# T-String-Unification:
# Unifying Prefixes in Non-Classical Proof Methods

Jens Otten[*]       Christoph Kreitz

*Fachgebiet Intellektik, Fachbereich Informatik*
*Technische Hochschule Darmstadt*
*Alexanderstr. 10, 64283 Darmstadt, Germany*
`{jeotten,kreitz}@intellektik.informatik.th-darmstadt.de`

**Abstract.** For an efficient proof search in non-classical logics, particular in intuitionistic and modal logics, two similar approaches have been established: Wallen's matrix characterization and Ohlbach's resolution calculus. Beside the usual term-unification both methods require a specialized string-unification to unify the so-called prefixes of atomic formulae (in Wallen's notation) or world-paths (in Ohlbach's notation). For this purpose we present an efficient algorithm, called T-String-Unification, which computes a *minimal* set of most general unifiers. By transforming systems of equations we obtain an elegant unification procedure, which is applicable to the intuitionistic logic $\mathcal{J}$ and the modal logic S4. With some modifications we are able to treat the modal logics D, K, D4, K4, S5, and T. We explain our method by an intuitive graphical presentation, prove correctness, completeness, minimality, and termination and investigate its complexity.

## 1 Introduction

Unification is one of the key operations which are used to guide an efficient search for a proof of a theorem in classical predicate logic. Within theorem provers based on the connection method [5, 6, 8, 4], resolution [17, 20], the tableaux calculus [3, 2], and others unification is required for making certain atomic formulae *complementary*. Complementarity is a key concept in the characterization of logical validity. In classical logic two atomic formulae (or atoms) are called complementary if they have the same predicate symbol but different *polarities* (or *signs*) and if their sub-terms can be made identical by some (*first-order-* or *quantifier-*) substitution. This substitution encodes the fact that certain quantified variables occurring in the formula to be proven have to be instantiated by terms in order to complete the proof. It can be computed by well-known algorithms for *term-unification* such as the algorithm of Herbrand & Robinson [7, 17] or Martelli-Montanari [11].

If proof methods like the above shall be extended to non-classical logics such as modal logics or intuitionistic logic it is not sufficient to consider only the terms occurring in atomic formulae. In his matrix characterization of non-classical validity Wallen [19] has shown that in addition the *prefixes* of atomic formulae need to be unified in order to make them complementary where a prefix of an atom essentially describes its position in the formula tree. Similarly Ohlbach's resolution calculi for modal logics [12, 13] require a unification of *world-paths* which characterize the modal context of an atom in a formula. Both conditions are nearly identical[2] and

---

[*] Supported by the Adolf Messer Stiftung
[2] In the rest of this paper we shall follow the Wallen's terminology and notation.

express the peculiarities and restrictions of these logics. In addition to the usual term unification, therefore, proof procedures for non-classical logics will have to incorporate a *prefix unification* algorithm.

Formally, a prefix of an atomic formula is a *string* over some alphabet $\mathcal{A}$ consisting of variables and constants. Two prefix strings $s$ and $t$ can be unified if there is a morphism $\sigma$ which assigns a string to each variable such that $\sigma(s){=}\sigma(t)$. In contrast to the usual term substitution Wallen [19] calls this morphism $\sigma$ a *modal* or *intuitionistic substitution*.

Consider, for instance, the strings $tabUlAR$ and $taSTeFuL$ where capital letters indicate variables and small letters refer to constants. These two strings can be unified by applying the substitution[3] $\sigma = \{U\backslash\varepsilon, A\backslash ea, R\backslash ux, S\backslash b, T\backslash l, F\backslash a, L\backslash x\}$ which yields the string *tableaux* in both cases.

In general, unification with respect to some equational theory $\mathcal{E}$ can be considered as the following problem: given a set of equations $\Gamma = \{s_1{=}t_1, \ldots, s_n{=}t_n\}$ one has to find a substitution $\sigma$ which solves all these equations w.r.t. the theory $\mathcal{E}$, i.e. $\sigma(s_i) =_{\mathcal{E}} \sigma(t_i)$ must hold for all $i$. The substitution $\sigma$ is called a *unifier* of $\Gamma$. It is a *most general unifier*, shortly $\sigma \in \mathrm{mgu}(\Gamma)$, if it is not an instance of some other unifier $\tau$ of $\Gamma$.

The theory $\mathcal{E}$ itself is defined by some set of axioms which have to hold in addition to the usual laws of reflexivity, symmetry, and transitivity. In the case of string unification (the *monoid problem* or *resolution problem for word equations*) the only axiom is the associativity of string concatenation. An algorithm for enumerating the most general unifiers of a set of string equations has first been presented by Plotkin [16] whereas Makanin [9, 10, 1] showed that it is possible to decide whether a set of string equations has a unifier or not. In general, however, the number of most general unifiers of a set of string equations is infinite. For $\Gamma = \{aX{=}Xa\}$, for instance, the set of most general unifiers is $\{\sigma \mid \sigma(X) = a^i, i \in \mathbb{N}_0\}$.

Fortunately, there are certain restrictions on the prefixes of atomic formulae which drastically simplify the unification problem for prefix strings. Prefixes are strings without duplicates and in two prefixes (corresponding to atoms of the same formula) equal characters can occur only within a common substring at the beginning of the two prefixes. These requirements, which we shall call *T-string property*, will allow us to write a much more efficient algorithm for unifying prefix strings. Since two prefixes contain no common variables (except for some identical substring at their beginning) one only has to unify the variables occurring in one string with substrings of the second and vice versa. The number of most general unifiers will become finite, although it may still grow exponentially with the length of the prefixes.

During *T-string unification* it may become necessary to consider additional variables. The strings $aX$ and $Yb$, for instance, can be unified by $\sigma_1 = \{Y\backslash a, X\backslash b\}$ but also by $\sigma_2 = \{Y\backslash ac, X\backslash cb\}$, $\sigma_3 = \{Y\backslash acd, X\backslash cdb\}$, $\sigma_4 = \{Y\backslash acde, X\backslash cdeb\}$, etc. None of these substitutions can be considered most general since they are all instances of $\sigma = \{Y\backslash aZ, X\backslash Zb\}$ where $Z$ is a new variable. This fact is illustrated by the following diagram and has to be taken into account when developing an algorithm for T-string unification.

---

[3] As usual we denote a substitution $\sigma$ by the set $\{x\backslash t \mid \sigma(x){=}t \wedge x{\neq}t\}$.
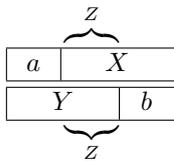
The unification algorithm which we shall present in this paper computes for any set of equations $\Gamma = \{s_1{=}t_1, \ldots, s_n{=}t_n\}$ (where all possible pairs of strings satisfy the T-string property) a *minimal* set of most general unifiers. The algorithm proceeds by transforming a set of equations in a way similar to the algorithm of Martelli & Montanari [11] for term-unification. For this purpose it uses transformation rules which step-wisely convert the set of equations into a set of substitutions which eventually will become the set of most general unifiers. This makes it possible to give an elegant description of our procedure and to adopt it to different modal logics by simple modifications of its transformation rules.

In the following section we shall briefly introduce all the mathematical preliminaries of T-string unification. Section 3 describes a general unification algorithm which can be used in proof procedures for intuitionistic logic and the modal logic S4. We shall present its transformation rules and investigate its computational complexity. In section 4 we shall then study T-string unification for various other modal logics. We conclude with a brief comparison with other approaches and a few remarks on applications and further investigations.

## 2 Preliminaries

In this section we shall briefly describe the basic concepts which are necessary for describing and investigating our T-string unification procedure.

**Definition 1 (Strings).** Let $\mathcal{V}$ be a set of *variables* and $\mathcal{C}$ a set of *constants* (with $\mathcal{V} \cap \mathcal{C} = \emptyset$). A *string* is a word over the *alphabet* $\mathcal{A} = \mathcal{V} \cup \mathcal{C}$. The *empty string* is denoted by $\varepsilon$. The set of strings over the alphabet $\mathcal{A}$ is denoted by $\mathcal{A}^*$. The *length* of a string $s$ is denoted by $|s|$. For a string $s$ the character at position $i$ is denoted by $s_i$. The *concatenation* of the strings $s$ and $t$ is denoted by $s \circ t$ (or briefly $st$).

*Example 1.* Let $\mathcal{V} = \{A, B, C, \ldots, Z\}$ and $\mathcal{C} = \{a, b, c, \ldots, z\}$. Then $tabUlAR$ and $taSTeFuL$ are strings over $\mathcal{V} \cup \mathcal{C}$.

The strings we intend to unify shall be used to represent the positions of atomic formulae within a formula tree. As such they satisfy certain restrictions which we call the T-string property (i.e. 'formula *tree string*'-property). Two strings $s$ and $t$ have the T-string property, iff

1. no character is repeated either in $s$ nor in $t$, and
2. equal characters occur only at the beginning of $s$ and $t$.

A set of strings $\mathcal{S}$ has the T-string property iff each pair of strings in $\mathcal{S}$ has the T-string property.

**Definition 2 (T-string property).** Let $s$ and $t$ be two strings over an alphabet $\mathcal{A}$. $s$ and $t$ have the *T-string property* iff

1. $\forall_{1 \leq i,j \leq |s|}: i \neq j \Rightarrow s_i \neq s_j$ and $\forall_{1 \leq i,j \leq |t|}: i \neq j \Rightarrow t_i \neq t_j$, and
2. $\exists_{0 \leq j \leq \min(|s|,|t|)}: (\forall_{1 \leq i \leq j}: s_i{=}t_i$ and $\forall_{i_1,i_2}: (j{<}i_1{\leq}|s|$ and $j{<}i_2{\leq}|t|) \Rightarrow s_{i_1} \neq t_{i_2})$.

Let $\mathcal{S} = \{s_1, \ldots, s_m\}$ ($m \geq 2$) be a set of strings. $\mathcal{S}$ has the *T-string property* iff $\forall_{1 \leq i,j \leq m} : s_i$ and $s_j$ have the T-string property. We call the strings in $\mathcal{S}$ *T-strings*.

*Remark.* The prefixes in Wallen's matrix characterization have exactly these attributes, since they are defined as branches in a formula tree.

*Example 2.* The strings $tabUlAR$ and $taSTeFuL$ have the T-string property.

**Definition 3 (Substitution).** A *substitution* $\sigma : \mathcal{V} \to (\mathcal{V} \cup \mathcal{C})^*$ is a mapping from the set of variables to the set of strings over $\mathcal{V} \cup \mathcal{C}$, which is almost everywhere (i.e. except for finitely many elements) the identity mapping. A substitution $\sigma$ can be represented by a set of variable string pairs, i.e. by $\{x \backslash s \mid \sigma(x) = s \text{ and } x \neq s\}$. The *application* $\sigma(t)$ of a substitution $\sigma$ to a string $t$ leads to the string $t'$ where the variables in $t$ are replaced by their values under $\sigma$.

*Example 3.* Let $\mathcal{V}$ and $\mathcal{C}$ defined as above. Then $\sigma = \{U \backslash \varepsilon, A \backslash ea, R \backslash Ux\}$ is a substitution. Its application to the string $tabUlAR$ yields $\sigma(tabUlAR) = tableaUx$.

**Definition 4 (Composition).** The *composition* $\tau(\sigma)$ of two substitutions $\sigma$ and $\tau$ is the combination of the corresponding functions, i.e. this yields the substitution $\{x \backslash \tau(t) \mid x \backslash t \in \sigma \text{ and } x \neq \tau(t)\} \cup \{x \backslash s \mid x \backslash s \in \tau \text{ and not } x \backslash t \in \sigma \text{ for some } t\}$. A substitution $\sigma$ is called *idempotent* if $\sigma(\sigma) = \sigma$.

*Example 4.* Let $\sigma = \{U \backslash \varepsilon, A \backslash ea, R \backslash Ux\}$ and $\tau = \{U \backslash v, A \backslash b, C \backslash De\}$. The composition $\rho = \tau(\sigma)$ results in $\rho = \{U \backslash \varepsilon, A \backslash ea, R \backslash vx, C \backslash De\}$.
The substitution $\tau$ is idempotent but $\sigma$ is not since $\sigma(\sigma) = \{U \backslash \varepsilon, A \backslash ea, R \backslash x\}$

*Remark.* From now on we shall always deal with idempotent substitutions.

**Definition 5 (Instance).** A substitution $\rho$ is called an *instance* of a substitution $\sigma$ (or $\sigma$ is *more general* than $\rho$), iff there is substitution $\tau$ with $\rho = \tau(\sigma)$.[4]

*Example 5.* In example 4 the substitution $\rho$ is an instance of $\sigma$, since $\rho = \tau(\sigma)$.

With the above notions we can now define unifiers and most general unifiers. Our definitions correspond to concepts introduced e.g. in [19, 12, 18].

**Definition 6 (T-Unifier).** Let $\Gamma = \{s_i = t_i \mid i = 1, \ldots, n\}$ be a system of equations for T-strings such that the set $\mathcal{S} = \{s_i \mid i = 1, \ldots, n\} \cup \{t_i \mid i = 1, \ldots, n\}$ has the T-string property. A substitution $\sigma$ *T-unifies* $\Gamma$ iff for every equation $s_i = t_i$ in $\Gamma$ we have $\sigma(s_i) = \sigma(t_i)$. $\sigma$ is called a *T-unifier* for $\Gamma$.

*Example 6.* $\sigma = \{U \backslash \varepsilon, A \backslash ea, R \backslash ux, S \backslash b, T \backslash l, F \backslash a, L \backslash x\}$ is a T-unifier for the set $\Gamma = \{tabUlAR = taSTeFuL\}$ since $\sigma(tabUlAR) = \sigma(taSTeFuL) = tableaux$.

**Definition 7 (Most General T-Unifiers).** Let $\Gamma = \{s_i = t_i \mid i = 1, .., n\}$ be a system of equations of T-strings. A set $\Sigma$ of substitutions is called a *(minimal) set of most general T-unifiers* for $\Gamma$, denoted mgu($\Gamma$), if the following properties hold:

---

[4] Note that every substitution $\sigma$ is an instance of itself, since $\sigma = \tau(\sigma)$ for $\tau = \emptyset$.

1. Every substitution $\sigma \in \Sigma$ is a T-unifier for $\Gamma$.            (*Correctness*)
2. Every T-unifier $\tau$ for $\Gamma$ is an instance of some $\sigma \in \Sigma$.     (*Completeness*)
3. No $\sigma \in \Sigma$ is an instance of another substitution $\sigma' \in \Sigma$.      (*Minimality*)

An element of mgu($\Gamma$) is called a *most general T-unifier* for $\Gamma$.

## 3   T-String-Unification for Intuitionistic Logic

In this section we present a general method for computing the set of most general T-unifiers. It can be used to unify prefixes in proof methods for the intuitionistic logic $\mathcal{J}$ or the modal logic S4 (see [19, 14]).[5] The method is *general* in the sense that there are no restrictions for the computed unifier: variables may be instantiated by arbitrary strings including the empty string.[6]

After examining the decidability of T-string-unification we shall give an intuitive graphical explanation of a method for computing T-unifiers. We shall then present a formal description of our technique which operates by transforming systems of equations according to given transformation rules. We proof correctness, completeness, minimality, and termination of our unification procedure, investigate its complexity and finally present an implementation of the algorithm.

### 3.1   Decidability

For two T-strings $s$ and $t$, i.e. for the equation system $\{s=t\}$, there is a simple way for deciding whether there exists a T-unifier or not.

**Lemma 8 (Decidability).** *Let $s'$ and $t'$ be two strings over the alphabet $\mathcal{V} \cup \mathcal{C}$ which have the T-string property. Furthermore let $u$ be the common string at the beginning of $s$ and $t$, i.e. $s'=us$ and $t'=ut$.*
*There is a T-unifier for the equation $\{s'=t'\}$ if and only if*

1. *$s = \varepsilon$ and $t = \varepsilon$, or*
2. *$s_1 \in \mathcal{V}$ and $t_{|t|} \in \mathcal{V}$, or*
3. *$s_{|s|} \in \mathcal{V}$ and $t_1 \in \mathcal{V}$, or*
4. *$s_1 \in \mathcal{V}$ and $s_{|s|} \in \mathcal{V}$ and if $s_i \in \mathcal{C}$ for some $i$ then $t_j \in \mathcal{V}$ for some $j$, or*
5. *$t_1 \in \mathcal{V}$ and $t_{|t|} \in \mathcal{V}$ and if $t_i \in \mathcal{C}$ for some $i$ then $s_j \in \mathcal{V}$ for some $j$.*

*Proof of lemma.* Analyze all possible combinations of variables and constants within the strings $s$ and $t$. The five cases describe all the combinations where unification is possible. In all other cases $s$ and $t$ cannot be unified.     □

**Theorem 9 (Complexity of Decidability).** *The existence of a T-unifier for the equation $s=t$, where $s$ and $t$ are T-strings, can be decided in time $\mathcal{O}(|s|+|t|)$.*

*Proof of theorem.* The five cases mentioned in lemma 8 can be checked in linear time with respect to the length of $s$ and $t$.     □

---

[5] There is a close relationship between these logics since $\mathcal{J}$ can be embedded into S4.

[6] This property follows from the fact that the *accessibility relation* of the modal logic S4 is reflexive and transitive.

## 3.2 A Graphical Representation

Before we give a detailed description of our unification procedure, we first want to introduce a graphical method for representing the set of most general T-unifiers for a given equation of T-strings. It is somewhat similar to Makanin's notion of an *equation with scheme* [9, 10].

*Example 7.* The strings $s=aBcDEf$ and $t=aBGhiJ$ fulfill the T-string property. If we want to unify them (i.e. find a most general T-unifier for $\{s=t\}$) we have to look for a mapping which assigns a substring of $t$ to each variable in $s$ and vice versa. We can represent this mapping by the following scheme:

| $a$ | $B$ | $c$ | $D$ | $E$ | $f$ |
|---|---|---|---|---|---|
| $a$ | $B$ | $G$ | $h$ | $i$ | $J$ |

$$\underbrace{\quad}_{X'} \qquad \underbrace{\quad}_{Y'}$$

The T-unifier derived from this scheme is $\sigma_1 = \{G\backslash cX', D\backslash X'hiY', J\backslash Y'Ef\}$ where $X'$ and $Y'$ are new variables which do not occur in the strings $s$ or $t$. It is necessary to introduce these variables in order to obtain a most general T-unifier.[7] To get *all* T-unifiers we just have to consider all possible correct schemes.

In general we have to perform the following steps in order to get a scheme representation of the most general T-unifiers for a T-string equation $s=t$: We start by writing down the first string $s$ in a way such that constants will receive a small slot and variables will receive a large slot.[8] In the row below we write down the second string $t$ such that

1. equal characters at the beginning of $s$ and $t$ appear in the same column,
2. a constant occurs only within the range of a variable (except for constants in the common substring at the beginning) and
3. the range of each variable may be stretched arbitrarily so that it may overlap with a variable of the first string.

In the first row (of the second string) we begin by stretching the variables as little as possible. Below we will then systematically enumerate all the possibilities for extending the range of one or more variables. Each of these rows (except the one representing the first string) represents a most general T-unifier and each most general T-unifier is represented by one row of the scheme.

*Example 8.* Continuing our previous example and applying this technique to unify the strings $s=aBcDEf$ and $t=aBGhiJ$ we get the following scheme:

| $a$ | $B$ | $c$ | $D$ | | $E$ | | $f$ |
|---|---|---|---|---|---|---|---|
| $a$ | $B$ | $G$ | | $h$ | $i$ | $J$ | |
| $a$ | $B$ | $G$ | $h$ | $i$ | $J$ | | |
| $a$ | $B$ | $G$ | | | $h$ | $i$ | $J$ |

---

[7] $\sigma' = \{G\backslash c, D\backslash hi, J\backslash Ef\}$ is *not* a most general T-unifier since it is an instance of $\sigma_1$.

[8] Technically it suffices to reserve a slot of the size of the second string $t$.

Given such a graphical scheme the unifier of $s$ and $t$ implicitly contained in each row can easily be computed:

1. Constants and variables occuring in the range of a variable will assigned to this variable and
2. new variables will have to be generated and assigned accordingly if two variables of $s$ and $t$ overlap.

Performing this final step will lead to the desired set of most general T-unifiers for the equation $\{s=t\}$.

*Example 9.* According to the scheme in example 8 we compute the following set of most general unifiers for the T-strings $s=aBcDEf$ and $t=aBGhiJ$:

$\sigma_0 = \{G\backslash cX', D\backslash X'h, E\backslash iY', J\backslash Y'f\}$  (first row),

$\sigma_1 = \{G\backslash cX', D\backslash X'hiY', J\backslash Y'Ef\}$  (second row), and

$\sigma_2 = \{G\backslash cDX', E\backslash X'hiY', J\backslash Y'f\}$  (third row)

where $X'$ and $Y'$ are new variables.

## 3.3 Transforming Equations

The intuitive graphical technique given above should be sufficient to understand the following formalized method for T-string-unification. Rather than by giving a recursive algorithm we consider the process of unification as a sequence of transformations on systems of equations. This concept is similar to the ideas of Martelli & Montanari ([11], see also [12, 18]).

To keep our notation simple we divide our system of equations into an *unsolved* part $\Gamma$ that initially contains the equations to be solved, and into a *solved* part $\sigma$ which represents a substitution. We start with a given system $\Gamma = \{s_i=t_i \mid i = 1,..,n\}$ of equations and an empty substitution $\sigma = \emptyset$ and stop with an empty system of equations $\Gamma = \emptyset$ and a substitution $\sigma = \{x_j\backslash t_j \mid j = 1,..,m\}$ representing an idempotent most general T-unifier. Each transformation step replaces a system $\Gamma, \sigma$ by a modified system $\Gamma', \sigma'$.

The algorithm is described by transformation rules which can be applied non-deterministically to the actual equation system. A sequence of transformations is *successful* if it transforms $\Gamma$ into an empty set of equations. Each successful chain of transformations corresponds to one most general T-unifier for $\Gamma$. For technical reasons we divide the right part $t$ of each equation into two parts $t^1$ and $t^2$, i.e. we consider equations of the form $s_i=t_i^1|t_i^2$.[9] Hence we will start with the slightly modified equation system $\Gamma = \{s_i=\varepsilon|t_i \mid i = 1,..,n\}$.

We shall first formalize the transformation algorithm and specify the transformation rules afterwards.

---

[9] As in the graphical representation of our method the vertical bar represents the end of a substring which will be assigned to a variable. Some transformation rules (e.g. R5, R7, and R9 in table 1) will perform this assignment while others (e.g. R10) move the vertical bar in order to create alternatives.

**Definition 10 (Transformation Algorithm).** Let $\Gamma = \{s_i = \varepsilon | t_i \mid i = 1, .., n\}$ be a system of equations where the $s_i, t_i$ are T-strings over an alphabet $\mathcal{V} \cup \mathcal{C}$. Let $\sigma = \emptyset$ and let $\mathcal{T} = \{\Pi_i \rightarrow \Delta_i, \Theta_i \mid i = 1, .., k\}$ be a set of transformation rules. The following (nondeterministic) *transformation algorithm* computes a set of most general T-unifiers for $\Gamma$.

**while** $\Gamma \neq \emptyset$ **do**

    a. select the leftmost equation $s = t$ from $\Gamma$

    b. set $\Gamma$ to $\Gamma \setminus \{s = t\}$

    c. select a transformation rule $\Pi \rightarrow \Delta, \Theta$ from $\mathcal{T}$ which is applicable to $\{s = t\}$;
        **if** no rule is applicable **then stop** with *failure*

    d. apply the rule $\Pi \rightarrow \Delta, \Theta$ to $\{s = t\}$
        let $\Delta', \Theta'$ be the result of this transformation

    e. apply the substitution $\Theta'$ to $\Gamma$ and $\sigma$   (compute $\Gamma := \Theta'(\Gamma)$ and $\sigma := \Theta'(\sigma)$)

    f. set $\Gamma$ to $\Delta' \cup \Gamma$; set $\sigma$ to $\sigma \cup \Theta'$

**stop** with *success* and return $\sigma$.

The set of most general T-unifiers consists of the results of all successfully finished transformations. In each case $\sigma$ represents an idempotent most general T-unifier.

*Remark.* The selection of an equation in step a. can be done arbitrarily. The algorithm will, however, be more comprehensible if we make this choice systematically and first solve one equation before considering the next one.

**Definition 11 (Transformation Rules for $\mathcal{J}$).** Let $\mathcal{V}$ be a set of Variables, $\mathcal{C}$ a set of Constants and $\mathcal{V}'$ a set of *auxiliary variables* with $\mathcal{V} \cap \mathcal{V}' = \emptyset$. The set $\mathcal{T}$ of *transformation rules* for the intuitionistic logic $\mathcal{J}$ consists of the following 10 rules:

| | | |
|---|---|---|
| R1. | $\{\varepsilon = \varepsilon \vert \varepsilon\}$ | $\rightarrow \{\}, \{\}$ |
| R2. | $\{\varepsilon = \varepsilon \vert t^+\}$ | $\rightarrow \{t^+ = \varepsilon \vert \varepsilon\}, \{\}$ |
| R3. | $\{Xs = \varepsilon \vert Xt\}$ | $\rightarrow \{s = \varepsilon \vert t\}, \{\}$ |
| R4. | $\{Cs = \varepsilon \vert Vt\}$ | $\rightarrow \{Vt = \varepsilon \vert Cs\}, \{\}$ |
| R5. | $\{Vs = z \vert \varepsilon\}$ | $\rightarrow \{s = \varepsilon \vert \varepsilon\}, \{V \backslash z\}$ |
| R6. | $\{Vs = \varepsilon \vert C_1 t\}$ | $\rightarrow \{s = \varepsilon \vert C_1 t\}, \{V \backslash \varepsilon\}$ |
| R7. | $\{Vs = z \vert C_1 C_2 t\}$ | $\rightarrow \{s = \varepsilon \vert C_2 t\}, \{V \backslash z C_1\}$ |
| R8. | $\{Vs^+ = \varepsilon \vert V_1 t\}$ | $\rightarrow \{V_1 t = V \vert s^+\}, \{\}$ |
| R9. | $\{Vs^+ = z^+ \vert V_1 t\}$ | $\rightarrow \{V_1 t = V' \vert s^+\}, \{V \backslash z^+ V'\}$ |
| R10. | $\{Vs = z \vert Xt\}$ | $\rightarrow \{Vs = zX \vert t\}, \{\}$, where $V \neq X$, and $s = \varepsilon$ or $t \neq \varepsilon$ or $X \in \mathcal{C}$ |

$s, t$ and $z$ denote (arbitrary) strings and $s^+, t^+, z^+$ non-empty strings. $X, V, V_1, C, C_1$ and $C_2$ denote single characters with $X \in \mathcal{V} \cup \mathcal{C} \cup \mathcal{V}'$, $V, V_1 \in \mathcal{V} \cup \mathcal{V}'$ (with $V \neq V_1$), and $C, C_1, C_2 \in \mathcal{C}$. $V' \in \mathcal{V}'$ is a new variable which does not occur in the substitution $\sigma$ computed so far.

**Table 1.** Transformation Rules for Intuitionistic Logic

*Example 10.* Consider our previous example, i.e. let $\Gamma = \{aBcDEf = \varepsilon | aBGhiJ\}$, $\sigma = \emptyset$, and $X', Y'$ auxiliary variables. We explain the unification algorithm by a successive application of transformation rules starting with the equation system $\Gamma$ and the empty substitution $\sigma$. According to our algorithm we always select the first equation of $\Gamma$. We shall write $\Gamma, \sigma \xrightarrow{\mathrm{R}i} \Gamma', \sigma'$ to denote that applying rule R$i$ to $\Gamma$ (and $\sigma$) leads to the modified system $\Gamma'$ and the modified substitution $\sigma'$.

During the first three steps the transformation is straightforward.

$\{aBcDEf=\varepsilon|aBGhiJ\}$, $\{\}$

$\xrightarrow{\text{R3}}$ $\{BcDEf=\varepsilon|BGhiJ\}$, $\{\}$

$\xrightarrow{\text{R3}}$ $\{cDEf=\varepsilon|GhiJ\}$, $\{\}$

$\xrightarrow{\text{R4}}$ $\{GhiJ=\varepsilon|cDEf\}$, $\{\}$

Now two rules (R6 and R10) are applicable. We first consider R6 and then R10.

1. $\xrightarrow{\text{R6}}$ $\{hiJ=\varepsilon|cDEf\}$, $\{G\backslash\varepsilon\} \longrightarrow \bot$ (no transformation rule applicable)

2. $\xrightarrow{\text{R10}}$ $\{GhiJ=c|DEf\}$, $\{\}$

   Again there are two possibilities (R9 and R10):

   (a) $\xrightarrow{\text{R9}}$ $\{DEf=X'|hiJ\}$, $\{G\backslash cX'\}$

       i. $\xrightarrow{\text{R7}}$ $\{Ef=\varepsilon|iJ\}$, $\{G\backslash cX', D\backslash X'h\}$

          A. $\xrightarrow{\text{R6}}$ $\{f=\varepsilon|iJ\}$, $\{G\backslash cX', D\backslash X'h, E\backslash\varepsilon\} \longrightarrow \bot$

          B. $\xrightarrow{\text{R10}}$ $\{Ef=i|J\}$, $\{G\backslash cX', D\backslash X'h\}$

            $\xrightarrow{\text{R9}}$ $\{J=Y'|f\}$, $\{G\backslash cX', D\backslash X'h, E\backslash iY'\}$

            $\xrightarrow{\text{R10}}$ $\{J=Y'f|\varepsilon\}$, $\{G\backslash cX', D\backslash X'h, E\backslash iY'\}$

            $\xrightarrow{\text{R5}}$ $\{\varepsilon=\varepsilon|\varepsilon\}$, $\{G\backslash cX', D\backslash X'h, E\backslash iY', J\backslash Y'f\}$

            $\xrightarrow{\text{R1}}$ $\{\}$, $\underline{\{G\backslash cX', D\backslash X'h, E\backslash iY', J\backslash Y'f\}}$     $\boxed{\sigma_0}$

       ii. $\xrightarrow{\text{R10}}$ $\{DEf=X'h|iJ\}$, $\{G\backslash cX'\}$

         $\xrightarrow{\text{R10}}$ $\{DEf=X'hi|J\}$, $\{G\backslash cX'\}$

         $\xrightarrow{\text{R9}}$ $\{J=Y'|Ef\}$, $\{G\backslash cX', D\backslash X'hiY'\}$

         $\xrightarrow{\text{R10}}$ $\{J=Y'E|f\}$, $\{G\backslash cX', D\backslash X'hiY'\}$

         $\xrightarrow{\text{R10}}$ $\{J=Y'Ef|\varepsilon\}$, $\{G\backslash cX', D\backslash X'hiY'\}$

         $\xrightarrow{\text{R5}}$ $\{\varepsilon=\varepsilon|\varepsilon\}$, $\{G\backslash cX', D\backslash X'hiY', J\backslash Y'Ef\}$

         $\xrightarrow{\text{R1}}$ $\{\}$, $\underline{\{G\backslash cX', D\backslash X'hiY', J\backslash Y'Ef\}}$     $\boxed{\sigma_1}$

   (b) $\xrightarrow{\text{R10}}$ $\{GhiJ=cD|Ef\}$, $\{\}$

       i. $\xrightarrow{\text{R9}}$ $\{Ef=X'|hiJ\}$, $\{G\backslash cDX'\}$

          A. $\xrightarrow{\text{R7}}$ $\{f=\varepsilon|iJ\}$, $\{G\backslash cDX', E\backslash X'h\} \longrightarrow \bot$

          B. $\xrightarrow{\text{R10}}$ $\{Ef = X'h|iJ\}$, $\{G\backslash cDX'\}$

            $\xrightarrow{\text{R10}}$ $\{Ef=X'hi|J\}$, $\{G\backslash cDX'\}$

            $\xrightarrow{\text{R9}}$ $\{J=Y'|f\}$, $\{G\backslash cDX', E\backslash X'hiY'\}$

            $\xrightarrow{\text{R10}}$ $\{J=Y'f|\varepsilon\}$, $\{G\backslash cDX', E\backslash X'hiY'\}$

            $\xrightarrow{\text{R5}}$ $\{\varepsilon=\varepsilon|\varepsilon\}$, $\{G\backslash cDX', E\backslash X'hiY', J\backslash Y'f\}$

            $\xrightarrow{\text{R1}}$ $\{\}$, $\underline{\{G\backslash cDX', E\backslash X'hiY', J\backslash Y'f\}}$     $\boxed{\sigma_2}$

       ii. $\xrightarrow{\text{R10}}$ $\{GhiJ=cDE|f\}$, $\{\}$

         $\xrightarrow{\text{R10}}$ $\{GhiJ=cDEf|\varepsilon\}$, $\{\}$

         $\xrightarrow{\text{R5}}$ $\{hiJ=\varepsilon|\varepsilon\}$, $\{G\backslash cDEf\} \longrightarrow \bot$

As a result we get three most general T-unifiers $\sigma_0, \sigma_1, \sigma_2$ which are identical with the ones evaluated with the graphical method in the previous subsection.

*Remark.* If we use a T-string-unification algorithm within a proof procedure we can replace the auxiliary variables by the empty string *after* unification, i.e. we apply the substitution $\{V'\backslash\varepsilon \mid V' \in \mathcal{V}'\}$ to each calculated most general T-unifier.

### 3.4 Correctness, Termination, Minimality and Completeness

In the following we prove correctness, termination, minimality and completeness of T-string-unification (or briefly T-unification).

**Lemma 12 (T-String Preserving).** *Let $\Gamma = \{s_1{=}t_1, \ldots, s_n{=}t_n\}$ be a system of equations where $\mathcal{S}{=}\{s_i \mid i{=}1, \ldots, n\} \cup \{t_i \mid i{=}1, \ldots, n\}$ has the T-string property and let $\sigma$ be a most general T-unifier for $\{s_1{=}t_1\}$. Then the set $\sigma(\mathcal{S}){:=}\{\sigma(u) \mid u \in \mathcal{S}\}$ also fulfills the T-string property.*

*Proof of lemma.* We can represent the elements of $\mathcal{S}$ as branches of a tree which splits if the elements become different. If we unify the two branches corresponding to $s_1$ and $t_1$, the result will still be a tree since all the additional variables we may insert are unique. □

Because of lemma 12 we can restrict ourselves to unifying systems containing only a single equation $s{=}t$ instead of having to unify a larger system $\Gamma$ all at once. After solving the first equation we may choose the next equation and so on since the strings of each equation fulfill the T-string property under the computed substitution. This insight will drastically simplify the proofs of correctness, minimality, termination, and completeness.

**Definition 13 (Result mgu$_\mathrm{T}$ of T-String-Unification).** Let $\Gamma$ be a system of equations. mgu$_\mathrm{T}(\Gamma)$ denotes the set of substitutions computed by the specified algorithm for T-string-unification.

**Theorem 14 (Correctness of T-Unification).** *Let $\Gamma{=}\{s{=}\varepsilon|t\}$ be an equation of T-strings. Then $mgu_\mathrm{T}(\Gamma)$ is correct, i.e. each $\sigma \in mgu_\mathrm{T}(\Gamma)$ is a T-unifier for $\Gamma$.*

*Proof of theorem.* Let $\Gamma = \{s{=}z|t\}$ be an equation and $k(\Gamma)$ be the number of rule applications necessary for computing a T-unifier for $\Gamma$. We prove the theorem by induction on $k(\Gamma)$ and show that $\sigma \in \mathrm{mgu}_\mathrm{T}(\Gamma)$ is correct for $n{=}k(\Gamma)$.
For $n{=}1$ we have $\Gamma = \{\varepsilon{=}\varepsilon|\varepsilon\}$. Applying rule R1 we obtain $\Gamma{=}\{\}$ and the substitution $\sigma{=}\{\}$ which is a correct T-unifier for $\Gamma$.
Let $\Gamma = \{s{=}z|t\}$ with $k(\Gamma) = n{+}1$ and R$i$ be a transformation rule which is applicable to $\{s{=}z|t\}$. Let $\Delta', \Theta'$ be the result of this application. By the induction hypothesis $\sigma \in \mathrm{mgu}_\mathrm{T}(\Delta')$ is a correct T-unifier for $\Delta'$ (if some $\sigma$ exists), since $k(\Delta') = n$. Then $\Theta' \cup \sigma$ is a correct T-unifier for $\Gamma$, since $(\Theta' \cup \sigma)(s) = (\Theta' \cup \sigma)(z\,t)$ if $\sigma$ is a correct T-unifier for $\Delta'$. □

**Theorem 15 (Termination of T-Unification).** *Let $\Gamma{=}\{s{=}\varepsilon|t\}$ be an equation of T-strings. Then the specified T-string-unification terminates on $\Gamma$.*

*Proof of theorem.* Let $\Gamma=\{s=z|t\}$. We define a well-ordered relation $<_\Gamma$ between systems of equations and show that after applying at most two transformation rules to $\Gamma$ the property $\Gamma'<_\Gamma\Gamma$ holds between $\Gamma$ and the resulting system $\Gamma'$. This property is defined as follows:

Let $\Gamma_1 = \{s_1=z_1|t_1\}$ and $\Gamma_2 = \{s_2=z_2|t_2\}$. The *system ordering* $<_\Gamma$ is a relation on system equations in the following way:

$$\Gamma_1<_\Gamma\Gamma_2 \;\Leftrightarrow\; \begin{cases} |s_1z_1t_1| < |s_2z_2t_2|, \text{ or} \\ |s_1z_1t_1| = |s_2z_2t_2| \text{ and } |z_1| > |z_2|, \text{ or} \\ \Gamma_1=\{\} \text{ and } \Gamma_2=\{\varepsilon=\varepsilon|\varepsilon\}. \end{cases}$$

Let $\Gamma=\{s=z|t\}$ be a system of equations and R$i$ a transformation rule which is applicable to $\Gamma$. Let $\Delta',\Theta'$ denote the result of this transformation. We distinguish three cases:

1. Applying one of the rules R1, R3, R5,R6, R7, R8, R9, or R10 we get $\Delta'<_\Gamma\Gamma$.
2. After applying rule R2, the only applicable rule to the resulting system $\Delta'$ would be R5. Let this application yield the system $\Delta''$. From 1. we conclude that $\Delta''<_\Gamma\Gamma$.
3. After applying rule R4 the only applicable rules to the result $\Delta'$ could be R6, R7 or R10. Let this application yield $\Delta''$. From 1. we conclude that $\Delta''<_\Gamma\Gamma$.

Since there are only finitely many transformation rules applicable to a system $\Gamma$ and the relation $<_\Gamma$ is well-ordered, every transformation chain will be finite. $\quad\square$

**Theorem 16 (Minimality of T-Unification).** *Let $\Gamma=\{s=\varepsilon|t\}$ be an equation of T-strings. Then $mgu_\mathrm{T}(\Gamma)$ is minimal, i.e. no substitution $\sigma \in mgu_\mathrm{T}(\Gamma)$ is an instance of another substitution $\sigma' \in mgu_\mathrm{T}(\Gamma)$.*

*Proof of theorem (sketch).* We show that the application of different rules to a system $\Gamma = \{s=z|t\}$ yields different assignments to at least one variable such that the assignments are not an instance of each other.

If one of the rules R1 to R5 is applicable to $\Gamma$ then no other rule will be applicable to $\Gamma$. If one of the rules R6 to R10 is applicable to $\{Vs=z|t\}$ there may be another rule which is also applicable. By a lengthy case analysis it can be verified that the application of different rules assigns different strings to the variable $V$. Therefore none of the computed substitutions are instances of each other. $\quad\square$

**Theorem 17 (Completeness of T-Unification).** *Let $\Gamma=\{s=\varepsilon|t\}$ be an equation of T-strings. Then $mgu_\mathrm{T}(\Gamma)$ is complete, i.e. each T-unifier of $\Gamma$ is an instance of some $\sigma \in mgu_\mathrm{T}(\Gamma)$.*

*Proof of theorem (sketch).* If we investigate the pattern of a string assigned to a variable we can determine that they must have a certain form. For example auxiliary variables may only occur at the beginning or at the end of such a string. By examining all these possibilities we can verify that each such assignment is calculated by the algorithm. $\quad\square$

### 3.5  Complexity

Since the complexity of the unification process is determined by the number of computed unifiers, we will now investigate the number of most general T-unifiers. As mentioned before this number is finite but can be extremely large.

**Theorem 18 (Complexity of T-Unification).** *In the* worst case *the number of most general T-unifiers for $s$ and $t$ grows exponentially w.r.t. the length of $s$ and $t$: The number of mgu's can be up to* $\frac{1}{2}\frac{(2n)!}{(n!)^2} \in \mathcal{O}(\frac{2^{2n}}{\sqrt{n}})$ *where* $n = \max(|s|,|t|)$.

*Proof of theorem.* In the worst case one string contains only variables and the other only constants. In this case unification is the same as *matching* and we have to consider every possible assignment of constant strings to variables. This number is equal to the number of possibilities to select $n$ elements out of $2n-1$ where $n$ is the length of the two unified strings. Hence we get $\binom{2n-1}{n} = \frac{(2n-1)!}{n!(n-1)!} = \frac{1}{2}\frac{(2n)!}{(n!)^2}$ most general T-unifiers. Using the approximation $n! \approx \sqrt{n} \cdot n^n$ we get for large $n$:

$$\frac{1}{2}\frac{\sqrt{2n}(2n)^{2n}}{(\sqrt{n}n^n)^2} = \frac{1}{\sqrt{2n}}\left(\frac{2n}{n}\right)^{2n} = \frac{1}{\sqrt{2n}}2^{2n} \in \mathcal{O}\left(\frac{2^{2n}}{\sqrt{n}}\right) \qquad \square$$

*Example 11.* Consider the strings $AB$ and $cd$ where $A, B$ are variables and $c, d$ are constants. The most general T-unifiers can be represented by the following scheme:

| $A$ | | $B$ | |
|---|---|---|---|
| $c$ | $d$ | $\varepsilon$ | |
| $c$ | | $d$ | |
| $\varepsilon$ | | $c$ | $d$ |

The number of most general T-unifiers is $\frac{1}{2}\frac{4!}{(2!)^2} = 3$.


### 3.6  Implementation of T-String-Unification

The implementation of our unification procedure has been written in Prolog but may easily be translated into a functional or an imperative programming language.

Strings are represented as Prolog lists, i.e. the string $w = v_1 v_2 \ldots v_n$ is represented by the list `W=[v1,v2,...,vN]`. As usual variables are written in capitals whereas constants are written in small letters. The predicate `tunify(S,[],T)` succeeds if the T-strings `S` and `T` can be unified. In this case the instantiated variables represent a most general T-unifier for `S` and `T`. All other most general T-unifiers will be computed via backtracking. Each clause of the predicate `tunify` corresponds to exactly one transformation rule as defined in table 1.

To solve a set of string equations the predicate `t_string_unify(G)` is used. `G` is a set of equations, i.e. a set whose elements have the form `S=T`, where `S` and `T` are strings. Once again the set of most general T-unifiers is calculated via backtracking.[10]

---

[10] The variables may be instantiated with nested lists. Therefore the predicate `flatten` is necessary which can be implemented as follows:
```
flatten(A,[A|B],B) :- (var(A); atom(A)) -> A\==[], !.   flatten([],A,A).
flatten([A|B],C,D) :- flatten(A,C,E), flatten(B,E,D).
```

```
t_string_unify([]).
t_string_unify([S=T|G])  :- flatten(S,S1,[]), flatten(T,T1,[]),
                              tunify(S1,[],T1), t_string_unify(G).
tunify([],[],[]).
tunify([],[],[X|T])       :- tunify([X|T],[],[]).
tunify([X1|S],[],[X2|T]) :- X1==X2, !, tunify(S,[],T).
tunify([C|S],[],[V|T])   :- atom(C),!,var(V),tunify([V|T],[],[C|S]).
tunify([V|S],Z,[])       :- V=Z, tunify(S,[],[]).
tunify([V|S],[],[C1|T])  :- atom(C1), V=[], tunify(S,[],[C1|T]).
tunify([V|S],Z,[C1,C2|T]):- atom(C1), atom(C2), append(Z,[C1],V),
                              tunify(S,[],[C2|T]).
tunify([V,X|S],[],[V1|T])   :- var(V1), tunify([V1|T],[V],[X|S]).
tunify([V,X|S],[Z1|Z],[V1|T]):- var(V1), append([Z1|Z],[Vnew],V),
                                tunify([V1|T],[Vnew],[X|S]).
tunify([V|S],Z,[X|T])    :- (S=[]; t\=[]; atom(X)) ->
                              append(Z,[X],Z1), tunify([V|S],Z1,T).
```

*Remark.* Note that the entered strings have to fulfill the T-string property. Otherwise no sensible output will be given.

## 4   T-String-Unification for Modal Logics

If we want to apply T-string-unification to the modal logics, we have to respect the accessibility relation for these logics. This means that we are not allowed to assign arbitrary strings to variables but only strings with certain restrictions on their length.

In the following we will treat the modal logics D, K, D4, K4, S4, S5 and T. For each logic we shall define the essential restrictions, specify the transformation rules and investigate the complexity of the corresponding unification process.

### 4.1   The Modal Logics D and K

The modal logics D and K, together with S5, are the simplest. Variables may only be instantiated with exact one single character, i.e. for every T-unifier $\sigma$ the following property must hold: $|\sigma(V)| = 1$ for all $V \in \mathcal{V}$. This restriction (see [19]) follows from the fact that the accessibility relation for D and K has no special properties which can be guaranteed. Accordingly we only need the transformation rules R1, R3, R4 of table 1 and have to add one new rule (R2):

| | | | |
|---|---|---|---|
| R1. | $\{\varepsilon = \varepsilon\|\varepsilon\}$ | $\rightarrow$ | $\{\}, \{\}$ |
| R2. | $\{Vs = \varepsilon\|Xt\}$ | $\rightarrow$ | $\{s = \varepsilon\|t\}, \{V\backslash X\}$, where $V \neq X$ |
| R3. | $\{Xs = \varepsilon\|Xt\}$ | $\rightarrow$ | $\{s = \varepsilon\|t\}, \{\}$ |
| R4. | $\{Cs = \varepsilon\|Vt\}$ | $\rightarrow$ | $\{Vt = \varepsilon\|Cs\}, \{\}$ |

$s$ and $t$ denote arbitrary strings, $X$, $V$ and $C$ denote a single character, constant, or variable, respectively, i.e. $X \in \mathcal{V} \cup \mathcal{C}$, $V \in \mathcal{V}$ and $C \in \mathcal{C}$.

**Table 2.** Transformation Rules for the Modal Logics D and K

*Example 12.* Consider the well known strings $aBcDEf$ and $aBGhiJ$. The following scheme represents the graphical unification:

| $a$ | $B$ | $c$ | $D$ | $E$ | $f$ |
|-----|-----|-----|-----|-----|-----|
| $a$ | $B$ | $G$ | $h$ | $i$ | $J$ |

The only most general T-unifier is $\sigma = \{G\backslash c, D\backslash h, E\backslash i, J\backslash f\}$.

In general there is at most one most general unifier for every unification problem in the modal logics K and D.[11]

## 4.2 The Modal Logics D4 and K4

In the modal logic D4 and K4 variables may only be instantiated with non-empty strings since the accessibility relation for D4 and K4 is transitive (see [19]). Thus for every T-unifier $\sigma$ the property $|\sigma(V)| \geq 1$ must hold for all $V \in \mathcal{V}$. The rules R1 to R4 and R7 to R10 are identical with those for intuitionistic logic in table 1. For the rules R5 and R6 we need an additional restriction. This results in the following rules:

> R1. – R4.  *see transformation rules for $\mathcal{J}$*
> R5. $\{Vs = z|\varepsilon\}$   $\rightarrow$  $\{s = \varepsilon|\varepsilon\}$, $\{V\backslash z\}$, where $z \neq \varepsilon$ or ($V \in \mathcal{V}'$ and $R_V^{(\sigma)}$)
> R6. $\{Vs = \varepsilon|C_1 t\}$ $\rightarrow$ $\{s = \varepsilon|C_1 t\}$, $\{V\backslash\varepsilon\}$, where $V \in \mathcal{V}'$ and $R_V^{(\sigma)}$
> R7. – R10.  *see transformation rules for $\mathcal{J}$*

$s, t$ and $z$ denote arbitrary strings, $V \in \mathcal{V} \cup \mathcal{V}'$, $C_1 \in \mathcal{C}$ and the restriction $R_V^{(\sigma)}$ is defined as
$R_V^{(\sigma)} := \begin{cases} \textit{false} \text{ , if } \tilde{V}\backslash V \in \sigma \text{ for some } \tilde{V} \in \mathcal{V} \\ \textit{true} \text{ , otherwise} \end{cases}$ , where $\sigma$ is the substitution computed so far.

**Table 3.** Transformation Rules for the Modal Logics D4 and K4

*Remark.* The auxiliary variables may be removed from the most general T-unifiers *after* unification so that the property above is respected.

*Example 13.* Consider the strings $aBCDeF$ and $aghIj$. The most general T-unifier can be calculated using the following scheme:

| $a$ | | $B$ | | $C$ | $D$ | $e$ | $F$ |
|-----|-----|-----|-----|-----|-----|-----|-----|
| $a$ | | $g$ | $h$ | | $I$ | | $j$ |
| $a$ | $g$ | $h$ | | $I$ | | | $j$ |

The most general T-unifiers are $\sigma_0 = \{B\backslash g, C\backslash hX', I\backslash X'DeY', F\backslash Y'j\}$ and $\sigma_1 = \{B\backslash ghX', I\backslash X'CDeY', F\backslash Y'j\}$.

As in the intuitionistic case the number of computed most general T-unifiers is finite but may grow exponentially with respect to the length of the strings.

---

[11] We want to point out that the existence of a T-unifier is not sufficient for the complementarity of two atoms in the modal logics K or K4. An additional criterion has to be fulfilled in these cases (see [19]).

## 4.3 The Modal Logics S4 and S5

The modal logic S4, where the accessibility relation is reflexive and transitive, can be treated in the same way as the intuitionistic logic. We can use the general T-string-unification algorithm (see section 3) without any additional restrictions.

The modal logic S5 is trivial since the strings we want to unify simply consist of a single character which is either a variable or a constant. This follows from the fact that the accessibility relation for S5 is an equivalence relation (see [19]). Therefore the set of most general T-unifiers is a singleton set and we only need three transformation rules:

$$
\begin{array}{lll}
\text{R1.} & \{V = \varepsilon|X\} & \to \ \{\}, \ \{V\backslash X\}, \text{ where } V \neq X \\
\text{R2.} & \{X = \varepsilon|X\} & \to \ \{\}, \ \{\} \\
\text{R3.} & \{C = \varepsilon|V\} & \to \ \{V = \varepsilon|C\}, \ \{\}
\end{array}
$$

$X, V$ and $C$ denote single characters with $X \in \mathcal{V} \cup \mathcal{C}$, $V \in \mathcal{V}$ and $C \in \mathcal{C}$.

**Table 4.** Transformation Rules for the Modal Logic S5

## 4.4 The Modal Logic T

In the modal logic T strings may only be instantiated with at most one character since the accessibility relation for T is reflexive (see [19]). Thus for every T-unifier $\sigma$ the following property must hold: $|\sigma(V)| \leq 1$ for all $V \in \mathcal{V}$. Whereas the transformation rules R1 to R3 can be taken unchanged from table 1 for intuitionistic logic we slightly have to modify the rules R5 and R8 to R10 and add new rules R6 and R7. This results in the following transformation rules:

$$
\begin{array}{lll}
\text{R1. -- R3.} & \textit{see transformation rules for } \mathcal{J} \\
\text{R5.} & \{Vs = z_v|\varepsilon\} & \to \ \{s = \varepsilon|\varepsilon\}, \ \{V\backslash z_v\} \\
\text{R6.} & \{s_1 V s_2 = z|Ct\} & \to \ \{s_1 = \varepsilon|z, s_2 = \varepsilon|t\}, \ \{V\backslash C\} \cup \sigma_V^{(\sigma)} \\
\text{R7.} & \{s_c^+ = \varepsilon|t_v^+\} & \to \ \{t_v^+ = \varepsilon|s_c^+\}, \ \{\} \\
\text{R8.} & \{Vs_v^+ = \varepsilon|V_1 t_v\} & \to \ \{V_1 t_v = V|s_v^+\}, \ \{\} \\
\text{R9.} & \{Vs_v^+ = z^+|V_1 t_v\} & \to \ \{V_1 t_v = V'|s_v^+\}, \ \{V\backslash z^+ V'\} \\
\text{R10.} & \{Xs = z|Vt\} & \to \ \{Xs = zV|t\}, \ \{\}, \text{ where } X \neq V, \text{ and } s = \varepsilon \text{ or } t \neq \varepsilon
\end{array}
$$

$s, t$ and $z$ denote arbitrary strings, $s^+, t^+, z^+$ non-empty strings; strings indexed with a $v$ may only contain variables, strings indexed with a $c$ must contain at least one constant; $X \in \mathcal{V} \cup \mathcal{C} \cup \mathcal{V}'$, $V, V_1 \in \mathcal{V} \cup \mathcal{V}'$ (with $V \neq V_1$), $V' \in \mathcal{V}'$ and $C \in \mathcal{C}$; the substitution $\sigma_V^{(\sigma)}$ is defined as $\sigma_V^{(\sigma)} := \{\tilde{V}\backslash\varepsilon \mid X\backslash t \in \sigma \text{ where } t \text{ contains } V \text{ and } \tilde{V} \text{ for some } X, \tilde{V} \in \mathcal{V} \cup \mathcal{V}' \text{ with } V \neq \tilde{V}\}$ where $\sigma$ is the substitution computed so far.

**Table 5.** Transformation Rules for the Modal Logic T

*Remark.* To ensure the above property variables which are not assigned to non-empty strings may be replaced by the empty string *after* the unification process, i.e. we apply the substitution $\{V\backslash\varepsilon \mid V \in (\mathcal{V} \cup \mathcal{V}')\}$ to every calculated most general T-unifier.

*Example 14.* Consider the strings $aBCDeF$, $aGhiJ$ and the following scheme representing the most general T-unifiers:

| $a$ | $B$ | $C$ | $D$ | $e$ | $F$ |
|-----|-----|-----|-----|-----|-----|
| $a$ | $h$ | $i$ | $\varepsilon$ | $J$ | $\varepsilon$ |
| $a$ | $h$ | $\varepsilon$ | $i$ | $J$ | $\varepsilon$ |
| $a$ | $G$ | $h$ | $i$ | $J$ | $\varepsilon$ |

The most general T-unifiers are $\sigma_0 = \{G\backslash\varepsilon, B\backslash h, C\backslash i, D\backslash\varepsilon, J\backslash e, F\backslash\varepsilon\}$, $\sigma_1 = \{G\backslash\varepsilon, B\backslash h, C\backslash\varepsilon, D\backslash i, J\backslash e, F\backslash\varepsilon\}$ and $\sigma_2 = \{B\backslash G, C\backslash h, D\backslash i, J\backslash e, F\backslash\varepsilon\}$.

Once again the number of most general unifiers is finite but may grow exponentially with respect to the length of the unified strings.

## 5  Conclusion

We have developed an efficient algorithm for unifying strings which correspond to prefixes of atomic formulae in a formula tree. It can be used to guide the search for a proof in various non-classical logics and plays a fundamental role within efficient proof procedures for these logics. Our method has been described following the terminology of Wallen [19] but it can be applied directly to Ohlbach's world-paths [12, 13] as well.

Our generic algorithm is based on a small set of transformation rules which encode the peculiarities and restrictions of a particular logic. By modifying this set of transformation rules we were able to adopt it to a variety of non-classical logics such as intuitionistic logic $\mathcal{J}$ and the modal logics D, K, D4, K4, S4, S5, and T. In the future we shall investigate how our algorithm can be extended to some subset of linear logic and other important non-classical logics. For these logics, however, a characterization for validity and the notion of a prefix still has to be developed.

Our algorithm is much simpler and considerably more efficient than other string unification algorithms developed so far. The algorithms described in [9, 10, 1] are developed for general string unification and do not take advantage of the special properties of prefix strings. Ohlbach's algorithm [13] does not compute a *minimal* set of unifiers and thus wastes computation time. Besides this one of the main advantages of our algorithm is that it generates unifiers step by step instead of computing them all at once. This will become particularly important when using the algorithm within a proof procedure since it will seldomly be the case that *all* possible unifiers have to be investigated during a proof search. Our algorithm will therefore lead to the implementation of a very efficient proof search procedure for non-classical logics.

Our unification algorithm has been implemented in Prolog and tested within a prototypical theorem prover for intuitionistic logic (see [15]). In the future we shall integrate it into a general proof procedure for non-classical logics (see e.g. [14]) and investigate the practical efficiency of the resulting proof technique. In particular we intend to compare it with a method which first translates these logics into classical logic and then uses one of the existing classical theorem provers.

# References

1. H. Abdulrab and J.-P. Pecuchet. Solving word equations. In C. Kirchner, editor, *Unification*, pages 353–375. Academic Press, London, 1990.
2. B. Beckert and J. Posegga. lean$T^A$P: Lean tableau-based theorem proving. In Alan Bundy, editor, *Proceedings of the 12$^{th}$ Conference on Automated Deduction*, LNAI 814, Springer Verlag, 1994.
3. E. W. Beth. *The foundations of mathematics*. North–Holland, 1959.
4. W. Bibel, S. Brüning, U. Egly, T. Rath. Komet. In Alan Bundy, editor, *Proceedings of the 12$^{th}$ Conference on Automated Deduction*, LNAI 814, pages 783–787. Springer Verlag, 1994.
5. W. Bibel. On matrices with connections. *Jour. of the ACM*, 28, p. 633–645, 1981.
6. W. Bibel. *Automated Theorem Proving*. Vieweg Verlag, 1987.
7. J. J. Herbrand. Recherches sur la théorie de la démonstration. *Travaux Soc. Sciences et Lettres Varsovie, Cl. 3 (Mathem., Phys.)*, page 128 pp., 1930. Engl. transl. in W. D. Goldfarb, ed. *J.J. Herbrand — Logical writings*, Reidel, 1971.
8. R. Letz, J. Schumann, S. Bayerl, W. Bibel. Setheo: A high-performance theorem prover. *Journal of Automated Reasoning*, 8:183–212, 1992.
9. G. S. Makanin. The problem of solvability of equations in a free semigroup. *Math. Sb.*, 103(145):147–236, 1977. English translation: Math. USSR Sb. 32.
10. G. S. Makanin. Algorithmic decidability of the rank of constant free equations in a free semigroup. *Dokl. Akad. Nauk, SSSR*, 243, 1978.
11. A. Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4:258–282, 1982.
12. H. J. Ohlbach. A resolution calculus for modal logics. Ph.D. Thesis (SEKI Report SR-88-08), Universität Kaiserslautern, 1988.
13. H. J. Ohlbach. A resolution calculus for modal logics. In E. Lusk and R. Overbeek, editors, *Proceedings of the 9$^{th}$ Conference on Automated Deduction*, LNCS 310, pages 500–515. Springer Verlag, 1988.
14. J. Otten, C. Kreitz. A connection based proof method for intuitionistic logic. In *Proceedings of the 4$^{th}$ Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 918, pp. 122–137, Springer Verlag, 1995.
15. J. Otten. Ein konnektionenorientiertes Beweisverfahren für intuitionistische Logik. Master's thesis, TH Darmstadt, 1995.
16. G. Plotkin. Building-in Equational Theories. *Machine Intelligence*, 7:73–90, Edinburgh University Press, 1972.
17. J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, January 1965.
18. M. Schmidt-Schauss. Unification in a combination of arbitrary disjoint equational theories. In E. Lusk and R. Overbeek, editors, *Proceedings of the 9$^{th}$ Conference on Automated Deduction*, LNCS 310, pages 378–392. Springer Verlag, 1988.
19. L. Wallen. *Automated deduction in nonclassical logic*. MIT Press, 1990.
20. L. Wos et. al. Automated reasoning contributes to mathematics and logic. In *Proceedings of the 10$^{th}$ Conference on Automated Deduction*, LNCS 449, pages 485–499. Springer Verlag 1990.