_subst, compose, replace

The following conjectures can easily be proved by induction w.r.t. is_subst.

$$\mathsf{def}(n) \wedge \mathsf{is\_subst}(\sigma) \Rightarrow \mathsf{def}(\mathsf{apply\_subst\_var}(\sigma, n)) \tag{172}$$

$$\mathsf{is\_subst}(\sigma) \Rightarrow \mathsf{def}(\mathsf{dom}(\sigma)) \tag{173}$$

$$\mathsf{def}(l) \wedge \mathsf{is\_subst}(\sigma) \Rightarrow \mathsf{def}(\mathsf{apply\_subst\_tll}(\sigma, l)) \tag{174}$$

Using (172), by structural induction one can also prove

$$\mathsf{def}(n) \wedge \mathsf{is\_subst}(\sigma) \Rightarrow \mathsf{def}(\mathsf{apply\_subst}(\sigma, n)). \tag{175}$$

In a similar way one can also prove

$$\mathsf{is\_subst}(\sigma) \wedge \mathsf{is\_subst}(\tau) \Rightarrow \mathsf{def}(\mathsf{special\_subst}(\sigma, \tau)) \tag{176}$$

$$\mathsf{is\_subst}(\sigma) \wedge \mathsf{is\_subst}(\tau) \Rightarrow \mathsf{def}(\mathsf{compose}(\sigma, \tau)) \tag{177}$$

$$\mathsf{def}(n, s) \wedge \mathsf{is\_subst}(\sigma) \Rightarrow \mathsf{def}(\mathsf{replace}(\sigma, n, s)). \tag{178}$$

### 5.3 Totality of matches

The following theorem states that matches is total.

$$\mathsf{def}(s, t) \Rightarrow \mathsf{def}(\mathsf{matches}(s, t)) \tag{179}$$

The theorem can be generalized to

$$\mathsf{def}(s, t, \sigma) \Rightarrow \mathsf{def}(\mathsf{matches\_aux}(s, t, \sigma)).$$

To prove this theorem we proceed in a similar way as in the proof of (24), i.e. we again generate the corresponding domain predicate using a relation which compares pairs of terms by the number of var- and func-occurrences in the first (or second) term. (This relation can easily be generated automatically, cf. [7].)

## 5.4 Definedness of matcher and mgu

The next conjecture states that the truth of matches implies the definedness of matcher.

$$\mathsf{matches}(s,t) \Rightarrow \mathsf{def}(\mathsf{matcher}(s,t)) \tag{180}$$

This conjecture can be generalized to

$$\mathsf{matches\_aux}(s,t,\sigma) \Rightarrow \mathsf{def}(\mathsf{matcher\_aux}(s,t,\sigma))$$

which can easily be proved by induction w.r.t. matches_aux.

In a similar way one can also prove the following related conjecture about unification.

$$\mathsf{unifies}(s,t) \Rightarrow \mathsf{def}(\mathsf{mgu}(s,t)) \tag{181}$$

Note however, that the totality of unifies is much harder to verify than the totality of matches. The reason is that for unifies one needs a much more complicated relation comparing terms by the number of *different* variables occurring in them. To our knowledge, there is no method which can synthesize this relation automatically. Hence, a fully automatic termination proof of unifies is not possible (it is only possible if a suitable relation is given to the system by the *user*). However, with our technique for induction proofs with partial functions, one can nevertheless verify the needed partial correctness statements about unification without proving the termination of unifies (cf. (247), (250), (260)).

## 5.5 Substitutions do not change Variables Outside Their Domain (pc)

The following theorem states that application of a substitution to a variable not in its domain does not change this variable.

$$\neg\mathsf{occurs}(n,\mathsf{dom}(\sigma)) \Rightarrow \mathsf{apply\_subst\_var}(\sigma,n) = \mathsf{var}(n,\mathsf{e}) \tag{182}$$

It can easily be proved by induction w.r.t. apply_subst_var.

## 5.6 Stability of is_subst Under appendterm (pc)

The following conjecture says that appending two substitutions again generates a substitution.

$$\mathsf{is\_subst}(\sigma) \wedge \mathsf{is\_subst}(\tau) \Rightarrow \mathsf{is\_subst}(\mathsf{appendterm}(\sigma,\tau)) \tag{183}$$

It can easily be proved by induction w.r.t. is_subst.

## 5.7 Distributivity of Substitutions Over addterm (pc)

The following theorem shows that substitutions are distributive over addterm.

$$\mathsf{addterm}(\mathsf{apply\_subst}(\sigma,s),\mathsf{apply\_subst}(\sigma,t)) = \mathsf{apply\_subst}(\sigma,\mathsf{addterm}(s,t)) \tag{184}$$

The theorem is proved by induction (resp. case analysis) w.r.t. addterm (using Rule 1$''$).

If $s = \mathsf{var}(n,\mathsf{e})$, then we have to prove

$$\mathsf{addterm}(\mathsf{apply\_subst\_var}(\sigma,n),\mathsf{apply\_subst}(\sigma,t)) = \mathsf{apply\_subst}(\sigma,\mathsf{var}(n,t))$$

which can be verified by symbolic evaluation. The case $s = \mathsf{func}(n,t,\mathsf{e})$ works in a similar way.

In an analogous way one can also prove that the definedness of the second termlist implies the definedness of the first termlist.

$$\mathsf{def}(\mathsf{apply\_subst}(\sigma,\mathsf{addterm}(s,t))) \Rightarrow \mathsf{def}(\mathsf{addterm}(\mathsf{apply\_subst}(\sigma,s),\mathsf{apply\_subst}(\sigma,t))) \tag{185}$$

## 5.8 Distributivity of Substitutions Over $\mathsf{appendterm}$ (pc)

The following theorem shows that substitutions are distributive over $\mathsf{appendterm}$.

$$\mathsf{apply\_subst}(\sigma, \mathsf{appendterm}(s_1, s_2)) = \mathsf{appendterm}(\mathsf{apply\_subst}(\sigma, s_1), \mathsf{apply\_subst}(\sigma, s_2)) \tag{186}$$

The proof is done by induction w.r.t. $\mathsf{appendterm}$. The base case is trivial and in the case $s_1 = \mathsf{var}(n, r)$ one needs (66) and (71) to reduce the induction conclusion to the induction hypothesis. The last step case is straightforward.

## 5.9 Applying $\mathsf{first}$ to $\mathsf{apply\_subst}$ (pc)

This theorem states that $\mathsf{first}$ and $\mathsf{apply\_subst}$ may be exchanged.

$$\mathsf{first}(\mathsf{apply\_subst}(\sigma, t)) = \mathsf{apply\_subst}(\sigma, \mathsf{first}(t)) \tag{187}$$

It can be proved by induction (resp. case analysis) w.r.t. $\mathsf{first}$ using $\mathsf{first}(\mathsf{addterm}(s, t)) = s$ (which can be proved by induction resp. case analysis w.r.t. $\mathsf{addterm}$). In a similar way one can also prove

$$\mathsf{def}(\mathsf{apply\_subst}(\sigma, \mathsf{first}(t))) \Rightarrow \mathsf{def}(\mathsf{first}(\mathsf{apply\_subst}(\sigma, t))) \tag{188}$$

and the corresponding statements for $\mathsf{tail}$:

$$\mathsf{tail}(\mathsf{apply\_subst}(\sigma, t)) = \mathsf{apply\_subst}(\sigma, \mathsf{tail}(t)) \tag{189}$$

$$\mathsf{def}(\mathsf{apply\_subst}(\sigma, \mathsf{tail}(t))) \Rightarrow \mathsf{def}(\mathsf{tail}(\mathsf{apply\_subst}(\sigma, t))). \tag{190}$$

## 5.10 $\mathsf{addterm}$ and $\mathsf{apply\_subst}$ (pc)

We also have

$$\mathsf{addterm}(s, t) = \mathsf{apply\_subst}(\sigma, r) \Rightarrow s = \mathsf{apply\_subst}(\sigma, \mathsf{first}(r)) \wedge t = \mathsf{apply\_subst}(\sigma, \mathsf{tail}(r)), \tag{191}$$

as this is a consequence of (67), (61), (187), and (189).

## 5.11 $\mathsf{appendterm}$ and $\mathsf{apply\_subst\_var}$ (pc)

This conjecture says that unnecessary substitution pairs can be ignored.

$$\mathsf{apply\_subst\_var}(\mathsf{appendterm}(\mathsf{var}(n, t), \sigma), n) = \mathsf{first}(t) \tag{192}$$

It can be proved by symbolic evaluation and the lemma $\mathsf{first}(\mathsf{appendterm}(t, \sigma)) = \mathsf{first}(t)$ (which is easily provable by induction (resp. case analysis) w.r.t. $\mathsf{first}$).

## 5.12 Substitutions Preserve Length (pc)

This theorem says that substitutions preserve the length of a term.

$$\mathsf{length}(t) = \mathsf{length}(\mathsf{apply\_subst}(\sigma, t)) \tag{193}$$

It can easily be proved by induction w.r.t. $\mathsf{apply\_subst}$ using $\mathsf{length}(\mathsf{addterm}(s, t)) = \mathsf{s}(\mathsf{length}(t))$ (which can be proved by induction resp. case analysis w.r.t. $\mathsf{addterm}$).

## 5.13 Length of Termlists Unifying With Variables (pc)

This conjecture says that if $t$ unifies with the variable $n$, then the length of $t$ is 1 (i.e. $t = \mathsf{first}(t)$).

$$\mathsf{apply\_subst\_var}(\sigma, n) = \mathsf{apply\_subst}(\sigma, t) \Rightarrow t = \mathsf{first}(t) \tag{194}$$

This can be proved by (193) and (63).

## 5.14 Equality of Substitutions on Termlists (pc)

The following theorem claims that if $\sigma$ and $\tau$ are the same on the termlist $t$ then this also holds for every variable occurring in $t$.

$$\mathsf{apply\_subst}(\sigma, t) = \mathsf{apply\_subst}(\tau, t) \wedge \mathsf{occurs}(n, t) \Rightarrow \mathsf{apply\_subst\_var}(\sigma, n) = \mathsf{apply\_subst\_var}(\tau, n) \quad (195)$$

The theorem is proved by an easy induction w.r.t. occurs, where in the cases $t = \mathsf{var}(m, r)$ one needs (67) to reduce the induction conclusion to the induction hypothesis.

## 5.15 Equality of a Substitution and an Appended Substitution (pc)

The next conjecture says that if $\sigma$ and the appended substitution $\mathsf{appendterm}(\tau, \sigma)$ behave the same on the termlist $r$, then one may also append substitution pairs from $\sigma$ in front of $\tau$.

$$\begin{aligned}&\mathsf{apply\_subst}(\mathsf{appendterm}(\tau, \sigma), r) = \mathsf{apply\_subst}(\sigma, r) \Rightarrow \\ &\quad \mathsf{apply\_subst}(\mathsf{appendterm}(\mathsf{appendterm}(\mathsf{var}(n, \mathsf{apply\_subst\_var}(\sigma, n)), \tau), \sigma), r) = \mathsf{apply\_subst}(\sigma, r) \quad (196)\end{aligned}$$

The conjecture can be proved by induction w.r.t. apply_subst using the lemmata (67) and (193).

## 5.16 Variables in the Result of Substitutions (pc)

The following conjecture states that if $\sigma$ is applied to the variable $m$, then another variable $n$ can only occur in the result if it also occurs in $\sigma$.

$$\neg n = m \wedge \mathsf{occurs}(n, \mathsf{apply\_subst\_var}(\sigma, m)) \Rightarrow \mathsf{occurs}(n, \sigma) \quad (197)$$

The proof is done by induction w.r.t. apply_subst_var. The base case is easy and if $\sigma = \mathsf{var}(k, t)$ and $\mathsf{eq}(k, n)$, then one needs (106), (107), (95), and (93). Otherwise the induction conclusion can be transformed into the hypothesis, (107), (95), and (93).

## 5.17 Elimination of Variables (pc)

This theorem states that if all occurrences of the variable $n$ in the term $r$ are replaced by $t$, then $n$ can only occur in the resulting term if it also occurs in $t$.

$$\mathsf{occurs}(n, \mathsf{apply\_subst}(\mathsf{var}(n, t), r)) \Rightarrow \mathsf{occurs}(n, t) \quad (198)$$

The conjecture is proved by structural induction on $r$. The case $r = \mathsf{e}$ is easy. If $r = \mathsf{var}(m, s)$ and $\mathsf{eq}(n, m)$, then the induction conclusion is implied by (66), (112), (93), (106), and the induction hypothesis. In the case $\mathsf{eq}(n, m) = \mathsf{false}$, to reduce the induction conclusion to the induction hypothesis one needs (66), (112), (93), (107), and (197). The last case $r = \mathsf{func}(m, s, r')$ is proved by reducing the induction conclusion to the induction hypothesis, (66), (112), (93), (107), and (105).

## 5.18 Variables in Substituted Termlists (Version 1) (pc)

This theorem says that if $\sigma$ is applied to a termlist $r$, then the resulting termlist only contains variables from $r$ and $\sigma$.

$$\mathsf{subseteq}(\mathsf{vars}(\mathsf{apply\_subst}(\sigma, r)), \mathsf{appendterm}(\mathsf{vars}(\sigma), \mathsf{vars}(r))) \quad (199)$$

The conjecture is proved by induction w.r.t. apply_subst. The base case ($r = \mathsf{e}$) is trivial. In the case $r = \mathsf{var}(n, r')$, the induction conclusion follows from the induction hypothesis and (66), (83), (85), (95), (100), and

$$\mathsf{subseteq}(\mathsf{vars}(\mathsf{apply\_subst\_var}(\sigma, n)), \mathsf{var}(n, \mathsf{vars}(\sigma)))$$

(which can be proved by induction w.r.t. apply_subst_var). In the case $r = \mathsf{func}(n, s, r')$, the induction conclusion is implied by both hypotheses, (85) and (102).

### 5.19 Variables in Substituted Termlists (Version 2) (pc)

This is a refinement of the above conjecture which allows to drop the first variable in $\sigma$.

$$\mathsf{subseteq}(\mathsf{vars}(\mathsf{apply\_subst}(\mathsf{var}(n,t),r)),\mathsf{appendterm}(\mathsf{vars}(t),\mathsf{vars}(r)) \tag{200}$$

It is a consequence of (199), (108), and (198).

### 5.20 Symbols in Substituted Termlists (Version 1) (pc)

The next conjecture states that if the variable $n$ occurs in the termlist $t$, then $\sigma(t)$ has at least as much symbols as $\sigma(n)$.

$$\mathsf{occurs}(n,t) \Rightarrow \mathsf{ge}(\mathsf{symbols}(\mathsf{apply\_subst}(\sigma,t)),\mathsf{symbols}(\mathsf{apply\_subst\_var}(\sigma,n))) \tag{201}$$

The conjecture is proved by induction w.r.t. $\mathsf{apply\_subst\_var}$. The base case $t = \mathsf{e}$ is trivial. If $t = \mathsf{var}(m,v)$ and $\mathsf{eq}(m,n)$, then the conjecture follows from (66), (80), and (49). If $\mathsf{eq}(m,n) = \mathsf{false}$, then the induction conclusion follows from the induction hypothesis, (66), (80), (48), and (36). The last case $t = \mathsf{func}(m,s,r)$ can be proved by transforming the induction conclusion into the induction hypothesis, (112), (49), (48), (36), (41), and (44).

### 5.21 Symbols in Substituted Termlists (Version 2) (pc)

The following conjecture refined the previous one by stating that if $t$ is not equal to the variable $n$, then the number of symbols in the instantiation of $t$ is strictly greater than the number of symbols in the instantiation of $n$.

$$\mathsf{occurs}(n,t) \wedge \neg t = \mathsf{var}(n,\mathsf{e}) \Rightarrow \mathsf{gt}(\mathsf{symbols}(\mathsf{apply\_subst}(\sigma,t)),\mathsf{symbols}(\mathsf{apply\_subst\_var}(\sigma,n))) \tag{202}$$

The conjecture is again proved by induction w.r.t. $\mathsf{apply\_subst\_var}$ (resp. by case analysis, as the induction hypotheses are not used). The base case $t = \mathsf{e}$ is trivial. If $t = \mathsf{var}(m,v)$ and $\mathsf{eq}(m,n)$, then the conjecture follows from (66), (80), (79), and (51). If $\mathsf{eq}(m,n) = \mathsf{false}$, then the conjecture follows from (66), (80), (79), (50), (201), and (37). The last case $t = \mathsf{func}(m,s,r)$ can be proved by (112), (201), (37), and (41).

### 5.22 Occur Failure (pc)

This conjecture states that if the variable $n$ occurs in a term $t$ (different from $n$), then $n$ and $t$ are not unifiable.

$$\mathsf{occurs}(n,t) \Rightarrow \neg\mathsf{apply\_subst\_var}(\sigma,n) = \mathsf{apply\_subst}(\sigma,t) \vee t = \mathsf{var}(n,\mathsf{e}) \tag{203}$$

The conjecture is a consequence of (202) and (40).

### 5.23 Application of an Unnecessary Pair (pc)

This theorem says that if $\sigma$ is a unifier of the variable $n$ and the term $t$, then one can replace $n$ by $t$ without changing the result of a subsequent $\sigma$-application.

$$\mathsf{apply\_subst\_var}(\sigma,n) = \mathsf{apply\_subst}(\sigma,t) \Rightarrow \mathsf{apply\_subst}(\sigma,\mathsf{apply\_subst}(\mathsf{var}(n,t),r)) = \mathsf{apply\_subst}(\sigma,r) \tag{204}$$

The conjecture is proved by induction w.r.t. $\mathsf{apply\_subst}$. The case $r = \mathsf{e}$ is trivial. If $r = \mathsf{var}(m,r')$ and $\mathsf{eq}(n,m)$, then the induction conclusion is implied by the induction hypothesis, (184), and (194). If $\mathsf{eq}(n,m) = \mathsf{false}$, the proof is similar (using also the lemma $(\mathsf{tail}(\mathsf{first}(t)) = \mathsf{e}$ which is provable by induction resp. case analysis w.r.t. $\mathsf{first}$). In the last case $r = \mathsf{func}(m,s,r')$, the induction conclusion can be directly reduced to the induction hypothesis.

## 5.24 Correctness of apply_subst_list (pc)

The following theorem is the correctness of apply_subst_list.

$$\mathsf{member}(\sigma, l) \Rightarrow \mathsf{member}(\mathsf{apply\_subst}(\sigma, t), \mathsf{apply\_subst\_list}(l, t)) \tag{205}$$

It can easily be proved by induction w.r.t. apply_subst_list using (55).

## 5.25 apply_subst_list and first (pc)

The next theorem states that first can be shifted to the front when using apply_subst_list.

$$\mathsf{apply\_subst\_list}(k, \mathsf{first}(t)) = \mathsf{first\_list}(\mathsf{apply\_subst\_list}(k, t)) \tag{206}$$

It can be proved by an easy induction w.r.t. apply_subst_list (using (187)). In a similar way one can also prove

$$\mathsf{apply\_subst\_list}(k, \mathsf{tail}(t)) = \mathsf{tail\_list}(\mathsf{apply\_subst\_list}(k, t)). \tag{207}$$

## 5.26 Decomposing the Application of Substitution Lists for Functions (pc)

The next theorem states that application of apply_subst_list to a term built with a function symbol can be decomposed using applytwice.

$$\mathsf{apply\_subst\_list}(l, \mathsf{func}(n, s, t)) = \mathsf{applytwice}(n, \mathsf{apply\_subst\_list}(l, s), \mathsf{apply\_subst\_list}(l, t)) \tag{208}$$

It can be proved by a straightforward induction w.r.t. apply_subst_list. In this way one can also prove

$$\mathsf{def}(\mathsf{apply\_subst\_list}(l, \mathsf{func}(n, s, t))) \Rightarrow \mathsf{def}(\mathsf{applytwice}(n, \mathsf{apply\_subst\_list}(l, s), \mathsf{apply\_subst\_list}(l, t))). \tag{209}$$

## 5.27 Decomposing the Application of Substitution Lists by first and tail (pc)

The following theorem states that application of substitution lists can be decomposed using first and tail.

$$\mathsf{apply\_subst\_list}(l, t) = \mathsf{addtermtwice}(\mathsf{apply\_subst\_list}(l, \mathsf{first}(t)), \mathsf{apply\_subst\_list}(l, \mathsf{tail}(t))) \tag{210}$$

The theorem is proved by induction w.r.t. apply_subst_list. If $l = \mathsf{empty}$, then the conjecture is trivial. If $l = \mathsf{add}(\sigma, l')$, then the induction conclusion reduces to

$$\begin{aligned}&\mathsf{add}(\mathsf{apply\_subst}(\sigma, t), \mathsf{apply\_subst\_list}(l', t)) = \\ &\mathsf{add}(\mathsf{apply\_subst}(\sigma, t), \mathsf{addtermtwice}(\mathsf{apply\_subst\_list}(l', \mathsf{first}(t)), \mathsf{apply\_subst\_list}(l', \mathsf{tail}(t)))\end{aligned}$$

which is a direct consequence of the induction hypothesis.

In the same way one can also prove that definedness of the left hand side of the equation in (210) implies definedness of the right hand side.

$$\mathsf{def}(\mathsf{apply\_subst\_list}(l, t)) \Rightarrow \mathsf{def}(\mathsf{addtermtwice}(\mathsf{apply\_subst\_list}(l, \mathsf{first}(t)), \mathsf{apply\_subst\_list}(l, \mathsf{tail}(t)))). \tag{211}$$

## 5.28 Distributivity of apply_subst_list over append (pc)

The next conjectures state that apply_subst_list is distributive over append. They can easily be proved by induction w.r.t. append.

$$\mathsf{apply\_subst\_list}(\mathsf{append}(l_1, l_2), t) \;=\; \mathsf{append}(\mathsf{apply\_subst\_list}(l_1, t), \mathsf{apply\_subst\_list}(l_2, t)) \tag{212}$$
$$\mathsf{def}(\mathsf{apply\_subst\_list}(\mathsf{append}(l_1, l_2), t)) \;\Leftrightarrow\; \mathsf{def}(\mathsf{append}(\mathsf{apply\_subst\_list}(l_1, t), \mathsf{apply\_subst\_list}(l_2, t))) \tag{213}$$

## 5.29 apply_subst_list and append_list (Version 1) (pc)

The next lemma simplifies expressions with apply_subst_list.

$$\begin{aligned}&\mathsf{apply\_subst\_list}(\mathsf{append\_list}(\mathsf{var}(m,\mathsf{e}),k),\mathsf{var}(n,\mathsf{e}))=\\&\mathsf{if}(\mathsf{eq}(m,n),\mathsf{first\_list}(k),\mathsf{apply\_subst\_list}(\mathsf{tail\_list}(k),\mathsf{var}(n,\mathsf{e}))).\end{aligned}\tag{214}$$

It can easily be proved by structural induction on $k$. In this way one can also prove

$$\begin{aligned}&\mathsf{def}(\mathsf{apply\_subst\_list}(\mathsf{append\_list}(\mathsf{var}(m,\mathsf{e}),k),\mathsf{var}(n,\mathsf{e})))\wedge\neg\mathsf{eq}(m,n)\Rightarrow\\&\mathsf{def}(\mathsf{apply\_subst\_list}(\mathsf{tail\_list}(k),\mathsf{var}(n,\mathsf{e})))\end{aligned}\tag{215}$$

and

$$\mathsf{def}(\mathsf{apply\_subst\_list}(\mathsf{append\_list}(\mathsf{var}(n,\mathsf{e}),k),\mathsf{var}(n,\mathsf{e})))\Rightarrow\mathsf{def}(\mathsf{first\_list}(k)).\tag{216}$$

## 5.30 apply_subst_list and append_list (Version 2) (pc)

The next conjecture states a similar fact.

$$\mathsf{eq}(m,n)=\mathsf{false}\Rightarrow\mathsf{apply\_subst\_list}(\mathsf{append\_list}(\mathsf{var}(m,\mathsf{first}(t)),k),\mathsf{var}(n,\mathsf{e}))=\mathsf{apply\_subst\_list}(k,\mathsf{var}(n,\mathsf{e})).\tag{217}$$

It can again be proved by structural induction w.r.t. $k$ where in the step case one needs the lemma $\mathsf{tail}(\mathsf{appendterm}(\mathsf{first}(t),\sigma))=\sigma$ which can easily be proved by induction (resp. case analysis) w.r.t. first. In this way one can also prove

$$\begin{aligned}&\mathsf{def}(\mathsf{apply\_subst\_list}(\mathsf{append\_list}(\mathsf{var}(m,\mathsf{first}(t)),k),\mathsf{var}(n,\mathsf{e})))\wedge\mathsf{eq}(m,n)=\mathsf{false}\Rightarrow\\&\mathsf{def}(\mathsf{apply\_subst\_list}(k,\mathsf{var}(n,\mathsf{e})))\end{aligned}\tag{218}$$

## 5.31 onlyconsistsof and append_list (Version 1) (pc)

The following theorem states that if a substitution list where each element starts with $\mathsf{var}(n,t)$ is applied to the variable $n$, then the resulting lists only contains $\mathsf{first}(t)$.

$$\mathsf{onlyconsistsof}(\mathsf{apply\_subst\_list}(\mathsf{append\_list}(\mathsf{var}(n,t),k),\mathsf{var}(n,\mathsf{e})),\mathsf{first}(t))\tag{219}$$

The conjecture is proved by structural induction on $k$, where in the step case one needs (55) and $\mathsf{first}(\mathsf{appendterm}(t,s))=\mathsf{first}(t)$ which can easily be proved by induction (resp. case analysis) w.r.t. first.

## 5.32 onlyconsistsof and append_list (Version 2) (pc)

The next conjecture says that if a substitution list $k$ only consists of one substitution $\sigma$, then application of $k$ to a termlist $t$ produces a list only containing $\sigma(t)$.

$$\mathsf{onlyconsistsof}(k,\sigma)\Rightarrow\mathsf{onlyconsistsof}(\mathsf{apply\_subst\_list}(k,t),\mathsf{apply\_subst}(\sigma,t))\tag{220}$$

The conjecture is easily proved by induction w.r.t. onlyconsistsof using (55).

## 5.33 Connection Between apply_subst_tll and append_list (pc)

The next theorems state the connection between apply_subst_tll and append_list.

$$\mathsf{apply\_subst\_tll}(\sigma,\mathsf{append\_list}(\mathsf{var}(n,\mathsf{e}),l))=\mathsf{append\_list}(\mathsf{apply\_subst\_var}(\sigma,n),\mathsf{apply\_subst\_tll}(\sigma,l))\tag{221}$$

$$\mathsf{def}(\mathsf{apply\_subst\_tll}(\sigma,\mathsf{append\_list}(\mathsf{var}(n,\mathsf{e}),l)))\Leftrightarrow\mathsf{def}(\mathsf{append\_list}(\mathsf{apply\_subst\_var}(\sigma,n),\mathsf{apply\_subst\_tll}(\sigma,l)))\tag{222}$$

Both theorems can be proved by induction w.r.t. append_list. For that purpose one has to replace the first argument $\mathsf{var}(n,\mathsf{e})$ of append_list by a new variable $q$ and add the premise $q=\mathsf{var}(n,\mathsf{e})$. In the base case $l=\mathsf{empty}$ the proofs are trivial. If $l=\mathsf{add}(s,l')$, then for (221) we obtain the induction conclusion

$$\begin{aligned}&\mathsf{add}(\mathsf{apply\_subst}(\sigma, \mathsf{appendterm}(\mathsf{var}(n, \mathsf{e}), s)), \mathsf{apply\_subst\_tll}(\sigma, \mathsf{append\_list}(\mathsf{var}(n, \mathsf{e}), l'))) =\\&\mathsf{add}(\mathsf{appendterm}(\mathsf{apply\_subst\_var}(\sigma, n), \mathsf{apply\_subst}(\sigma, s)),\\&\qquad \mathsf{append\_list}(\mathsf{apply\_subst\_var}(\sigma, n), \mathsf{apply\_subst\_tll}(\sigma, l'))).\end{aligned}$$

This follows from the induction hypothesis, (193), (185), and (66). The proof for (222) is completely analogous.

## 5.34 Distributivity of apply_subst_tll over append (pc)

The next conjecture states that apply_subst_tll is distributive over append.

$$\mathsf{apply\_subst\_tll}(\sigma, \mathsf{append}(l_1, l_2)) = \mathsf{append}(\mathsf{apply\_subst\_tll}(\sigma, l_1), \mathsf{apply\_subst\_tll}(\sigma, l_2)) \tag{223}$$

The conjecture is proved by induction w.r.t. append. The case $l_1 = \mathsf{empty}$ is trivial and in the case $l_1 = \mathsf{add}(t, l)$ the induction conclusion reduces to the induction hypothesis. In a similar way one can also prove

$$\mathsf{def}(\mathsf{append}(\mathsf{apply\_subst\_tll}(\sigma, l_1), \mathsf{apply\_subst\_tll}(\sigma, l_2))) \Leftrightarrow \mathsf{def}(\mathsf{apply\_subst\_tll}(\sigma, \mathsf{append}(l_1, l_2))). \tag{224}$$

## 5.35 Connection Between apply_subst and apply_subst_tll (pc)

This theorem states that if $t$ is a member of $k$, then $\sigma(t)$ is a member of the list containing all $\sigma$-instantiations of elements from $k$.

$$\mathsf{member}(t, k) \Rightarrow \mathsf{member}(\mathsf{apply\_subst}(\sigma, t), \mathsf{apply\_subst\_tll}(\sigma, k)) \tag{225}$$

The conjecture can be proved by an easy induction w.r.t. apply_subst_tll using (55).

## 5.36 Disjointness of Instantiated Lists (pc)

The following theorem states that if two instantiated lists are disjoint, then so are the original lists.

$$\mathsf{disjoint\_list}(\mathsf{apply\_subst\_tll}(\sigma, l), \mathsf{apply\_subst\_tll}(\sigma, k)) \Rightarrow \mathsf{disjoint\_list}(l, k) \tag{226}$$

The conjecture can be proved by induction w.r.t. apply_subst_tll using $\sigma$ and $l$ as induction variables. The base case is easy and in the step case the induction conclusion follows from the induction hypothesis, (122), (92), (101), and (225).

## 5.37 Substitution Outside of Domain (pc)

The next conjecture states that application of a substitution to a variable outside of its domain does not change the variable.

$$\neg\mathsf{occurs}(n, \mathsf{dom}(\sigma)) \Rightarrow \mathsf{apply\_subst\_var}(\sigma, n) = \mathsf{var}(n, \mathsf{e}) \tag{227}$$

It can be directly proved by induction w.r.t. dom.

## 5.38 Distributivity of dom over appendterm (pc)

The following conjecture states that the domain of an appended substitution can be obtained by appending the two subdomains.

$$\mathsf{dom}(\mathsf{appendterm}(s, t)) = \mathsf{appendterm}(\mathsf{dom}(s), \mathsf{dom}(t)) \tag{228}$$

The conjecture can be proved by induction w.r.t. dom. In the step case one needs the lemma $\mathsf{tail}(\mathsf{appendterm}(r, t)) = \mathsf{appendterm}(\mathsf{tail}(r), t)$ which can easily be proved by induction w.r.t. tail.

## 5.39 Appending Substitutions (Version 1) (pc)

The following conjecture says that if a variable already occurs in $\tau$'s domain, then appending a substitution to $\tau$ in the back does not change the result of applying the substitution to $n$.

$$\mathsf{occurs}(n, \mathsf{dom}(\tau)) \Rightarrow \mathsf{apply\_subst\_var}(\mathsf{appendterm}(\tau, \sigma), n) = \mathsf{apply\_subst\_var}(\tau, n) \tag{229}$$

The conjecture is proved by induction w.r.t. dom. In the step case ($\tau = \mathsf{var}(m, r)$) we have to distinguish the cases depending on the truth of $\mathsf{eq}(n, m)$. If this is true, then the conjecture is implied by (192). Otherwise, the proof is similar as for (228).

## 5.40 Appending Substitutions (Version 2) (pc)

This theorem states that if $n$ is not in the domain of the second substitution, then we only have to regard the first substitution.

$$\neg\mathsf{occurs}(n, \mathsf{dom}(\sigma)) \Rightarrow \mathsf{apply\_subst\_var}(\mathsf{appendterm}(\tau, \sigma), n) = \mathsf{apply\_subst\_var}(\tau, n) \tag{230}$$

The theorem can be proved by induction w.r.t. apply_subst_var using $\tau$ and $n$ as induction variables. In the base case $\tau = \mathsf{e}$ we need (227). In the case $\tau = \mathsf{var}(m, t)$ and $\mathsf{eq}(n, m)$ one needs $\mathsf{first}(\mathsf{appendterm}(t, s)) = \mathsf{appendterm}(\mathsf{first}(t), s)$ and in the other case one needs the corresponding statement for tail.

## 5.41 Appending Substitutions (Version 3) (pc)

This is the symmetric counterpart of conjecture (230).

$$\neg\mathsf{occurs}(n, \mathsf{dom}(\tau)) \Rightarrow \mathsf{apply\_subst\_var}(\mathsf{appendterm}(\tau, \sigma), n) = \mathsf{apply\_subst\_var}(\sigma, n). \tag{231}$$

It can be proved by induction w.r.t. dom where the step case is similar to the last case in the previous proof.

## 5.42 Appending Substitutions on Disjoint Domains (Version 1) (pc)

The next theorem says that those parts of substitutions which concern only variables that do not occur in the termlist can be omitted.

$$\mathsf{disjoint}(\mathsf{dom}(\mu), \mathsf{vars}(t)) \Rightarrow \mathsf{apply\_subst}(\mathsf{appendterm}(\sigma, \mu), t) = \mathsf{apply\_subst}(\sigma, t) \tag{232}$$

The theorem is proved by induction w.r.t. apply_subst. The base case $t = \mathsf{e}$ is trivial. In the case $t = \mathsf{var}(n, r)$, the induction conclusion is implied by the induction hypothesis and (121), (100), (123), and (230). In the case $t = \mathsf{func}(n, s, r)$ the induction conclusion can be transformed into the induction hypothesis and (121), (98), (100).

## 5.43 Appending Substitutions on Disjoint Domains (Version 2) (pc)

This is the symmetric counterpart to the previous theorem.

$$\mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(t)) \Rightarrow \mathsf{apply\_subst}(\mathsf{appendterm}(\sigma, \mu), t) = \mathsf{apply\_subst}(\mu, t) \tag{233}$$

Its proof is similar to the proof of (232), but instead of (230) we now need (231).

## 5.44 Domain of Renamed Substitutions (pc)

This conjecture states that if $m$ is in the domain of $\sigma$, then $m + n$ is in the domain of the substitution that results from $\sigma$ by renaming all variables (by adding $n$ to them).

$$\mathsf{occurs}(m, \mathsf{dom}(\sigma)) = \mathsf{occurs}(\mathsf{plus}(m, n), \mathsf{dom}(\mathsf{rename\_dom}(\sigma, n))) \tag{234}$$

The conjecture is proved by induction w.r.t. dom. The case $\sigma = \mathsf{e}$ is easy and if $\sigma = \mathsf{var}(k, r)$ then in the case $\mathsf{eq}(m, k)$ the conjecture follows from (54) and (47) and otherwise the induction hypothesis, (47), and $\mathsf{tail}(\mathsf{appendterm}(\mathsf{first}(s), t)) = t$ imply the induction conclusion.

## 5.45 Applying Renamed Substitutions (pc)

This theorem states that application of a substitution to $n$ is the same as application of the renamed substitution to the renamed variable.

$$\mathsf{occurs}(m, \mathsf{dom}(\sigma)) \Rightarrow \mathsf{apply\_subst\_var}(\sigma, m) = \mathsf{apply\_subst\_var}(\mathsf{rename\_dom}(\sigma, n), \mathsf{plus}(m, n)) \tag{235}$$

The conjecture is proved by induction w.r.t. $\mathsf{apply\_subst\_var}$. The base case is trivial and in the case $\sigma = \mathsf{var}(k, r)$ we have to distinguish two cases. If $\mathsf{eq}(k, m)$ then the conjecture follows from (54) and $\mathsf{first}(\mathsf{appendterm}(\mathsf{first}(s), t)) = \mathsf{first}(s)$. Otherwise, the induction conclusion is implied by the induction hypothesis, (54), and (47).

## 5.46 matcher computes Substitutions (pc)

The next conjecture states that every matcher is a substitution.

$$\mathsf{is\_subst}(\mathsf{matcher}(s, t)) \tag{236}$$

This conjecture can be evaluated and generalized to

$$\mathsf{is\_subst}(\mathsf{matcher\_aux}(s, t, \sigma)).$$

This is proved by induction w.r.t. $\mathsf{matcher\_aux}$.

## 5.47 Domain of matcher (pc)

The following theorem states that a matcher only changes variables from the term to be matched.

$$\mathsf{subseteq}(\mathsf{dom}(\mathsf{matcher}(s, t)), \mathsf{vars}(s)) \tag{237}$$

The theorem is transformed to

$$\mathsf{subseteq}(\mathsf{dom}(\mathsf{matcher\_aux}(s, t, \sigma)), \mathsf{appendterm}(\mathsf{vars}(s), \mathsf{dom}(\sigma)))$$

which is proved by induction w.r.t. $\mathsf{matcher\_aux}$. The base case is trivial and if $s = \mathsf{var}(n, r)$ and $\mathsf{occurs}(n, \mathsf{dom}(\sigma))$, then the induction conclusion can be reduced to the induction hypothesis and (102), (100), (91), and (95). If $\neg\mathsf{occurs}(n, \mathsf{dom}(\sigma))$, then one needs (98), (100), and (85). In the final case, the reduction of the induction conclusion into the hypothesis is also easy.

## 5.48 Already Computed Matcher is not Changed (pc)

The next conjecture says that when computing $\mathsf{matcher\_aux}(s, t, \sigma)$, then the resulting substitution behaves like $\sigma$ on $\sigma$'s domain.

$$\mathsf{occurs}(n, \mathsf{dom}(\sigma)) \Rightarrow \mathsf{apply\_subst\_var}(\mathsf{matcher\_aux}(s, t, \sigma), n) = \mathsf{apply\_subst\_var}(\sigma, n) \tag{238}$$

The conjecture is proved by induction w.r.t. $\mathsf{matcher\_aux}$. The base case is easy. If $s = \mathsf{func}(m, s, r)$ or if $s = \mathsf{var}(m, r)$ and $\mathsf{occurs}(m, \sigma)$, then the induction conclusion is a direct consequence of the induction hypothesis. In the case $s = \mathsf{var}(m, r)$ and $\mathsf{occurs}(m, \sigma) = \mathsf{false}$ we have $\neg n = m$. Hence, the induction conclusion follows from the hypothesis, (93), (228), and (100).

## 5.49 Correctness of matcher (pc)

This is the correctness theorem for the matching algorithm.

$$\mathsf{matches}(s, t) \Rightarrow t = \mathsf{apply\_subst}(\mathsf{matcher}(s, t), s) \tag{239}$$

It can be evaluated and generalized to

$$\mathsf{matches\_aux}(s,t,\sigma) \Rightarrow t = \mathsf{apply\_subst}(\mathsf{matcher}(s,t,\sigma),s).$$

We prove this conjecture by induction w.r.t. $\mathsf{matches\_aux}$. If $s = \mathsf{e}$, then (55) implies $t = \mathsf{e}$. Hence, in the base case the proof is trivial.

If $s = \mathsf{var}(n,r)$, then (55) implies $\neg t = \mathsf{e}$. If $\mathsf{occurs}(n,\mathsf{dom}(\sigma))$, then we have $\mathsf{first}(t) = \mathsf{apply\_subst}(\sigma, n)$. By (238) and (61), the induction conclusion can be reduced to the induction hypothesis. If $\neg\mathsf{occurs}(n, \mathsf{dom}(\sigma))$, by (61) the induction conclusion can again be reduced to the induction hypothesis, (238), and (61). Finally, the last step case can be proved using (186) and (78).

## 5.50 Correctness of matches (pc)

The next theorem states that a term matches each of its instantiations.

$$\mathsf{matches}(t,\mathsf{apply\_subst}(\sigma,t)) \tag{240}$$

It can be generalized to

$$\mathsf{apply\_subst}(\mathsf{appendterm}(\tau,\sigma),t) = \mathsf{apply\_subst}(\sigma,t) \Rightarrow \mathsf{matches\_aux}(t,\mathsf{apply\_subst}(\sigma,t),\tau).$$

This conjecture can be proved by induction w.r.t. $\mathsf{matches\_aux}$. The base case $t = \mathsf{e}$ is trivial. In the first step case ($t = \mathsf{var}(n,r)$), by (65) we have to consider two cases. If $\mathsf{occurs}(n,\mathsf{dom}(\tau))$, then the induction conclusion follows from (67), (229), and the induction hypothesis. If $\mathsf{occurs}(n,\mathsf{dom}(\tau)) = \mathsf{false}$, then one needs the lemma (196). Finally, in the case $t = \mathsf{func}(n,s,r)$, the induction conclusion follows from the induction hypothesis, (193), and (55).

## 5.51 Renaming for matches (pc)

The next theorem states that if $s$ matches $t$, then this also holds if $s$ is renamed.

$$\mathsf{matches}(s,t) \Rightarrow \mathsf{matches}(\mathsf{rename}(s,n),t) \tag{241}$$

The conjecture can be transformed into

$$\mathsf{matches\_aux}(s,t,\sigma) \Rightarrow \mathsf{matches\_aux}(\mathsf{rename}(s,n),t,\mathsf{rename\_dom}(\sigma,n)).$$

We prove this conjecture by induction w.r.t. $\mathsf{matches\_aux}$. The base case $s = \mathsf{e}$ is trivial. In the case $s = \mathsf{var}(m,r)$ one needs (234), (235), and (128) to transform the induction conclusion into the hypothesis. In the last case $s = \mathsf{func}(m,u,r)$ this can be done by (130) and (129).

## 5.52 Renaming for matcher (pc)

The next theorem states that if $r$ only contains variables from $l$, then application of the matcher of $l$ and $t$ to $r$ is the same as application of the corresponding matcher if $l$ and $r$ are renamed.

$$\mathsf{subseteq}(\mathsf{vars}(r),\mathsf{vars}(l)) \Rightarrow \mathsf{apply\_subst}(\mathsf{matcher}(l,t),r) = \mathsf{apply\_subst}(\mathsf{matcher}(\mathsf{rename}(l,n),t),\mathsf{rename}(r,n)) \tag{242}$$

We perform a structural induction on $r$. The case $r = \mathsf{e}$ is trivial. In the case $r = \mathsf{func}(m,s,r')$ the induction conclusion is implied by the induction hypothesis, (95), (98), and (100). In the case $r = \mathsf{var}(m,r')$ to reduce the induction conclusion to the induction hypothesis one needs the conjecture

$$\mathsf{occurs}(m,\mathsf{vars}(l)) \Rightarrow \mathsf{apply\_subst\_var}(\mathsf{matcher}(l,t),m) = \mathsf{apply\_subst\_var}(\mathsf{matcher}(\mathsf{rename}(l,n),t),\mathsf{plus}(m,n)).$$

This can be transformed into

$$\begin{aligned}\mathsf{occurs}(m,\mathsf{vars}(l)) \Rightarrow\ & \mathsf{apply\_subst\_var}(\mathsf{matcher\_aux}(l,t,\sigma),m) = \\ & \mathsf{apply\_subst\_var}(\mathsf{matcher\_aux}(\mathsf{rename}(l,n),t,\mathsf{rename\_dom}(\sigma,n)),\mathsf{plus}(m,n)).\end{aligned}$$

To prove this conjecture we use an induction w.r.t. $\mathsf{matcher\_aux}$. The base case is trivial (since the premise evaluates to $\mathsf{false}$) and in the step cases with $l = \mathsf{var}(k,r)$ the induction conclusion follows from the induction hypothesis and (234). The case $l = \mathsf{func}(k,q,r)$ can be proved using (129).

## 5.53 Matcher is Most General (Version 1) (pc)

This conjecture says that for every variable from $l$, the matcher of $l$ with $\sigma(l)$ behaves like $\sigma$.

$$\mathsf{occurs}(n, \mathsf{vars}(l)) \Rightarrow \mathsf{apply\_subst\_var}(\sigma, n) = \mathsf{apply\_subst\_var}(\mathsf{matcher}(l, \mathsf{apply\_subst}(\sigma, l)), n). \tag{243}$$

The conjecture is a consequence of (240), (239), and (195).

## 5.54 Matcher is Most General (Version 2) (pc)

This conjecture says that for every $r$ containing only variables from $l$, the matcher of $l$ with $\sigma(l)$ behaves like $\sigma$.

$$\mathsf{subseteq}(\mathsf{vars}(r), \mathsf{vars}(l)) \Rightarrow \mathsf{apply\_subst}(\sigma, r) = \mathsf{apply\_subst}(\mathsf{matcher}(l, \mathsf{apply\_subst}(\sigma, l)), r) \tag{244}$$

This conjecture can be proved by induction w.r.t. apply_subst. The base case is trivial. In the case $r = \mathsf{var}(n, r')$ one needs (95), (100), and (243) to transform the induction conclusion into the hypothesis. In the last case, the conclusion can be transformed into (86) and both induction hypotheses.

## 5.55 Adding New Elements Produces no Duplicates (pc)

The next conjecture states that if $\sigma$ is a substitution without duplicates and if one adds a new substitution pair for a variable outside of $\sigma$'s domain, then the resulting substitution has no duplicates either.

$$\mathsf{no\_duplicates}(\sigma) \wedge \neg\mathsf{occurs}(n, \mathsf{dom}(\sigma)) \Rightarrow \mathsf{no\_duplicates}(\mathsf{appendterm}(\mathsf{var}(n, \mathsf{first}(t)), \sigma)) \tag{245}$$

The conjecture is proved by symbolic evaluation of no_duplicates (and $\mathsf{tail}(\mathsf{appendterm}(\mathsf{first}(t), \sigma)) = \sigma$, which can easily be proved by induction (resp. case analysis) w.r.t. first).

## 5.56 Matcher Contains no Duplicates (pc)

The following theorem says that a matcher contains no duplicates.

$$\mathsf{no\_duplicates}(\mathsf{matcher}(s, t)) \tag{246}$$

The conjecture is transformed into

$$\mathsf{no\_duplicates}(\sigma) \Rightarrow \mathsf{no\_duplicates}(\mathsf{matcher\_aux}(s, t, \sigma)).$$

This conjecture is proved by induction w.r.t. matcher_aux. The base case is trivial and if $s = \mathsf{func}(m, s, r)$ or if $s = \mathsf{var}(m, r)$ and $\mathsf{occurs}(m, \sigma)$, then the induction conclusion is a direct consequence of the induction hypothesis. In the case $s = \mathsf{var}(m, r)$ and $\mathsf{occurs}(m, \sigma) = \mathsf{false}$, the induction conclusion follows from the hypothesis and (245).

## 5.57 Correctness of unifies (pc)

The following theorem states the correctness of the algorithm unifies.

$$\mathsf{apply\_subst}(\sigma, s) = \mathsf{apply\_subst}(\sigma, t) \Rightarrow \mathsf{unifies}(s, t) \tag{247}$$

For this theorem a method for induction proofs with partial functions is very advantageous, because we need an induction w.r.t. unifies. Of course, unifies is total, but this is hard to prove automatically. But with our method we can perform an induction w.r.t. unifies without verifying its termination.

In the base case we have $s = \mathsf{e}$. Now the conjecture follows from

$$\mathsf{e} = \mathsf{apply\_subst}(\sigma, t) \Rightarrow \mathsf{eqterm}(\mathsf{e}, t)$$

which can be proved by induction w.r.t. apply_subst.

In the second case, we have $s = \mathsf{var}(n_1, r_1)$ and $t = \mathsf{e}$. Now the conjecture is implied by (65) and (55).

In the case $s = \mathsf{var}(n_1, r_1)$, $t = \mathsf{var}(n_2, r_2)$, we only have to show that the premise of the induction conclusion implies the premise of the induction hypothesis. This follows from (67) and (204).

If $s = \mathsf{var}(n_1, r_1)$ and $t = \mathsf{func}(n_2, s_2, r_2)$ then by (67), (73) and (203), the premise of the induction conclusion implies $\neg\mathsf{occurs}(n_1, \mathsf{appendterm}(s_2, \mathsf{e}))$. Hence, again we only have to show that the premise of the induction hypothesis is implied by the premise of the induction conclusion. This can be done using (67) and (204).

If $s = \mathsf{func}(n_1, s_1, r_1)$ and $t = \mathsf{var}(n_2, r_2)$ then the induction conclusion directly follows from the induction hypothesis.

Finally, if $s = \mathsf{func}(n_1, s_1, r_1)$ and $t = \mathsf{func}(n_2, s_2, r_2)$, then the premise of the induction conclusion is $\mathsf{func}(n_1, \mathsf{apply\_subst}(\sigma, s_1), \mathsf{apply\_subst}(\sigma, r_1)) = \mathsf{func}(n_2, \mathsf{apply\_subst}(\sigma, s_2), \mathsf{apply\_subst}(\sigma, r_2))$. By (54) and (193) this implies $\mathsf{eq}(n_1, n_2)$ and $\mathsf{eq}(\mathsf{length}(s_1), \mathsf{length}(s_2))$. Now the induction conclusion follows from the induction hypothesis and (186).

## 5.58 Relation between Matching and Unification (pc)

This theorem relates matching and unification.

$$\mathsf{matches}(s, \mathsf{apply\_subst}(\sigma, t)) \wedge \mathsf{disjoint}(\mathsf{vars}(t), \mathsf{vars}(s)) \wedge \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(s)) \Rightarrow \mathsf{unifies}(s, t) \tag{248}$$

By (239), $\mathsf{matches}(s, \mathsf{apply\_subst}(\sigma, t))$ implies $\mathsf{apply\_subst}(\mathsf{matcher}(s, \mathsf{apply\_subst}(\sigma, t)), s) = \mathsf{apply\_subst}(\sigma, t)$. By (121), (123), (232), and (233), this implies

$$\begin{aligned} &\mathsf{apply\_subst}(\mathsf{appendterm}(\mathsf{matcher}(s, \mathsf{apply\_subst}(\sigma, t)), \sigma), s) = \\ &\mathsf{apply\_subst}(\mathsf{appendterm}(\mathsf{matcher}(s, \mathsf{apply\_subst}(\sigma, t)), \sigma), t). \end{aligned}$$

Hence, the conjecture is implied by (247).

## 5.59 mgu generates Substitutions (pc)

The following conjecture says that the result of mgu is always a substitution.

$$\mathsf{is\_subst}(\mathsf{mgu}(s, t)) \tag{249}$$

It can easily be proved by induction w.r.t. mgu using (59).

## 5.60 Domain of mgu (pc)

The next theorem says that $\mathsf{mgu}(s, t)$ only changes variables occurring in $s$ or $t$.

$$\mathsf{subseteq}(\mathsf{dom}(\mathsf{mgu}(s, t)), \mathsf{appendterm}(\mathsf{vars}(s), \mathsf{vars}(t))) \tag{250}$$

The theorem is proved by induction w.r.t. mgu. The base case ($s = t = \mathsf{e}$) is trivial. In the case $s = \mathsf{var}(n_1, r_1)$ and $\mathsf{var}(n_1, \mathsf{e}) = \mathsf{first}(t)$ the induction conclusion can be reduced to the induction hypothesis, (61), (100), (102), and (95). If $s = \mathsf{var}(n_1, r_1)$ and $\neg\mathsf{var}(n_1, \mathsf{e}) = \mathsf{first}(t)$, then by (55), to reduce the induction conclusion to the induction hypothesis we need (66), (64), (249), (200), (83), (98), (100), (85), (102), (95), and (114). If $s = \mathsf{func}(n_1, s_1, r_1)$ and $t = \mathsf{var}(n_2, r_2)$, then the induction conclusion is a direct consequence of the induction hypothesis. Finally, in the last step case, the induction conclusion is transformed into the induction hypothesis and (81).

## 5.61 Definedness of special_subst and of apply_subst

The following conjecture states that if $\mathsf{special\_subst}(\sigma, \tau)$ is defined and true, then the application of $\sigma$ on the domain of $\tau$ is also defined.

$$\mathsf{special\_subst}(\sigma, \tau) \Rightarrow \mathsf{def}(\mathsf{apply\_subst}(\sigma, \mathsf{dom}(\tau))) \tag{251}$$

The conjecture is proved by induction w.r.t. special_subst. In the case $\tau = \mathsf{e}$ it is trivial. Otherwise we have $\tau = \mathsf{var}(n, t)$ and the induction conclusion can be transformed into

$$\mathsf{eqterm}(\mathsf{apply\_subst\_var}(\sigma, n), \mathsf{apply\_subst}(\sigma, \mathsf{first}(t))) \wedge \mathsf{special\_subst}(\sigma, \mathsf{tail}(t))) \Rightarrow$$
$$\mathsf{def}(\mathsf{addterm}(\mathsf{apply\_subst\_var}(\sigma, n), \mathsf{apply\_subst}(\sigma, \mathsf{dom}(\mathsf{tail}(t))))).$$

This is a consequence of the induction hypothesis, (19), and (193).
In a similar way one can also prove

$$\mathsf{special\_subst}(\sigma, \tau) \Rightarrow \mathsf{def}(\mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst}(\tau, \mathsf{dom}(\tau)))). \tag{252}$$

## 5.62 Correctness of special_subst (pc)

The following theorem states the correctness of special_subst.

$$\mathsf{special\_subst}(\sigma, \tau) \Rightarrow \mathsf{apply\_subst}(\sigma, t) = \mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst}(\tau, t)) \tag{253}$$

The theorem is proved by induction w.r.t. apply_subst. The base case ($t = \mathsf{e}$) reduces to a tautology. We now consider the remaining two cases.

**Case 1:** $t = \mathsf{var}(n, t')$

Using (184) and (185), the induction conclusion is evaluated to

$$\begin{aligned} &\mathsf{special\_subst}(\sigma, \tau) \Rightarrow \\ &\qquad \mathsf{addterm}(\mathsf{apply\_subst\_var}(\sigma, n), \mathsf{apply\_subst}(\sigma, t')) \\ &\quad = \ \mathsf{addterm}(\mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst\_var}(\tau, n)), \mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst}(\tau, t'))) \end{aligned}$$

and using (67), (193), and (15), it is transformed further into the induction hypothesis and

$$\mathsf{special\_subst}(\sigma, \tau) \Rightarrow \mathsf{apply\_subst\_var}(\sigma, n) = \mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst\_var}(\tau, n)).$$

This conjecture is proved by induction w.r.t. apply_subst_var (using $\tau$ and $n$ as induction variables). If $\tau = \mathsf{e}$, then it can be proved by symbolic evaluation. Otherwise we have $\tau = \mathsf{var}(m, r)$. If $\mathsf{eqterm}(m, n)$ is false, then the induction conclusion is transformed into the induction hypothesis. Otherwise (if $m = n$) the induction conclusion is transformed into

$$\mathsf{eqterm}(\mathsf{apply\_subst\_var}(\sigma, n), \mathsf{apply\_subst}(\sigma, \mathsf{first}(r))) \Rightarrow \mathsf{apply\_subst\_var}(\sigma, n) = \mathsf{apply\_subst}(\sigma, \mathsf{first}(r))$$

which is an instantiation of (55).

**Case 2:** $t = \mathsf{func}(n, u, t')$

The induction conclusion reduces to

$$\begin{aligned} \mathsf{special\_subst}(\sigma, \tau) \Rightarrow &\mathsf{func}(\mathsf{apply\_subst}(\sigma, u), \mathsf{apply\_subst}(\sigma, t)) = \\ &\mathsf{func}(\mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst}(\tau, u)), \mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst}(\tau, t))) \end{aligned}$$

which is implied by the induction hypotheses.

## 5.63 Correctness of compose (pc)

The next theorem states that compose indeed composes substitutions.

$$\mathsf{apply\_subst}(\mathsf{compose}(\sigma_1, \sigma_2), t) = \mathsf{apply\_subst}(\sigma_2, \mathsf{apply\_subst}(\sigma_1, t)) \tag{254}$$

The theorem is proved by induction w.r.t. apply_subst. The base case ($t = \mathsf{e}$) is trivial. We now consider the two remaining cases.

**Case 1:** $t = \mathsf{var}(n, r)$

Now (using (184) and (185)) the induction conclusion can be evaluated to

$$\begin{array}{l}\mathsf{addterm}(\mathsf{apply\_subst\_var}(\mathsf{compose}(\sigma_1, \sigma_2), n), \mathsf{apply\_subst}(\mathsf{compose}(\sigma_1, \sigma_2), r)) = \\ \mathsf{addterm}(\mathsf{apply\_subst}(\sigma_2, \mathsf{apply\_subst\_var}(\sigma_1, n)), \mathsf{apply\_subst}(\sigma_2, \mathsf{apply\_subst}(\sigma_1, r))).\end{array}$$

Using (67) and (15) this can be transformed into the induction hypothesis and into

$$\mathsf{apply\_subst\_var}(\mathsf{compose}(\sigma_1, \sigma_2), n) = \mathsf{apply\_subst}(\sigma_2, \mathsf{apply\_subst\_var}(\sigma_1, n))$$

which can be further evaluated to

$$\begin{array}{l}\mathsf{apply\_subst\_var}(\mathsf{compose\_aux}(\sigma_1, \sigma_2, \mathsf{disjoint\_union}(\mathsf{dom}(\sigma_1), \mathsf{dom}(\sigma_2))), n) = \\ \mathsf{apply\_subst}(\sigma_2, \mathsf{apply\_subst\_var}(\sigma_1, n)).\end{array}$$

We perform the following case analysis by Rule 6$''$.

**Case 1.1:** $\mathsf{occurs}(n, \mathsf{disjoint\_union}(\mathsf{dom}(\sigma_1), \mathsf{dom}(\sigma_2)))$
In this case, the conjecture can be generalized to

$$\mathsf{occurs}(n, v) \Rightarrow \mathsf{apply\_subst\_var}(\mathsf{compose\_aux}(\sigma_1, \sigma_2, v), n) = \mathsf{apply\_subst}(\sigma_2, \mathsf{apply\_subst\_var}(\sigma_1, n))$$

This lemma is proved by induction w.r.t. $\mathsf{compose\_aux}$. The case $v = \mathsf{e}$ is trivial, since the premise is $\mathsf{false}$. If $v = \mathsf{var}(n, q)$, we have to distinguish two subcases. If $\mathsf{eqterm}(\mathsf{apply\_subst}(\sigma_1, \mathsf{apply\_subst\_var}(\sigma_2, n)), \mathsf{var}(n, \mathsf{e}))$ $\wedge\, \mathsf{not}(\mathsf{occurs}(n, \mathsf{dom}(\sigma_1))) = \mathsf{true}$, then the proof is straightforward. Otherwise, the induction conclusion reduces to

$$\begin{array}{rl} & \mathsf{apply\_subst\_var}(\mathsf{var}(n, \mathsf{addterm}(\mathsf{apply\_subst}(\sigma_2, \mathsf{apply\_subst\_var}(\sigma_1, n)), \ldots)), n) \\ = & \mathsf{apply\_subst}(\sigma_2, \mathsf{apply\_subst\_var}(\sigma_1, n))\end{array}$$

which can be transformed into a tautology. Finally, in the case $v = \mathsf{var}(m, q)$ (where $m \neq n$) the induction conclusion can be evaluated to the induction hypothesis.

**Case 1.2:** $\neg\mathsf{occurs}(n, \mathsf{disjoint\_union}(\mathsf{dom}(\sigma_1), \mathsf{dom}(\sigma_2)))$
In this case we have

$$\mathsf{apply\_subst}(\sigma_2, \mathsf{apply\_subst\_var}(\sigma_1, n)) = \mathsf{var}(n, \mathsf{e})$$

(which follows from (111) and (182)). Hence, it suffices to prove

$$\neg\mathsf{occurs}(n, v) \Rightarrow \mathsf{apply\_subst\_var}(\mathsf{compose\_aux}(\sigma_1, \sigma_2, v), n) = \mathsf{var}(n, \mathsf{e}).$$

This can be proved by induction w.r.t. $\mathsf{compose\_aux}$. The case $v = \mathsf{e}$ is again trivial. In the case $v = \mathsf{var}(m, q)$ the premise implies $m \neq n$. But then the induction conclusion can be immediately transformed into the induction hypothesis.

**Case 2:** $t = \mathsf{func}(n, s, r)$

The induction conclusion can be evaluated to

$$\begin{array}{rl} & \mathsf{func}(n, \mathsf{apply\_subst}(\mathsf{compose}(\sigma_1, \sigma_2), s), \mathsf{apply\_subst}(\mathsf{compose}(\sigma_1, \sigma_2), r)) \\ = & \mathsf{func}(n, \mathsf{apply\_subst}(\sigma_2, \mathsf{apply\_subst}(\sigma_1, s)), \mathsf{apply\_subst}(\sigma_2, \mathsf{apply\_subst}(\sigma_1, r))).\end{array}$$

This can be immediately transformed into the induction hypotheses.

## 5.64 Relation between compose and special_subst (pc)

The following theorem says that if $\sigma = \sigma \circ \tau$ holds, then $\sigma$ is more special than $\tau$.

$$\sigma = \mathsf{compose}(\tau, \sigma) \Rightarrow \mathsf{special\_subst}(\sigma, \tau) \tag{255}$$

By Rule 4″ this can be transformed into (251), (252),

$$\sigma = \mathsf{compose}(\tau, \sigma) \Rightarrow \mathsf{apply\_subst}(\sigma, \mathsf{dom}(\tau)) = \mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst}(\tau, \mathsf{dom}(\tau)))$$

(which can be proved by (254)) and

$$\mathsf{apply\_subst}(\sigma, \mathsf{dom}(\tau)) = \mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst}(\tau, \mathsf{dom}(\tau))) \Rightarrow \mathsf{special\_subst}(\sigma, \tau).$$

For this conjecture we perform an induction w.r.t. special_subst. If $\tau = \mathsf{e}$ then it is obviously true. In the case $\tau = \mathsf{var}(n, t)$, by symbolic evaluation the induction conclusion can be transformed into

$$\begin{array}{l}\mathsf{addterm}(\mathsf{apply\_subst\_var}(\sigma, n), \mathsf{apply\_subst}(\sigma, \mathsf{dom}(\mathsf{tail}(t)))) = \\ \mathsf{apply\_subst}(\sigma, \mathsf{addterm}(\mathsf{first}(t), \mathsf{apply\_subst}(\tau, \mathsf{dom}(\mathsf{tail}(t))))) \Rightarrow \\ \quad \mathsf{apply\_subst\_var}(\sigma, n) = \mathsf{apply\_subst}(\sigma, \mathsf{first}(t)) \wedge \mathsf{special\_subst}(\sigma, \mathsf{tail}(t)).\end{array}$$

This is a consequence of the induction hypothesis and (184), (185), (15), (16), (67), (193), (63).

## 5.65 Removing Unnecessary Variables when Composing With Empty Substitution (pc)

The following conjecture states that if $n$ does not occur in the intended domain of a composition with the empty substitution, then the corresponding variable term pair may be deleted.

$$\neg\mathsf{occurs}(n, v) \Rightarrow \mathsf{compose\_aux}(\mathsf{e}, \mathsf{var}(n, r), v) = \mathsf{compose\_aux}(\mathsf{e}, \mathsf{tail}(r), v) \tag{256}$$

The conjecture is proved by a straightforward induction w.r.t. compose_aux.

## 5.66 Composition with Empty Substitution (pc)

The next theorem states that composition with the empty substitution does not change substitutions.

$$\mathsf{no\_duplicates}(\sigma) \Rightarrow \mathsf{compose\_aux}(\mathsf{e}, \sigma, \mathsf{dom}(\sigma)) = \sigma \tag{257}$$

The conjecture is proved by induction w.r.t. dom. The base case is trivial and in the step case the induction conclusion can be reduced to the induction hypothesis and (256).

## 5.67 Removing Unnecessary Pairs from a Composition (pc)

The next conjecture says that if $\sigma$ is a unifier of the variable $n$ and the term $t$, then one may remove a substitution pair $n/t$ from $\tau$ when composing $\tau$ and $\sigma$.

$$\begin{array}{l}\mathsf{apply\_subst\_var}(\sigma, n) = \mathsf{apply\_subst}(\sigma, t) \wedge \mathsf{not}(\mathsf{occurs}(n, \mathsf{dom}(\tau))) \Rightarrow \\ \qquad \mathsf{compose\_aux}(\mathsf{appendterm}(\mathsf{var}(n, t), \tau), \sigma, v) = \mathsf{compose\_aux}(\tau, \sigma, v)\end{array} \tag{258}$$

The conjecture is proved by induction w.r.t. compose_aux. The base case is trivial and in the step case ($v = \mathsf{var}(m, t)$) we have to consider two cases.

If $n = m$, then the conjecture follows from

$$\mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst\_var}(\mathsf{appendterm}(\mathsf{var}(n, t), \tau), n)) = \mathsf{apply\_subst\_var}(\sigma, n)$$

(which is a consequence of (192) and the premise $\mathsf{apply\_subst\_var}(\sigma, n) = \mathsf{apply\_subst}(\sigma, t)$) and

$$\mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst\_var}(\tau, n)) = \mathsf{apply\_subst\_var}(\sigma, n)$$

which is a consequence of (227) and the premise $\mathsf{not}(\mathsf{occurs}(n, \mathsf{dom}(\tau)))$.

If $n \neq m$ then the theorem is a consequence of

$$\mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst\_var}(\mathsf{appendterm}(\mathsf{var}(n,t),\tau), m)) = \mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst\_var}(\tau, m))$$

which follows from (54).

## 5.68 Elimination of Variables in Compositions (pc)

The next theorem states that if the first argument and the second argument of compose_aux are inverse on a variable $n$, then this variable may be deleted from the domain list.

$$\begin{aligned}&\mathsf{apply\_subst}(\sigma, t) = \mathsf{var}(n, \mathsf{e}) \wedge \neg\mathsf{occurs}(n, \mathsf{dom}(\sigma)) \Rightarrow\\ &\quad \mathsf{compose\_aux}(\mathsf{appendterm}(\mathsf{var}(n,t),\tau), \sigma, \mathsf{disjoint\_union}(\mathsf{appendterm}(v_1, \mathsf{var}(n,\mathsf{e})), v_2)) =\\ &\quad \mathsf{compose\_aux}(\mathsf{appendterm}(\mathsf{var}(n,t),\tau), \sigma, \mathsf{disjoint\_union}(v_1, v_2))\end{aligned} \tag{259}$$

The conjecture is proved by induction w.r.t. disjoint_union. In the base case we have $v_2 = \mathsf{e}$. Hence, we have to prove

$$\begin{aligned}&\mathsf{apply\_subst}(\sigma, t) = \mathsf{var}(n, \mathsf{e}) \wedge \neg\mathsf{occurs}(n, \mathsf{dom}(\sigma)) \Rightarrow\\ &\quad \mathsf{compose\_aux}(\mathsf{appendterm}(\mathsf{var}(n,t),\tau), \sigma, \mathsf{appendterm}(v_1, \mathsf{var}(n,\mathsf{e}))) =\\ &\quad \mathsf{compose\_aux}(\mathsf{appendterm}(\mathsf{var}(n,t),\tau), \sigma, v_1).\end{aligned}$$

For this conjecture we use an induction w.r.t. compose_aux. The base case $v_1 = \mathsf{e}$ can be proved using (192). In the two step cases, the induction conclusion is directly implied by the induction hypothesis.

Now we consider the step cases of the outer disjoint_union-induction, i.e. $v_2 = \mathsf{var}(m, v_2')$. If $\mathsf{occurs}(m, v_1)$, then by (93) and (98), the induction conclusion is implied by the induction hypothesis. If $\neg\mathsf{occurs}(m, v_1)$ and $\mathsf{occurs}(m, \mathsf{appendterm}(v_1, \mathsf{var}(n, \mathsf{e})))$, then (55) and (112) imply $n = m$. Hence, the conclusion of the induction conclusion is a tautology. Finally, in the remaining case the induction conclusion can be evaluated to the induction hypothesis.

## 5.69 mgu is Most General (pc)

The following theorem proves that the mgu is really most general.

$$\mathsf{no\_duplicates}(\sigma) \wedge \mathsf{apply\_subst}(\sigma, s) = \mathsf{apply\_subst}(\sigma, t) \Rightarrow \sigma = \mathsf{compose}(\mathsf{mgu}(s,t), \sigma) \tag{260}$$

Again we benefit from our technique for induction proofs with partial functions, because we need an induction w.r.t. mgu. Of course, if $s$ and $t$ unify then $\mathsf{mgu}(s,t)$ is total. However, this is very hard to prove automatically. On the other hand, with our method for induction proofs with (possibly) partial functions, we can perform an induction w.r.t. mgu without having to verify its termination.

In the base case we have $s = t = \mathsf{e}$. Now the conjecture follows from (257). If $s = \mathsf{func}(n, u, r)$, then the proof is straightforward and if $s = \mathsf{var}(n, r)$, then we distinguish two cases (omitting the premise $\mathsf{no\_duplicates}(\sigma)$).

**Case 1:** $\mathsf{first}(t) = \mathsf{var}(n, \mathsf{e})$

By (55) and (191), the premise of the induction conclusion implies the premise $\mathsf{apply\_subst}(\sigma, r) = \mathsf{apply\_subst}(\sigma, \mathsf{tail}(t))$ of the induction hypothesis. Moreover, the conclusion of the induction conclusion can be evaluated to the conclusion of the induction hypothesis.

**Case 2:** $\neg\mathsf{first}(t) = \mathsf{var}(n, \mathsf{e})$

By (55) and (191), the premise of the induction conclusion implies $\mathsf{apply\_subst\_var}(\sigma, n) = \mathsf{apply\_subst}(\sigma, \mathsf{first}(t))$ and $\mathsf{apply\_subst}(\sigma, r) = \mathsf{apply\_subst}(\sigma, \mathsf{tail}(t))$. Hence, by (204) this implies the premise

$$\mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst}(\mathsf{var}(n, \mathsf{first}(t)), r)) = \mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst}(\mathsf{var}(n, \mathsf{first}(t)), \mathsf{tail}(t)))$$

of the induction hypothesis.
Using (228) and (58), the conclusion of the induction conclusion is evaluated to

$$\begin{aligned}\sigma = \mathsf{compose\_aux}(&\mathsf{appendterm}(\mathsf{var}(n, \mathsf{first}(t)), \mathsf{mgu}(\mathsf{apply\_subst}(\mathsf{var}(n, \mathsf{first}(t)), r),\\ &\qquad\qquad \mathsf{apply\_subst}(\mathsf{var}(n, \mathsf{first}(t)), \mathsf{tail}(t)))),\\ &\sigma,\\ &\mathsf{disjoint\_union}(\mathsf{dom}(\sigma), \mathsf{var}(n, \mathsf{dom}(\mathsf{mgu}(\mathsf{apply\_subst}(\mathsf{var}(n, \mathsf{first}(t)), r),\\ &\qquad\qquad \mathsf{apply\_subst}(\mathsf{var}(n, \mathsf{first}(t)), \mathsf{tail}(t))))))).\end{aligned}$$

We now perform a case analysis (Rule $6''$) according to $\mathsf{disjoint\_union}$.

**Case 2.1:** $\mathsf{occurs}(n, \mathsf{dom}(\sigma))$

Hence, $\mathsf{disjoint\_union}(\mathsf{dom}(\sigma), \mathsf{var}(n, \mathsf{dom}(\mathsf{mgu}(\ldots))))$ can be evaluated to $\mathsf{disjoint\_union}(\mathsf{dom}(\sigma), \mathsf{dom}(\mathsf{mgu}(\ldots)))$. Now the conclusion

$$\sigma = \mathsf{compose\_aux}(\mathsf{mgu}(\mathsf{apply\_subst}(\mathsf{var}(n, \mathsf{first}(t)), r), \mathsf{apply\_subst}(\mathsf{var}(n, \mathsf{first}(t)), \mathsf{tail}(t))), \sigma, \mathsf{disjoint\_union}(\ldots))$$

of the induction hypothesis and (258), (203), (250), (198), (112), (93) imply the conclusion of the induction conclusion.

**Case 2.2:** $\neg\mathsf{occurs}(n, \mathsf{dom}(\sigma))$

Now $\mathsf{disjoint\_union}(\mathsf{dom}(\sigma), \mathsf{var}(n, \mathsf{dom}(\mathsf{mgu}(\ldots))))$ is evaluated to $\mathsf{disjoint\_union}(\mathsf{appendterm}(\mathsf{dom}(\sigma), \mathsf{var}(n, \mathsf{e})), \mathsf{dom}(\mathsf{mgu}(\ldots)))$. By (227), the premise of the induction conclusion implies $\mathsf{apply\_subst}(\sigma, \mathsf{first}(t)) = \mathsf{var}(n, \mathsf{e})$. Together with (259) and (258), (203), (250), (198), (112), (93), the conclusion of the induction hypothesis implies the conclusion of the induction conclusion.

### 5.70 replace generates Substitutions (pc)

The following conjecture states that the result of replace is a substitution.

$$\mathsf{is\_subst}(\sigma) \Rightarrow \mathsf{is\_subst}(\mathsf{replace}(\sigma, n, t)) \tag{261}$$

The conjecture is proved by induction w.r.t. replace. The base case is trivial. In the step case where $\mathsf{eq}(m, n) = \mathsf{true}$, the induction conclusion follows from (65) and (64). In the other case the induction conclusion is implied by the induction hypothesis and (183), (59), (58).

## 6 Theorems about Rewriting

In this section we prove theorems about rewriting.

### 6.1 Definedness of rewrites_rule, rewrites_matcher, rewrites, rule

The first conjecture says that $\mathsf{rewrites\_rule}(t, s, l, r)$ is total provided $s$ is not e.

$$\mathsf{def}(t, s, l, r) \wedge \neg s = \mathsf{e} \Rightarrow \mathsf{def}(\mathsf{rewrites\_rule}(t, s, l, r)) \tag{262}$$

The conjecture is easily proved by structural induction on $s$.

In a similar way (using also already proved theorems about definedness) one can verify the following statements.

$$\mathsf{def}(l,r) \wedge \neg s = \mathsf{e} \wedge \mathsf{rewrites\_rule}(t,s,l,r) \Rightarrow \mathsf{def}(\mathsf{rewrites\_matcher}(t,s,l,r)) \tag{263}$$

$$\mathsf{def}(t,s) \wedge \neg s = \mathsf{e} \wedge \mathsf{trs}(R) \Rightarrow \mathsf{def}(\mathsf{rewrites}(t,s,R) \tag{264}$$

$$\mathsf{def}(\mathsf{rewrites}(t,s,R)) \Rightarrow \mathsf{def}(\mathsf{rule}(t,s,R)) \tag{265}$$

## 6.2 Definedness of rewrite_rule, rewrite_rule_list, rewrite_list

Now we prove the definedness theorem for rewrite_rule.

$$\mathsf{def}(t,l) \wedge \mathsf{length}(r) = \mathsf{s}(0) \Rightarrow \mathsf{def}(\mathsf{rewrite\_rule}(t,l,r)) \tag{266}$$

This theorem can be proved by structural induction on $t$ using (236), (193) and

$$\mathsf{def}(n,l,t) \wedge \mathsf{length}(r) = \mathsf{s}(0) \Rightarrow \mathsf{def}(\mathsf{addtail}(\mathsf{add}(r,\mathsf{apply}(n,l)),t))$$

(which can also be proved easily).
In a similar way one can show

$$\mathsf{def}(k,l) \wedge \mathsf{length}(r) = \mathsf{s}(0) \Rightarrow \mathsf{def}(\mathsf{rewrite\_rule\_list}(k,l,r)) \tag{267}$$

$$\mathsf{def}(k) \wedge \mathsf{trs}(R) \Rightarrow \mathsf{def}(\mathsf{rewrite\_list}(k,R)). \tag{268}$$

## 6.3 Definedness of rewrites_rule implies Non-Emptiness

The following conjecture says that if rewrites_rule is defined and the first argument is not empty, then the second one is not empty either.

$$\mathsf{def}(\mathsf{rewrites\_rule}(\mathsf{addterm}(t,q),s,l,r)) \Rightarrow \neg s = \mathsf{e} \tag{269}$$

The conjecture is proved by induction (resp. case analysis) w.r.t. addterm. In all cases, the term rewrites_rule( ...) can be reduced to a term containing first($s$) (in the first argument of an if). Hence, the conjecture follows from (55) and (59).

## 6.4 Decomposing rewrites_rule with addterm (Version 1) (pc)

The next theorem says that if $s$ rewrites to $t$, then this also holds if a term $q$ is added in the front.

$$\mathsf{rewrites\_rule}(s,t,l,r) \Rightarrow \mathsf{rewrites\_rule}(\mathsf{addterm}(q,s),\mathsf{addterm}(q,t),l,r) \tag{270}$$

The conjecture is proved by induction (resp. case analysis) w.r.t. addterm. In both cases ($q = \mathsf{var}(n,\mathsf{e})$ and $q = \mathsf{func}(n,q',\mathsf{e})$), the conjecture can be proved by symbolic evaluation.

## 6.5 Decomposing rewrites_rule with addterm (Version 2) (pc)

The next theorem says that if $s$ rewrites to $t$, then this also holds if a termlist $q$ is added in the back.

$$\mathsf{rewrites\_rule}(s,t,l,r) \Rightarrow \mathsf{rewrites\_rule}(\mathsf{addterm}(s,q),\mathsf{addterm}(t,q),l,r) \tag{271}$$

The conjecture is also proved by induction (resp. case analysis) w.r.t. both occurrences of addterm. In the cases where $s = \mathsf{var}(n,\mathsf{e})$, the premise reduces to false, because $\mathsf{rewrites\_rule}(\mathsf{e}.\mathsf{e},l,r)$ is false. If $s = \mathsf{func}(n,s',\mathsf{e})$, then the second or the third disjunct must hold and the premise of the implication can be reduced to the conclusion.

## 6.6 Composing $\mathsf{rewrites\_rule}$ with $\mathsf{addterm}$ (pc)

The next theorem is a kind of converse to the preceding ones. It states that when rewriting a list of terms, the rewriting is either done in the first element or in the tail.

$$\begin{aligned}&\mathsf{rewrites\_rule}(\mathsf{addterm}(t_1, t_2), s, l, r) \Rightarrow \\ &\quad \mathsf{rewrites\_rule}(t_1, \mathsf{first}(s), l, r) \wedge t_2 = \mathsf{tail}(s) \ \vee \ \mathsf{rewrites\_rule}(t_2, \mathsf{tail}(s), l, r) \wedge t_1 = \mathsf{first}(s)\end{aligned} \tag{272}$$

The theorem is also easily proved by induction (resp. case analysis) w.r.t. addterm and symbolic evaluation.

## 6.7 Length Preservation Under Rewriting (pc)

The next theorem says that application of rewrite rules to termlists preserves their length.

$$\mathsf{rewrites\_rule}(s, t, l, r) \Rightarrow \mathsf{length}(s) = \mathsf{length}(t) \tag{273}$$

It can be proved by a straightforward induction w.r.t. rewrites_rule.

## 6.8 $\mathsf{rewrites\_rule}$ under Contexts (pc)

The following theorem states that rewriting remains stable under function contexts.

$$\mathsf{rewrites\_rule}(s, t, l, r) \Rightarrow \mathsf{rewrites\_rule}(\mathsf{func}(n, s, r'), \mathsf{func}(n, t, r'), l, r) \tag{274}$$

It can easily be proved by symbolic evaluation and Rule $4''$.

## 6.9 Rewriting with Renamed Rules (pc)

The next theorem says that if $t$ rewrites to $s$ with the rule $l \to r$, then this also works with the rule where $l$ and $r$ have been renamed.

$$\mathsf{rewrites\_rule}(t, s, l, r) \wedge \mathsf{subseteq}(\mathsf{vars}(r), \mathsf{vars}(l)) \Rightarrow \mathsf{rewrites\_rule}(t, s, \mathsf{rename}(l, n), \mathsf{rename}(r, n)) \tag{275}$$

The conjecture can easily be proved by induction w.r.t. rewrites_rule, where in the third case one needs (241) and (131). In a similar way one can also prove

$$\mathsf{def}(\mathsf{rewrites\_rule}(t, s, l, r)) \wedge \mathsf{subseteq}(\mathsf{vars}(r), \mathsf{vars}(l)) \Rightarrow \mathsf{def}(\mathsf{rewrites\_rule}(t, s, \mathsf{rename}(l, n), \mathsf{rename}(r, n))) \tag{276}$$

## 6.10 Rewriting of Instantiated Rules (pc)

This theorem states that rules may be instantiated.

$$\begin{aligned}&\mathsf{length}(l) = \mathsf{s}(0) \wedge \mathsf{length}(r) = \mathsf{s}(0) \wedge \mathsf{first\_is\_func}(l) \wedge \mathsf{subseteq}(\mathsf{vars}(r), \mathsf{vars}(l)) \Rightarrow \\ &\qquad\qquad \mathsf{rewrites\_rule}(\mathsf{apply\_subst}(\sigma, l), \mathsf{apply\_subst}(\sigma, r), l, r)\end{aligned} \tag{277}$$

By induction (resp. case analysis) w.r.t. first_is_func and length one can determine that the premise implies $l = \mathsf{func}(n, u, \mathsf{e})$. Then the conjecture follows from (240), (55), (244), and (193).

## 6.11 Correctness of $\mathsf{rule}$ (pc)

The following theorem states that rule indeed returns a rule which allows the desired reduction.

$$\mathsf{rewrites}(t, s, R) \Rightarrow \mathsf{rewrites\_rule}(t, s, \mathsf{first}(\mathsf{rule}(t, s, R)), \mathsf{second}(\mathsf{rule}(t, s, R))). \tag{278}$$

The conjecture can easily be proved by induction w.r.t. rewrites.

### 6.12 rule only Generates Rules from the TRS (pc)

This theorem states that rule only returns rules of the TRS.

$$\mathsf{in}(\mathsf{first}(\mathsf{rule}(t,s,R)),\mathsf{second}(\mathsf{rule}(t,s,R)),R) \tag{279}$$

The proof is a straightforward induction w.r.t. rule.

### 6.13 rewrites_matcher Generates Substitutions (pc)

The next theorem is similar to (236).

$$\mathsf{is\_subst}(\mathsf{rewrites\_matcher}(t,s,l,r)) \tag{280}$$

It can easily be proved by induction w.r.t. rewrites_matcher using (236).

### 6.14 Domain of rewrites_matcher (pc)

The next conjecture states that $\mathsf{rewrites\_matcher}(t,s,l,r)$ only changes variables from $l$.

$$\mathsf{subseteq}(\mathsf{dom}(\mathsf{rewrites\_matcher}(t,s,l,r)),\mathsf{vars}(l)) \tag{281}$$

It can be proved by induction w.r.t. rewrites_matcher. In all cases except the last one, the induction conclusion is implied by the induction hypothesis. In the last case, the conjecture follows from (237).

### 6.15 apply and rewrite_rule (pc)

The following conjecture states the connection between apply and func when using rewrite_rule.

$$\mathsf{subseteq\_list}(\mathsf{apply}(n,\mathsf{rewrite\_rule}(t,l,r)),\mathsf{rewrite\_rule}(\mathsf{func}(n,t,\mathsf{e}),l,r)) \tag{282}$$

It can be proved by symbolic evaluation and (97), (162), (96), and (147).

### 6.16 addtail and rewrite_rule (pc)

The following conjecture states the connection between addtail and addterm when using rewrite_rule.

$$\mathsf{subseteq\_list}(\mathsf{addtail}(\mathsf{rewrite\_rule}(s,l,r),t),\mathsf{rewrite\_rule}(\mathsf{addterm}(s,t),l,r)) \tag{283}$$

The conjecture is proved by induction (resp. case analysis) w.r.t. addterm. In the case $s = \mathsf{var}(n,\mathsf{e})$, it is easy to prove. In the case $s = \mathsf{func}(n,u,\mathsf{e})$ one needs (74), (147), and (99).

### 6.17 tail_list and rewrite_rule (pc)

The following conjecture states the connection between tail and tail_list when using rewrite_rule.

$$\mathsf{subseteq\_list}(\mathsf{rewrite\_rule}(\mathsf{tail}(t),l,r),\mathsf{tail\_list}(\mathsf{rewrite\_rule}(t,l,r))) \tag{284}$$

The conjecture is proved by induction (resp. case analysis) w.r.t. tail. In the case $t = \mathsf{var}(n,t')$, the conjecture follows from (166). In the case $t = \mathsf{func}(n,u,t')$ one needs (141), (101), and (166).

### 6.18 tail_list and rewrite_rule (Version 2) (pc)

The following conjecture states the (converse) connection between tail and tail_list when using rewrite_rule.

$$\mathsf{member}(t,\mathsf{tail\_list}(\mathsf{rewrite\_rule}(s,l,r))) \Rightarrow t = \mathsf{tail}(s) \vee \mathsf{member}(t,\mathsf{rewrite\_rule}(\mathsf{tail}(s),l,r)) \tag{285}$$

The conjecture is proved by induction (resp. case analysis) w.r.t. tail. In the case $t = \mathsf{var}(n,t')$, it again follows from (166). In the case $t = \mathsf{func}(n,u,t')$ one needs (141), (166), and (168).

## 6.19 Exchanging $\mathsf{rewrite\_rule\_list}$ and $\mathsf{append}$ (pc)

The following theorem says that $\mathsf{rewrite\_rule\_list}$ is distributive over $\mathsf{append}$.

$$\mathsf{append}(\mathsf{rewrite\_rule\_list}(k_1, l, r), \mathsf{rewrite\_rule\_list}(k_2, l, r)) = \mathsf{rewrite\_rule\_list}(\mathsf{append}(k_1, k_2), l, r) \tag{286}$$

The conjecture is easily proved by induction w.r.t. $\mathsf{append}$ using (72).

## 6.20 $\mathsf{rewrites\_rule}$ implies $\mathsf{rewrite\_rule}$ (pc)

The following theorem states that if $t$ reduces to $s$ according to $\mathsf{rewrites\_rule}$, then $s$ is also a member of $\mathsf{rewrite\_rule}(t, l, r)$.

$$\mathsf{rewrites\_rule}(t, s, l, r) \Rightarrow \mathsf{member}(s, \mathsf{rewrite\_rule}(t, l, r)) \tag{287}$$

We prove the theorem by induction w.r.t. $\mathsf{rewrites\_rule}$. The base case $t = \mathsf{e}$ is easy. In the case $t = \mathsf{var}(n, t')$, the induction conclusion can be reduced to the induction hypothesis using (167). If $t = \mathsf{func}(n, u, t')$ we consider three cases according to the definition of $\mathsf{rewrites\_rule}$. In the first case, the induction conclusion can be proved by the hypothesis and (167), (94), (101), (96). In the second case the induction conclusion is transformed into (151), (94), (99), (96). Finally, the third case can be proved using (151) and (94).

## 6.21 $\mathsf{rewrite\_rule}$ implies $\mathsf{rewrite\_rule\_list}$ (pc)

The following conjecture states the connection between $\mathsf{rewrite\_rule}$ and $\mathsf{rewrite\_rule\_list}$.

$$\mathsf{member}(t, k) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_rule}(t, l, r), \mathsf{rewrite\_rule\_list}(k, l, r)) \tag{288}$$

We use Rule $1''$ to perform an induction w.r.t. $\mathsf{member}$. The case $k = \mathsf{empty}$ is trivial. If $k = \mathsf{add}(s, k')$ and $t = s$, then the induction conclusion can be reduced to the induction hypothesis and (99). If $k = \mathsf{add}(s, k')$ and $t \neq s$, then the induction conclusion is transformed into the induction hypothesis, (96), and (101).

## 6.22 Correctness of $\mathsf{replace}$ (pc)

The next conjecture states that if $\sigma$ applied to $n$ rewrites to $s$, then $\mathsf{replace}(\sigma, n, s)$ is a member of $\mathsf{all\_reductions}$. In other words, $\mathsf{rewrites\_rule}$ can be used to construct an element from $\mathsf{all\_reductions}$.

$$\mathsf{rewrites\_rule}(\mathsf{apply\_subst\_var}(\sigma, n), s, l, r) \Rightarrow \mathsf{member}(\mathsf{replace}(\sigma, n, s), \mathsf{all\_reductions}(\sigma, l, r)) \tag{289}$$

The conjecture is proved by induction w.r.t. $\mathsf{replace}$ (or $\mathsf{apply\_subst\_var}$). In the base case ($\sigma = \mathsf{e}$), the premise reduces to $\mathsf{false}$. In the case $\sigma = \mathsf{var}(n, t)$, the induction conclusion can be proved using (94), (99), (164), (159), and (287). Finally, in the case $\sigma = \mathsf{var}(n, t)$ (with $m \neq n$), the induction conclusion can be reduced to the induction hypothesis using (94), (101), and (164).

## 6.23 Connection between $\mathsf{rewrite\_rule\_list}$ and $\mathsf{rewrite\_list}$ (pc)

The next conjecture says that $\mathsf{rewrite\_rule\_list}$ is a subset of $\mathsf{rewrite\_list}$ if the rule used is a member of the TRS.

$$\mathsf{in}(l, r, R) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k, l, r), \mathsf{rewrite\_list}(k, R)) \tag{290}$$

The conjecture can be proved by induction w.r.t. $\mathsf{in}$. In the case $\mathsf{first}(R) = l, \mathsf{second}(R) = r$, the induction conclusion is implied by the induction hypothesis and (99). In the other induction step case, the induction conclusion follows from the induction hypothesis and (101).

## 6.24 Connection between $\mathsf{rewrite\_rule}$ and $\mathsf{rewrite\_rule\_list}$ (pc)

The next theorem states that $\mathsf{rewrite\_rule}$ is a subset of $\mathsf{rewrite\_rule\_list}$.

$$\mathsf{member}(t, k) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_rule}(t, l, r), \mathsf{rewrite\_rule\_list}(k, l, r)) \tag{291}$$

The conjecture can be proved by induction w.r.t. $\mathsf{rewrite\_rule\_list}$. The base case is easy. In the step case where $k = \mathsf{add}(t, k')$, the conjecture follows from (55) and (99). In the other step case one needs the induction hypothesis, (101), and (96).

## 6.25 Stability of subseteq_list under rewrite_rule_list (pc)

The next theorem states that subsets are preserved under rewrite_rule_list.

$$\mathsf{subseteq\_list}(k_1, k_2) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k_1, l, r), \mathsf{rewrite\_rule\_list}(k_2, l, r)) \tag{292}$$

The conjecture can be proved by induction w.r.t. $\mathsf{rewrite\_rule\_list}(k_1, l, r)$, where in the step case one needs (103) and (291).

## 6.26 Stability of subseteq_list under rewrite_list (pc)

The next theorem states that subsets are preserved under rewrite_list.

$$\mathsf{subseteq\_list}(k_1, k_2) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k_1, R), \mathsf{rewrite\_list}(k_2, R)) \tag{293}$$

We prove the theorem by induction w.r.t. rewrite_list. The base case is trivial and the step case can be proved using (86) and (292).

## 6.27 Exchanging rewrite_rule_list and apply (pc)

The next conjecture says that one may exchange rewrite_rule_list and apply (yielding subsets).

$$\mathsf{subseteq\_list}(\mathsf{apply}(n, \mathsf{rewrite\_rule\_list}(k, l, r)), \mathsf{rewrite\_rule}(\mathsf{apply}(n, k), l, r)) \tag{294}$$

We prove the conjecture by induction w.r.t. apply (resp. w.r.t. rewrite_rule_list). The base case is easy and in the step case, the induction conclusion can be transformed into the hypothesis and (158), (103), (282).

## 6.28 Exchanging rewrite_list and apply (pc)

The next conjecture says that one may exchange rewrite_list and apply (yielding subsets).

$$\mathsf{subseteq\_list}(\mathsf{apply}(n, \mathsf{rewrite\_list}(k, R)), \mathsf{rewrite\_list}(\mathsf{apply}(n, k), R)) \tag{295}$$

The conjecture can be proved by induction w.r.t. rewrite_list. The base case is trivial and in the step case the induction conclusion can be reduced to the induction hypothesis, (158), (103), and (294).

## 6.29 subseteq_list of rewrite_rule_list with apply (pc)

The following theorem says that if no new terms can be generated from $\mathsf{apply}(n, k)$ with a rule, then this also holds for $k$.

$$\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(\mathsf{apply}(n, k), l, r), \mathsf{apply}(n, k)) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k, l, r), k). \tag{296}$$

By (96) and (294), this can be transformed into

$$\mathsf{subseteq\_list}(\mathsf{apply}(n, \mathsf{rewrite\_rule\_list}(k, l, r)), \mathsf{apply}(n, k)) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k, l, r), k).$$

This is a consequence of (152).

## 6.30 subseteq_list of rewrite_list with apply (pc)

The following theorem says that if no new terms can be generated from $\mathsf{apply}(n, k)$ with the TRS $R$, then this also holds for $k$.

$$\mathsf{subseteq\_list}(\mathsf{rewrite\_list}(\mathsf{apply}(n, k), R), \mathsf{apply}(n, k)) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k, R), k) \tag{297}$$

This can be proved by induction w.r.t. rewrite_list using (86), (96), (101), and (296).

## 6.31 Exchanging rewrite_rule_list and addtail (pc)

The next conjecture says that one may exchange rewrite_rule_list and addtail (yielding subsets).

$$\mathsf{subseteq\_list}(\mathsf{addtail}(\mathsf{rewrite\_rule\_list}(k,l,r),t),\mathsf{rewrite\_rule\_list}(\mathsf{addtail}(k,t),l,r)) \tag{298}$$

We prove the conjecture by induction w.r.t. addtail (resp. w.r.t. rewrite_rule_list). The base case is easy and in the step case, the induction conclusion can be transformed into the hypothesis and (156), (103), (283). In a similar way one can also prove

$$\mathsf{subseteq\_list}(\mathsf{addfirst}(t,\mathsf{rewrite\_rule\_list}(k,l,r)),\mathsf{rewrite\_rule\_list}(\mathsf{addfirst}(t,k),l,r)). \tag{299}$$

## 6.32 Exchanging rewrite_list and addtail (pc)

The next conjecture says that one may exchange rewrite_list and addtail (yielding subsets).

$$\mathsf{subseteq\_list}(\mathsf{addtail}(\mathsf{rewrite\_list}(k,R),t),\mathsf{rewrite\_list}(\mathsf{addtail}(k,t),R)) \tag{300}$$

The conjecture can be proved by induction w.r.t. rewrite_list. The base case is trivial and in the step case the induction conclusion can be reduced to the induction hypothesis, (156), (103), and (298).

## 6.33 subseteq_list of rewrite_rule_list with addtail (pc)

The following theorem says that if no new terms can be generated from $\mathsf{addtail}(k,t)$ with a rule, then this also holds for $k$.

$$\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(\mathsf{addtail}(k,t),l,r),\mathsf{addtail}(k,t)) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k,l,r),k) \tag{301}$$

By (96) and (298), this can be transformed into

$$\mathsf{subseteq\_list}(\mathsf{addtail}(\mathsf{rewrite\_rule\_list}(k,l,r),t),\mathsf{addtail}(k,t)) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k,l,r),k).$$

This is a consequence of (153). In a similar way one can also prove

$$\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(\mathsf{addfirst}(t,k),l,r),\mathsf{addfirst}(t,k)) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k,l,r),k). \tag{302}$$

## 6.34 subseteq_list of rewrite_list with addtail (pc)

The following theorem says that if no new terms can be generated from $\mathsf{addtail}(k,t)$ with the TRS $R$, then this also holds for $k$.

$$\mathsf{subseteq\_list}(\mathsf{rewrite\_list}(\mathsf{addtail}(k,t),R),\mathsf{addtail}(k,t)) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k,R),k) \tag{303}$$

This can be proved by induction w.r.t. rewrite_list using (86), (96), (99), (101), and (301).

## 6.35 Exchanging rewrite_rule_list and tail_list (pc)

The next conjecture says that one may exchange rewrite_rule_list and tail_list (yielding subsets).

$$\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(\mathsf{tail\_list}(k),R),\mathsf{tail\_list}(\mathsf{rewrite\_list}(k,R))) \tag{304}$$

We prove the conjecture by induction w.r.t. tail_list (resp. w.r.t. rewrite_rule_list). The base case is easy and in the step case, the induction conclusion can be transformed into the hypothesis and (141), (103), (284). In a similar way one can also prove

$$\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(\mathsf{first\_list}(k),R),\mathsf{first\_list}(\mathsf{rewrite\_list}(k,R))). \tag{305}$$

## 6.36 Exchanging rewrite_list and tail_list (pc)

The next conjecture says that one may exchange rewrite_list and tail_list (yielding subsets).

$$\mathsf{subseteq\_list}(\mathsf{rewrite\_list}(\mathsf{tail\_list}(k), R), \mathsf{tail\_list}(\mathsf{rewrite\_list}(k, R))) \tag{306}$$

The conjecture can be proved by induction w.r.t. rewrite_list. The base case is trivial and in the step case the induction conclusion can be reduced to the induction hypothesis, (141), (103), and (304). In a similar way one can also prove

$$\mathsf{subseteq\_list}(\mathsf{rewrite\_list}(\mathsf{first\_list}(k), R), \mathsf{first\_list}(\mathsf{rewrite\_list}(k, R))). \tag{307}$$

## 6.37 subseteq_list of rewrite_rule_list with tail_list (pc)

The following theorem says that if no new terms can be generated from $k$ with one rule, then this also holds for tail_list.

$$\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k, l, r), k) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(\mathsf{tail\_list}(k), l, r), \mathsf{tail\_list}(k)) \tag{308}$$

By (96) and (304), this can be transformed into

$$\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k, l, r), k) \Rightarrow \mathsf{subseteq\_list}(\mathsf{tail\_list}(\mathsf{rewrite\_rule\_list}(k, l, r)), \mathsf{tail\_list}(k)).$$

This is a consequence of (145).

## 6.38 subseteq_list of rewrite_list with first_list and tail_list (pc)

The following theorem says that if no new terms can be generated from $k$ with a TRS, then this also holds for first_list and tail_list.

$$\begin{aligned} \mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k, R), k) \Rightarrow \quad & \mathsf{subseteq\_list}(\mathsf{rewrite\_list}(\mathsf{first\_list}(k), R), \mathsf{first\_list}(k)) \\ \wedge \quad & \mathsf{subseteq\_list}(\mathsf{rewrite\_list}(\mathsf{tail\_list}(k), R), \mathsf{tail\_list}(k)) \end{aligned} \tag{309}$$

We will only show the proof of

$$\mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k, R), k) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_list}(\mathsf{tail\_list}(k), R), \mathsf{tail\_list}(k))$$

(the proof of the corresponding statement for first_list works in an analogous way). The above statement can be proved by induction w.r.t. rewrite_list using (86), (96), (101), and (308).

## 6.39 subseteq_list of rewrite_rule_list with tail_list (Version 1) (pc)

The following theorem says that the elements of $\mathsf{tail\_list}(\mathsf{rewrite\_rule\_list}(k, l, r))$ only come from $\mathsf{tail\_list}(k)$ and $\mathsf{rewrite\_rule\_list}(\mathsf{tail\_list}(k), l, r)$.

$$\begin{aligned} & \mathsf{member}(t, \mathsf{tail\_list}(\mathsf{rewrite\_rule\_list}(k, l, r))) \Rightarrow \\ & \qquad \mathsf{member}(t, \mathsf{tail\_list}(k)) \vee \mathsf{member}(t, \mathsf{rewrite\_rule\_list}(\mathsf{tail\_list}(k), l, r)) \end{aligned} \tag{310}$$

We prove the conjecture by induction w.r.t. tail_list (resp. w.r.t. rewrite_rule_list). The base case is easy and in the step case, the induction conclusion can be transformed into the hypothesis and (141), (113), (94), (99), (101), and (285).

## 6.40 $\mathsf{subseteq\_list}$ of $\mathsf{rewrite\_list}$ with $\mathsf{tail\_list}$ (Version 2) (pc)

The following theorem says that the elements of $\mathsf{tail\_list}(\mathsf{rewrite\_list}(k, R))$ only come from $\mathsf{tail\_list}(k)$ and $\mathsf{rewrite\_list}(\mathsf{tail\_list}(k), R)$.

$$\mathsf{member}(t, \mathsf{tail\_list}(\mathsf{rewrite\_list}(k, R))) \Rightarrow \mathsf{member}(t, \mathsf{tail\_list}(k)) \vee \mathsf{member}(t, \mathsf{rewrite\_list}(\mathsf{tail\_list}(k), R)) \quad (311)$$

The theorem can be proved by induction w.r.t. $\mathsf{rewrite\_list}$ using (141), (113), (94), (99), (101), and (310). In an analogous way one can also prove

$$\mathsf{member}(t, \mathsf{first\_list}(\mathsf{rewrite\_list}(k, R))) \Rightarrow \mathsf{member}(t, \mathsf{first\_list}(k)) \vee \mathsf{member}(t, \mathsf{rewrite\_list}(\mathsf{first\_list}(k), R)). \quad (312)$$

## 6.41 Instantiated Left-Hand Sides are Replaced by Instantiated Right-Hand sides by $\mathsf{rewrite\_rule\_list}$ (pc)

This conjecture says that if an instantiated left-hand side is a member of $k$, then the corresponding instantiated right-hand side is a member of $\mathsf{rewrite\_rule\_list}(k, l, r)$.

$$\begin{aligned}&\mathsf{length}(l) = \mathsf{s}(0) \wedge \mathsf{length}(r) = \mathsf{s}(0) \wedge \mathsf{subseteq}(\mathsf{vars}(r), \mathsf{vars}(l)) \wedge \\ &\mathsf{first\_is\_func}(l) \wedge \mathsf{member}(\mathsf{apply\_subst}(\sigma, l), k) \Rightarrow \\ &\qquad \mathsf{member}(\mathsf{apply\_subst}(\sigma, r), \mathsf{rewrite\_rule\_list}(k, l, r))\end{aligned} \quad (313)$$

The conjecture follows from (288), (94), (287), and (277).

## 6.42 $\mathsf{disjoint\_list}$ of $\mathsf{rewrite\_rule\_list}$ and $\mathsf{apply\_subst\_list}$ (pc)

The following theorem says that if none of the terms in $\mathsf{apply\_subst\_list}(k', r)$ can be reached from $k$ with the rule $l \to r$, then $k$ and $\mathsf{apply\_subst\_list}(k', l)$ are disjoint.

$$\begin{aligned}&\mathsf{length}(l) = \mathsf{s}(0) \wedge \mathsf{length}(r) = \mathsf{s}(0) \wedge \mathsf{subseteq}(\mathsf{vars}(r), \mathsf{vars}(l)) \wedge \\ &\mathsf{first\_is\_func}(l) \wedge \mathsf{disjoint\_list}(\mathsf{rewrite\_rule\_list}(k, l, r), \mathsf{apply\_subst\_list}(k', r)) \Rightarrow \\ &\qquad \mathsf{disjoint\_list}(k, \mathsf{apply\_subst\_list}(k', l))\end{aligned} \quad (314)$$

Using (124), we can transform the conjecture into

$$\mathsf{disjoint\_list}(\mathsf{apply\_subst\_list}(k', r), \mathsf{rewrite\_rule\_list}(k, l, r)) \Rightarrow \mathsf{disjoint\_list}(\mathsf{apply\_subst\_list}(k', l), k).$$

Now we perform an induction w.r.t. $\mathsf{apply\_subst\_list}$. In the case $\mathsf{member}(\mathsf{apply\_subst}(\sigma, l), k)$ the conjecture follows from (313). Otherwise the induction conclusion can be transformed into the induction hypothesis.

## 6.43 $\mathsf{disjoint\_list}$ of $\mathsf{rewrite\_list}$ and $\mathsf{apply\_subst\_list}$ (pc)

The following theorem says that if none of the terms in $\mathsf{apply\_subst\_list}(k', r)$ can be reached from $k$ and if $l \to r$ is a rule of $R$, then $k$ and $\mathsf{apply\_subst\_list}(k', l)$ are disjoint.

$$\mathsf{trs}(R) \wedge \mathsf{disjoint\_list}(\mathsf{rewrite\_list}(k, R), \mathsf{apply\_subst\_list}(k', r)) \wedge \mathsf{in}(l, r, R) \Rightarrow \mathsf{disjoint\_list}(k, \mathsf{apply\_subst\_list}(k', l)) \quad (315)$$

This can be proved by induction w.r.t. $\mathsf{in}$. If $l = \mathsf{first}(R)$ and $r = \mathsf{second}(R)$, then the conjecture follows from (122), (92), (99), (169), (170), and (314). In the other step case, the induction conclusion can be reduced to the induction hypothesis, (122), (92), (101), (169), (170), and (314).

### 6.44 Exchanging $\mathsf{apply\_subst\_tll}$ and $\mathsf{rewrite\_list}$ (pc)

The next theorem states the connection one obtains when exchanging $\mathsf{apply\_subst\_tll}$ and $\mathsf{rewrite\_list}$

$$\mathsf{subseteq\_list}(\mathsf{apply\_subst\_tll}(\sigma, \mathsf{rewrite\_list}(k, R)), \mathsf{rewrite\_list}(\mathsf{apply\_subst\_tll}(\sigma, k), R)) \tag{316}$$

Using Rule 1″, we apply an induction w.r.t. $\mathsf{rewrite\_list}$. The case $R = \mathsf{e}$ is trivial and in the case $R = \mathsf{func}(n, s, t)$ we obtain the induction conclusion

$$\begin{aligned}&\mathsf{subseteq\_list}(\mathsf{apply\_subst\_tll}(\sigma, \mathsf{append}(\mathsf{rewrite\_rule\_list}(k, \mathsf{func}(n, s, \mathsf{e}), \mathsf{first}(t)), \mathsf{rewrite\_list}(k, \mathsf{tail}(t)))),\\ &\qquad \mathsf{append}(\mathsf{rewrite\_rule\_list}(\mathsf{apply\_subst\_tll}(\sigma, k), \mathsf{func}(n, s, \mathsf{e}), \mathsf{first}(t)),\\ &\qquad\qquad \mathsf{rewrite\_list}(\mathsf{apply\_subst\_tll}(\sigma, k), \mathsf{tail}(t)))).\end{aligned}$$

Using (223), (224), (26), and (99), (101), (96), and (86), this can be transformed into the induction hypothesis and (after generalization) into

$$\mathsf{subseteq\_list}(\mathsf{apply\_subst\_tll}(\sigma, \mathsf{rewrite\_rule\_list}(k, l, r)), \mathsf{rewrite\_rule\_list}(\mathsf{apply\_subst\_tll}(\sigma, k), l, r)).$$

This conjecture is now proved by induction w.r.t. $\mathsf{rewrite\_rule\_list}$. The base case ($k = \mathsf{empty}$) is again trivial. In the step case one proceeds in an analogous way as above. In this way this conjecture is transformed into

$$\mathsf{subseteq\_list}(\mathsf{apply\_subst\_tll}(\sigma, \mathsf{rewrite\_rule}(t, l, r)), \mathsf{rewrite\_rule}(\mathsf{apply\_subst}(\sigma, t), l, r)).$$

Finally, this conjecture is proved by induction w.r.t. $\mathsf{rewrite\_rule}$. The base case is again trivial. In the case $t = \mathsf{var}(m, q)$, the induction conclusion can be transformed (using (221), (222)) into the following formula

$$\begin{aligned}&\mathsf{subseteq\_list}(\mathsf{append\_list}(p, \mathsf{apply\_subst\_tll}(\sigma, \mathsf{rewrite\_rule}(q, l, r))),\\ &\qquad \mathsf{rewrite\_rule}(\mathsf{addterm}(p, \mathsf{apply\_subst}(\sigma, q)), l, r)).\end{aligned}$$

By induction (resp. case analysis) w.r.t. $\mathsf{addterm}$ and (101) in both cases the conjecture can be reduced to

$$\begin{aligned}&\mathsf{subseteq\_list}(\mathsf{append\_list}(p, \mathsf{apply\_subst\_tll}(\sigma, \mathsf{rewrite\_rule}(q, l, r))),\\ &\qquad \mathsf{append\_list}(p, \mathsf{rewrite\_rule}(\mathsf{apply\_subst}(\sigma, q), l, r))).\end{aligned}$$

Now the conjecture follows from the induction hypothesis and (165).
In a similar way one can also prove the conjecture

$$\mathsf{def}(\mathsf{rewrite\_list}(\mathsf{apply\_subst\_tll}(\sigma, k), R)) \wedge \mathsf{trs}(R) \Rightarrow \mathsf{def}(\mathsf{apply\_subst\_tll}(\sigma, \mathsf{rewrite\_list}(k, R))) \tag{317}$$

because $\mathsf{rewrite\_list}$ is total if $R$ is a TRS (268).

### 6.45 Monotonicity of $\mathsf{rewrite\_list}$ (pc)

The next theorem states that the list of all terms obtained by rewriting using rules of $R$ is a subset of those terms obtained by rewriting using the whole TRS $R$.

$$\begin{aligned}&\mathsf{in}(l, r, R) \wedge \mathsf{in}(l', r', R) \Rightarrow\\ &\quad \mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k, \mathsf{addterm}(l', \mathsf{addterm}(r', \mathsf{addterm}(l, \mathsf{addterm}(r, \mathsf{e}))))), \mathsf{rewrite\_list}(k, R))\end{aligned} \tag{318}$$

By symbolic evaluation and Rule 4″ this can be transformed into (86), (28), and

$$\mathsf{in}(l, r, R) \Rightarrow \mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k, l, r), \mathsf{rewrite\_list}(k, R)).$$

This theorem can be proved by induction w.r.t. $\mathsf{rewrite\_list}$. The base case ($R = \mathsf{e}$) is trivial. In the step case we have $R = \mathsf{func}(n, s, t)$. Evaluation of $\mathsf{in}(l, r, \mathsf{func}(n, s, t)$ suggests the following case analysis. If $l = \mathsf{func}(n, s, \mathsf{e})$ and $r = \mathsf{first}(t)$, then the conjecture is a consequence of (99). Otherwise the induction conclusion can be transformed into the induction hypothesis.

### 6.46 Stability of $\mathsf{rewrites\_list*\_exists}$ under Subsets

This theorem says that if a term in $k_1$ can be reached from $k$, then this also holds for any superlist $k_2$.

$$\mathsf{subseteq\_list}(k_1, k_2) \wedge \mathsf{rewrites\_list*\_exists}(k, k_1, R) \Rightarrow \mathsf{rewrites\_list*\_exists}(k, k_2, R) \tag{319}$$

It can easily be proved by induction w.r.t. $\mathsf{rewrites\_list*\_exists}(k, k_1, R)$ using (122). (This proof was also sketched in [9].) Note that in this way we proved inductive truth of this conjecture (instead of just *partial* truth). This stronger statement is needed in the proof of subsequent theorems. (For example, in the proof of (396) it is needed to ensure that the truth of $\mathsf{rewrites\_list*\_exists}(\ldots, k, \ldots)$ implies definedness of $\mathsf{rewrites\_list*\_exists}(\ldots, \mathsf{append}(k, \mathsf{rewrite\_list}(k, R)), \ldots)$.)

### 6.47 $\mathsf{rewrites\_rule\_list*\_exists}$ implies $\mathsf{rewrites\_list*\_exists}$

The next theorem states that if an element of $k_1$ reduces to an element of $k$ with a rule from $R$, then this also works with the whole trs $R$.

$$\mathsf{rewrites\_rule\_list*\_exists}(k_1, k, l, r) \wedge \mathsf{in}(l, r, R) \wedge \mathsf{subseteq\_list}(k_1, k_2) \Rightarrow \mathsf{rewrites\_list*\_exists}(k_2, k, R) \tag{320}$$

We prove the conjecture by induction w.r.t. $\mathsf{rewrites\_list*\_exists}(k_1, k, l, r)$.

In the case $\mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k_2, R), k_2)$, we first add the premise $\mathsf{ge}(\mathsf{setdiff}(k_2, k_1), \mathsf{setdiff}(k_2, k_1))$ which we then generalize to $\mathsf{ge}(n, \mathsf{setdiff}(k_2, k_1))$. Now we perform another induction w.r.t. $n$ and $k_1$ (this is a structural induction about $n$, where $k_1$ is changed as in the algorithm $\mathsf{rewrites\_rule\_list*\_exists}$, cf. the extension of Rule $1''$ and $2''$ by allowing arbitrary instantiations in induction hypotheses [9]). If $n = \mathsf{0}$, then by (136), (290), (292), (96) we know that $\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k_1, l, r), k_1)$ also holds, i.e. in this case the conjecture is trivial. If $n = \mathsf{s}(m)$, in the only interesting case for $k_1$, we obtain the following induction conclusion (of the inner $n$-induction).

$$\begin{aligned}&\mathsf{rewrites\_rule\_list*\_exists}(\mathsf{append}(k_1, \mathsf{rewrite\_rule\_list}(k_1, l, r)), k, l, r) \wedge \mathsf{in}(l, r, R) \wedge \\ &\mathsf{subseteq\_list}(k_1, k_2) \wedge \mathsf{ge}(\mathsf{s}(m), \mathsf{setdiff}(k_2, k_1)) \\ &\quad \Rightarrow \mathsf{false}.\end{aligned}$$

By (99), (292), and (138) this can be transformed into the induction hypothesis.

Finally we prove the remaining case of the outer $\mathsf{rewrites\_list*\_exists}$-induction. Here, the induction conclusion follows from the induction hypothesis, (290), and (292).

### 6.48 Stability of $\mathsf{rewrites\_rule\_list*}$ under Subsets

The following theorem states that if $s$ can be reached from a list $k_1$, then this also holds for every superlist $k_2$.

$$\mathsf{subseteq\_list}(k_1, k_2) \wedge \mathsf{rewrites\_rule\_list*}(k_1, s, l, r) \Rightarrow \mathsf{rewrites\_rule\_list*}(k_2, s, l, r) \tag{321}$$

The conjecture can be proved by induction w.r.t. $\mathsf{rewrites\_rule\_list*}(k_1, s, l, r)$. If $\mathsf{member}(s, k_1)$ then the conjecture follows from (94).

In the case $\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k_2, l, r), k_2)$, we proceed in a similar way as in the proof of (320). Hence, we add the premise $\mathsf{ge}(\mathsf{setdiff}(k_2, k_1), \mathsf{setdiff}(k_2, k_1))$ which we then generalize to $\mathsf{ge}(n, \mathsf{setdiff}(k_2, k_1))$. Now we perform another induction w.r.t. $n$ and $k_1$. If $n = \mathsf{0}$, then (136), (292), and (96) imply $\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k_1, l, r), k_1)$, i.e. in this case the conjecture is trivial. If $n = \mathsf{s}(m)$, in the only interesting case for $k_1$, we obtain the following induction conclusion (of the inner $n$-induction).

$$\begin{aligned}&\mathsf{subseteq\_list}(k_1, k_2) \wedge \mathsf{ge}(\mathsf{s}(m), \mathsf{setdiff}(k_2, k_1)) \wedge \\ &\mathsf{rewrites\_rule\_list*}(\mathsf{append}(k_1, \mathsf{rewrite\_rule\_list}(k_1, l, r)), s, l, r) \Rightarrow \\ &\quad \mathsf{rewrites\_rule\_list*}(\mathsf{append}(k_2, \mathsf{rewrite\_rule\_list}(k_2, l, r)), s, l, r)\end{aligned}$$

By (99), (292), and (138) this can be transformed into the induction hypothesis.

Finally we prove the remaining case of the outer $\mathsf{rewrites\_rule\_list*}$-induction. Here, the induction conclusion follows from the induction hypothesis, (103), and (292).

## 6.49 Stability of $\mathsf{rewrites\_list*}$ under Subsets

The following theorem says that if $t$ can be reached from an element of $k_1$ in the TRS $R$, then this also works with any superset $k_2$.

$$\mathsf{rewrites\_list*}(k_1, t, R) \wedge \mathsf{subseteq\_list}(k_1, k_2) \Rightarrow \mathsf{rewrites\_list*}(k_2, t, R) \tag{322}$$

The conjecture can be proved by induction w.r.t. $\mathsf{rewrites\_list*}(k_1, t, R)$ (where $k_2$ is also changed appropriately, i.e. we again use a merged induction relation whose well-foundedness is guaranteed by $\mathsf{def}(\mathsf{rewrites\_list*}(k_1, t, R))$). If $\mathsf{member}(t, k_1)$ then the conjecture follows from (94).

In the case $\mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k_2, R), k_2)$, we proceed in a similar way as in the proofs of (320) and (321). Hence, we add the premise $\mathsf{ge}(\mathsf{setdiff}(k_2, k_1), \mathsf{setdiff}(k_2, k_1))$ which we then generalize to $\mathsf{ge}(n, \mathsf{setdiff}(k_2, k_1))$. Now we perform another induction w.r.t. $n$. If $n = \mathsf{0}$, then (136), (293), and (96) imply $\mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k_1, R), k_1)$, i.e. in this case the conjecture is trivial. If $n = \mathsf{s}(m)$, in the only interesting case for $k_1$, we obtain the following induction conclusion (of the inner $n$-induction).

$$\begin{aligned}&\mathsf{subseteq\_list}(k_1, k_2) \wedge \mathsf{ge}(\mathsf{s}(m), \mathsf{setdiff}(k_2, k_1)) \wedge \mathsf{rewrites\_list*}(\mathsf{append}(k_1, \mathsf{rewrite\_list}(k_1, R)), t, R) \Rightarrow\\ &\quad \mathsf{rewrite\_list}(\mathsf{append}(k_2, \mathsf{rewrite\_list}(k_2, R)), t, R)\end{aligned}$$

By (99), (293), and (138) this can be transformed into the induction hypothesis.

Finally we prove the remaining case of the outer $\mathsf{rewrites\_list*}$-induction. Here, the induction conclusion follows from the induction hypothesis, (103), and (293).

## 6.50 Stability of $\mathsf{rewrites*}$ and $\mathsf{rewrites\_list*}$ under Substitutions

The next theorem is the stability of reductions under substitutions.

$$\begin{aligned}&\mathsf{def}(\mathsf{apply\_subst}(\sigma, s)) \wedge \mathsf{def}(\mathsf{apply\_subst}(\sigma, t)) \wedge \mathsf{rewrites*}(s, t, R) \Rightarrow\\ &\qquad \mathsf{rewrites*}(\mathsf{apply\_subst}(\sigma, s), \mathsf{apply\_subst}(\sigma, t), R)\end{aligned} \tag{323}$$

The theorem can be generalized to

$$\begin{aligned}&\mathsf{def}(\mathsf{apply\_subst\_tll}(\sigma, k)) \wedge \mathsf{def}(\mathsf{apply\_subst}(\sigma, t)) \wedge \mathsf{rewrites\_list*}(k, t, R) \Rightarrow\\ &\qquad \mathsf{rewrites\_list*}(\mathsf{apply\_subst\_tll}(\sigma, k), \mathsf{apply\_subst}(\sigma, t), R).\end{aligned} \tag{324}$$

It can be proved by induction w.r.t. $\mathsf{rewrites\_list*}(k, t, R)$. If $\mathsf{member}(t, k)$, then the conjecture follows from (225). If $\mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k, R), k)$, then the conjecture is trivial. Otherwise the induction conclusion can be evaluated to

$$\begin{aligned}&\mathsf{def}(\ldots) \wedge \mathsf{def}(\ldots) \wedge \mathsf{rewrites\_list*}(\mathsf{append}(k, \mathsf{rewrite\_list}(k, R)), t, R) \Rightarrow\\ &\quad \mathsf{rewrites\_list*}(\mathsf{apply\_subst\_tll}(\sigma, k), \mathsf{apply\_subst}(\sigma, t), R).\end{aligned}$$

In the case $\mathsf{subseteq\_list}(\mathsf{rewrite\_list}(\mathsf{apply\_subst\_tll}(\sigma, k), R), \mathsf{apply\_subst\_tll}(\sigma, k))$, we have

$$\mathsf{subseteq\_list}(\mathsf{apply\_subst\_tll}(\sigma, \mathsf{append}(k, \mathsf{rewrite\_list}(k, R))), \mathsf{apply\_subst\_tll}(\sigma, k))$$

by (223), (103), (96), (316), and (86). Hence, by (322) the induction conclusion can be transformed into the induction hypothesis. Otherwise the induction conclusion can be evaluated to

$$\begin{aligned}&\mathsf{def}(\ldots) \wedge \mathsf{def}(\ldots) \wedge \mathsf{rewrites\_list*}(\mathsf{append}(k, \mathsf{rewrite\_list}(k, R))t, R) \Rightarrow\\ &\quad \mathsf{rewrites\_list*}(\mathsf{append}(\mathsf{apply\_subst\_tll}(\sigma, k), \mathsf{rewrite\_rule}(\mathsf{apply\_subst\_tll}(\sigma, k), R)), \mathsf{apply\_subst}(\sigma, t), R).\end{aligned}$$

This can be transformed into the induction hypothesis and (223), (103), (96), (316), (322).

### 6.51 Splitting Appended Lists when using $\mathsf{rewrites\_rule\_list*}$

The next conjecture states a kind of converse to the above conjecture.

$$\mathsf{rewrites\_rule\_list*}(\mathsf{append}(k_1, k_2), t, l, r) \Rightarrow \mathsf{rewrites\_rule\_list*}(k_1, t, l, r) \vee \mathsf{rewrites\_rule\_list*}(k_2, t, l, r). \quad (325)$$

We prove the conjecture by induction w.r.t. $\mathsf{rewrites\_rule\_list*}$. (Formally, we replace $\mathsf{append}(k_1, k_2)$ by a new variable $k$ and add the premise $k = \mathsf{append}(k_1, k_2)$. Now $k_1$ and $k_2$ are also changed appropriately by the induction relation, i.e. we use the merged induction relations of all three calls of $\mathsf{rewrites\_rule\_list*}$.) The cases $k_1 = \mathsf{empty}$ or $k_2 = \mathsf{empty}$ can easily be proved. In the case $\mathsf{member}(t, \mathsf{append}(k_1, k_2))$ the conjecture follows from (113). If

$$\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(\mathsf{append}(k_1, k_2), l, r), \mathsf{append}(k_1, k_2)),$$

then the proof is trivial. Otherwise, the induction conclusion is

$$\begin{aligned}&\mathsf{rewrites\_rule\_list*}(\mathsf{append}(\mathsf{append}(k_1, k_2), \mathsf{rewrite\_rule\_list}(\mathsf{append}(k_1, k_2), l, r))t, l, r) \Rightarrow\\ &\quad \mathsf{rewrites\_rule\_list*}(k_1, t, l, r) \vee \mathsf{rewrites\_rule\_list*}(k_2, t, l, r).\end{aligned}$$

Due to (321) and (103), this can be transformed into

$$\begin{aligned}&\mathsf{rewrites\_rule\_list*}(\mathsf{append}(\mathsf{append}(k_1, k_2), \mathsf{rewrite\_rule\_list}(\mathsf{append}(k_1, k_2), l, r))t, l, r) \Rightarrow\\ &\quad \mathsf{rewrites\_rule\_list*}(\mathsf{append}(k_1, \mathsf{rewrite\_rule\_list}(k_1, l, r))t, l, r) \vee\\ &\quad \mathsf{rewrites\_rule\_list*}(\mathsf{append}(k_1, \mathsf{rewrite\_rule\_list}(k_2, l, r))t, l, r)\end{aligned}$$

even if $\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k_1), k_1)$ or $\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k_2), k_2)$ holds. Using (286), (72), and

$$\mathsf{rewrites\_rule\_list*}(\mathsf{append}(k, k'), t, l, r) \Rightarrow \mathsf{rewrites\_rule\_list*}(\mathsf{append}(k', k), t, l, r)$$

(which holds due to (321) and (115)), this can now be reduced further to the induction hypothesis.

### 6.52 Connection between $\mathsf{rewrites\_rule\_list*}$ and $\mathsf{addfirst}$

The next conjecture states that one may add a new first element to the first two arguments of $\mathsf{rewrites\_rule\_list*}$.

$$\mathsf{rewrites\_rule\_list*}(k, \mathsf{tail}(t), l, r) \Rightarrow \mathsf{rewrites\_rule\_list*}(\mathsf{addfirst}(\mathsf{first}(t), k), t, l, r) \quad (326)$$

The conjecture is proved by induction w.r.t. $\mathsf{rewrites\_rule\_list*}(k, \mathsf{tail}(t), l, r)$. If $k = \mathsf{empty}$ then the proof is trivial. If $\mathsf{member}(\mathsf{tail}(t), k)$, then the proof is done using (160). If $\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(\mathsf{addfirst}(\mathsf{first}(t), k), l, r), \mathsf{addfirst}(\mathsf{first}(t), k))$, then the conjecture follows from (302). Otherwise, the induction conclusion can be transformed into the induction hypothesis, (321), and

$$\begin{aligned}&\mathsf{subseteq\_list}(\mathsf{addfirst}(\mathsf{first}(t), \mathsf{append}(k, \mathsf{rewrite\_rule\_list}(k, l, r))),\\ &\qquad \mathsf{append}(\mathsf{addfirst}(\mathsf{first}(t), k), \mathsf{rewrite\_rule\_list}(\mathsf{addfirst}(\mathsf{first}(t), k), l, r))).\end{aligned}$$

This conjecture can be proved using (157), (299), and (103).

### 6.53 Connection between $\mathsf{rewrites\_rule\_list*}$ and $\mathsf{addtail}$

The next conjecture states that $\mathsf{rewrites\_rule\_list*}$ applied to $\mathsf{addtail}$ may be split.

$$\mathsf{rewrites\_rule\_list*}(k, \mathsf{first}(t), l, r) \wedge \mathsf{rewrites\_rule}(s, \mathsf{tail}(t), l, r) \Rightarrow \mathsf{rewrites\_rule\_list*}(\mathsf{addtail}(k, s), t, l, r) \quad (327)$$

The conjecture is proved by induction w.r.t. $\mathsf{rewrites\_rule\_list*}(k, \mathsf{first}(t), l, r)$. If $k = \mathsf{empty}$ then the proof is straightforward. If $\mathsf{member}(\mathsf{first}(t), k)$, then the proof is done using (159), (321), (326). If $\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(\mathsf{addtail}(k, s), l, r), \mathsf{addtail}(k, s))$, then the conjecture follows from (301). Otherwise, the induction conclusion can be transformed into the induction hypothesis, (321), and

$$\begin{aligned}&\mathsf{subseteq\_list}(\mathsf{addtail}(\mathsf{append}(k, \mathsf{rewrite\_rule\_list}(k, l, r)), s),\\ &\qquad \mathsf{append}(\mathsf{addtail}(k, s), \mathsf{rewrite\_rule\_list}(\mathsf{addtail}(k, s), l, r))).\end{aligned}$$

This conjecture can be proved using (156), (298), and (103).

## 6.54 Decomposing $\mathsf{rewrites\_rule\_list*}$ with $\mathsf{all\_combinations}$

The following theorem says that $\mathsf{rewrites\_rule\_list*}$ can be split using $\mathsf{first}$ and $\mathsf{tail}$.

$$\begin{aligned}\mathsf{rewrites\_rule\_list*}(k_1, \mathsf{first}(t), l, r) \wedge \mathsf{rewrites\_rule\_list*}(k_2, \mathsf{tail}(t), l, r) \wedge \mathsf{def}(\mathsf{all\_combinations}(k_1, k_2)) \Rightarrow \\ \mathsf{rewrites\_rule\_list*}(\mathsf{all\_combinations}(k_1, k_2), t, l, r)\end{aligned} \tag{328}$$

The theorem is proved by induction w.r.t. $\mathsf{all\_combinations}$. The base case is trivial and in the case $k_2 = \mathsf{add}(s, k')$ we obtain the induction conclusion

$$\begin{aligned}&\mathsf{rewrites\_rule\_list*}(k_1, \mathsf{first}(t), l, r) \wedge \mathsf{rewrites\_rule\_list*}(\mathsf{add}(s, k'), \mathsf{tail}(t), l, r) \wedge \mathsf{def}(\ldots) \Rightarrow \\ &\mathsf{rewrites\_rule\_list*}(\mathsf{append}(\mathsf{addtail}(k_1, s), \mathsf{all\_combinations}(k_1, k')), t, l, r).\end{aligned}$$

Using (327), (325), and (321), the induction conclusion can be transformed into the induction hypothesis.

## 6.55 Decomposing $\mathsf{rewrites\_rule*}$ with $\mathsf{first}$ and $\mathsf{tail}$

The following theorem says that $\mathsf{rewrites\_rule*}$ can be split using $\mathsf{first}$ and $\mathsf{tail}$.

$$\mathsf{rewrites\_rule*}(\mathsf{first}(s), \mathsf{first}(t), l, r) \wedge \mathsf{rewrites\_rule*}(\mathsf{tail}(s), \mathsf{tail}(t), l, r) \Rightarrow \mathsf{rewrites\_rule*}(s, t, l, r) \tag{329}$$

The theorem can be evaluated and transformed into (328).

## 6.56 Applying $\mathsf{appendterm}$ in the Arguments of $\mathsf{rewrites\_rule*}$

The following conjecture says that one may append a term without changing the result of $\mathsf{rewrites\_rule*}$.

$$\mathsf{rewrites\_rule*}(t, q, l, r) \wedge \mathsf{def}(s) \Rightarrow \mathsf{rewrites\_rule*}(\mathsf{appendterm}(s, t), \mathsf{appendterm}(s, q), l, r) \tag{330}$$

The conjecture can be proved by induction w.r.t. $\mathsf{appendterm}$. In both step cases, the induction conclusion can be transformed into the induction hypothesis and (329).

## 6.57 Decomposing $\mathsf{rewrites\_rule\_list*}$ with $\mathsf{apply}$

The next conjecture says that if an element of $k$ rewrites to $u$, then an element of $\mathsf{apply}(n, k)$ rewrites to $\mathsf{func}(n, u, \mathsf{e})$.

$$\mathsf{rewrites\_rule\_list*}(k, u, l, r) \wedge \mathsf{def}(n) \Rightarrow \mathsf{rewrites\_rule\_list*}(\mathsf{apply}(n, k), \mathsf{func}(n, u, \mathsf{e}), l, r). \tag{331}$$

The conjecture is proved by induction w.r.t. $\mathsf{rewrites\_rule\_list*}(k, u, l, r)$. The case $k = \mathsf{empty}$ is trivial. If $\mathsf{member}(u, k)$, then the conjecture follows from (161). If $\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(\mathsf{apply}(n, k), l, r), \mathsf{apply}(n, k))$, then the conjecture can be proved using (296). Finally, in the remaining case, the induction conclusion can be transformed into the induction hypothesis, (158), (294), and (321).

## 6.58 Decomposing $\mathsf{rewrites\_rule*}$ with Contexts

The following theorem says that $\mathsf{rewrites\_rule*}$ can be split using the function context.

$$\mathsf{rewrites\_rule*}(q, t, l, r) \wedge \mathsf{rewrites\_rule*}(s, u, l, r) \wedge \mathsf{def}(n) \Rightarrow \mathsf{rewrites\_rule*}(\mathsf{func}(n, s, q), \mathsf{func}(n, u, t), l, r). \tag{332}$$

The theorem is a consequence of (329) and (331).

## 6.59 Connection between $\mathsf{rewrite^*\_all}$ and $\mathsf{append\_list}$

The next conjecture says something similar about $\mathsf{rewrite^*\_all}$ and $\mathsf{append\_list}$.

$$\mathsf{rewrite^*\_all}(t,k,l,r) \wedge \mathsf{def}(s) \Rightarrow \mathsf{rewrite^*\_all}(\mathsf{appendterm}(s,t), \mathsf{append\_list}(s,k), l, r) \tag{333}$$

The conjecture is proved by induction w.r.t. $\mathsf{append\_list}$. If $k = \mathsf{empty}$, then the conjecture is trivially proved. Otherwise (if $k = \mathsf{add}(q,k')$), the induction conclusion is

$$\begin{aligned}&\mathsf{rewrites\_rule^*}(t,q,l,r) \wedge \mathsf{rewrite^*\_all}(t,k',l,r) \Rightarrow\\ &\quad \mathsf{rewrites\_rule^*}(\mathsf{appendterm}(s,t), \mathsf{appendterm}(s,q), l, r) \wedge\\ &\quad \mathsf{rewrite^*\_all}(\mathsf{appendterm}(s,t), \mathsf{append\_list}(s,k'), l, r).\end{aligned}$$

This is a consequence of (330) and the induction hypothesis.

## 6.60 Stability of $\mathsf{rewrites\_list^*\_all}$ under Subsets

The next conjecture says that if each element of $k$ can be reached from a list $k_1$, then this also holds for every superlist $k_2$.

$$\mathsf{subseteq\_list}(k_1,k_2) \wedge \mathsf{rewrites\_list^*\_all}(k_1,k,l,r) \Rightarrow \mathsf{rewrites\_list^*\_all}(k_2,k,l,r) \tag{334}$$

The conjecture can be immediately proved by induction w.r.t. $\mathsf{rewrites\_list^*\_all}$ using conjecture (321).

## 6.61 Stability of $\mathsf{rewrites\_list^*\_exists}$ under Rule Application

The next theorem states the stability of $\mathsf{rewrites\_list^*\_exists}$ under rule application.

$$\begin{aligned}&\mathsf{rewrites\_list^*\_exists}(k, \mathsf{apply\_subst\_list}(k',l), R) \wedge \mathsf{trs}(R) \wedge \mathsf{in}(l,r,R) \Rightarrow\\ &\qquad\qquad \mathsf{rewrites\_list^*\_exists}(k, \mathsf{apply\_subst\_list}(k',r), R)\end{aligned} \tag{335}$$

This theorem can be proved by induction w.r.t. $\mathsf{rewrites\_list^*\_exists}$, where in the case $\neg\mathsf{disjoint\_list}(k, \mathsf{apply\_subst\_list}(k',r))$ one needs the conjectures (315), (122), (101).

## 6.62 $\mathsf{rewrite\_rule\_list}$ implies $\mathsf{rewrites\_rule\_list^*\_exists}$

The following theorem states that if $t$ is a member of $\mathsf{rewrite\_rule\_list}(k,l,r)$, then this can also be verified using $\mathsf{rewrites\_rule\_list^*\_exists}$.

$$\mathsf{member}(t, \mathsf{rewrite\_rule\_list}(k,l,r)) \wedge \mathsf{length}(r) = \mathsf{s}(0) \Rightarrow \mathsf{rewrites\_rule\_list^*\_exists}(k, \mathsf{add}(t,\mathsf{empty}), l, r) \tag{336}$$

The conjecture is proved by induction w.r.t. $\mathsf{rewrites\_rule\_list^*\_exists}$. The base case ($k = \mathsf{empty}$) is trivial. In the step case ($k = \mathsf{add}(s,k')$), $\mathsf{member}(t, \mathsf{rewrite\_rule\_list}(k,l,r))$ and $\mathsf{disjoint\_list}(\mathsf{add}(t,\mathsf{empty}),k)$ imply $\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k,l,r),k) = \mathsf{false}$. Hence, the induction conclusion can be transformed into the induction hypothesis, (319), and (101).

## 6.63 $\mathsf{rewrites\_rule}$ implies $\mathsf{rewrites\_rule\_list^*\_exists}$

The next theorem says that if $s$ rewrites to $t$ in one step, then this can also be verified with $\mathsf{rewrites\_rule\_list^*\_exists}$.

$$\mathsf{rewrites\_rule}(s,t,l,r) \wedge \mathsf{length}(r) = \mathsf{s}(0) \Rightarrow \mathsf{rewrites\_rule\_list^*\_exists}(\mathsf{add}(s,\mathsf{empty}), \mathsf{add}(t,\mathsf{empty}), l, r) \tag{337}$$

This theorem is a consequence of (287), (288), and (336).

### 6.64 $\mathsf{rewrites\_rule}$ implies $\mathsf{rewrites\_list*\_exists}$

The next theorem says something similar about $\mathsf{rewrites\_list*\_exists}$.

$$\mathsf{rewrites\_rule}(s,t,l,r) \wedge \mathsf{in}(l,r,R) \Rightarrow \mathsf{rewrites\_list*\_exists}(\mathsf{add}(s,\mathsf{empty}),\mathsf{add}(t,\mathsf{empty}),R) \tag{338}$$

This conjecture can be transformed into (69), (320), and (337).

### 6.65 $\mathsf{rewrites\_rule}$ implies $\mathsf{rewrites\_list*\_all}$

The following theorem is a similar conjecture for $\mathsf{rewrites\_list*\_all}$.

$$\mathsf{rewrites\_rule}(s,t,l,r) \Rightarrow \mathsf{rewrites\_list*\_all}(\mathsf{add}(s,\mathsf{empty}),\mathsf{add}(t,\mathsf{empty}),l,r) \tag{339}$$

We perform symbolic evaluation on $\mathsf{rewrites\_list*\_all}$ according to Rule 3″. In the case $t = s$, the theorem is easily proved. Otherwise, by (55) and (287), we obtain that $\mathsf{subseteq\_list}(\mathsf{rewrite\_rule}(s,l,r),\mathsf{add}(s,\mathsf{e})) = \mathsf{false}$. Hence, the conclusion of the implication can be evaluated to

$$\mathsf{rewrites\_rule\_list*}(\mathsf{append}(\mathsf{add}(s,\mathsf{empty}),\mathsf{rewrite\_rule}(s,l,r)),t,l,r).$$

This in turn can be evaluated to $\mathsf{true}$, because (287), (99), and (94) imply $\mathsf{member}(t,\mathsf{rewrite\_rule}(s,l,r))$.

### 6.66 $\mathsf{rewrite^*\_all}$ implies $\mathsf{rewrites\_list*\_all}$

The next theorem shows that $\mathsf{rewrite^*\_all}$ implies $\mathsf{rewrites\_list*\_all}$.

$$\mathsf{rewrite^*\_all}(t,k,l,r) \Rightarrow \mathsf{rewrites\_list*\_all}(\mathsf{add}(t,\mathsf{empty}),k,l,r) \tag{340}$$

The theorem can be proved by an easy induction w.r.t. $\mathsf{rewrite^*\_all}$. The induction conclusion can be directly reduced to the induction hypothesis.

### 6.67 $\mathsf{rewrites\_list*\_all}$ implies $\mathsf{rewrites\_rule\_list*}$

The next conjecture states that if every element of $k_2$ can be reached from $k_1$ and $s$ is a member of $k_2$, then $s$ can also be reached from $k_1$.

$$\mathsf{member}(s,k_2) \wedge \mathsf{rewrites\_list*\_all}(k_1,k_2,l,r) \Rightarrow \mathsf{rewrites\_rule\_list*}(k_1,s,l,r) \tag{341}$$

The conjecture is easily proved by Rule 1″ (using induction w.r.t. $\mathsf{member}$).

### 6.68 $\mathsf{rewrites\_rule\_list*}$ implies $\mathsf{rewrites\_rule\_list*\_exists}$

The next theorem shows the connection between $\mathsf{rewrites\_rule\_list*}$ and $\mathsf{rewrites\_rule\_list*\_exists}$.

$$\mathsf{member}(s,k_2) \wedge \mathsf{rewrites\_rule\_list*}(k_1,s,l,r) \Rightarrow \mathsf{rewrites\_rule\_list*\_exists}(k_1,k_2,l,r) \tag{342}$$

We prove the conjecture by induction w.r.t. $\mathsf{rewrites\_rule\_list*}$. The cases $k_1 = \mathsf{empty}$ or $k_2 = \mathsf{empty}$ can easily be verified. Otherwise, in the only interesting case we have $\mathsf{disjoint\_list}(k_1,k_2)$. By (118), this implies $\mathsf{member}(s,k_1) = \mathsf{false}$. Hence, in this case the induction conclusion can be evaluated to the induction hypothesis.

### 6.69 $\mathsf{rewrites\_rule*}$ implies $\mathsf{rewrite^*\_exists}$

The next theorem is the correctness theorem for $\mathsf{rewrite^*\_exists}$.

$$\mathsf{member}(t,k) \wedge \mathsf{rewrites\_rule*}(s,t,l,r) \Rightarrow \mathsf{rewrite^*\_exists}(s,k,l,r)) \tag{343}$$

By symbolic evaluation and generalization, it can be transformed into (342).

## 6.70 $\mathsf{rewrites\_list*\_all}$ implies $\mathsf{rewrites\_rule\_list*\_exists}$ for Non-Disjoint Lists

The next theorem states that if all elements of $k_3$ can be reached from $k_2$ where $k_3$ and $k_1$ are not disjoint, then there exists an element of $k_1$ which reachable from $k_2$.

$$\mathsf{not}(\mathsf{disjoint\_list}(k_1, k_3)) \wedge \mathsf{rewrites\_list*\_all}(k_2, k_3, l, r) \Rightarrow \mathsf{rewrites\_rule\_list*\_exists}(k_2, k_1, l, r) \tag{344}$$

We prove the conjecture by induction w.r.t. $\mathsf{disjoint\_list}$. The base case $k_1 = \mathsf{empty}$ is trivial. If $k_1 = \mathsf{add}(s, k')$ and $\mathsf{member}(s, k_3)$, then the conjecture follows from (341) and (342). If $k_1 = \mathsf{add}(s, k')$ and $\mathsf{member}(s, k_3) = \mathsf{false}$, then the induction conclusion can be reduced to the induction hypothesis, (334), and (101).

## 6.71 Splitting $\mathsf{rewrites\_rule\_list*\_exists}$ (pc)

The next theorem says that if an element of $\mathsf{add}(t, k')$ can be reached from $k$, then either $t$ can already be reached from $k$ or an element of $k'$ can be reached from $k$.

$$\mathsf{rewrites\_rule\_list*\_exists}(k, \mathsf{add}(t, k'), l, r) \Rightarrow \mathsf{rewrites\_rule\_list*}(k, t, l, r) \vee \mathsf{rewrites\_rule\_list*\_exists}(k, k', l, r) \tag{345}$$

This conjecture can be proved by a straightforward induction w.r.t. $\mathsf{rewrites\_rule\_list*}$ (or also w.r.t. $\mathsf{rewrites\_rule\_list*\_exists}$). In a similar way one can also prove

$$\mathsf{def}(\mathsf{rewrites\_rule\_list*}(k, t, l, r)) \wedge \mathsf{def}(k') \Rightarrow \mathsf{def}(\mathsf{rewrites\_rule\_list*\_exists}(k, \mathsf{add}(t, k'), l, r)) \tag{346}$$

$$\mathsf{def}(\mathsf{rewrites\_rule\_list*\_exists}(k, k', l, r)) \wedge \mathsf{def}(t) \Rightarrow \mathsf{def}(\mathsf{rewrites\_rule\_list*\_exists}(k, \mathsf{add}(t, k'), l, r)) \tag{347}$$

## 6.72 $\mathsf{rewrite^*\_exists}$ for First Elements and Tails of Termlists

The following theorem relates $\mathsf{rewrite^*\_exists}$ for first elements and tails of termlists.

$$\mathsf{rewrite^*\_all}(\mathsf{first}(s), \mathsf{first\_list}(k), l, r) \wedge \mathsf{rewrite^*\_exists}(\mathsf{tail}(s), \mathsf{tail\_list}(k), l, r) \Rightarrow \mathsf{rewrite^*\_exists}(s, k, l, r) \tag{348}$$

We prove the conjecture by structural induction on $k$. If $k = \mathsf{empty}$, then $\mathsf{tail\_list}(k) = \mathsf{empty}$, hence the conjecture is trivial. Otherwise, we have $k = \mathsf{add}(t, k')$ and $\mathsf{rewrite^*\_all}(\mathsf{first}(s), \mathsf{first\_list}(k), l, r)$ is evaluated to $\mathsf{rewrites\_rule*}(\mathsf{first}(s), \mathsf{first}(t), l, r) \wedge \mathsf{rewrite^*\_all}(\mathsf{first}(s), \mathsf{first\_list}(k'), l, r)$. Note that by (345), (346), and (347), we can replace $\mathsf{rewrite^*\_exists}(\mathsf{tail}(s), \mathsf{tail\_list}(k), l, r)$ by $\mathsf{rewrites\_rule*}(\mathsf{tail}(s), \mathsf{tail}(t), l, r) \vee \mathsf{rewrite^*\_exists}(\mathsf{tail}(s), \mathsf{tail\_list}(k'), l, r)$. If the first part of this disjunction is true, then the conjecture can be proved using (329) and (343). Otherwise, the induction conclusion is implied by the hypothesis and (319).

## 6.73 $\mathsf{rewrite^*\_exists}$ for Contexts

The following theorem relates $\mathsf{rewrite^*\_exists}$ for contexts.

$$\mathsf{rewrite^*\_all}(q, \mathsf{apply\_subst\_list}(k, t), l, r) \wedge \mathsf{rewrite^*\_exists}(s, \mathsf{apply\_subst\_list}(k, u), l, r) \Rightarrow \\ \mathsf{rewrite^*\_exists}(\mathsf{func}(n, s, q), \mathsf{apply\_subst\_list}(k, \mathsf{func}(n, u, t), l, r)) \tag{349}$$

The proof is similar to the proof of (348), i.e. we prove the conjecture by structural induction on $k$ (resp. by induction w.r.t. $\mathsf{apply\_subst\_list}$). If $k = \mathsf{empty}$, then the conjecture is trivial. Otherwise, we have $k = \mathsf{add}(\sigma, k')$ and $\mathsf{rewrite^*\_all}(q, \mathsf{apply\_subst\_list}(k, t), l, r)$ is evaluated to $\mathsf{rewrites\_rule*}(q, \mathsf{apply\_subst}(\sigma, t), l, r) \wedge \mathsf{rewrite^*\_all}(q, \mathsf{apply\_subst\_list}(k', t), l, r)$. By (345), (346), and (347), we can replace $\mathsf{rewrite^*\_exists}(s, \mathsf{add}(\mathsf{apply\_subst}(\sigma, u), \mathsf{apply\_subst\_list}(k', u)), l, r)$ by $\mathsf{rewrites\_rule*}(s, \mathsf{apply\_subst}(\sigma, u), l, r) \vee \mathsf{rewrite^*\_exists}(s, \mathsf{apply\_subst\_list}(k', u), l, r)$. If the first part of this disjunction is true, then the conjecture can be proved using (332) and (343). Otherwise, the induction conclusion is implied by the hypothesis and (319).

## 6.74 Correctness of rewrite_rule (pc)

The following conjecture says that if $\mathsf{func}(n, u, \mathsf{e})$ can be rewritten on top position, then the result of this rewrite is also computed by rewrite_rule.

$$\begin{aligned}&\mathsf{matches}(l, \mathsf{func}(n, u, \mathsf{e})) \Rightarrow \\ &\qquad \mathsf{member}(\mathsf{apply\_subst}(\mathsf{matcher}(l, \mathsf{func}(n, u, \mathsf{e})), r), \mathsf{rewrite\_rule}(\mathsf{func}(n, u, \mathsf{e}), l, r))\end{aligned} \tag{350}$$

This conjecture can be proved by (94), (99), and repeated symbolic evaluation.

## 6.75 rewrites_rule implies rewrites_rule* (pc)

The next theorem says that if $s$ rewrites to $t$ in one step, then $s$ also rewrites to $t$ in arbitrary many steps (i.e. rewrites_rule is a sub-relation of rewrites_rule*).

$$\mathsf{rewrites\_rule}(s, t, l, r) \Rightarrow \mathsf{rewrites\_rule*}(s, t, l, r) \tag{351}$$

The theorem can be proved by induction w.r.t. rewrites_rule. The base case $s = \mathsf{e}$ is trivial. The case $s = \mathsf{var}(n, s')$ follows from the induction hypothesis and (329). If $s = \mathsf{func}(n, u, s')$ we have to regard three cases according to the definition of rewrites_rule. In the first case, the conjecture again follows from (329). In the second case one needs (332). The third case can be proved by (350) and (329).

## 6.76 Correctness of rewrite*_all (pc)

The following theorem is the correctness theorem for rewrite*_all.

$$\mathsf{member}(t, k) \wedge \mathsf{rewrite^*\_all}(s, k, l, r) \Rightarrow \mathsf{rewrites\_rule*}(s, t, l, r) \tag{352}$$

It can be proved by a straightforward induction w.r.t. member.

## 6.77 Splitting rewrite*_all using addterm and addtermtwice (pc)

The following lemma states that rewrite*_all can be split using addterm and addtermtwice.

$$\mathsf{rewrite^*\_all}(s, k_1, l, r) \wedge \mathsf{rewrite^*\_all}(t, k_2, l, r) \Rightarrow \mathsf{rewrite^*\_all}(\mathsf{addterm}(s, t), \mathsf{addtermtwice}(k_1, k_2), l, r) \tag{353}$$

It can be proved by induction w.r.t. addtermtwice. The base case is trivial and in the step case one also needs (329). In a similar way one can also prove

$$\begin{aligned}&\mathsf{def}(\mathsf{rewrite^*\_all}(\mathsf{addterm}(s, t), \mathsf{addtermtwice}(k_1, k_2), l, r)) \Rightarrow \\ &\qquad \mathsf{def}(\mathsf{rewrite^*\_all}(s, k_1, l, r)) \wedge \mathsf{def}(\mathsf{rewrite^*\_all}(t, k_2, l, r)).\end{aligned} \tag{354}$$

## 6.78 Splitting rewrite*_all using append (pc)

The following lemma states a similar conjecture for rewrite*_all and append.

$$\mathsf{rewrite^*\_all}(s, k_1, l, r) \wedge \mathsf{rewrite^*\_all}(s, k_2, l, r) \Rightarrow \mathsf{rewrite^*\_all}(s, \mathsf{append}(k_1, k_2), l, r) \tag{355}$$

It can easily be proved by induction w.r.t. append. In this way one can also prove

$$\mathsf{def}(\mathsf{rewrite^*\_all}(s, \mathsf{append}(k_1, k_2), l, r)) \Rightarrow \mathsf{def}(\mathsf{rewrite^*\_all}(s, k_1, l, r)) \wedge \mathsf{def}(\mathsf{rewrite^*\_all}(s, k_2, l, r)). \tag{356}$$

## 6.79 Splitting rewrite*_all using applytwice (pc)

The following theorem shows how rewrite*_all can be decomposed using applytwice.

$$\mathsf{rewrite^*\_all}(s, k_1, l, r) \wedge \mathsf{rewrite^*\_all}(t, k_2, l, r) \Rightarrow \mathsf{rewrite^*\_all}(\mathsf{func}(n, s, t), \mathsf{applytwice}(n, k_1, k_2), l, r) \tag{357}$$

It can easily be proved by induction w.r.t. applytwice using (332).

## 6.80 Splitting $\mathsf{rewrite^*\_all}$ using $\mathsf{apply}$ (pc)

The following theorem shows a similar fact for $\mathsf{apply}$.

$$\mathsf{rewrite^*\_all}(s,k,l,r) \Rightarrow \mathsf{rewrite^*\_all}(\mathsf{func}(n,s,t), \mathsf{addtail}(\mathsf{apply}(n,k),t)) \tag{358}$$

This conjecture can be proved in a similar way using an induction w.r.t. $\mathsf{apply}$ and the conjecture (332).

## 6.81 $\mathsf{rewrites\_rule*}$ implies $\mathsf{rewrite^*\_all}$ if a List only Contains one Element (pc)

The next conjecture says that if $s$ rewrites to $t$, then $s$ also rewrites to all elements in a list consisting only of $t$'s.

$$\mathsf{rewrites\_rule*}(s,t,l,r) \wedge \mathsf{onlyconsistsof}(k,t) \Rightarrow \mathsf{rewrite^*\_all}(s,k,l,r) \tag{359}$$

The proof is easily done by induction w.r.t. $\mathsf{onlyconsistsof}$ using (55). In this way one can also prove

$$\mathsf{def}(\mathsf{rewrite^*\_all}(s,k,l,r)) \wedge \mathsf{onlyconsistsof}(k,t) \Rightarrow \mathsf{def}(\mathsf{rewrites\_rule*}(s,t,l,r)). \tag{360}$$

## 6.82 Connection between $\mathsf{rewrite^*\_all}$ and $\mathsf{rewrite\_rule}$ (pc)

The following conjecture states an obvious connection between $\mathsf{rewrite^*\_all}$ and $\mathsf{rewrite\_rule}$.

$$\mathsf{rewrite^*\_all}(t, \mathsf{rewrite\_rule}(t,l,r), l, r) \tag{361}$$

The conjecture is proved by induction w.r.t. $\mathsf{rewrite\_rule}$. In the case where $t = \mathsf{e}$ it can be evaluated to $\mathsf{true}$. If $t = \mathsf{var}(n,t')$, then the induction conclusion is transformed into

$$\mathsf{rewrite^*\_all}(\mathsf{var}(n,t'), \mathsf{append\_list}(\mathsf{var}(n,\mathsf{e}), \mathsf{rewrite\_rule}(t',l,r)), l, r).$$

Rule $4''$ transforms this into the induction hypothesis and the conjecture (333). Finally, we consider the case where $t = \mathsf{func}(n,u,t')$. Using (355) and (356), the induction conclusion is transformed into

$$\mathsf{rewrite^*\_all}(\mathsf{func}(n,u,t'), \mathsf{addtail}(\mathsf{if}(\ldots), t'), l, r)$$

and

$$\mathsf{rewrite^*\_all}(\mathsf{func}(n,u,t'), \mathsf{append\_list}(\mathsf{func}(n,u,\mathsf{e}), \mathsf{rewrite\_rule}(t',l,r)), l, r).$$

The second conjecture can be proved using the induction hypothesis and (333). For the first conjecture we perform a case analysis w.r.t. $\mathsf{matches}(l, \mathsf{func}(n,u,\mathsf{e}))$.

**Case 1:** $\mathsf{matches}(l, \mathsf{func}(n,u,\mathsf{e})) = \mathsf{false}$

We have to prove

$$\mathsf{rewrite^*\_all}(\mathsf{func}(n,u,t'), \mathsf{addtail}(\mathsf{apply}(n, \mathsf{rewrite\_rule}(u,l,r)), t'), l, r)$$

which is a consequence of the induction hypothesis and (358).

**Case 2:** $\mathsf{matches}(l, \mathsf{func}(n,u,\mathsf{e})) = \mathsf{true}$

Now the induction conclusion can be evaluated to

$$\begin{aligned}&\mathsf{rewrite^*\_all}(\mathsf{func}(n,u,t'),\\ &\qquad\quad \mathsf{add}(\mathsf{addterm}(\mathsf{apply\_subst}(\mathsf{matcher}(l, \mathsf{func}(n,u,\mathsf{e})), r), t'),\\ &\qquad\qquad \mathsf{addtail}(\mathsf{apply}(n, \mathsf{rewrite\_rule}(u,l,r)), t')), l, r)\end{aligned}$$

which can be further evaluated to

$$\begin{aligned}&\mathsf{rewrites\_rule*}(\mathsf{func}(n,u,t'), \mathsf{addterm}(\mathsf{apply\_subst}(\mathsf{matcher}(l, \mathsf{func}(n,u,\mathsf{e})), r), t'), l, r)\\ &\wedge\ \mathsf{rewrite^*\_all}(\mathsf{func}(n,u,t'), \mathsf{addtail}(\mathsf{apply}(n, \mathsf{rewrite\_rule}(u,l,r)), t'), l, r).\end{aligned}$$

The second conjunct is proved as in Case 1. The first conjunct can be transformed into (329), (351), and

$$\mathsf{rewrites\_rule}(\mathsf{func}(n,u,\mathsf{e}), \mathsf{apply\_subst}(\mathsf{matcher}(l, \mathsf{func}(n,u,\mathsf{e})), r), l, r)$$

which can be proved by symbolic evaluation.

## 6.83 Rewriting via Substitutions Carries Over To Terms (pc)

This lemma states that if $\sigma$ is modified by rewriting one term in the range (yielding the substitution $\sigma'$), then for every term $t$ we have that $\sigma(t)$ rewrites to $\sigma'(t)$.

$$\mathsf{rewrite^*\_all}(\mathsf{apply\_subst}(\sigma, t), \mathsf{apply\_subst\_list}(\mathsf{all\_reductions}(\sigma, l, r), t), l, r) \tag{362}$$

We prove the lemma by induction w.r.t. the algorithm apply_subst using Rule 1″.

**Case 1:** $t = \mathsf{e}$

In this case, Rule 3″ and Rule 5″ transform the conjecture into

$$\mathsf{rewrite^*\_all}(\mathsf{e}, \mathsf{apply\_subst\_list}(k, \mathsf{e}), l, r).$$

This conjecture can be proved by induction w.r.t. apply_subst_list. If $k = \mathsf{empty}$, then symbolic evaluation transforms the conjecture to true and if $k = \mathsf{add}(\sigma, k')$, then the induction conclusion can be evaluated to

$$\mathsf{rewrites\_rule*}(\mathsf{e}, \mathsf{e}, l, r) \wedge \mathsf{rewrite^*\_all}(\mathsf{e}, \mathsf{apply\_subst\_list}(k', \mathsf{e}), l, r).$$

The first part of this conjunction is evaluated to true and the second one is the induction hypothesis.

**Case 2:** $t = \mathsf{var}(n, t')$

By symbolic evaluation, (61), (210), (211), (353) (using Rule 3″ and 4″), the induction conclusion can be transformed into the induction hypothesis and into

$$\mathsf{rewrite^*\_all}(\mathsf{apply\_subst\_var}(\sigma, n), \mathsf{apply\_subst\_list}(\mathsf{all\_reductions}(\sigma, l, r), \mathsf{var}(n, \mathsf{e})), l, r).$$

To prove this conjecture, we use an induction w.r.t. apply_subst_var. If $\sigma = \mathsf{e}$, then the conjecture can be proved by symbolic evaluation. Let us now consider the case $\sigma = \mathsf{var}(m, q)$. Using (212), (213), (355), and (356), the conjecture can be split into

$$\begin{aligned}&\mathsf{rewrite^*\_all}(\mathsf{apply\_subst\_var}(\mathsf{var}(m, q), n),\\ &\qquad \mathsf{apply\_subst\_list}(\mathsf{append\_list}(\mathsf{var}(m, \mathsf{e}), \mathsf{addtail}(\mathsf{rewrite\_rule}(\mathsf{first}(q), l, r), \mathsf{tail}(q))), \mathsf{var}(n, \mathsf{e})), l, r)\end{aligned}$$

and

$$\begin{aligned}&\mathsf{rewrite^*\_all}(\mathsf{apply\_subst\_var}(\mathsf{var}(m, q), n),\\ &\qquad \mathsf{apply\_subst\_list}(\mathsf{append\_list}(\mathsf{var}(m, \mathsf{first}(q)), \mathsf{all\_reductions}(\mathsf{tail}(q), l, r)), \mathsf{var}(n, \mathsf{e})), l, r).\end{aligned}$$

Note that in the case $q = \mathsf{e}$ these conjectures are immediately transformed into tautologies. Otherwise, we examine two cases depending on the result of $\mathsf{eq}(m, n)$.

**Case 2.1:** $\mathsf{eq}(m, n) = \mathsf{false}$

Now the two conjectures can be transformed into (214), (215), (217), (218),

$$\begin{aligned}&\mathsf{rewrite^*\_all}(\mathsf{apply\_subst\_var}(\mathsf{tail}(q), n),\\ &\qquad \mathsf{apply\_subst\_list}(\mathsf{tail\_list}(\mathsf{addtail}(\mathsf{rewrite\_rule}(\mathsf{first}(q), l, r), \mathsf{tail}(q))), \mathsf{var}(n, \mathsf{e})), l, r)\end{aligned}$$

(which can be proved using (168), (359), (360), and (220)) and

$$\mathsf{rewrite^*\_all}(\mathsf{apply\_subst\_var}(\mathsf{tail}(q), n), \mathsf{apply\_subst\_list}(\mathsf{all\_reductions}(\mathsf{tail}(q), l, r), \mathsf{var}(n, \mathsf{e})), l, r)$$

(which is the induction hypothesis).

**Case 2.2:** $\mathsf{eq}(m, n) = \mathsf{true}$

Now the two conjectures can be transformed into (214), (215), (219), (56), (57),

$$\mathsf{rewrite^*\_all}(\mathsf{first}(q), \mathsf{first\_list}(\mathsf{addtail}(\mathsf{rewrite\_rule}(\mathsf{first}(q), l, r), \mathsf{tail}(q))), l, r)$$

(which can be proved using (148) and (361)) and

$$\mathsf{onlyconsistsof}(k, \mathsf{first}(q)) \Rightarrow \mathsf{rewrite^*\_all}(\mathsf{first}(q), k, l, r)$$

(which is a consequence of (359) and (360)).

**Case 3:** $t = \mathsf{func}(n, u, t')$

This time by symbolic evaluation, (208), and (209), the induction conclusion is transformed into

$$\begin{aligned}&\mathsf{rewrite^*\_all}(\mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{apply\_subst}(\sigma, t')),\\ &\qquad \mathsf{applytwice}(n, \mathsf{apply\_subst\_list}(\mathsf{all\_reductions}(\sigma, l, r), u), \mathsf{apply\_subst\_list}(\mathsf{all\_reductions}(\sigma, l, r), t')),\\ &\qquad l, r).\end{aligned}$$

This is a consequence of the induction hypotheses and (357).
Similar to (362) one can also prove

$$\begin{aligned}&\mathsf{def}(\mathsf{apply\_subst}(\sigma, t)) \wedge \mathsf{def}(\mathsf{apply\_subst\_list}(\mathsf{all\_reductions}(\sigma, l, r), t)) \Rightarrow\\ &\quad \mathsf{def}(\mathsf{rewrite^*\_all}(\mathsf{apply\_subst}(\sigma, t), \mathsf{apply\_subst\_list}(\mathsf{all\_reductions}(\sigma, l, r), t), l, r)).\end{aligned} \tag{363}$$

# 7 Theorems about Narrowing

In this section we prove theorems about the algorithms which compute narrowing.

## 7.1 Definedness of is_narrowlist, add_narrowlist, apply_narrowlist, back_narrowlist, remove_subst

The next theorems state that is_narrowlist is total and that add_narrowlist, apply_narrowlist, back_narrowlist, and remove_subst are defined for lists which represent narrowings. They can easily be proved by induction w.r.t. is_narrowlist.

$$\mathsf{def}(k) \Rightarrow \mathsf{def}(\mathsf{is\_narrowlist}(k)) \tag{364}$$
$$\mathsf{def}(n, l) \wedge \mathsf{is\_narrowlist}(l) \Rightarrow \mathsf{def}(\mathsf{apply\_narrowlist}(n, l)) \tag{365}$$
$$\mathsf{def}(t, l) \wedge \mathsf{is\_narrowlist}(l) \wedge \mathsf{length}(t) = \mathsf{s}(0) \Rightarrow \mathsf{def}(\mathsf{add\_narrowlist}(t, l)) \tag{366}$$
$$\mathsf{def}(t, l) \wedge \mathsf{is\_narrowlist}(l) \Rightarrow \mathsf{def}(\mathsf{back\_narrowlist}(l, t)) \tag{367}$$
$$\mathsf{def}(l) \wedge \mathsf{is\_narrowlist}(l) \Rightarrow \mathsf{def}(\mathsf{remove\_subst}(l)) \tag{368}$$

## 7.2 Preservation of is_narrowlist under add_narrowlist, apply_narrowlist, back_narrowlist (pc)

The following theorems state that if add_narrowlist, apply_narrowlist, or back_narrowlist are applied to a list representing a narrowing, then the resulting list also represents a narrowing.

$$\mathsf{is\_narrowlist}(l) \Rightarrow \mathsf{is\_narrowlist}(\mathsf{add\_narrowlist}(t, l)) \tag{369}$$
$$\mathsf{is\_narrowlist}(l) \Rightarrow \mathsf{is\_narrowlist}(\mathsf{apply\_narrowlist}(n, l)) \tag{370}$$
$$\mathsf{is\_narrowlist}(l) \Rightarrow \mathsf{is\_narrowlist}(\mathsf{back\_narrowlist}(l, t)) \tag{371}$$

These theorems are easily proved by induction w.r.t. is_narrowlist.

## 7.3 narrow generates Lists Representing Narrowings (pc)

The following theorem shows that the result of narrow represents a narrowing.

$$\mathsf{length}(r) = \mathsf{s}(0) \Rightarrow \mathsf{is\_narrowlist}(\mathsf{narrow}(t, l, r)) \tag{372}$$

The theorem is proved by induction w.r.t. narrow. The base case ($t = \mathsf{e}$) is trivial. The case $t = \mathsf{var}(n, t')$ is easily proved using (369). Finally, the case $t = \mathsf{func}(n, s, t')$ follows from (369), (370), (371), (249), (193), and

$$\mathsf{is\_narrowlist}(l_1) \wedge \mathsf{is\_narrowlist}(l_2) \Rightarrow \mathsf{is\_narrowlist}(\mathsf{append}(l_1, l_2))$$

which is easily proved.

## 7.4 Definedness of special

The following theorem shows that special is defined if its first argument is a substitution and its third argument represents a narrowing.

$$\mathsf{def}(\sigma, s, l) \wedge \mathsf{is\_narrowlist}(l) \wedge \mathsf{is\_subst}(\sigma) \Rightarrow \mathsf{def}(\mathsf{special}(\sigma, s, l)) \tag{373}$$

The theorem can easily be proved by induction w.r.t. is_narrowlist using already proved theorems about the definedness of special's auxiliary functions.

## 7.5 Relation between special and append (on the First Argument) (pc)

The following theorem relates special and append (on the first argument).

$$\mathsf{special}(\sigma, s, l_1) \Rightarrow \mathsf{special}(\sigma, s, \mathsf{append}(l_1, l_2)) \tag{374}$$

By induction w.r.t. special we obtain two induction formulas. The first one (in the case $l_1 = \mathsf{empty}$) can be evaluated to a tautology and in the induction step we obtain the following induction conclusion (after symbolic evaluation)

$$\mathsf{special}(\sigma, s, \mathsf{add}(\tau, \mathsf{add}(q, l_1'))) \Rightarrow \mathsf{special}(\sigma, s, \mathsf{add}(\tau, \mathsf{add}(q, \mathsf{append}(l_1', l_2)))).$$

By symbolic evaluation of special and the induction hypothesis, this conjecture is easily proved.

## 7.6 Relation between special and append (on the Second Argument) (pc)

The next theorem relates special and append (on the second argument).

$$\mathsf{hasevenlength}(l_1) \wedge \mathsf{special}(\sigma, s, l_2) \Rightarrow \mathsf{special}(\sigma, s, \mathsf{append}(l_1, l_2)) \tag{375}$$

The theorem is proved by induction w.r.t. hasevenlength. If $l_1 = \mathsf{empty}$ or $l_1 = \mathsf{add}(t_1, \mathsf{empty})$, then the proof is trivial. The only remaining case ($l_1 = \mathsf{add}(t_1, \mathsf{add}(t_2, l_1'))$) yields the following induction conclusion (after symbolic evaluation)

$$\mathsf{hasevenlength}(l_1') \wedge \mathsf{special}(\sigma, s, l_2) \Rightarrow \mathsf{special}(\sigma, s, \mathsf{add}(t_1, \mathsf{add}(t_2, \mathsf{append}(l_1', l_2)))).$$

A case analysis depending on the truth of $\mathsf{special}(\sigma, t_1) \wedge s = \mathsf{apply\_subst}(t_1, t_2)$ proves the conjecture immediately (resp. reduces the conclusion the induction hypothesis).

## 7.7 Relation between $\mathsf{special}$, $\mathsf{back\_narrowlist}$, and $\mathsf{if}$ (Version 1) (pc)

The next two theorems state facts similar to (374) and (375) using $\mathsf{back\_narrowlist}$ and $\mathsf{if}$.

$$\begin{aligned} & b \wedge \mathsf{special}(\sigma, s, \mathsf{back\_narrowlist}(\mathsf{add}(\tau, \mathsf{add}(q, \mathsf{empty})), t)) \Rightarrow \\ & \quad \mathsf{special}(\sigma, s, \mathsf{back\_narrowlist}(\mathsf{if}(b, \mathsf{add}(\tau, \mathsf{add}(q, l)), l), t)) \end{aligned} \tag{376}$$

By symbolic evaluation, this conjecture is transformed into

$$\mathsf{special}(\sigma, s, \mathsf{back\_narrowlist}(\mathsf{add}(\tau, \mathsf{add}(q, \mathsf{empty})), t)) \Rightarrow \mathsf{special}(\sigma, s, \mathsf{back\_narrowlist}(\mathsf{add}(\tau, \mathsf{add}(q, l)), t)).$$

Now the premise can be evaluated further to

$$\mathsf{special}(\sigma, s, \mathsf{add}(\tau, \mathsf{add}(\mathsf{addterm}(q, \mathsf{apply\_subst}(\tau, t)), \mathsf{empty})))$$

The conjecture trivially holds if the premise is $\mathsf{false}$. Otherwise it can be evaluated to

$$\mathsf{special\_subst}(\sigma, \tau) \wedge \mathsf{eqterm}(s, \mathsf{apply\_subst}(\sigma, \mathsf{addterm}(q, \mathsf{apply\_subst}(\tau, t)))).$$

The conclusion can be evaluated to

$$\mathsf{special}(\sigma, s, \mathsf{add}(\tau, \mathsf{add}(\mathsf{addterm}(q, \mathsf{apply\_subst}(\tau, t)), \mathsf{back\_narrowlist}(l, t))))$$

and further to

$$\mathsf{special\_subst}(\sigma, \tau) \wedge \mathsf{eqterm}(s, \mathsf{apply\_subst}(\sigma, \mathsf{addterm}(q, \mathsf{apply\_subst}(\tau, t)))) \vee \ldots$$

Hence, the premise implies the conclusion.

## 7.8 Relation between $\mathsf{special}$, $\mathsf{back\_narrowlist}$, and $\mathsf{if}$ (Version 2) (pc)

The following theorem is similar to (376).

$$\mathsf{special}(\sigma, s, \mathsf{back\_narrowlist}(l, t)) \Rightarrow \mathsf{special}(\sigma, s, \mathsf{back\_narrowlist}(\mathsf{if}(b, \mathsf{add}(\tau, \mathsf{add}(q, l)), l), t)) \tag{377}$$

We prove the theorem by a case analysis w.r.t. $b$ using Rule $6''$. In the case $b = \mathsf{false}$, we obtain a tautology. Otherwise, the conjecture can be transformed into

$$\mathsf{special}(\sigma, s, \mathsf{back\_narrowlist}(l, t)) \Rightarrow \mathsf{special}(\sigma, s, \mathsf{back\_narrowlist}(\mathsf{add}(\tau, \mathsf{add}(q, l)), t))$$

and by symbolic evaluation we obtain

$$\mathsf{special}(\sigma, s, \mathsf{back\_narrowlist}(l, t)) \Rightarrow \mathsf{special}(\sigma, s, \mathsf{add}(\tau, \mathsf{add}(\mathsf{addterm}(q, \mathsf{apply\_subst}(\tau, t)), \mathsf{back\_narrowlist}(l, t))))$$

resp. the tautology

$$\mathsf{special}(\sigma, s, \mathsf{back\_narrowlist}(l, t)) \Rightarrow \ldots \vee \mathsf{special}(\sigma, s, \mathsf{back\_narrowlist}(l, t)).$$

## 7.9 Monotonicity of $\mathsf{special}$ w.r.t. $\mathsf{addterm}$ (pc)

The following theorem states that adding an instantiated term to the front does not change the result of $\mathsf{special}$.

$$\mathsf{special}(\sigma, s, l) \Rightarrow \mathsf{special}(\sigma, \mathsf{addterm}(\mathsf{apply\_subst}(\sigma, r), s), \mathsf{add\_narrowlist}(r, l)) \tag{378}$$

We prove the conjecture (using Rule $1''$) by induction w.r.t. $\mathsf{special}$. The base case ($l = \mathsf{e}$) is trivial. For the step case ($l = \mathsf{add}(\tau, \mathsf{add}(q, l'))$), the conclusion of the induction conclusion is evaluated to

$$\mathsf{special}(\sigma, \mathsf{addterm}(\mathsf{apply\_subst}(\sigma, r), s), \mathsf{add}(\tau, \mathsf{add}(\mathsf{addterm}(\mathsf{apply\_subst}(\tau, r), q), \mathsf{add\_narrowlist}(r, l'))))$$

and further to

$$\begin{aligned} & \mathsf{special\_subst}(\sigma, \tau) \wedge \mathsf{eqterm}(\mathsf{addterm}(\mathsf{apply\_subst}(\sigma, r), s), \mathsf{apply\_subst}(\sigma, \mathsf{addterm}(\mathsf{apply\_subst}(\tau, r), q))) \\ \vee \quad & \mathsf{special}(\sigma, \mathsf{addterm}(\mathsf{apply\_subst}(\sigma, r), s), \mathsf{add\_narrowlist}(r, l')). \end{aligned}$$

We consider two cases.

**Case 1:** $\mathsf{special\_subst}(\sigma,\tau) \wedge \mathsf{eqterm}(s, \mathsf{apply\_subst}(\sigma, q))$

Using (184), (185), and (55), the conjecture can be transformed into

$$\mathsf{addterm}(\mathsf{apply\_subst}(\sigma, r), s) = \mathsf{addterm}(\mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst}(\tau, r)), \mathsf{apply\_subst}(\sigma, q))$$

which (under the premises of this case) can be transformed further into

$$\mathsf{apply\_subst}(\sigma, r) = \mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst}(\tau, r)).$$

Under the above premises, this is a consequence of (253).

**Case 2: Otherwise**

In this case the induction conclusion can be transformed into the induction hypothesis using Rule 4″.

## 7.10 Monotonicity of special w.r.t. Function Context (pc)

The following theorem states something similar for function context and adding an instantiated term in the back.

$$\mathsf{special}(\sigma, s, l) \Rightarrow \mathsf{special}(\sigma, \mathsf{func}(n, s, \mathsf{apply\_subst}(\sigma, r)), \mathsf{back\_narrowlist}(\mathsf{apply\_narrowlist}(n, l), r)) \tag{379}$$

The conjecture is proved by induction w.r.t. special. The base case ($l = \mathsf{e}$) is trivial. In the step case ($l = \mathsf{add}(\tau, \mathsf{add}(q, l'))$) the induction conclusion can be evaluated (using Rule 3″ and (55)) to

$$\begin{aligned} &\mathsf{special\_subst}(\sigma,\tau) \wedge s = \mathsf{apply\_subst}(\sigma, q) \vee \mathsf{special}(\sigma, s, l') \Rightarrow \\ &\quad \mathsf{special\_subst}(\sigma,\tau) \wedge \mathsf{func}(n, s, \mathsf{apply\_subst}(\sigma, r)) = \\ &\qquad\qquad \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, q), \mathsf{apply\_subst}(\sigma, \mathsf{apply\_subst}(\tau, r))) \vee \\ &\quad \mathsf{special}(\sigma, \mathsf{func}(n, s, \mathsf{apply\_subst}(\sigma, r)), \mathsf{back\_narrowlist}(\mathsf{apply\_narrowlist}(n, l'), r)). \end{aligned}$$

Using Rule 4″, this can be transformed further into the induction hypothesis and (253).

## 7.11 Using special for the Critical Pair Approach (pc)

The following theorem proves the correctness of special's use for the critical pair approach.

$$\begin{aligned} &\mathsf{length}(s) = \mathsf{s}(0) \wedge \mathsf{length}(r) = \mathsf{s}(0) \wedge \\ &\mathsf{special}(\sigma, s, \mathsf{narrow}(l, \mathsf{rename}(l', \mathsf{s}(\mathsf{max}(\mathsf{vars}(l)))), \mathsf{rename}(r', \mathsf{s}(\mathsf{max}(\mathsf{vars}(l)))))) \Rightarrow \\ &\quad \mathsf{in}(\mathsf{apply\_subst}(\sigma, r), s, \mathsf{apply\_subst}(\sigma, \mathsf{cp\_rule}(l, r, l', r'))) \end{aligned} \tag{380}$$

Using (378), (373), (372), and (70), by Rule 3″, 4″, and 5″, the conjecture can be transformed into

$$\mathsf{special}(\sigma, s, l) \Rightarrow \mathsf{membereven}(s, \mathsf{apply\_subst}(\sigma, \mathsf{remove\_subst}(l))).$$

This conjecture is proved by induction w.r.t. special. If $l = \mathsf{e}$, then it is obviously true. If $l = \mathsf{add}(\tau, \mathsf{add}(q, l'))$, then we consider two cases.

**Case 1:** $\mathsf{special\_subst}(\sigma,\tau) \wedge s = \mathsf{apply\_subst}(\sigma, q)$

We have to prove

$$\mathsf{membereven}(s, \mathsf{apply\_subst}(\sigma, \mathsf{appendterm}(q, \mathsf{remove\_subst}(l'))))$$

which (using (77) and (55)) can be transformed into

$$\mathsf{first}(s) = \mathsf{first}(\mathsf{apply\_subst}(\sigma, \mathsf{appendterm}(q, \mathsf{remove\_subst}(l'))))$$

and

$$\mathsf{tail}(s) = \mathsf{second}(\mathsf{apply\_subst}(\sigma, \mathsf{appendterm}(q, \mathsf{remove\_subst}(l')))).$$

These conjectures can be proved by (187), (188), (189), (190), (75), (76), and (61).

**Case 2: Otherwise**

Now the induction conclusion can be transformed into

$$\mathsf{special}(\sigma, s, l') \Rightarrow \mathsf{membereven}(s, \mathsf{apply\_subst}(\sigma, \mathsf{appendterm}(q, \mathsf{remove\_subst}(l'))))$$

which can be transformed further into the induction hypothesis (similar as in Case 1).

### 7.12 Soundness of special (pc)

The following theorem is the soundness of special.

$$\sigma = \mathsf{compose}(\tau, \sigma) \wedge s = \mathsf{apply\_subst}(\sigma, q) \Rightarrow \mathsf{special}(\sigma, s, \mathsf{add}(\tau, \mathsf{add}(q, \mathsf{empty}))). \tag{381}$$

By Rule $3''$ and Rule $4''$, it can be transformed into (255).

## 8 Theorems about Joinability

The next section contains theorems about the algorithms which check joinability.

### 8.1 Reflexivity of joinable

This theorem proves the reflexivity of joinable.

$$\mathsf{joinable}(t, t) \tag{382}$$

Its proof can immediately be reduced to the proof of (125).

### 8.2 Commutativity of Joinability (pc)

Now we show that joinability is commutative.

$$\mathsf{joinable}(s, t, R) = \mathsf{joinable}(t, s, R) \tag{383}$$

$$\mathsf{joinable\_list}(k_1, k_2, R) = \mathsf{joinable\_list}(k_2, k_1, R) \tag{384}$$

Obviously, conjecture (383) is a direct consequence of (384). The latter conjecture is proved by an easy induction w.r.t. joinable_list using (124). In a similar way one can also prove

$$\mathsf{def}(\mathsf{joinable}(s, t, R)) \Leftrightarrow \mathsf{def}(\mathsf{joinable}(t, s, R)) \tag{385}$$

$$\mathsf{def}(\mathsf{joinable\_list}(k_1, k_2, R)) \Leftrightarrow \mathsf{def}(\mathsf{joinable\_list}(k_2, k_1, R)) \tag{386}$$

(by showing both directions of the theorems separately and using induction w.r.t. the arguments of joinable_list in the premise).

### 8.3 Monotonicity of joinable_list

The following theorem says that if $k_1$ and $k_2$ are joinable, then this also holds for all superlists of $k_1$ and $k_2$.

$$\mathsf{subseteq\_list}(k_1, k_1') \wedge \mathsf{subseteq\_list}(k_2, k_2') \wedge \mathsf{trs}(R) \wedge \mathsf{joinable\_list}(k_1, k_2, R) \Rightarrow \mathsf{joinable\_list}(k_1', k_2', R) \tag{387}$$

The proof is done by induction w.r.t. merged induction relations suggested by $\mathsf{joinable\_list}(k_1, k_2, R)$ and $\mathsf{joinable\_list}(k_1', k_2', R)$, cf. the extensions of Rule $1''$ in [9]. If $\mathsf{disjoint\_list}(k_1', k_2') = \mathsf{false}$, then the proof is trivial. Otherwise we also have $\mathsf{disjoint\_list}(k_1, k_2) = \mathsf{true}$ (by (122)).

In the case where $\mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k_1, R), k_1) \wedge \mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k_2, R), k_2)$, we proceed in a similar way as in the proofs of (320), (321), and (322). So we add the premise

$$\mathsf{ge}(\mathsf{setdiff}(k_1', k_1), \mathsf{setdiff}(k_1', k_1)) \wedge \mathsf{ge}(\mathsf{setdiff}(k_2', k_2), \mathsf{setdiff}(k_2', k_2))$$

which we then generalize to $\mathsf{ge}(n_1, \mathsf{setdiff}(k_1', k_1)) \wedge \mathsf{ge}(n_2, \mathsf{setdiff}(k_2', k_2))$. Now we perform another induction w.r.t. $n_1$ and $n_2$. If $n_i = \mathsf{0}$, then (136), (293), and (96) imply $\mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k_i, R), k_i)$. Hence, if both $n_1$ and $n_2$ are 0, then the conjecture is trivial. If $n_1 = \mathsf{s}(m_1)$ and $n_2 = \mathsf{0}$, in the only interesting case for $k_1$ and $k_2$, we obtain the following induction conclusion (of the inner $n_1$-induction).

$$\begin{array}{l}\mathsf{subseteq\_list}(k_1, k_1') \wedge \mathsf{subseteq\_list}(k_2, k_2') \wedge \mathsf{trs}(R) \wedge \mathsf{ge}(\mathsf{s}(m_1), \mathsf{setdiff}(k_1', k_1)) \wedge \\ \mathsf{ge}(\mathsf{0}, \mathsf{setdiff}(k_2', k_2)) \wedge \mathsf{joinable\_list}(\mathsf{append}(k_1, \mathsf{rewrite\_list}(k_1, R)), \mathsf{append}(k_2, \mathsf{rewrite\_list}(k_2, R)), R) \Rightarrow \\ \quad \mathsf{joinable\_list}(\mathsf{append}(k_1', \mathsf{rewrite\_list}(k_1', R)), \mathsf{append}(k_2', \mathsf{rewrite\_list}(k_2', R)), R)\end{array}$$

By (99), (293), (138), (136), and (137), this can be transformed into the induction hypothesis. The cases where $n_2$ is not 0 work in an analogous way.

Finally we prove the remaining case of the outer joinable_list-induction. Here, the induction conclusion follows from the induction hypothesis, (103), and (293).

## 8.4 Joinability from Joinability of Tails (pc)

The following theorem states that for two lists of terms $t$ and $s$, joinability of $t$ and $s$ follows from joinability of their tails (provided their first elements are the same).

$$\mathsf{first}(s) = \mathsf{first}(t) \wedge \mathsf{joinable}(\mathsf{tail}(s), \mathsf{tail}(t), R) \;\Rightarrow\; \mathsf{joinable}(s, t, R) \tag{388}$$

By symbolic evaluation and generalization, the theorem is transformed into

$$\begin{array}{l}\mathsf{subseteq\_list}(\mathsf{first\_list}(k_1), \mathsf{first\_list}(k_2)) \wedge \mathsf{subseteq\_list}(\mathsf{first\_list}(k_2), \mathsf{first\_list}(k_1)) \wedge \\ \mathsf{joinable\_list}(\mathsf{tail\_list}(k_1), \mathsf{tail\_list}(k_2), R) \;\Rightarrow \\ \quad \mathsf{joinable\_list}(k_1, k_2, R).\end{array}$$

We prove this conjecture by induction w.r.t. joinable_list. By (146) and (309), in the only interesting case we obtain the induction conclusion

$$\begin{array}{l}\mathsf{subseteq\_list}(\mathsf{first\_list}(k_1), \mathsf{first\_list}(k_2)) \wedge \mathsf{subseteq\_list}(\mathsf{first\_list}(k_2), \mathsf{first\_list}(k_1)) \wedge \\ \mathsf{joinable\_list}(\mathsf{append}(\mathsf{tail\_list}(k_1), \mathsf{rewrite\_list}(\mathsf{tail\_list}(k_1), R)), \\ \qquad\qquad \mathsf{append}(\mathsf{tail\_list}(k_2), \mathsf{rewrite\_list}(\mathsf{tail\_list}(k_2), R)), R) \;\Rightarrow \\ \quad \mathsf{joinable\_list}(\mathsf{append}(k_1, \mathsf{rewrite\_list}(k_1, R)), \mathsf{append}(k_2, \mathsf{rewrite\_list}(k_2, R)), R).\end{array}$$

This can be reduced to the induction hypothesis and (142), (307), (312), (306), (141), (387).
In a similar way one can also prove

$$\mathsf{def}(\mathsf{joinable}(s, t, R)) \wedge \neg s = \mathsf{e} \wedge \neg t = \mathsf{e} \wedge \mathsf{first}(s) = \mathsf{first}(t) \Rightarrow \mathsf{def}(\mathsf{joinable}(\mathsf{tail}(s), \mathsf{tail}(t), R)). \tag{389}$$

## 8.5 Joinability from Joinability of First Elements (pc)

The following theorem states that for two lists of terms $t$ and $s$, joinability of $t$ and $s$ follows from joinability of their first elements (provided their tails are the same).

$$\mathsf{tail}(s) = \mathsf{tail}(t) \wedge \mathsf{joinable}(\mathsf{first}(s), \mathsf{first}(t), R) \;\Rightarrow\; \mathsf{joinable}(s, t, R) \tag{390}$$

The proof for this theorem is analogous to the one of (388). In a similar way one can also prove

$$\mathsf{def}(\mathsf{joinable}(s, t, R)) \wedge \neg s = \mathsf{e} \wedge \neg t = \mathsf{e} \wedge \mathsf{tail}(s) = \mathsf{tail}(t) \Rightarrow \mathsf{def}(\mathsf{joinable}(\mathsf{first}(s), \mathsf{first}(t), R)). \tag{391}$$

## 8.6 Stability of joinable under Contexts

The following theorem states that joinability is stable under context.

$$\mathsf{joinable}(s, t, R) \;\Rightarrow\; \mathsf{joinable}(\mathsf{func}(n, s, r), \mathsf{func}(n, t, r), R) \tag{392}$$

By symbolic evaluation and generalization, it can be transformed into

$$\mathsf{joinable\_list}(k_1, k_2, R) \;\Rightarrow\; \mathsf{joinable\_list}(\mathsf{addtail}(\mathsf{apply}(n, k_1), r), \mathsf{addtail}(\mathsf{apply}(n, k_2), r), R).$$

This conjecture is proved by induction w.r.t. $\mathsf{joinable\_list}(k_1, k_2, R)$. Using (154), (155), (297), (303), in the only interesting case we obtain the induction conclusion

$$\begin{aligned}&\mathsf{joinable\_list}(\mathsf{append}(k_1,\mathsf{rewrite\_list}(k_1,R)),\mathsf{append}(k_2,\mathsf{rewrite\_list}(k_2,R)),R) \Rightarrow\\ &\quad \mathsf{joinable\_list}(\mathsf{append}(\mathsf{addtail}(\mathsf{apply}(n,k_1),r),\mathsf{rewrite\_list}(\mathsf{addtail}(\mathsf{apply}(n,k_1),r),R)),\\ &\qquad\qquad \mathsf{append}(\mathsf{addtail}(\mathsf{apply}(n,k_2),r),\mathsf{rewrite\_list}(\mathsf{addtail}(\mathsf{apply}(n,k_2),r),R)),\\ &\qquad\qquad R)\end{aligned}$$

Rule $4''$ transforms this into (387) and

$$\begin{aligned}&\mathsf{subseteq\_list}(\mathsf{addtail}(\mathsf{apply}(n,\mathsf{append}(k,\mathsf{rewrite\_list}(k,R))),r),\\ &\qquad \mathsf{append}(\mathsf{addtail}(\mathsf{apply}(n,k),r),\mathsf{rewrite\_list}(\mathsf{addtail}(\mathsf{apply}(n,k),r),R))).\end{aligned}$$

This conjecture can be proved by (103), (158), (156), (295), (300).

## 8.7 Stability of joinable under Substitutions (pc)

The following theorem says that if two termlists are joinable, then so are all their instantiations.

$$\mathsf{trs}(R) \wedge \mathsf{joinable}(s,t,R) \Rightarrow \mathsf{joinable}(\mathsf{apply\_subst}(\sigma,s),\mathsf{apply\_subst}(\sigma,t),R) \tag{393}$$

By symbolic evaluation and generalization, the conjecture is transformed into

$$\mathsf{joinable\_list}(l,k,R) \Rightarrow \mathsf{joinable\_list}(\mathsf{apply\_subst\_tll}(\sigma,l),\mathsf{apply\_subst\_tll}(\sigma,k),R).$$

We prove the conjecture by induction w.r.t. joinable_list using $l, k, R$ as induction variables. The case $\mathsf{disjoint\_list}(l,k) = \mathsf{false}$ can be proved using (226). If the premise of the implication evaluates to false, then the proof is trivial. Otherwise, the induction conclusion is evaluated to

$$\begin{aligned}&\mathsf{joinable\_list}(\mathsf{append}(l,\mathsf{rewrite\_list}(l,R)),\mathsf{append}(k,\mathsf{rewrite\_list}(k,R)),R) \Rightarrow\\ &\quad \mathsf{joinable\_list}(\mathsf{apply\_subst\_tll}(\sigma,l),\mathsf{apply\_subst\_tll}(\sigma,k),R).\end{aligned}$$

Due to (387) and (103), this can be transformed into

$$\begin{aligned}&\mathsf{joinable\_list}(\mathsf{append}(l,\mathsf{rewrite\_list}(l,R)),\mathsf{append}(k,\mathsf{rewrite\_list}(k,R)),R) \Rightarrow\\ &\quad \mathsf{joinable\_list}(\mathsf{append}(\mathsf{apply\_subst\_tll}(\sigma,l),\mathsf{rewrite\_list}(\mathsf{apply\_subst\_tll}(\sigma,l),R)),\\ &\qquad\qquad \mathsf{append}(\mathsf{apply\_subst\_tll}(\sigma,k),\mathsf{rewrite\_list}(\mathsf{apply\_subst\_tll}(\sigma,k),R)),\\ &\qquad\qquad R)\end{aligned}$$

even if $\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k_1),k_1)$ and $\mathsf{subseteq\_list}(\mathsf{rewrite\_rule\_list}(k_2),k_2)$ hold. By (223), (224), (387), (316), and (317), it can be transformed further into the induction hypothesis.

## 8.8 Rewriting implies Joinability (pc)

The following theorem states that if a termlist rewrites to another (in arbitrary many steps), then both terms are joinable.

$$\mathsf{trs}(R) \wedge \mathsf{rewrites*}(s,t,R) \Rightarrow \mathsf{joinable}(s,t,R) \tag{394}$$

Symbolic evaluation and generalization (Rule $5''$) transforms the conjecture into

$$\mathsf{trs}(R) \wedge \mathsf{rewrites\_list*}(k,t,R) \Rightarrow \mathsf{joinable\_list}(k,\mathsf{add}(t,\mathsf{empty}),R).$$

This conjecture is proved by induction w.r.t. rewrites_list*. If $\mathsf{member}(t,k)$ holds then by (124), the conjecture is proved. Otherwise, in the only interesting case we obtain the induction conclusion

$$\begin{aligned}&\mathsf{trs}(R) \wedge \mathsf{rewrites\_list*}(\mathsf{rewrite\_list}(k,R),t,R) \Rightarrow\\ &\quad \mathsf{joinable\_list}(\mathsf{append}(k,\mathsf{rewrite\_list}(k,R)),\mathsf{append}(\mathsf{add}(t,\mathsf{empty}),\mathsf{rewrite\_list}(\mathsf{add}(t,\mathsf{empty}),R)),R)\end{aligned}$$

and the induction hypothesis

$$\mathsf{trs}(R) \wedge \mathsf{rewrites\_list*}(\mathsf{rewrite\_list}(k,R),t,R) \Rightarrow \mathsf{joinable\_list}(\mathsf{rewrite\_list}(k,R),\mathsf{add}(t,\mathsf{empty}),R).$$

Hence, the conjecture can be proved using (101) and (387).

## 8.9 Stability of joinable_pairs under Substitutions (pc)

This theorem says that if all pairs in the list $l$ are joinable, then this is also true for all instantiated pairs.

$$\mathsf{trs}(R) \wedge \mathsf{joinable\_pairs}(l, R) \wedge \mathsf{in}(s, t, \mathsf{apply\_subst}(\sigma, l)) \Rightarrow \mathsf{joinable}(s, t, R). \tag{395}$$

The theorem is proved by induction w.r.t. joinable_pairs. The base case ($l = \mathsf{e}$) is easy, because the second premise evaluates to false. Now we consider the two step cases.

**Case 1:** $l = \mathsf{var}(n, l')$

The induction conclusion can be transformed into

$$\begin{aligned}&\mathsf{rewrites^*}(\mathsf{first}(l'), \mathsf{var}(n, \mathsf{e})) \wedge \mathsf{joinable\_pairs}(\mathsf{tail}(l'), R) \wedge\\ &\mathsf{in}(s, t, \mathsf{addterm}(\mathsf{apply\_subst\_var}(\sigma, n), \mathsf{apply\_subst}(\sigma, l'))) \Rightarrow\\ &\mathsf{joinable}(s, t, R).\end{aligned}$$

Now we perform a case analysis as suggested by in. In the case $s = \mathsf{apply\_subst\_var}(\sigma, n)$, $t = \mathsf{apply\_subst}(\sigma, \mathsf{first}(l'))$, the conjecture follows from (187), (188), (323), (394), and (383). Otherwise, the conclusion can be transformed into the induction hypothesis using (189) and (190).

**Case 2:** $l = \mathsf{func}(n, u, l')$

In this case we have the (transformed) induction conclusion

$$\begin{aligned}&\mathsf{joinable}(\mathsf{func}(n, u, \mathsf{e}), \mathsf{first}(l')) \wedge \mathsf{joinable\_pairs}(\mathsf{tail}(l'), R) \wedge\\ &\mathsf{in}(s, t, \mathsf{addterm}(\mathsf{func}(n, \mathsf{apply\_subst\_var}(\sigma, u), \mathsf{apply\_subst}(\sigma, l')))) \Rightarrow\\ &\quad \mathsf{joinable}(s, t, R).\end{aligned}$$

Again we perform a case analysis w.r.t. the result of in. If $s = \mathsf{apply\_subst}(\sigma, \mathsf{func}(n, u, \mathsf{e}))$, $t = \mathsf{apply\_subst}(\sigma, \mathsf{first}(l'))$, then the theorem follows from (393). Otherwise, the induction conclusion can again be transformed into the induction hypothesis, (189), and (190).

## 8.10 rewrites_list*_exists implies joinable_list (pc)

The following theorem says that if one of the termlists in $k_1$ reduces to one of the termlists in $k_2$, then $k_1$ and $k_2$ are joinable.

$$\mathsf{rewrites\_list^*\_exists}(k_1, k_2, R) \Rightarrow \mathsf{joinable\_list}(k_1, k_2, R) \tag{396}$$

The theorem can easily be proved by induction w.r.t. joinable_list, where the induction conclusion can be transformed (using Rule 4″) into (319), (99), and the induction hypothesis.

## 8.11 Connection between rewrite*_exists, rewrite*_all, and joinable_list (pc)

The next theorem states that if one of the termlists in $k_1$ rewrites to one of the termlists in $k_3$ and if each of the termlists in $k_3$ can be reached by rewriting one of the termlists in $k_2$, then $k_1$ and $k_2$ are joinable.

$$\mathsf{rewrites\_list^*\_exists}(k_1, k_3, R)) \wedge \mathsf{rewrites\_list^*\_all}(k_2, k_3, l, r) \wedge \mathsf{in}(l, r, R) \Rightarrow \mathsf{joinable\_list}(k_1, k_2, R) \tag{397}$$

The theorem is proved by induction w.r.t. joinable_list. In the only interesting case we have $\mathsf{disjoint\_list}(k_1, k_2)$. If $\mathsf{disjoint\_list}(k_1, k_3) = \mathsf{false}$, then the theorem is proved by (344), (320), (396). Otherwise (if $\mathsf{disjoint\_list}(k_1, k_3)$), we make a case analysis w.r.t. the truth of $\mathsf{subseteq\_list}(\mathsf{rewrite\_list}(k_1, R), k_1)$. If this holds, then the theorem is trivial. Otherwise, the induction conclusion is

$$\begin{aligned}&\mathsf{rewrites\_list^*\_exists}(\mathsf{append}(k_1, \mathsf{rewrite\_list}(k_1, R)), k_3, R) \wedge \mathsf{rewrites\_list^*\_all}(k_2, k_3, l, r) \wedge \mathsf{in}(l, r, R) \Rightarrow\\ &\quad \mathsf{joinable\_list}(\mathsf{append}(k_1, \mathsf{rewrite\_list}(k_1, R)), \mathsf{append}(k_2, \mathsf{rewrite\_list}(k_2, R)), R)\end{aligned}$$

Using (334) and (99) this can be transformed into the induction hypothesis.

## 8.12 Connection between $\mathsf{rewrite^*\_exists}$, $\mathsf{all\_reductions}$, and $\mathsf{joinable}$ (pc)

The following theorem states a fact needed for the critical pair lemma.

$$\begin{aligned}&\mathsf{rewrite^*\_exists}(s, \mathsf{apply\_subst\_list}(\mathsf{all\_reductions}(\sigma, l', r'), l), l', r')\\ &\quad\Rightarrow \Big(\mathsf{subseteq}(\mathsf{vars}(r), \mathsf{vars}(l)) \wedge \mathsf{subseteq}(\mathsf{vars}(r'), \mathsf{vars}(l')) \wedge\\ &\qquad \mathsf{first\_is\_func}(l) \wedge \mathsf{first\_is\_func}(l') \Rightarrow\\ &\qquad \mathsf{joinable}(s, \mathsf{apply\_subst}(\sigma, r), \mathsf{addterm}(l', \mathsf{addterm}(r', \mathsf{addterm}(l, \mathsf{addterm}(r, \mathsf{e})))))\Big)\end{aligned} \tag{398}$$

By Rule $3''$, $4''$, and $5''$, it can be transformed into

$$\begin{aligned}&\mathsf{rewrites\_rule\_list^*\_exists}(k, \mathsf{apply\_subst\_list}(k', l), l', r') \wedge \mathsf{def}(R, r) \wedge \mathsf{trs}(R) \wedge \mathsf{in}(l, r, R) \Rightarrow\\ &\qquad\qquad \mathsf{rewrites\_list^*\_exists}(k, \mathsf{apply\_subst\_list}(k', r), R)\end{aligned}$$

(which is implied by (320) and (335)), (362), (340), and (397).

## 8.13 Joinability for Rules from a TRS (pc)

The next theorem says that if two termlists are joinable with rules from $R$, then they are also joinable with $R$.

$$\begin{aligned}&\mathsf{trs}(R) \wedge \mathsf{in}(l, r, R) \wedge \mathsf{in}(l', r', R) \wedge \mathsf{joinable}(s, t, \mathsf{addterm}(l', \mathsf{addterm}(r', \mathsf{addterm}(l, \mathsf{addterm}(r, \mathsf{e}))))) \Rightarrow\\ &\qquad\qquad \mathsf{joinable}(s, t, R)\end{aligned} \tag{399}$$

By symbolic evaluation and generalization, the theorem is transformed into a modified one where $\mathsf{joinable}(s, t, \mathsf{addterm}(l', \mathsf{addterm}(r', \mathsf{addterm}(l, \mathsf{addterm}(r, \mathsf{e})))))$ is replaced by $\mathsf{joinable\_list}(k_1, k_2, \mathsf{addterm}(l', \mathsf{addterm}(r', \mathsf{addterm}(l, \mathsf{addterm}(r, \mathsf{e})))))$ and $\mathsf{joinable}(s, t, R)$ is replaced by $\mathsf{joinable\_list}(k_1, k_2, R)$. This conjecture is proved w.r.t. the induction suggested by the term $\mathsf{joinable\_list}(k_1, k_2, R)$. The if's in the result of joinable_list lead to several cases. All of them are trivial except the one corresponding to the step case. Here, the induction conclusion is

$$\begin{aligned}&\mathsf{trs}(R) \wedge \mathsf{in}(l, r, R) \wedge \mathsf{in}(l', r', R) \wedge \mathsf{joinable\_list}(k_1, k_2, \mathsf{addterm}(l', \mathsf{addterm}(r', \mathsf{addterm}(l, \mathsf{addterm}(r, \mathsf{e}))))) \Rightarrow\\ &\quad \mathsf{joinable\_list}(k_1, k_2, R).\end{aligned}$$

Using (318), (103), and (387), this can be transformed into

$$\begin{aligned}&\mathsf{trs}(R) \wedge \mathsf{in}(l, r, R) \wedge \mathsf{in}(l', r', R) \wedge\\ &\mathsf{joinable\_list}(\mathsf{append}(k_1, \mathsf{rewrite\_list}(k_1, R)), \mathsf{append}(k_2, \mathsf{rewrite\_list}(k_2, R)),\\ &\qquad\qquad \mathsf{addterm}(l', \mathsf{addterm}(r', \mathsf{addterm}(l, \mathsf{addterm}(r, \mathsf{e}))))) \Rightarrow\\ &\qquad \mathsf{joinable\_list}(\mathsf{append}(k_1, \mathsf{rewrite\_list}(k_1, R)), \mathsf{append}(k_2, \mathsf{rewrite\_list}(k_2, R)), R),\end{aligned}$$

which is the induction hypothesis.

## 8.14 Correctness of $\mathsf{jcp}$

This theorem shows that jcp indeed guarantees that all critical pairs are joinable.

$$\mathsf{jcp}(R) \wedge \mathsf{in}(l_1, r_1, R) \wedge \mathsf{in}(l_2, r_2, R) \Rightarrow \mathsf{joinable\_pairs}(\mathsf{cp\_rule}(l_1, r_1, l_2, r_2), R) \tag{400}$$

Symbolic evaluation transforms the conjecture into

$$\mathsf{jcp\_aux1}(R, R, R) \wedge \mathsf{in}(l_1, r_1, R) \wedge \mathsf{in}(l_2, r_2, R) \Rightarrow \mathsf{joinable\_pairs}(\mathsf{cp\_rule}(l_1, r_1, l_2, r_2), R)$$

which can be generalized (using Rule $5''$) to

$$\mathsf{jcp\_aux1}(R_1, R_2, R) \wedge \mathsf{in}(l_1, r_1, R_1) \wedge \mathsf{in}(l_2, r_2, R_2) \Rightarrow \mathsf{joinable\_pairs}(\mathsf{cp\_rule}(l_1, r_1, l_2, r_2), R).$$

This conjecture is proved by induction w.r.t. jcp_aux1. The formula in the base case ($R_1 = \mathsf{e}$) reduces to a tautology, since the premise $\mathsf{in}(l_1, r_1, \mathsf{e})$ reduces to false. In the step case, we have $R_1 = \mathsf{func}(n, s, t)$. Symbolic evaluation of $\mathsf{in}(l_1, r_1, \mathsf{func}(n, s, t))$ suggests the following case analysis.

**Case 1:** $l_1 = \mathsf{func}(n, s, \mathsf{e}), r_1 = \mathsf{first}(t)$

In this case we have to prove

$l_1 = \mathsf{func}(n, s, \mathsf{e}) \wedge r_1 = \mathsf{first}(t) \wedge \mathsf{jcp\_aux2}(l_1, r_1, R_2, R) \wedge \mathsf{in}(l_2, r_2, R) \Rightarrow \mathsf{joinable\_pairs}(\mathsf{cp\_rule}(l_1, r_1, l_2, r_2), R)$.

This can be proved by induction w.r.t. jcp_aux2. The base case is again trivial. In the step case we have $R_2 = \mathsf{func}(m, s', t')$. If $l_2 = \mathsf{func}(m, s', t')$ and $r_2 = \mathsf{first}(t')$, then the conjecture can be immediately transformed into a tautology. Otherwise the induction conclusion is evaluated to the induction hypothesis

$$l_1 = \mathsf{func}(n, s, \mathsf{e}) \wedge r_1 = \mathsf{first}(t) \wedge \mathsf{jcp\_aux2}(l_1, r_1, \mathsf{tail}(t'), R) \wedge \mathsf{in}(l_2, r_2, \mathsf{tail}(t')) \Rightarrow$$
$$\mathsf{joinable\_pairs}(\mathsf{cp\_rule}(l_1, r_1, l_2, r_2), R).$$

**Case 2: Otherwise**

In this case, the induction conclusion is evaluated to the induction hypothesis

$$\mathsf{jcp\_aux1}(\mathsf{tail}(t), R_2, R) \wedge \mathsf{in}(l_1, r_1, \mathsf{tail}(t)) \wedge \mathsf{in}(l_2, r_2, R_2) \Rightarrow \mathsf{joinable\_pairs}(\mathsf{cp\_rule}(l_1, r_1, l_2, r_2), R).$$

# 9 The Critical Pair Lemma

In Section 9.1 we first present a crucial lemma needed in the proof of the critical pair lemma. This lemma itself follows in Section 9.2.

## 9.1 Every Non-Joinable Local Divergence is an Instantiation of a Critical Pair (pc)

The following theorem is needed in the proof of the critical pair lemma. It states that every local divergence is joinable or an instantiation of a critical pair.

$$\begin{array}{l}
\mathsf{is_subst}(\sigma) \wedge \mathsf{no_duplicates}(\sigma) \wedge \mathsf{subseteq}(\mathsf{vars}(r), \mathsf{vars}(l)) \wedge \mathsf{subseteq}(\mathsf{vars}(\hat{r}), \mathsf{vars}(\hat{l})) \\
\wedge \mathsf{length}(\hat{r}) = \mathsf{s}(0) \wedge \mathsf{subseteq}(\mathsf{dom}(\sigma), \mathsf{vars}(l)) \wedge \mathsf{first_is_func}(l) \wedge \mathsf{first_is_func}(\hat{l}) \\
\wedge\ \mathsf{rewrites_rule}(\mathsf{apply_subst}(\sigma, l), s, \hat{l}, \hat{r}) \ \Rightarrow \\
\qquad \mathsf{in}(\mathsf{apply_subst}(\sigma, r), s, \\
\qquad\quad \mathsf{apply_subst}( \\
\qquad\qquad \mathsf{appendterm}(\sigma, \mathsf{rewrites_matcher}(\mathsf{apply_subst}(\sigma, l), s, \mathsf{rename}(\hat{l}, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l)))), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{rename}(\hat{r}, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l)))))), \\
\qquad\qquad \mathsf{cp_rule}(l, r, \hat{l}, \hat{r}))) \\
\quad \vee \quad \mathsf{joinable}(s, \mathsf{apply_subst}(\sigma, r), \mathsf{addterm}(\hat{l}, \mathsf{addterm}(\hat{r}, \mathsf{addterm}(l, \mathsf{addterm}(r, \mathsf{e})))))
\end{array} \tag{401}$$

By Rule 4″, this conjecture can be replaced by (62), (281), (121), (232), (380), (398), (273), (193), (183), (280), (373), (372), (175), (132), (130), (131), and

$$\begin{array}{l}
\mathsf{is_subst}(\sigma) \wedge \mathsf{no_duplicates}(\sigma) \wedge \mathsf{subseteq}(\mathsf{vars}(r'), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{vars}(l), \mathsf{vars}(l')) \\
\wedge\ \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \wedge \mathsf{length}(r') = \mathsf{s}(0) \wedge l' = \mathsf{rename}(\hat{l}, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l)))) \\
\wedge\ r' = \mathsf{rename}(\hat{r}, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l)))) \wedge \mathsf{rewrites_rule}(\mathsf{apply_subst}(\sigma, l), s, \hat{l}, \hat{r}) \ \Rightarrow \\
\qquad \mathsf{special}(\mathsf{appendterm}(\sigma, \mathsf{rewrites_matcher}(\mathsf{apply_subst}(\sigma, l), s, l', r')), s, \mathsf{narrow}(l, l', r')) \\
\quad \vee\ \mathsf{rewrite^*_exists}(s, \mathsf{apply_subst_list}(\mathsf{all_reductions}(\sigma, \hat{l}, \hat{r}), l), \hat{l}, \hat{r})
\end{array}$$

To prove this conjecture, we apply an induction w.r.t. narrow. Hence, by Rule 1″ the original conjecture is transformed into new induction formulas and by case analysis (using Rule 6″) we obtain the following new conjectures to be proved.

**Case 1:** $l = \mathsf{e}$

In this case, $\mathsf{rewrites\_rule}(\mathsf{apply\_subst}(\sigma, l), s, \hat{l}, \hat{r})$ reduces to $\mathsf{false}$, i.e. the conjecture is a tautology (provable with Rule $4''$).

**Case 2:** $l = \mathsf{var}(n, t)$

In this case, $\mathsf{apply\_subst}(\sigma, l)$ is evaluated to $\mathsf{addterm}(\mathsf{apply\_subst\_var}(\sigma, n), \mathsf{apply\_subst}(\sigma, t))$. Now using Rule $4''$,

$$\mathsf{rewrites\_rule}(\mathsf{addterm}(\mathsf{apply\_subst\_var}(\sigma, n), \mathsf{apply\_subst}(\sigma, t)), s, \hat{l}, \hat{r})$$

can be replaced by an instantiation of conjecture (272), (262), (269), and $\mathsf{rewrites\_rule}(\mathsf{apply\_subst\_var}(\sigma, n), \mathsf{first}(s), \hat{l}, \hat{r}) \wedge \mathsf{apply\_subst}(\sigma, t) = \mathsf{tail}(s) \;\vee\; \mathsf{rewrites\_rule}(\mathsf{apply\_subst}(\sigma, t), \mathsf{tail}(s), \hat{l}, \hat{r}) \wedge \mathsf{apply\_subst\_var}(\sigma, n) = \mathsf{first}(s)$. By Rule $4''$ we can now perform the following case analysis.

**Case 2.1:** $\mathsf{rewrites\_rule}(\mathsf{apply\_subst\_var}(\sigma, n), \mathsf{first}(s), \hat{l}, \hat{r}), \mathsf{apply\_subst}(\sigma, t) = \mathsf{tail}(s)$

We omit the premises to ease readability. Then the first part of the disjunction of the conclusion can be deleted, i.e. by Rule $4''$, the conjecture can be transformed into

$$\mathsf{rewrite}^*\mathsf{\_exists}(s, \mathsf{apply\_subst\_list}(\mathsf{all\_reductions}(\sigma, \hat{l}, \hat{r}), l), \hat{l}, \hat{r}).$$

By (289), (261), (262), (175), (205), and (343), Rule $4''$ transforms this conjecture into

$$\mathsf{rewrites\_rule*}(s, \mathsf{apply\_subst}(\mathsf{replace}(\sigma, n, \mathsf{first}(s)), l), \hat{l}, \hat{r}).$$

This can be transformed into (329), (187),

$$\mathsf{rewrites\_rule*}(\mathsf{first}(s), \mathsf{apply\_subst}(\mathsf{replace}(\sigma, n, \mathsf{first}(s)), \mathsf{first}(l)), \hat{l}, \hat{r})$$

(this can be proved by (55) and symbolic evaluation), and

$$\mathsf{rewrites\_rule*}(\mathsf{tail}(s), \mathsf{apply\_subst}(\mathsf{replace}(\sigma, n, \mathsf{first}(s)), \mathsf{tail}(l)), \hat{l}, \hat{r}).$$

Using (55) this can be transformed into

$$\mathsf{rewrites\_rule*}(\mathsf{apply\_subst}(\sigma, t), \mathsf{apply\_subst}(\mathsf{replace}(\sigma, n, \mathsf{first}(s)), t), \hat{l}, \hat{r}),$$

which can be proved by (289), (352), (205), (362), and (363).

**Case 2.2:** $\mathsf{rewrites\_rule}(\mathsf{apply\_subst}(\sigma, t), \mathsf{tail}(s), \hat{l}, \hat{r}) \wedge \mathsf{apply\_subst\_var}(\sigma, n) = \mathsf{first}(s)$

We have to prove

$$\mathsf{is\_subst}(\sigma) \wedge \ldots \wedge r' = \mathsf{rename}(\hat{r}, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l)))) \wedge \bar{l} = \mathsf{apply\_subst}(\sigma, l) \wedge \mathsf{rewrites\_rule}(\bar{l}, s, \hat{l}, \hat{r}) \Rightarrow \ldots$$

We now apply an induction w.r.t. $\mathsf{rewrites\_rule}$ where all cases are either trivial or can be solved as in Case 2.1 except the one where $\bar{l} = \mathsf{var}(m, t')$ (here, the premises imply $t' = \mathsf{apply\_subst}(\sigma, t)$). Now (omitting unnecessary premises) the induction conclusion is symbolically evaluated to

$$\begin{aligned}
&\mathsf{is_subst}(\sigma) \wedge \mathsf{subseteq}(\mathsf{vars}(r'), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{var}(n, \mathsf{vars}(t))), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \\
&\wedge \mathsf{length}(r') = \mathsf{s}(0) \wedge \mathsf{rewrites_rule}(\mathsf{apply_subst}(\sigma, t), \mathsf{tail}(s), \hat{l}, \hat{r}) \;\Rightarrow \\
&\quad \mathsf{special}(\mathsf{appendterm}(\sigma, \mathsf{rewrites_matcher}(\mathsf{apply_subst}(\sigma, t), \mathsf{tail}(s), l', r')), s, \mathsf{add_narrowlist}(\mathsf{var}(n, \mathsf{e}), \\
&\qquad\qquad \mathsf{narrow}(t, l', r'))) \\
&\quad \vee \mathsf{rewrite}^*\mathsf{_exists}(s, \mathsf{apply_subst_list}(\mathsf{all_reductions}(\sigma, \hat{l}, \hat{r}), \mathsf{var}(n, t)), \hat{l}, \hat{r})
\end{aligned}$$

and the induction hypothesis is

$$\begin{array}{l}\quad \mathsf{is\_subst}(\sigma) \wedge \mathsf{subseteq}(\mathsf{vars}(r'), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{vars}(t), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \\ \wedge\, \mathsf{length}(r') = \mathsf{s}(0) \wedge \mathsf{rewrites\_rule}(\mathsf{apply\_subst}(\sigma, t), \mathsf{tail}(s), \hat{l}, \hat{r}) \;\Rightarrow \\ \qquad \mathsf{special}(\mathsf{appendterm}(\sigma, \mathsf{rewrites\_matcher}(\mathsf{apply\_subst}(\sigma, t), \mathsf{tail}(s), l', r')), \mathsf{tail}(s), \\ \qquad\qquad \mathsf{narrow}(t, l', r')) \\ \quad\; \vee\, \mathsf{rewrite^*\_exists}(\mathsf{tail}(s), \mathsf{apply\_subst\_list}(\mathsf{all\_reductions}(\sigma, \hat{l}, \hat{r}), t), \hat{l}, \hat{r}).\end{array}$$

Using Rules 4″ and 5″, this induction formula can be transformed into (55), (61), (87), (91), (121), (378), (175), (373), (372), (280), (183), (206), (207), (348), and (362).

**Case 3:** $l = \mathsf{func}(n, u, t)$

In this case, $\mathsf{apply\_subst}(\sigma, l)$ can be evaluated to $\mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{apply\_subst}(\sigma, t))$. We again perform an induction w.r.t. rewrites_rule and consider the different cases according to the algorithm rewrites_rule.

**Case 3.1:** $\mathsf{eqterm}(\mathsf{first}(s), \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e})), \mathsf{rewrites\_rule}(\mathsf{apply\_subst}(\sigma, t), \mathsf{tail}(s), \hat{l}, \hat{r})$

This case has some similarities with Case 2.2. Omitting unnecessary premises, the induction conclusion can be evaluated to

$$\begin{array}{l}\quad \mathsf{is\_subst}(\sigma) \wedge \mathsf{subseteq}(\mathsf{vars}(r'), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{appendterm}(\mathsf{vars}(u), \mathsf{vars}(t)), \mathsf{vars}(l')) \\ \wedge\, \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \wedge \mathsf{length}(r') = \mathsf{s}(0) \wedge \mathsf{rewrites\_rule}(\mathsf{apply\_subst}(\sigma, t), \mathsf{tail}(s), \hat{l}, \hat{r}) \;\Rightarrow \\ \qquad \mathsf{special}(\mathsf{appendterm}(\sigma, \mathsf{rewrites\_matcher}(\mathsf{apply\_subst}(\sigma, t), \mathsf{tail}(s), l', r'), s, \mathsf{narrow}(\mathsf{func}(n, u, t), l', r')) \\ \quad\; \vee\, \mathsf{rewrite^*\_exists}(s, \mathsf{apply\_subst\_list}(\mathsf{all\_reductions}(\sigma, \hat{l}, \hat{r}), \mathsf{func}(n, u, t)), \hat{l}, \hat{r})\end{array}$$

and the induction hypothesis is

$$\begin{array}{l}\quad \mathsf{is\_subst}(\sigma) \wedge \mathsf{subseteq}(\mathsf{vars}(r'), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{vars}(t), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \\ \wedge\, \mathsf{length}(r') = \mathsf{s}(0) \wedge \mathsf{rewrites\_rule}(\mathsf{apply\_subst}(\sigma, t), \mathsf{tail}(s), \hat{l}, \hat{r}) \;\Rightarrow \\ \qquad \mathsf{special}(\mathsf{appendterm}(\sigma, \mathsf{rewrites\_matcher}(\mathsf{apply\_subst}(\sigma, t), \mathsf{tail}(s), l', r')), \mathsf{tail}(s), \mathsf{narrow}(t, l', r')) \\ \quad\; \vee\, \mathsf{rewrite^*\_exists}(\mathsf{tail}(s), \mathsf{apply\_subst\_list}(\mathsf{all\_reductions}(\sigma, \hat{l}, \hat{r}), t), \hat{l}, \hat{r}).\end{array}$$

Rule 4″ and Rule 5″ transform this conjecture into (55), (61), (100), (91), (121), (375), (163), (378), (175), (373), (372), (280), (183), (206), (207), (348), and (362).

**Case 3.2:** $\mathsf{first\_is\_func}(s), \mathsf{eq}(\mathsf{func\_name}(s), n), \mathsf{eqterm}(\mathsf{tail}(s)), \mathsf{apply\_subst}(\sigma, t),$ $\mathsf{rewrites\_rule}(\mathsf{apply\_subst}(\sigma, u), \mathsf{func\_args}(s), \hat{l}, \hat{r})$

Omitting unnecessary premises, the induction conclusion can be evaluated to

$$\begin{array}{l}\quad \mathsf{is\_subst}(\sigma) \wedge \mathsf{subseteq}(\mathsf{vars}(r'), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{appendterm}(\mathsf{vars}(u), \mathsf{vars}(t)), \mathsf{vars}(l')) \\ \wedge\, \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \wedge \mathsf{length}(r') = \mathsf{s}(0) \wedge \mathsf{rewrites\_rule}(\mathsf{apply\_subst}(\sigma, u), \mathsf{func\_args}(s), \hat{l}, \hat{r}) \;\Rightarrow \\ \qquad \mathsf{special}(\mathsf{appendterm}(\sigma, \mathsf{rewrites\_matcher}(\mathsf{apply\_subst}(\sigma, u), \mathsf{func\_args}(s), l', r')), s, \\ \qquad\qquad \mathsf{narrow}(\mathsf{func}(n, u, t), l', r')) \\ \quad\; \vee\, \mathsf{rewrite^*\_exists}(s, \mathsf{apply\_subst\_list}(\mathsf{all\_reductions}(\sigma, \hat{l}, \hat{r}), \mathsf{func}(n, u, t)), \hat{l}, \hat{r})\end{array}$$

and the induction hypothesis is

$$\begin{array}{l}\quad \mathsf{is\_subst}(\sigma) \wedge \mathsf{subseteq}(\mathsf{vars}(r'), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{vars}(u), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \\ \wedge\, \mathsf{length}(r') = \mathsf{s}(0) \wedge \mathsf{rewrites\_rule}(\mathsf{apply\_subst}(\sigma, u), \mathsf{func\_args}(s), \hat{l}, \hat{r}) \;\Rightarrow \\ \qquad \mathsf{special}(\mathsf{appendterm}(\sigma, \mathsf{rewrites\_matcher}(\mathsf{apply\_subst}(\sigma, u), \mathsf{func\_args}(s), l', r')), \mathsf{func\_args}(s), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{narrow}(u, l', r')) \\ \quad\; \vee\, \mathsf{rewrite^*\_exists}(\mathsf{func\_args}(s), \mathsf{apply\_subst\_list}(\mathsf{all\_reductions}(\sigma, \hat{l}, \hat{r}), u), \hat{l}, \hat{r}).\end{array}$$

This conjecture can be transformed into (55), (68), (61), (98), (91), (121), (374), (376), (379), (175), (373), (372), (280), (183), (349), and (362).

**Case 3.3:** $\mathsf{matches}(\hat{l}, \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e})), \mathsf{eqterm}(\mathsf{first}(s), \mathsf{apply\_subst}(\mathsf{matcher}(\hat{l}, \mathsf{func}(n, u, \mathsf{e})), \hat{r})),$ $\mathsf{eqterm}(\mathsf{tail}(s), \mathsf{apply\_subst}(\sigma, t))$

In this case, we use Rule 4$''$ to omit the second part of the disjunction, i.e. we obtain the following conjecture after symbolic evaluation

$$\begin{aligned}&\mathsf{is\_subst}(\sigma) \wedge \mathsf{no\_duplicates}(\sigma) \wedge \mathsf{subseteq}(\mathsf{vars}(r'), \mathsf{vars}(l')) \wedge \mathsf{length}(r') = \mathsf{s}(0) \wedge\\ &\mathsf{disjoint}(\mathsf{appendterm}(\mathsf{vars}(u), \mathsf{vars}(t)), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \wedge\\ &l' = \mathsf{rename}(\hat{l}, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l)))) \wedge r' = \mathsf{rename}(\hat{r}, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l)))) \;\Rightarrow\\ &\quad \mathsf{special}(\mathsf{appendterm}(\sigma, \mathsf{matcher}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e}))), s, \mathsf{narrow}(\mathsf{func}(n, u, t), l', r')).\end{aligned}$$

Using Rule 4$''$ and (241), (242), (374), (175), (373), (372), (130), (280), (183), (376), (91), (121), it can be transformed into

$$\begin{aligned}&\mathsf{matches}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e})) \wedge \mathsf{disjoint}(\mathsf{vars}(u), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \Rightarrow\\ &\quad \mathsf{unifies}(l', \mathsf{func}(n, u, \mathsf{e}))\end{aligned}$$

(which is implied by an instantiation of (248)) and

$$\begin{aligned}&\mathsf{no\_duplicates}(\sigma) \wedge \mathsf{matches}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e})) \wedge \mathsf{subseteq}(\mathsf{vars}(r'), \mathsf{vars}(l')) \wedge\\ &\mathsf{disjoint}(\mathsf{appendterm}(\mathsf{vars}(u), \mathsf{vars}(t)), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \Rightarrow\\ &\mathsf{special}(\mathsf{appendterm}(\sigma, \mathsf{matcher}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e}))),\\ &\qquad s,\\ &\qquad \mathsf{add}(\mathsf{mgu}(l', \mathsf{func}(n, u, \mathsf{e})),\\ &\qquad\quad \mathsf{add}(\mathsf{addterm}(\mathsf{apply\_subst}(\mathsf{mgu}(l', \mathsf{func}(n, u, \mathsf{e})), r'), \mathsf{apply\_subst}(\mathsf{mgu}(l', \mathsf{func}(n, u, \mathsf{e})), t)), \mathsf{empty})))\end{aligned}$$

(because the third argument of $\mathsf{special}$ results from evaluation of $\mathsf{back\_narrowlist}(\mathsf{add}(\mathsf{mgu}(l', \mathsf{func}(n, u, \mathsf{e})),$ $\mathsf{add}(\mathsf{apply\_subst}(\mathsf{mgu}(l', \mathsf{func}(n, u, \mathsf{e})), r'), \mathsf{empty})), t)))$. By (381) this can be transformed into

$$\begin{aligned}&\mathsf{no\_duplicates}(\sigma) \wedge \mathsf{matches}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e})) \wedge \mathsf{disjoint}(\mathsf{vars}(u), \mathsf{vars}(l'))\\ &\wedge \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \Rightarrow\\ &\quad \mathsf{appendterm}(\sigma, \mathsf{matcher}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e}))) =\\ &\quad \mathsf{compose}(\mathsf{mgu}(l', \mathsf{func}(n, u, \mathsf{e})), \mathsf{appendterm}(\sigma, \mathsf{matcher}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e}))))\end{aligned}$$

and

$$\begin{aligned}&\mathsf{matches}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e})) \wedge \mathsf{subseteq}(\mathsf{vars}(r'), \mathsf{vars}(l'))\\ &\wedge \mathsf{disjoint}(\mathsf{appendterm}(\mathsf{vars}(u), \mathsf{vars}(t)), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \Rightarrow\\ &\quad s = \mathsf{apply\_subst}(\mathsf{appendterm}(\sigma, \mathsf{matcher}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e}))),\\ &\qquad\qquad \mathsf{addterm}(\mathsf{apply\_subst}(\mathsf{mgu}(l', \mathsf{func}(n, u, \mathsf{e})), r'), \mathsf{apply\_subst}(\mathsf{mgu}(l', \mathsf{func}(n, u, \mathsf{e})), t))).\end{aligned}$$

Using (260), the first conjecture is transformed into

$$\begin{aligned}&\mathsf{matches}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e})) \wedge \mathsf{disjoint}(\mathsf{vars}(u), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \Rightarrow\\ &\quad \mathsf{appendterm}(\sigma, \mathsf{matcher}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e})))(l') =\\ &\quad \mathsf{appendterm}(\sigma, \mathsf{matcher}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e})))(\mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e}))\end{aligned}$$

which is in turn transformed into instantiations of (232), (233), (237), (121), (91), and (239).
Using (184), (61), and (55), the second conjecture above can be transformed into two new conjectures

$$\begin{aligned}&\mathsf{matches}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e})) \wedge \mathsf{subseteq}(\mathsf{vars}(r'), \mathsf{vars}(l'))\\ &\wedge \mathsf{disjoint}(\mathsf{appendterm}(\mathsf{vars}(u), \mathsf{vars}(t)), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \Rightarrow\\ &\quad \mathsf{apply\_subst}(\mathsf{matcher}(l', \mathsf{func}(n, u, \mathsf{e})), r') =\\ &\quad \mathsf{apply\_subst}(\mathsf{appendterm}(\sigma, \mathsf{matcher}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e}))), \mathsf{apply\_subst}(\mathsf{mgu}(l', \mathsf{func}(n, u, \mathsf{e})), r'))\end{aligned}$$

and

$$\begin{aligned}&\mathsf{matches}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e})) \wedge \mathsf{disjoint}(\mathsf{appendterm}(\mathsf{vars}(u), \mathsf{vars}(t)), \mathsf{vars}(l')) \wedge\\ &\mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \Rightarrow\\ &\quad \mathsf{apply\_subst}(\sigma, t) = \mathsf{apply\_subst}(\mathsf{appendterm}(\sigma, \mathsf{matcher}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e}))),\\ &\qquad\qquad \mathsf{apply\_subst}(\mathsf{mgu}(l', \mathsf{func}(n, u, \mathsf{e})), t)).\end{aligned}$$

By using (254) and the original first conjecture, these conjectures can be transformed into

$$\begin{aligned}&\mathsf{matches}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e})) \wedge \mathsf{subseteq}(\mathsf{vars}(r'), \mathsf{vars}(l'))\\ &\wedge \mathsf{disjoint}(\mathsf{appendterm}(\mathsf{vars}(u), \mathsf{vars}(t)), \mathsf{vars}(l')) \wedge \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \Rightarrow\\ &\quad \mathsf{apply\_subst}(\mathsf{matcher}(l', \mathsf{func}(n, u, \mathsf{e})), r') =\\ &\quad \mathsf{apply\_subst}(\mathsf{appendterm}(\sigma, \mathsf{matcher}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e}))), r')\end{aligned}$$

and

$$\begin{aligned}&\mathsf{matches}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e})) \wedge \mathsf{disjoint}(\mathsf{appendterm}(\mathsf{vars}(u), \mathsf{vars}(t)), \mathsf{vars}(l'))\\ &\wedge \mathsf{disjoint}(\mathsf{dom}(\sigma), \mathsf{vars}(l')) \Rightarrow\\ &\quad \mathsf{apply\_subst}(\sigma, t) = \mathsf{apply\_subst}(\mathsf{appendterm}(\sigma, \mathsf{matcher}(l', \mathsf{func}(n, \mathsf{apply\_subst}(\sigma, u), \mathsf{e}))), t).\end{aligned}$$

Now both these conjectures again follow from (232), (233), (237), (121), (91), and (239).

## 9.2 Critical Pair Lemma (pc)

In this section we prove (a variant of) the critical pair lemma of Knuth and Bendix [14] which states that if all critical pairs of a TRS are joinable, then the TRS is locally confluent[2].

$$\mathsf{trs}(R) \wedge \mathsf{jcp}(R) \wedge \mathsf{rewrites}(r, s, R) \wedge \mathsf{rewrites}(r, t, R) \;\Rightarrow\; \mathsf{joinable}(s, t, R). \tag{402}$$

By Rule 4$''$, (402) is transformed into instantiations of (278), (262), and a version of (402) where $\mathsf{rewrites}(r, s, R)$ is replaced by $\mathsf{rewrites\_rule}(r, s, \mathsf{first}(\mathsf{rule}(r, s, R)), \mathsf{second}(\mathsf{rule}(r, s, R)))$ (and a similar replacement is done for $\mathsf{rewrites}(r, s, R)$). By another application of Rule 4$''$ we obtain (279) and a modified version of the above conjecture, where $\mathsf{in}(\mathsf{first}(\mathsf{rule}(r, s, R)), \mathsf{second}(\mathsf{rule}(r, s, R)), R)$ and the similar conjecture for $r$ and $t$ are added as additional conjuncts in the premise. Now four application of Rule 5$''$ (to generalize the terms $\mathsf{first}(\mathsf{rule}(r, s, R))$ and $\mathsf{second}(\mathsf{rule}(r, s, R))$ etc.) results in

$$\begin{aligned}&\mathsf{trs}(R) \wedge \mathsf{jcp}(R) \wedge \mathsf{in}(l_1, r_1, R) \wedge \mathsf{in}(l_2, r_2, R) \wedge\\ &\qquad \mathsf{rewrites\_rule}(r, s, l_1, r_1) \wedge \mathsf{rewrites\_rule}(r, t, l_2, r_2) \quad\Rightarrow\quad \mathsf{joinable}(s, t, R).\end{aligned} \tag{403}$$

We prove (403) by induction w.r.t. the algorithm $\mathsf{rewrites\_rule}$ (i.e. w.r.t. the merged induction schemes of $\mathsf{rewrites\_rule}(r, s, l_1, r_1)$ and $\mathsf{rewrites\_rule}(r, t, l_2, r_2)$). So we apply Rule 1$''$ for the proof of (403) and subsequently we decompose the resulting induction formulas using Rule 6$''$ for case analyses. In the following we will consider the resulting conjectures. First note that in all cases where $\mathsf{rewrites\_rule}(r, s, l_1, r_1)$ or $\mathsf{rewrites\_rule}(r, t, l_2, r_2)$ can be symbolically evaluated to $\mathsf{false}$ (using Rule 3$''$) the conjecture reduces to a tautology (provable with Rule 4$''$). Therefore in the following we will only consider the other remaining cases. Moreover, to ease readability we will omit the premises $\mathsf{trs}(R) \wedge \mathsf{jcp}(R) \wedge \mathsf{in}(l_1, r_1, R) \wedge \mathsf{in}(l_2, r_2, R)$ as they remain unchanged in all induction conclusions and hypotheses.

**Case 1:** $r = \mathsf{var}(n, r'), \mathsf{eqterm}(\mathsf{first}(s), \mathsf{var}(n, \mathsf{e})) = \mathsf{true}, \mathsf{eqterm}(\mathsf{first}(t), \mathsf{var}(n, \mathsf{e})) = \mathsf{true}$

Here, symbolic evaluation transforms the induction conclusion into

$$\mathsf{rewrites\_rule}(r', \mathsf{tail}(s), l_1, r_1) \wedge \mathsf{rewrites\_rule}(r', \mathsf{tail}(t), l_2, r_2) \Rightarrow \mathsf{joinable}(s, t, R)$$

and the induction hypothesis is

$$\mathsf{rewrites\_rule}(r', \mathsf{tail}(s), l_1, r_1) \wedge \mathsf{rewrites\_rule}(r', \mathsf{tail}(t), l_2, r_2) \Rightarrow \mathsf{joinable}(\mathsf{tail}(s), \mathsf{tail}(t), R).$$

Now Rule 4$''$ transforms this conjecture into instantiations of (55), (382), (388), and (389).

**Case 2:** $r = \mathsf{func}(n, u, r'), \mathsf{eqterm}(\mathsf{first}(s), \mathsf{func}(n, u, \mathsf{e})) = \mathsf{true}, \mathsf{eqterm}(\mathsf{first}(t), \mathsf{func}(n, u, \mathsf{e})) = \mathsf{true}$

The proof for this case is almost identical to Case 1.

---

[2]This formulation is slightly different from the one in [9, Section 7], because in [9] we omitted the data type $\mathsf{tll}$ for the sake of brevity.

**Case 3:** $r = \mathsf{func}(n, u, r'), \mathsf{eqterm}(\mathsf{first}(s), \mathsf{func}(n, u, \mathsf{e})) = \mathsf{true}, \mathsf{first\_is\_func}(t), \mathsf{eq}(\mathsf{func\_name}(t), n),$ $\mathsf{eqterm}(\mathsf{tail}(t), r'), \mathsf{rewrites\_rule}(u, \mathsf{func\_args}(t), l_2, r_2)$

Using symbolic evaluation and conjecture (55) (by Rule 4″), the induction conclusion is transformed into

$$\mathsf{rewrites\_rule}(\mathsf{tail}(t), \mathsf{tail}(s), l_1, r_1) \wedge \mathsf{rewrites\_rule}(u, \mathsf{func\_args}(t), l_2, r_2) \Rightarrow \mathsf{joinable}(s, t, R).$$

By conjecture (274) and (262), $\mathsf{rewrites\_rule}(u, \mathsf{func\_args}(t), l_2, r_2)$ can be transformed into $\mathsf{rewrites\_rule}(\mathsf{func}(n, u, \mathsf{e}), \mathsf{func}(n, \mathsf{func\_args}(t), \mathsf{e}), l_2, r_2)$. Hence, by Rule 4″ we can drop the induction hypothesis and by (55) and (68) we obtain

$$\mathsf{rewrites\_rule}(\mathsf{tail}(t), \mathsf{tail}(s), l_1, r_1) \wedge \mathsf{rewrites\_rule}(\mathsf{first}(s), \mathsf{first}(t), l_2, r_2) \Rightarrow \mathsf{joinable}(s, t, R).$$

This follows from conjecture (270), (271), (338), (339), and (397).

**Case 4:** $r = \mathsf{func}(n, u, r'), \mathsf{eqterm}(\mathsf{first}(s), \mathsf{func}(n, u, \mathsf{e})) = \mathsf{true}, \mathsf{matches}(l_2, \mathsf{func}(n, u, \mathsf{e})),$ $\mathsf{eqterm}(\mathsf{first}(t), \mathsf{apply\_subst}(\mathsf{matcher}(l_2, \mathsf{func}(n, u, \mathsf{e})), r_2)), \mathsf{eqterm}(\mathsf{tail}(t), r')$

Using symbolic evaluation and conjecture (55) (by Rule 4″), the induction conclusion is transformed into

$$\mathsf{rewrites\_rule}(\mathsf{tail}(t), \mathsf{tail}(s), l_1, r_1) \wedge \mathsf{rewrites\_rule}(\mathsf{first}(s), \mathsf{first}(t), l_2, r_2) \Rightarrow \mathsf{joinable}(s, t, R)$$

and

$$\mathsf{matches}(l_2, \mathsf{func}(n, u, \mathsf{e})) \Rightarrow \mathsf{rewrites\_rule}(\mathsf{func}(n, u, \mathsf{e}), \mathsf{apply\_subst}(\mathsf{matcher}(l_2, \mathsf{func}(n, u, \mathsf{e})), r_2), l_2, r_2).$$

The first conjecture again follows from (270), (271), (338), (339), (397) and the second conjecture can be proved by symbolic evaluation and the conjectures (69), (193), (63).

**Case 5:** $r = \mathsf{func}(n, u, r'), \mathsf{first\_is\_func}(s), \mathsf{eq}(\mathsf{func\_name}(s), n), \mathsf{eqterm}(\mathsf{tail}(s), r'),$ $\mathsf{rewrites\_rule}(u, \mathsf{func\_args}(s), l_1, r_1), \mathsf{eqterm}(\mathsf{first}(t), \mathsf{func}(n, u, \mathsf{e}))$

This case is similar to Case 3.

**Case 6:** $r = \mathsf{func}(n, u, r'), \mathsf{first\_is\_func}(s), \mathsf{eq}(\mathsf{func\_name}(s), n), \mathsf{eqterm}(\mathsf{tail}(s), r'),$ $\mathsf{rewrites\_rule}(u, \mathsf{func\_args}(s), l_1, r_1), \mathsf{first\_is\_func}(t), \mathsf{eq}(\mathsf{func\_name}(t), n), \mathsf{eqterm}(\mathsf{tail}(t), r')$

Symbolic evaluation transforms the induction conclusion into

$$\mathsf{rewrites\_rule}(u, \mathsf{func\_args}(s), l_1, r_1) \wedge \mathsf{rewrites\_rule}(u, \mathsf{func\_args}(t), l_2, r_2) \Rightarrow \mathsf{joinable}(s, t, R)$$

and the induction hypothesis is

$$\mathsf{rewrites\_rule}(u, \mathsf{func\_args}(s), l_1, r_1) \wedge \mathsf{rewrites\_rule}(u, \mathsf{func\_args}(t), l_2, r_2) \Rightarrow \\ \mathsf{joinable}(\mathsf{func\_args}(s), \mathsf{func\_args}(t), R).$$

Now Rule 4″ transforms this conjecture into instantiations of (55), (68), (382), (390), (391), and (392).

**Case 7:** $r = \mathsf{func}(n, u, r'), \mathsf{first\_is\_func}(s), \mathsf{eq}(\mathsf{func\_name}(s), n),$ $\mathsf{eqterm}(\mathsf{tail}(s), r'), \mathsf{rewrites\_rule}(u, \mathsf{func\_args}(s), l_1, r_1), \mathsf{matches}(l_2, \mathsf{func}(n, u, \mathsf{e})),$ $\mathsf{eqterm}(\mathsf{first}(t), \mathsf{apply\_subst}(\mathsf{matcher}(l_2, \mathsf{func}(n, u, \mathsf{e})), r_2)), \mathsf{eqterm}(\mathsf{tail}(t), r')$

By symbolic evaluation and Rule 4″ (using conjectures (55), (391), (180)), the induction conclusion is transformed into

$$\mathsf{tail}(s) = \mathsf{tail}(t) \wedge \mathsf{joinable}(\mathsf{first}(s), \mathsf{first}(t), R) \Rightarrow \mathsf{joinable}(s, t, R),$$

(which follows from conjecture (382) and (390)),

$$\begin{array}{l}
\mathsf{in}(\mathsf{first}(t), \mathsf{first}(s), \\
\quad \mathsf{apply_subst}(\mathsf{appendterm}(\mathsf{matcher}(l_2, \mathsf{func}(n, u, \mathsf{e})), \mathsf{rewrites_matcher}(\mathsf{func}(n, u, \mathsf{e}), \\
\qquad \mathsf{first}(s), \\
\qquad \mathsf{rename}(l_1, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l_2)))), \\
\qquad \mathsf{rename}(r_1, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l_2))))))), \\
\qquad \mathsf{cp_rule}(l_2, r_2, l_1, r_1)) \\
\Rightarrow \mathsf{joinable}(\mathsf{first}(s), \mathsf{first}(t), R)
\end{array}$$

(which can be generalized and then proved by conjectures (400), (395), (383), (385), (263), (169), (275), (276), (175), (280), (183)),

$$\begin{array}{l}
\mathsf{rewrites_rule}(\mathsf{apply_subst}(\mathsf{matcher}(l_2, \mathsf{func}(n, u, \mathsf{e})), l_2), \mathsf{first}(s), l_1, r_1) \Rightarrow \\
\quad \mathsf{in}(\mathsf{apply_subst}(\mathsf{matcher}(l_2, \mathsf{func}(n, u, \mathsf{e})), r_2), \mathsf{first}(s), \\
\qquad \mathsf{apply_subst}(\mathsf{appendterm}(\mathsf{matcher}(l_2, \mathsf{func}(n, u, \mathsf{e})), \\
\qquad\quad \mathsf{rewrites_matcher}(\mathsf{apply_subst}(\mathsf{matcher}(l_2, \mathsf{func}(n, u, \mathsf{e})), l_2), \\
\qquad\qquad \mathsf{first}(s), \\
\qquad\qquad \mathsf{rename}(l_1, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l_2)))), \\
\qquad\qquad \mathsf{rename}(r_1, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l_2))))))), \\
\qquad \mathsf{cp_rule}(l_2, r_2, l_1, r_1)) \\
\quad \vee \mathsf{joinable}(\mathsf{first}(s), \mathsf{apply_subst}(\mathsf{matcher}(l_2, \mathsf{func}(n, u, \mathsf{e})), r_2), R)
\end{array}$$

(which (after using the conjectures (169), (237), (121), (69), (236), (246), and (399)) can be generalized to conjecture (401) from Sect. 9.1),

$$\mathsf{matches}(l_2, \mathsf{func}(n, u, \mathsf{e})) \Rightarrow \mathsf{func}(n, u, \mathsf{e}) = \mathsf{apply\_subst}(\mathsf{matcher}(l_2, \mathsf{func}(n, u, \mathsf{e})), l_2)$$

(which can be generalized to conjecture (239)), and

$$\mathsf{rewrites\_rule}(\mathsf{func}(n, u, \mathsf{e}), \mathsf{first}(s), l_1, r_1)$$

(which can be proved by symbolic evaluation and by Rule 4$''$ under the above premises).

**Case 8:** $r = \mathsf{func}(n, u, r'), \mathsf{matches}(l_1, \mathsf{func}(n, u, \mathsf{e})),$
$\mathsf{eqterm}(\mathsf{first}(s), \mathsf{apply\_subst}(\mathsf{matcher}(l_1, \mathsf{func}(n, u, \mathsf{e})), r_1)),$
$\mathsf{eqterm}(\mathsf{tail}(s), r'), \mathsf{eqterm}(\mathsf{first}(t), \mathsf{func}(n, u, \mathsf{e}))$

This case is similar to Case 4.

**Case 9:** $r = \mathsf{func}(n, u, r'), \mathsf{matches}(l_1, \mathsf{func}(n, u, \mathsf{e})),$
$\mathsf{eqterm}(\mathsf{first}(s), \mathsf{apply\_subst}(\mathsf{matcher}(l_1, \mathsf{func}(n, u, \mathsf{e})), r_1)),$
$\mathsf{eqterm}(\mathsf{tail}(s), r'), \mathsf{first\_is\_func}(t), \mathsf{eq}(\mathsf{func\_name}(t), n), \mathsf{eqterm}(\mathsf{tail}(t), r'),$
$\mathsf{rewrites\_rule}(u, \mathsf{func\_args}(t), l_2, r_2)$

This case is similar to Case 7.

**Case 10:** $r = \mathsf{func}(n, u, r'), \mathsf{matches}(l_1, \mathsf{func}(n, u, \mathsf{e})),$
$\mathsf{eqterm}(\mathsf{first}(s), \mathsf{apply\_subst}(\mathsf{matcher}(l_1, \mathsf{func}(n, u, \mathsf{e})), r_1)),$
$\mathsf{eqterm}(\mathsf{tail}(s), r'), \mathsf{matches}(l_2, \mathsf{func}(n, u, \mathsf{e})),$
$\mathsf{eqterm}(\mathsf{first}(t), \mathsf{apply\_subst}(\mathsf{matcher}(l_2, \mathsf{func}(n, u, \mathsf{e})), r_2))$

This case has some similarity to Case 7, too (i.e. it generates almost the same subgoals). By symbolic evaluation and Rule 4$''$ (using conjecture (55), (391), (180)), the conjecture is transformed into

$$\mathsf{tail}(s) = \mathsf{tail}(t) \wedge \mathsf{joinable}(\mathsf{first}(s), \mathsf{first}(t), R) \Rightarrow \mathsf{joinable}(s, t, R),$$

$$
\begin{aligned}
&\mathsf{in}(\mathsf{first}(t), \mathsf{first}(s),\\
&\quad \mathsf{apply\_subst}(\mathsf{appendterm}(\mathsf{matcher}(l_2, \mathsf{func}(n,u,\mathsf{e})), \mathsf{rewrites\_matcher}(\mathsf{func}(n,u,\mathsf{e}),\\
&\qquad\qquad \mathsf{first}(s),\\
&\qquad\qquad \mathsf{rename}(l_1, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l_2)))),\\
&\qquad\qquad \mathsf{rename}(r_1, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l_2))))))),\\
&\qquad \mathsf{cp\_rule}(l_2, r_2, l_1, r_1))\\
&\Rightarrow \mathsf{joinable}(\mathsf{first}(s), \mathsf{first}(t), R),
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{rewrites\_rule}(\mathsf{apply\_subst}(\mathsf{matcher}(l_2, \mathsf{func}(n,u,\mathsf{e})), l_2), \mathsf{first}(s), l_1, r_1) \Rightarrow\\
&\quad \mathsf{in}(\mathsf{apply\_subst}(\mathsf{matcher}(l_2, \mathsf{func}(n,u,\mathsf{e})), r_2), \mathsf{first}(s),\\
&\qquad \mathsf{apply\_subst}(\mathsf{appendterm}(\mathsf{matcher}(l_2, \mathsf{func}(n,u,\mathsf{e})),\\
&\qquad\qquad \mathsf{rewrites\_matcher}(\mathsf{apply\_subst}(\mathsf{matcher}(l_2, \mathsf{func}(n,u,\mathsf{e})), l_2),\\
&\qquad\qquad\quad \mathsf{first}(s),\\
&\qquad\qquad\quad \mathsf{rename}(l_1, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l_2)))),\\
&\qquad\qquad\quad \mathsf{rename}(r_1, \mathsf{s}(\mathsf{max}(\mathsf{vars}(l_2))))))),\\
&\qquad \mathsf{cp\_rule}(l_2, r_2, l_1, r_1))\\
&\quad \vee\, \mathsf{joinable}(\mathsf{first}(s), \mathsf{apply\_subst}(\mathsf{matcher}(l_2, \mathsf{func}(n,u,\mathsf{e})), r_2), R),
\end{aligned}
$$

$$\mathsf{matches}(l_2, \mathsf{func}(n,u,\mathsf{e})) \Rightarrow \mathsf{func}(n,u,\mathsf{e}) = \mathsf{apply\_subst}(\mathsf{matcher}(l_2, \mathsf{func}(n,u,\mathsf{e})), l_2),$$

$$\mathsf{matches}(l_1, \mathsf{func}(n,u,\mathsf{e})) \Rightarrow \mathsf{func}(n,u,\mathsf{e}) = \mathsf{apply\_subst}(\mathsf{matcher}(l_1, \mathsf{func}(n,u,\mathsf{e})), l_1),$$

$$\mathsf{rewrites\_rule}(\mathsf{apply\_subst}(\mathsf{matcher}(l_1, \mathsf{func}(n,u,\mathsf{e})), l_1), \mathsf{apply\_subst}(\mathsf{matcher}(l_1, \mathsf{func}(n,u,\mathsf{e})), r_1), l_1, r_1)$$

(this last conjecture can be generalized to (169), (170), and (277)).

# References


[1] A. Bouhoula & M. Rusinowitch. Implicit Induction in Conditional Theories. *Journal of Automated Reasoning*, 14:189-235, 1995.

[2] R. S. Boyer & J S. Moore. *A Computational Logic.* Academic Press, 1979.

[3] J. Brauburger & J. Giesl. Termination Analysis for Partial Functions. In *Proc. 3rd SAS*, Aachen, Germany, LNCS 1145, 1996. Extended version appeared as Technical Report IBN 96/33, TU Darmstadt, Germany.

[4] F. Bronsard, U. S. Reddy, & R. W. Hasker. Induction Using Term Orders. *Journal of Automated Reasoning*, 16:3-37, 1996.

[5] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, & A. Smaill. Rippling: A Heuristic for Guiding Inductive Proofs. *Artificial Intelligence*, 62:185-253, 1993.

[6] M. Dauchet. Simulation of Turing Machines by a Left-Linear Rewrite Rule. In *Proc. RTA '89*, Chapel Hill, NC, LNCS 355, 1989.

[7] J. Giesl. Termination Analysis for Functional Programs using Term Orderings. In *Proc. 2nd SAS*, Glasgow, Scotland, LNCS 983, 1995.

[8] J. Giesl. Termination of Nested and Mutually Recursive Algorithms. *Journal of Automated Reasoning*, 19:1-29, 1997.

[9] J. Giesl. Induction Proofs with Partial Functions. Technical Report IBN 98/48, TU Darmstadt, Germany, 1998.

[10] G. Huet. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. *Journal of the ACM* 27(4):797-821, 1980.

[11] D. Kapur & D. R. Musser. Proof by Consistency. *Artificial Intelligence*, 31:125-157, 1987.

[12] D. Kapur & M. Subramaniam. New Uses of Linear Arithmetic in Automated Theorem Proving by Induction. *Journal of Automated Reasoning*, 16:39-78, 1996.

[13] D. Kapur & M. Subramaniam. Automating Induction over Mutually Recursive Functions. In *Proc. 5th AMAST*, Munich, Germany, LNCS 1101, 1996.

[14] D. E. Knuth & P. B. Bendix. Simple Word Problems in Universal Algebras. In J. Leech (ed.), *Computational Problems in Abstract Algebra*, pp. 263-297, Pergamon Press, Oxford, 1970.

[15] Z. Manna & R. Waldinger. Deductive Synthesis of the Unification Algorithm. *Science of Computer Programming*, 1:5-48, 1981.

[16] T. Nipkow. More Church-Rosser Proofs (in ISABELLE/HOL). In *Proc. CADE-13*, New Brunswick, NJ, LNAI 1104, 1996.

[17] L. C. Paulson. Verifying the Unification Algorithm in LCF. *Science of Computer Programming*, 5:143-169, 1985.

[18] J. A. Robinson. A Machine Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12:23-41, 1965.

[19] N. Shankar. A Mechanical Proof of the Church-Rosser Theorem, *Journal of the ACM*, 35(3):475-522, 1988.

[20] C. Walther. Mathematical Induction. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson (eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming*, vol. 2, Oxford University Press, 1994.

[21] H. Zhang, D. Kapur, & M. S. Krishnamoorthy. A Mechanizable Induction Principle for Equational Specifications. In *Proc. CADE-9*, Argonne, IL, LNCS 310, 1988.

# Contents