J. Symbolic Computation (1994) **17**, 421–455

Regular Path Expressions in Feature Logic[†]

ROLF BACKOFEN

DFKI, Stuhlsatzenhausweg 3, W6600 Saarbrücken, Germany

(Received 30 May 1994)

We examine the existential fragment of a feature logic, which is extended by regular path expressions. A regular path expression is a subterm relation, where the allowed paths for the subterms are restricted to any given regular language. In the area of computational linguistics, this notion has been introduced as "functional uncertainty". We will prove that satisfiability is decidable by constructing a quasi-terminating rule system.

1. Introduction

Feature descriptions are used as the main data structure of so-called unification grammars, which are currently a popular family of declarative formalisms for processing natural language (Shieber (1986)). More recently, feature descriptions have been proposed as a constraint system for logic programming (Aït-Kaci & Nasr (1986), (Aït-Kaci *et al.* 1987), Aït-Kaci & Podelski (1991), (Aït-Kaci *et al.* 1992), Smolka & Treinen (1992)). They provide for a partial description of abstract objects by means of functional attributes called features. As an example consider the feature description (in matrix notation):

$$x \quad : \quad \exists y \left[\begin{array}{lcl} \mathit{woman} & & \\ \mathit{father} & : & \left[\begin{array}{l} \mathit{engineer} \\ \mathit{age} : \quad y \end{array} \right] \\ & & \\ \mathit{husband} & : & \left[\begin{array}{l} \mathit{painter} \\ \mathit{age} : \quad y \end{array} \right] \end{array} \right],$$

which may be read as saying that x is a woman whose father is an engineer, whose husband is a painter and whose father and husband are both of the same age.

Feature description have been proposed in various forms with various formalizations (Aït-Kaci (1986), Kasper & Rounds (1986), Rounds & Kasper (1986), Johnson (1988),

[†] This work was supported by a research grant, ITW 9002 0, from the German Bundesministerium für Forschung und Technologie to the DFKI project DISCO

0747–7171/94/050421 + 35 $08.00/0

Smolka (1992), Johnson (1991)). We will follow the logical approach introduced by Smolka (1992), where feature descriptions are standard first order formulae interpreted in first order structures. In this formalization features are considered as functional relations. Atomic formulae (which we will call atomic constraints) are of the form $A(x)$ or xfy, where x, y are first order variables, A is some sort predicate and f is a feature (written in infix notation). Then we can express the above feature description by the (admittedly less suggestive) formula

$$\begin{array}{lll} \exists y, x_1, x_2 & (& \mathit{woman}(x) \;\wedge \\ & & x \; \mathit{father} \; x_1 \;\wedge\; \mathit{engineer}(x_1) \;\wedge\; x_1 \; \mathit{age} \; y) \;\wedge \\ & & x \; \mathit{husband} \; x_2 \;\wedge\; \mathit{painter}(x_2) \;\wedge\; x_2 \; \mathit{age} \; y \;). \end{array}$$

This feature logic has been investigated in detail. A complete axiomatization of the standard model (so-called feature graphs) is given in Backofen & Smolka (1993). There it was shown that the standard model is elementarily equivalent to a tree model. Additionally, a connection to first order constructor terms has been examined in Smolka & Treinen (1992).

In this paper we will be concerned with an extension to feature descriptions introduced as "functional uncertainty" by Kaplan & Zaenen (1988), and Kaplan & Maxwell III (1988). This extension is made by adding a subterm relation, where the allowed paths for the subterms are restricted to any given regular language. It was invented for handling so-called long-distance dependencies in the grammar formalism LFG (Kaplan & Bresnan (1982)). For a detailed description the reader is referred to Kaplan & Zaenen (1988). Further applications can be found in Keller (1991).

To accomplish this extension we must first generalize the constraints of the form xfy to constraints of the form xwy, where $w = f_1 \ldots f_n$ is a string of features (called a feature path). Such feature paths are interpreted using simple relational composition.

This generalization is just syntactic sugar (see Smolka (1988)). This is no longer the case if we add functional uncertainty in the form of constraints xLy, where L is a regular expression denoting a regular language of feature paths. A constraint xLy holds if there is a word $w \in L$ such that xwy holds. By this existential interpretation a constraint xLy can be seen as the disjunction

$$xLy = \bigvee \{xwy \mid w \in L\}.$$

Certainly, this disjunction may be infinite, thus functional uncertainty yields additional expressivity. Note that a constraint xwy is a special case of a functional uncertainty constraint.

To see some possible application of functional uncertainty, we briefly recall an example that is given in Kaplan & Maxwell III (1988), page 1. Consider the topicalized sentence

Mary John telephoned yesterday.

Using s as a variable denoting the whole sentence, the LFG-like clause

$$s \; \mathit{topic} \; x \wedge s \; \mathit{obj} \; x$$

specifies that in s *Mary* should be interpreted as the object of the relation *telephoned*. The sentence could be extended by introducing additional complement predicates, as e.g. in sentences like *Mary John claimed that Bill telephoned*; *Mary John claimed that Bill said that ... Henry telephoned yesterday*; .... For this family of sentences the clauses $s\ topic\ x \wedge s\ comp\ obj\ x$, $s\ topic\ x \wedge s\ comp\ comp\ obj\ x$ and so on would be appropriate; specifying all possibilities would yield an infinite disjunction. This changes if we make use of functional uncertainty allowing to specify the above as the single clause

$$s\ topic\ x \wedge s\ comp^*\ obj\ x.$$

Kaplan & Maxwell III (1988) have shown that the satisfiability problem of the pure existential fragment (i.e. the satisfiability of formulae built with $A(x)$, xLy and equations $x \doteq y$) is decidable, provided that a certain acyclicity condition is met. (Baader *et al.* 1993) have shown that satisfiability is undecidable if we add unrestricted negation. It has, however, remained an open problem whether satisfiability of the purely existential fragment is decidable in the absence of additional conditions (such as acyclicity). In this paper we will show that it is indeed decidable.

Short versions of this paper not containing the full proofs have appeared in Backofen (1993a) and Backofen (1993b).

2. The Method

We will first sketch the method for testing satisfiability of the standard feature descriptions, and then turn to the systems as extended by functional uncertainty. To get a good intuition note that some sort of tree model is canonical for satisfiability; a pure existential formula is satisfiable if it is satisfiable in this tree model. Thus, the feature paths used in the language can be compared directly with paths in trees.

Consider a clause $\phi = xp_1y_1 \wedge xp_2y_2$ (in the rest of the paper we will call pure conjunctive formulae clauses). Although only subterm relations for x, y_1 and x, y_2 are contained in this clause, an additional subterm or equality relation can be implied depending on the paths p_1 and p_2. If p_1 equals p_2, we know that y_1 and y_2 must be equal, which implies that ϕ is equivalent to $xp_1y_1 \wedge y_1 \doteq y_2$. If p_1 is a prefix of p_2 and hence $p_2 = p_1p'$, we can transform ϕ into the equivalent formula $xp_1y_1 \wedge y_1p'y_2$, thus additionally stating that y_2 is a subterm of y_1. The reverse case is handled similarly. If neither prefix nor equality holds between the paths, there is nothing to do. By and large, clauses where the last condition holds for every x and every pair of different constraints $xp_1y_1 \in \phi$ and $xp_2y_2 \in \phi$ are the solved forms of Smolka (1988), which are satisfiable.

If we consider a clause of the form $\phi = xL_1y_1 \wedge xL_2y_2$, then we have again to check the relation between y_1 and y_2. But now there is in general no unique relation determined by ϕ, since this depends on which paths p_1 and p_2 are used out of L_1 and L_2. Hence, we have to select non-deterministically a relation between p_1 and p_2 before we can calculate the relation between y_1 and y_2. In the following, we will often just say "guess" instead of "select non-deterministically".

But there is a problem with the original syntax, namely that it does not allow one to express any relation between the chosen paths[†]. Therefore, we extend the syntax by introducing so-called path variables (written $\alpha, \beta, \alpha', \ldots$), which are interpreted as feature paths. If we use in addition the modified subterm relation $x\alpha y$ and a restriction constraint $\alpha \dot{\in} L$, a path expression xLy can be expressed by the equivalent clause $x\alpha y \wedge \alpha \dot{\in} L$ (α new).

Using this extended (two-sorted) syntax we are now able to reason about the relations between different path variables. To do this we introduce additional constraints $\alpha \doteq \beta$ (equality), $\alpha \dot{\prec} \beta$ (prefix) and $\alpha \dot{\amalg} \beta$ (divergence). Divergence holds if neither equality nor prefix does. Now we can describe a normal form equivalent to the solved clauses in Smolka's work, which we will call pre-solved clauses. A clause ϕ is *pre-solved* if for each pair of different constraints $x\alpha y_1$ and $x\beta y_2$ in ϕ there is a constraint $\alpha \dot{\amalg} \beta$ in ϕ. Additionally, we require pre-solved clauses to contain at most one constraint $\alpha \dot{\in} L$ for each path variable α. We call these clauses pre-solved, since these clauses are not necessarily satisfiable: it may happen that the divergence constraints together with the restrictions of the form $\alpha \dot{\in} L$ are inconsistent (think of the clause $\alpha \dot{\in} f^+ \wedge \beta \dot{\in} ff^+ \wedge \alpha \dot{\amalg} \beta$, e.g.). But pre-solved clauses have the property that if we find a valuation for the path variables, then the clause is satisfiable.

Our algorithm first transforms a clause into a set of pre-solved clauses, which is (when viewed as a disjunction) equivalent to the initial clause. In a second phase the pre-solved clauses are checked for satisfiability with respect to the path variables. In both phases we use a set of deterministic and non-deterministic transformation rules.

Before starting with the technical part we will illustrate the first phase, since it is the more difficult. For the rest of the paper we will write clauses as sets of atomic constraints. Consider the clause $\gamma = \{x\alpha y,\ \alpha \dot{\in} L_1,\ x\beta z,\ \beta \dot{\in} L_2\}$. Initially, one guesses the relation between the path variables α and β. In our example there are four different possibilities. Therefore, γ can be expressed equivalently by the set of clauses

$$\begin{aligned}
\gamma_1 &= \{\alpha \dot{\amalg} \beta,\ x\alpha y,\ \alpha \dot{\in} L_1,\ x\beta z,\ \beta \dot{\in} L_2\}\\
\gamma_2 &= \{\alpha \doteq \beta,\ x\alpha y,\ \alpha \dot{\in} L_1,\ x\beta z,\ \beta \dot{\in} L_2\}\\
\gamma_3 &= \{\alpha \dot{\prec} \beta,\ x\alpha y,\ \alpha \dot{\in} L_1,\ x\beta z,\ \beta \dot{\in} L_2\}\\
\gamma_4 &= \{\beta \dot{\prec} \alpha,\ x\alpha y,\ \alpha \dot{\in} L_1,\ x\beta z,\ \beta \dot{\in} L_2\}.
\end{aligned}$$

The clause γ_1 is pre-solved. For the others we must evaluate the relation between α and β as follows. In γ_2 we substitute α for β and y for z, which yields

$$\{y \doteq z,\ x\alpha y,\ \alpha \dot{\in} L_1,\ \alpha \dot{\in} L_2\}.$$

We keep only the equality constraint for the first order variables since we are interested only in their valuation. Combining $\{\alpha \dot{\in} L_1, \alpha \dot{\in} L_2\}$ into $\{\alpha \dot{\in} (L_1 \cap L_2)\}$ will then give us an equivalent pre-solved clause. For γ_3 we know that the variable β can be split up into two parts, one of them covered by α. We can use concatenation of path variables to

[†] Maxwell and Kaplan solved this problem by using operations on regular languages such as intersection and calculating prefix languages directly. The use of this method forced them to introduce a new variable each time a transformation rule was applied. For a feature description that contains a cycle of the form $xL_1y_1 \wedge \ldots y_{n-1}L_nx$ this resulted in the introduction of an infinite number of variables.

express this, that means we can replace β by the term $\alpha{\cdot}\beta'$ with β' new. This would lead to the clause

$$\{\alpha \mathrel{\dot{\prec}} \alpha{\cdot}\beta',\ x\alpha y,\ \alpha \mathrel{\dot{\in}} L_1,\ x\alpha{\cdot}\beta' z,\ \alpha{\cdot}\beta' \mathrel{\dot{\in}} L_2\}.$$

But this could easily be expressed more simply. First, the constraint $\alpha \mathrel{\dot{\prec}} \alpha{\cdot}\beta'$ is superfluous. Second, the constraint $x\alpha{\cdot}\beta' z$ in combination with $x\alpha y$ can also be expressed by $\{x\alpha y,\ y\beta' z\}$. We now obtain the clause

$$\gamma_3' = \{x\alpha y,\ \alpha \mathrel{\dot{\in}} L_1,\ y\beta' z,\ \alpha{\cdot}\beta' \mathrel{\dot{\in}} L_2\}.$$

This shows that we do not need concatenation of path variables within subterm agreements, and we will avoid them for simplicity.

The only thing that remains in order to achieve a pre-solved clause is to resolve the constraint $\alpha{\cdot}\beta' \mathrel{\dot{\in}} L_2$. To do this we have to guess a *decomposition* P, S of L_2 with $P{\cdot}S = \{ps \mid p \in P, s \in S\} \subseteq L_2$ such that $\alpha \mathrel{\dot{\in}} P$ and $\beta' \mathrel{\dot{\in}} S$ holds. In general, there can be an infinite number of decompositions (think of the possible decompositions of the language f^*g). But as we use regular languages, there is a finite set of regular decompositions which covers all possibilities. Finally, reducing $\{\alpha \mathrel{\dot{\in}} L_1,\ \alpha \mathrel{\dot{\in}} P\}$ to $\{\alpha \mathrel{\dot{\in}} (L_1 \cap P)\}$ will yield a pre-solved clause.

Note that the evaluation of the prefix relation in γ_3 has the additional effect of introducing a new constraint $y\beta' z$. In general this implies that after the evaluation of prefix constraints there may be some path variables whose relation is unknown. Hence, after reducing the terms of form $\alpha \doteq \beta$ or $\alpha \mathrel{\dot{\prec}} \beta$, we may have to repeat the non-deterministic choice of relation between path variables. In the end, the only remaining constraints between path variables are of form $\alpha \mathrel{\dot{\amalg}} \beta$.

Now let's turn to an additional point we have to consider, namely that the rules we present will (naturally) loop in some cases. Roughly speaking, one can say that this occurs if a cycle in the graph co-incides with a cycle in the regular language. To see this let us vary the above example and let γ be the clause

$$\{x\alpha x,\ \alpha \mathrel{\dot{\in}} f,\ x\beta z,\ \beta \mathrel{\dot{\in}} f^*g\}$$

Then a possibly looping derivation could be

$\{\alpha \mathrel{\dot{\prec}} \beta,\ x\alpha x,\ \alpha \mathrel{\dot{\in}} f,\ x\beta z,\ \beta \mathrel{\dot{\in}} f^*g\}$	adding relation $\alpha \mathrel{\dot{\prec}} \beta$
$\{x\alpha x,\ \alpha \mathrel{\dot{\in}} f,\ x\beta' z,\ \alpha{\cdot}\beta' \mathrel{\dot{\in}} f^*g\}$	splitting β into $\alpha{\cdot}\beta'$
$\{x\alpha x,\ \alpha \mathrel{\dot{\in}} f,\ x\beta' z,\ \alpha \mathrel{\dot{\in}} f^*,\ \beta' \mathrel{\dot{\in}} f^*g\}$	decomposing $\alpha{\cdot}\beta' \mathrel{\dot{\in}} f^*g$
$\{x\alpha x,\ \alpha \mathrel{\dot{\in}} f,\ x\beta' z,\ \beta' \mathrel{\dot{\in}} f^*g\}$	joining α-restrictions

But we will prove that we get a quasi-terminating rule system, which means that the rule system may cycle, but produces only finitely many different clauses (see Dershowitz (1987)). This is achieved by the following measures: first, we will guarantee that the rules do not introduce additional variables; second, we restrict concatenation to length 2; and third, we will show that the rule system produces only finitely many regular languages. In order to show that our rewrite system is complete, we must additionally show that every solution can be found in a pre-solved clause.

3. Preliminaries

Throughout this paper we assume a signature consisting of a set $\mathcal{S}$ of **sorts** $A, B, \ldots$, a set $\mathcal{F}$ of **features** $f, g, \ldots$, a set $\mathcal{X}$ of **first order variables** $x, y, \ldots$, and a set $\mathcal{P}$ of **path variables** $\alpha, \beta, \ldots$ We use a finite set of features and infinite sets of variables and sorts. The sets $\mathcal{S} \cup \mathcal{F}$, $\mathcal{X}$ and $\mathcal{P}$ are pairwise disjoint.

A **path** is a finite string of features. We identify a feature f with the string (f) consisting of a single feature. We say that a path u is a **prefix** of a path v (written $u \prec v$) if there is a non-empty path w such that $v = uw$. Note that $\prec$ is neither symmetric nor reflexive. We say that two paths u, v **diverge** (written $u \amalg v$) if there are features f, g with $f \neq g$, and possibly empty paths w, w_1, w_2, such that $u = wfw_1 \wedge v = wgw_2$. It is clear that $\amalg$ is a symmetric relation.

Proposition 3.1. *Given two paths u and v, then exactly one of the relations $u = v$, $u \prec v$, $u \succ v$ or $u \amalg v$ holds.*

A **path term**, denoted $p, q, \ldots$, is either a path variable α or a concatenation of path variables $\alpha \cdot \beta$. We will allow complex path terms only in divergence and restriction constraints, but not in prefix or equality constraints. The set of atomic constraints is given by

$$
\begin{array}{rll}
c \rightarrow & Ax & \textbf{sort restriction} \\
& x \doteq y & \textbf{agreement} \\
& x\ f_1 \ldots f_n\ y & \textbf{subterm agreement 1} \\
& x \alpha y & \textbf{subterm agreement 2} \\
& p \mathrel{\dot\in} L & \textbf{path restriction} \\
& p \mathrel{\dot\amalg} q & \textbf{divergence} \\
& \alpha \mathrel{\dot\prec} \beta & \textbf{prefix} \\
& \alpha \doteq \beta & \textbf{path equality}
\end{array}
$$

We exclude empty paths in subterm agreements, since $x \epsilon y$ is equivalent to $x \doteq y$. Therefore, we require $f_1 \ldots f_n \in \mathcal{F}^+$. L is a regular expression denoting a regular language $\mathcal{L}(L) \subseteq \mathcal{F}^+$. In the following we will not differentiate between the regular expression and the language it denotes, and we will feel free to mix both.

A **clause** is either the special symbol $\bot$ ("false") or a finite set of atomic constraints denoting their conjunction. We will say that a path term $\alpha \cdot \beta$ is **contained** (or **used**) **in** some clause ϕ if ϕ contains either a constraint $\alpha \cdot \beta \mathrel{\dot\in} L$ or a constraint $\alpha \cdot \beta \mathrel{\dot\amalg} q$.[†] Constraints of the form $p \mathrel{\dot\in} L$, $p \mathrel{\dot\amalg} q$, $\alpha \mathrel{\dot\prec} \beta$ and $\alpha \doteq \beta$ will be called **path constraints**. We use the term $\mathcal{V}ars_{\mathcal{X}}(\phi)$ to denote the set of first order variables used in some clause ϕ, and the term $\mathcal{V}ars_{\mathcal{P}}(\phi)$ to denote the set of path variables used in ϕ.

An **interpretation** $\mathcal{I}$ is a standard first order structure, where every feature $f \in \mathcal{F}$ is interpreted as a binary, functional relation $f^{\mathcal{I}}$, and where sort symbols are interpreted as disjoint, unary predicates (hence $A^{\mathcal{I}} \cap B^{\mathcal{I}} = \emptyset$ for $A \neq B$). A **valuation** is a pair

[†] We will not distinguish between $p \mathrel{\dot\amalg} q$ and $q \mathrel{\dot\amalg} p$.

$(V_\mathcal{X}, V_\mathcal{P})$, where $V_\mathcal{X}$ is a standard first order valuation of the variables in $\mathcal{X}$ and $V_\mathcal{P}$ is a function $V_\mathcal{P} : \mathcal{P} \to \mathcal{F}^+$. We define $V_\mathcal{P}(\alpha{\cdot}\beta)$ to be $V_\mathcal{P}(\alpha)V_\mathcal{P}(\beta)$.

The **validity** of an atomic constraint in an interpretation $\mathcal{I}$ under a valuation $(V_\mathcal{X}, V_\mathcal{P})$ is defined as follows:

$$
\begin{array}{lll}
(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} Ax & :\Longleftrightarrow & V_\mathcal{X}(x) \in A^\mathcal{I} \\
(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} x \doteq y & :\Longleftrightarrow & V_\mathcal{X}(x) = V_\mathcal{X}(y) \\
(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} x\ f_1 \dots f_n\ y & :\Longleftrightarrow & V_\mathcal{X}(x)\ \ f_1^\mathcal{I} \circ \dots \circ f_n^\mathcal{I}\ \ V_\mathcal{X}(y) \\
(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} x\alpha y & :\Longleftrightarrow & (V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} x\ V_\mathcal{P}(\alpha)\ y \\
(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} p \mathrel{\dot\in} L & :\Longleftrightarrow & V_\mathcal{P}(p) \in L \\
(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} p \mathrel{\dot\diamond} q & :\Longleftrightarrow & V_\mathcal{P}(p) \diamond V_\mathcal{P}(q) \text{ for } \diamond \in \{\amalg, \prec, =\}
\end{array}
$$

Note that subterm agreement 2 is the only constraint where an interaction between $V_\mathcal{X}$ and $V_\mathcal{P}$ happens. The validity of sort restriction, agreement and subterm agreement 1 depend only on $V_\mathcal{X}$ and $\mathcal{I}$. Hence, we will sometimes omit the path valuation $V_\mathcal{P}$ and write $V_\mathcal{X} \models_\mathcal{I} \phi$ if ϕ consists only of these forms of constraint and ϕ is valid under $\mathcal{I}$ and $V_\mathcal{X}$. Similar, validity of path constraints depend only on the path valuation. We will write $V_\mathcal{P} \models \phi$ if ϕ is a clause consisting of path constraints that are valid under $V_\mathcal{P}$.

For checking satisfiability of clauses we will use a set of deterministic and non-deterministic transformation rules. Which set of rules is used will depend on the initial clause. Let ϕ be a clause and r be a rule instance. We say that r is **applicable** on ϕ if ϕ matches the definition of r and the application conditions noted in the definition of r are satisfied. We will write $\phi \to_r \gamma$ if r is applicable on ϕ and the result of the application is γ. For a set of rules $\mathcal{R}$ we say $\phi \to_\mathcal{R} \gamma$ if there is an $r \in \mathcal{R}$ with $\phi \to_r \gamma$. ϕ is called **$\mathcal{R}$-irreducible** if no rule instance $r \in \mathcal{R}$ applies to ϕ. We will say that a clause ϕ is $\mathcal{R}$-reducible if ϕ is not $\mathcal{R}$-irreducible. A sequence

$$\phi_0 \to_{r_0} \phi_1 \cdots \phi_i \to_{r_i} \phi_{i+1} \cdots$$

is called a **derivation**. A clause γ is called a **$\mathcal{R}$-derivative of** ϕ if there is a derivation from ϕ to γ that uses only rule instances of $\mathcal{R}$.

Since we have a two-sorted logic, we have to redefine the notions of soundness and preservingness. For a set $\xi \subseteq \mathcal{X}$ we define $=_\xi$ to be the following relation on first order valuation:

$$V_\mathcal{X} =_\xi V'_\mathcal{X} \quad \text{iff} \quad \text{for all } x \in \xi \text{ the equation } V_\mathcal{X}(x) = V'_\mathcal{X}(x) \text{ holds.}$$

Similarly, we define $=_\pi$ with $\pi \subseteq \mathcal{P}$ for path valuations. Let $\vartheta \subseteq \mathcal{X} \cup \mathcal{P}$ be a set of variables. For a given interpretation $\mathcal{I}$ we say that a valuation $(V_\mathcal{X}, V_\mathcal{P})$ is a **ϑ-solution** of a clause ϕ in $\mathcal{I}$ if there is a valuation $(V'_\mathcal{X}, V'_\mathcal{P})$ in $\mathcal{I}$ such that

$$V_\mathcal{X} =_{\mathcal{X}\cap\vartheta} V'_\mathcal{X}, \quad V_\mathcal{P} =_{\mathcal{P}\cap\vartheta} V'_\mathcal{P} \ \text{ and } \ (V'_\mathcal{X}, V'_\mathcal{P}) \models_\mathcal{I} \phi.$$

The set of all ϑ-solutions of ϕ in $\mathcal{I}$ is denoted by $[\![\phi]\!]^\mathcal{I}_\vartheta$. We call $\mathcal{X}$-solutions just solutions and write $[\![\phi]\!]^\mathcal{I}$ instead of $[\![\phi]\!]^\mathcal{I}_\mathcal{X}$.† A clause ϕ is **ϑ-equivalent** to a clause γ (resp. a set of

† By calling $\mathcal{X}$-solutions just solutions we intend to suggest that X-solutions are the interesting one. The are two reasons for concentrating on $\mathcal{X}$-solutions rather than $\mathcal{X} \cup \mathcal{P}$-solutions. First, there are only first order variables in the original Kaplan/Maxwell syntax, and we have added path variables as an

clauses Γ) if for every interpretation $\mathcal{I}$, $[\![\phi]\!]^{\mathcal{I}}_{\vartheta} = [\![\gamma]\!]^{\mathcal{I}}_{\vartheta}$ (resp. $[\![\phi]\!]^{\mathcal{I}}_{\vartheta} = \bigcup_{\gamma\in\Gamma}[\![\gamma]\!]^{\mathcal{I}}_{\vartheta}$). Again we use equivalent as short for $\mathcal{X}$-equivalent.

A rule R is **ϑ-sound** if $\phi \rightarrow_R \gamma$ implies $[\![\phi]\!]^{\mathcal{I}}_{\vartheta} \supseteq [\![\gamma]\!]^{\mathcal{I}}_{\vartheta}$ for every interpretation $\mathcal{I}$. R is called **ϑ-preserving** if $\phi \rightarrow_R \gamma$ implies $[\![\phi]\!]^{\mathcal{I}}_{\vartheta} \subseteq [\![\gamma]\!]^{\mathcal{I}}_{\vartheta}$ for every $\mathcal{I}$. And R is **globally ϑ-preserving** if

$$\forall \mathcal{I}: \quad [\![\phi]\!]^{\mathcal{I}}_{\vartheta} \subseteq \bigcup_{\phi\rightarrow_R\gamma} [\![\gamma]\!]^{\mathcal{I}}_{\vartheta}.$$

4. Feature Trees

In this section we will establish two different interpretations, namely the feature tree structure and the rational feature tree structure. These interpretations are canonical for satisfiability. This means that if a clause is satisfiable, then it is also satisfiable in these interpretations. These models were introduced in (Aït-Kaci *et al.* 1992), Smolka & Treinen (1992) and Backofen & Smolka (1993). In Backofen & Smolka (1993) a complete axiomatization of the the full first order theory of these models with respect to a restricted syntax has been set up. The restricted syntax uses only Ax, xfy and $x \doteq y$ as atomic constraints.

A **tree domain** is a nonempty set $D \subseteq \mathcal{F}^*$ of paths that is **prefix-closed**, that is, if $wu \in D$, then $w \in D$. Note that every tree domain contains the empty path.

A **feature tree** is a partial function $\sigma\colon \mathcal{F}^* \rightarrow \mathcal{S}$ whose domain is a tree domain. The paths in the domain of a feature tree represent the nodes of the tree; the empty path represents its root. We use D_σ to denote the domain of a feature tree σ. A feature tree is called **finite [infinite]** if its domain is finite [infinite]. The letters σ and τ will always denote feature trees.

The subtree $w^{-1}\sigma$ of a feature tree σ at a path $w \in D_\sigma$ is the feature tree defined by (in relational notation)

$$w^{-1}\sigma \quad := \quad \{(u, A) \mid (wu, A) \in \sigma\}.$$

A feature tree σ is called a **subtree** of a feature tree τ if σ is a subtree of τ at some path $w \in D_\tau$, and a **direct subtree** if $w = f$ for some feature f.

A feature tree σ is called **rational** if σ has only finitely many distinct subtrees.

The **feature tree structure** $\mathcal{T}$ is defined as follows:

the universe of $\mathcal{T}$ is the set of all feature trees
$\sigma \in A^{\mathcal{T}}$ iff $\sigma(\varepsilon) = A$ (i.e., σ's root is labeled with A)
$(\sigma, \tau) \in f^{\mathcal{T}}$ iff $f \in D_\sigma$ and $\tau = f^{-1}\sigma$ (i.e., τ is the subtree of σ at f).

additional data structure. Our $\mathcal{X}$-solutions will be solutions for the corresponding clauses in the original syntax. And second, all of the rules we will present will preserve the solutions (i.e., $\mathcal{X}$-solutions) of a clause, but not necessarily the $\mathcal{X} \cup \mathcal{P}$-solutions.

The **rational feature tree structure** $\mathcal{R}$ is the substructure of $\mathcal{T}$ consisting only of rational feature trees.

5. Prime, Pre-Solved and Solved Clauses

In this section, we will define the input and output clauses for both phases of the algorithm.

Let ϕ be some clause and x, y be distinct variables. We say that ϕ **binds** y **to** x if $x \doteq y \in \phi$ and y occurs only once in ϕ.† Here it is important that we consider equations as directed, that is, we assume that $x \doteq y$ is different from $y \doteq x$. We say that ϕ **eliminates** y if ϕ binds y to some variable x. A clause is called **basic** if it is either $\bot$ or:

1 ϕ contains no constraint of the form $x\, f_1 \ldots f_n\, y$,
2 an equation $x \doteq y$ appears in ϕ if and only if ϕ eliminates y, and
3 for every path variable α used in ϕ there is *at most* one constraint $x\alpha y \in \phi$.

A clause ϕ is called **prime** if it is basic, does not contain a path term of the form $\alpha{\cdot}\beta$ and does not contain an atomic constraint of form $p \mathrel{\dot{\amalg}} q$, $\alpha \mathrel{\dot{\prec}} \beta$ or $\alpha \doteq \beta$.

The following two clauses are not basic. The first clause does not satisfy condition 2, and the second clause does not satisfy condition 3:

$$\{y \doteq z,\ \ x\alpha y,\ x\alpha' z,\ \alpha \mathrel{\dot{\in}} f,\ \alpha' \mathrel{\dot{\in}} (f \cup g)\}$$
$$\{\alpha \mathrel{\dot{\amalg}} \alpha',\ x\alpha y,\ y\alpha z,\ x\alpha' z,\ \alpha \mathrel{\dot{\in}} f,\ \alpha' \mathrel{\dot{\in}} (f \cup g)\}$$

On the other hand, the clause

$$\{y \doteq z,\ \alpha \mathrel{\dot{\amalg}} \alpha',\ x\alpha y,\ x\alpha' y',\ \alpha \mathrel{\dot{\in}} f,\ \alpha' \mathrel{\dot{\in}} (f \cup g)\}$$

is an example of a basic clause which is not prime.

As mentioned, Kaplan and Maxwell stated the satisfiability problem for functional uncertainty in an unsorted syntax. Essentially, this syntax consists of the atomic constraints Ax, $x\, f_1 \ldots f_n\, y$ and $x \doteq y$ together with the additional constraint xLy. This constraint is interpreted as

$$xLy = \bigvee \{xwy \mid w \in L\}.$$

It is easy to show that every clause in this syntax can be transformed into an equivalent prime clause.

† Note that we diverge from the standard practise in treating equality as binding its right argument. This is for uniformity with constraints involving the $\dot{\prec}$ relation, since they will always share a left argument which we wish to avoid renaming.
We find the commuted notation with $\dot{\succ}$ in place of the prefix relation $\dot{\prec}$ even less natural.

PROPOSITION 5.1. *Every clause ϕ in the Kaplan/Maxwell syntax can be translated into a prime clause γ such that for every interpretation $\mathcal{I}$ and for every first order valuation $V_{\mathcal{X}}$*

$$V_{\mathcal{X}} \models_{\mathcal{I}} \phi \iff \textit{there is a } V_{\mathcal{P}} \textit{ with } (V_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{I}} \gamma.$$

PROOF. The translation can be defined by the two rewrite rules

$$\text{(Rename)} \qquad \frac{\{xLy\} \cup \psi}{\{x\alpha y, \alpha \mathrel{\dot\in} L\} \cup \psi} \qquad \alpha \text{ new}$$

$$\text{(Elim)} \qquad \frac{\{x \doteq y\} \cup \psi}{\{x \doteq y\} \cup \psi[x \leftarrow y]} \qquad x \neq y, \quad x \in \mathcal{V}ars_{\mathcal{X}}(\psi)$$

It is easy to check that the system consisting of these two rules will always terminate and that the result satisfies the required conditions. □

This proposition shows that it suffices to check satisfiability of prime clauses in order to check satisfiability of clauses in the Kaplan/Maxwell syntax. Hence, prime clauses are the input clauses for the first phase.

Taking as an example the clause $s\ topic\ x \ \wedge\ s\ comp^*\ obj\ x$ in the original Kaplan/Maxwell syntax, the prime clause that is yielded by the above translation is

$$s\alpha_1 x \wedge s\alpha_2 x \wedge \alpha_1 \mathrel{\dot\in} topic \wedge \alpha_2 \mathrel{\dot\in} comp^*\ obj.$$

Now we turn to the output clauses of the first phase. A basic clause is said to be **pre-solved** if it is either $\perp$ or the following holds:

1 $Ax \in \phi$ and $Bx \in \phi$ implies $A = B$,
2 $\alpha \mathrel{\dot\in} L \in \phi$ and $\alpha \mathrel{\dot\in} L' \in \phi$ implies $L = L'$,
3 $\alpha \mathrel{\dot\in} \emptyset$ is not in ϕ,
4 ϕ contains no term of form $\alpha \cdot \beta$,
5 ϕ contains no constraint of form $\alpha \doteq \beta$ or $\alpha \mathrel{\dot\prec} \beta$, and
6 $\alpha \mathrel{\dot{\amalg}} \beta \in \phi$ if and only if $\alpha \neq \beta$, $x\alpha y \in \phi$ and $x\beta z \in \phi$.

For example the clause

$$\left\{ \begin{array}{c} \alpha_1 \mathrel{\dot{\amalg}} \alpha_2,\ \alpha_1 \mathrel{\dot{\amalg}} \alpha_3,\ \alpha_2 \mathrel{\dot{\amalg}} \alpha_3,\ x\alpha_1 y_1,\ x\alpha_2 y_2,\ x\alpha_3 y_3, \\ \alpha_1 \mathrel{\dot\in} (f \cup g),\ \alpha_2 \mathrel{\dot\in} (f^+ \cup h^+),\ \alpha_3 \mathrel{\dot\in} h(f \cup g) \end{array} \right\}$$

is pre-solved. The following clauses are not pre-solved. The first violates condition 4, the second violates condition 5, and the third and fourth do not satisfy condition 6:

$$\{\alpha_1 \dot{\amalg}\, \alpha_2, x\alpha_1 y_1,\ x\alpha_2 y_2,\ \alpha_1 \dot{\in} (f \cup g),\ \alpha_1{\cdot}\alpha_2 \dot{\in} (f^+ \cup h^+)\}$$
$$\{\alpha_1 \dot{\prec} \alpha_2,\ x\alpha_1 y_1,\ x\alpha_2 y_2,\ \alpha_1 \dot{\in} (f \cup g),\ \alpha_2 \dot{\in} (f^+ \cup h^+)\}$$
$$\{x\alpha_1 y_1,\ x\alpha_2 y_2,\ \alpha_1 \dot{\in} (f \cup g),\ \alpha_2 \dot{\in} (f^+ \cup h^+)\}$$
$$\{\alpha_1 \dot{\amalg}\, \alpha_2,\ x\alpha_1 y_1,\ y_1\alpha_2 y_2,\ \alpha_1 \dot{\in} (f \cup g),\ \alpha_2 \dot{\in} (f^+ \cup h^+)\}$$

LEMMA 5.2. *Let ϕ be a pre-solved clause different from $\bot$. Then ϕ is satisfiable iff there is a path valuation $V_{\mathcal{P}}$ with $V_{\mathcal{P}} \models \phi_p$, where ϕ_p is the set of path constraints in ϕ.*

PROOF. Without loss of generality we can assume that for every $x \in \mathcal{V}ars_{\mathcal{X}}(\phi)$ there is a sort restriction $Ax \in \phi$. Let

$$\gamma = \{xV_{\mathcal{P}}(\alpha)y \mid x\alpha y \in \phi\} \cup \{Ax \in \phi\}.$$

Then for each interpretation $\mathcal{I}$ and each first order valuation $V_{\mathcal{X}}$ we have $(V_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{I}} \phi$ iff $V_{\mathcal{X}} \models_{\mathcal{I}} \gamma$.

Let $\mathcal{T}$ be the feature tree model as defined before. For a feature tree σ and each word $w \in \mathcal{F}^+$ we define $w\sigma$ to be the feature tree

$$w\sigma = \{(wu, A) \mid (u, A) \in \sigma\}.$$

It is easy to check that $w^{-1}w\sigma = \sigma$ holds, but not in general $ww^{-1}\sigma = \sigma$.

For every $n \in \mathcal{N}$ we define $V^n_{\mathcal{X}}$ to be the following first order valuation on $\mathcal{V}ars_{\mathcal{X}}(\gamma)$:

1 $V^0_{\mathcal{X}}(x) = \{(\epsilon, A)\}$, where $Ax \in \gamma$,
2 $V^{n+1}_{\mathcal{X}}(x) = \{(\epsilon, A)\} \uplus \biguplus_{xwy \in \gamma} wV^n_{\mathcal{X}}(y)$, where $Ax \in \gamma$.

The union in the definition of $V^{n+1}_{\mathcal{X}}(x)$ is a disjoint union as $w \in \mathcal{F}^+$ and $\forall xwy, xuz \in \gamma : w \neq u \Rightarrow w \amalg u$ by the pre-solved conditions 5–6. Thus we can prove by induction that for each $n \geq 1$

1 $xwy \in \gamma$ implies $w^{-1}V^n_{\mathcal{X}}(x) = V^{n-1}_{\mathcal{X}}(y)$,
2 $V^n_{\mathcal{X}}(x) \supseteq V^{n-1}_{\mathcal{X}}(x)$, and
3 $V^n_{\mathcal{X}}(x)$ is a partial function $\mathcal{F}^\star \to \mathcal{S}$.

Now we define $V_{\mathcal{X}}$ to be the valuation with

$$V_{\mathcal{X}}(x) = \bigcup_{n \in \mathcal{N}} V^n_{\mathcal{X}}(x)$$

By the above propositions for $V^n_{\mathcal{X}}$ we know that $w^{-1}V_{\mathcal{X}}(x) = V_{\mathcal{X}}(y)$ holds for each $xwy \in \gamma$. Although $V_{\mathcal{X}}(x)$ is a partial function $\mathcal{F}^\star \to \mathcal{S}$ for every x, it is not yet a valuation in $\mathcal{T}$ since the $V_{\mathcal{X}}(x)$ are not necessarily prefix-closed. This can be overcome by defining $V'_{\mathcal{X}}$ to be the valuation

$$V'_{\mathcal{X}}(x) = V_{\mathcal{X}}(x) \uplus \{\, (w, A) \mid \quad \forall C : (w, C) \notin V_{\mathcal{X}}(x) \\ \wedge\ \exists u \neq \epsilon, B : (wu, B) \in V_{\mathcal{X}}(x) \,\},$$

where A is an arbitrary but fixed sort symbol. Then again $w^{-1}V_{\mathcal{X}}(x) = V_{\mathcal{X}}(y)$ holds for each $xwy \in \gamma$. This implies that $V'_{\mathcal{X}}(x) w^{\mathcal{T}} V'_{\mathcal{X}}(y)$ holds and hence $V'_{\mathcal{X}} \models_{\mathcal{T}} \gamma$. But then we get $(V'_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{T}} \phi$. Since the feature trees $V'_{\mathcal{X}}(x)$ are even rational, we get also $(V_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{R}} \phi$ $\square$

Note that this implies that the structure $\mathcal{T}$ (resp. $\mathcal{R}$) is **canonical** for pre-solved clauses; that is, a normal form clause is satisfiable if it is satisfiable in $\mathcal{T}$ (resp. $\mathcal{R}$). Since in the first phase we transform each prime clause into an equivalent set of pre-solved clauses, we know that $\mathcal{T}$ is also canonical for prime clauses.

In the second phase we will check satisfiability of a pre-solved clause by transforming it into an equivalent set of solved clauses. A clause ϕ is called **solved** if it is either $\perp$ or

1 $Ax \in \phi$ and $Bx \in \phi$ implies $A = B$,
2 $\alpha \mathrel{\dot\in} L \in \phi$ and $\alpha \mathrel{\dot\in} L' \in \phi$ implies $L = L'$,
3 $\alpha \mathrel{\dot\in} \emptyset$ is not in ϕ,
4 ϕ contains no term of form $\alpha{\cdot}\beta$,
5 ϕ contains no constraint of form $\alpha \doteq \beta$, $\alpha \mathrel{\dot\prec} \beta$ or $\alpha \mathrel{\dot{\amalg}} \beta$, and
6 for every pair of variables α, β such that $\alpha \neq \beta$, $x\alpha y \in \phi$ and $x\beta z \in \phi$, we have $\phi \models \alpha \mathrel{\dot{\amalg}} \beta$.

Here $\phi \models \gamma$ means that for every $\mathcal{I}$ and every $(V_{\mathcal{X}}, V_{\mathcal{P}})$ in $\mathcal{I}$, $\quad (V_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{I}} \phi$ implies $(V_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{I}} \gamma$. Note that the definitions of pre-solvedness and solvedness differ in the last two conditions and that every solved clause is also a prime clause. The clause

$$\{x\alpha y,\ x\alpha' z, \alpha \mathrel{\dot\in} (f \cup g),\ \alpha' \mathrel{\dot\in} h\}$$

is a solved clause, whereas

$$\phi = \{x\alpha y,\ x\alpha' z, \alpha \mathrel{\dot\in} (f \cup g),\ \alpha' \mathrel{\dot\in} gh\}$$

is not solved since $\phi \not\models \alpha \mathrel{\dot{\amalg}} \alpha'$.

Lemma 5.3. *Every solved clause different from $\perp$ is satisfiable.*

Proof. For every solved clause ϕ there is a $\mathcal{X} \cup \mathcal{P}$-equivalent clause γ such that γ is pre-solved, namely

$$\gamma = \{\alpha \mathrel{\dot{\amalg}} \beta \mid \alpha \neq \beta \wedge x\alpha y \in \phi \wedge x\beta z \in \phi\} \cup \phi.$$

Thus, a solved clause ϕ is (by lemma 5.2) satisfiable if there is a path valuation $V_{\mathcal{P}}$ with $V_{\mathcal{P}} \models \phi$. Now conditions 2–5 in the definition of solvedness guarantee that ϕ contains only path constraints of the form $\alpha \mathrel{\dot{\in}} L_\alpha$ with $L_\alpha \neq \emptyset$ (but no path constraints of the form $\alpha{\cdot}\alpha' \mathrel{\dot{\in}} L$, $\alpha \doteq \alpha'$, $\alpha \mathrel{\dot{\prec}} \alpha'$ or $p \mathrel{\dot{\amalg}} q$). Hence, every path valuation $V_\mathcal{P}$ with $V_\mathcal{P}(\alpha) \in L_\alpha$ for every $\alpha \in \mathcal{V}ars_\mathcal{P}(\alpha)$ satisfies $V_\mathcal{P} \models \phi$. Since $L_\alpha \neq \emptyset$ for every $\alpha \in \mathcal{V}ars_\mathcal{P}(\alpha)$, we know that there is at least one path valuation $V_\mathcal{P}$ with $V_\mathcal{P} \models \phi$. □

6. The First Phase

6.1. A Set of Rules

The first rule is the non-deterministic addition of relational constraints between path variables. In one step we will add the relations between one fixed variable α and all other path variables β which are used under the same node x as α. We will consider only the constraints $\alpha \doteq \beta$, $\alpha \mathrel{\dot{\amalg}} \beta$ and $\alpha \mathrel{\dot{\prec}} \beta$ but not $\alpha \mathrel{\dot{\succ}} \beta$. Thus the rule can be described by the following pseudo code:

> **Choose** $x \in \mathcal{V}ars_\mathcal{X}(\phi)$ (don't care)
>   **Choose** $x\alpha y \in \phi$ (don't know)
>   **For each** $x\beta z \in \phi$ **with** α different from β and $\alpha \mathrel{\dot{\amalg}} \beta \notin \phi$
>   **add** $\alpha \mathrel{\dot{\diamond}_\beta} \beta$ with $\dot{\diamond}_\beta \in \{\doteq, \dot{\prec}, \dot{\amalg}\}$ (don't know)

Formally, this rule is written as

$$\text{(PathRel)} \quad \frac{\{x\alpha y\} \cup \psi}{\{\alpha \mathrel{\dot{\diamond}_\beta} \beta \mid x\beta z \in \psi \wedge \alpha \neq \beta \wedge \alpha \mathrel{\dot{\amalg}} \beta \notin \psi\} \cup \{x\alpha y\} \cup \psi}$$

where $\dot{\diamond}_\beta \in \{\doteq, \dot{\prec}, \dot{\amalg}\}$.

This rule will only be applied if

> ψ contains no prefix and path equality constraint,
> ψ contains no path concatenation, and
> the rule adds at least one constraint.

Although we have restricted the relations $\dot{\diamond}_\beta$ to $\{\doteq, \dot{\prec}, \dot{\amalg}\}$, this rule is globally preserving since we have non-deterministically chosen $x\alpha y$. To see this let ϕ be a clause, $\mathcal{I}$ be an interpretation and $(V_\mathcal{X}, V_\mathcal{P})$ be a valuation in $\mathcal{I}$ with $(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} \phi$. To find an instance of (PathRel) such that $(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} \gamma$ where γ is the result of applying this instance, we choose $x\alpha y \in \phi$ with $V_\mathcal{P}(\alpha)$ $\prec$-minimal in

$$\{V_\mathcal{P}(\beta) \mid x\beta z \in \phi\}.$$

Then for each $x\beta z \in \phi$ with $\alpha \neq \beta$ and $\alpha \mathrel{\dot{\amalg}} \beta \notin \phi$ we add $\alpha \mathrel{\dot{\diamond}_\beta} \beta$ where $V_\mathcal{P}(\alpha) \diamond_\beta V_\mathcal{P}(\beta)$ holds. Note that $\dot{\diamond}_\beta$ equals $\dot{\succ}$ will not occur since we have chosen a path variable α the interpretation of which is $\prec$-minimal. Therefore, the restriction $\dot{\diamond}_\beta \in \{\doteq, \dot{\prec}, \dot{\amalg}\}$ is satisfied.

The definition of (PathRel) is more complex than the naive one in the introduction. The reason for this is that only by using this special definition can we maintain the condition that concatenation of path variables is restricted to binary concatenation. To see this suppose that we had added both $\beta_1 \dot{\prec} \alpha$ and $\alpha \dot{\prec} \beta_2$ to a clause γ. Then first splitting up the variable β_2 into $\alpha{\cdot}\beta_2'$ and then α into $\beta_1{\cdot}\alpha'$ will result in a substitution of β_2 in γ by $\beta_1{\cdot}\alpha'{\cdot}\beta_2'$. By the definition of (PathRel) we have ensured that this does not happen.

The second non-deterministic rule is used in the decomposition of regular languages. For decomposition we have the following rules:

$$\text{(DecClash)} \qquad \frac{\{\alpha{\cdot}\beta \,\dot{\in}\, L\} \cup \psi}{\bot} \qquad \text{if } \{w \in L \mid |w| > 1\} = \emptyset$$

$$(\text{LangDec}_\Lambda) \qquad \frac{\{\alpha{\cdot}\beta \,\dot{\in}\, L\} \cup \psi}{\{\alpha \,\dot{\in}\, P\} \cup \{\beta \,\dot{\in}\, S\} \cup \psi} \qquad P{\cdot}S \subseteq L$$

where $L, P, S \subseteq F^+$, $L, P, S \in \Lambda$, and L contains a path w with $|w| > 1$.

For a specific instance of the rule family LangDec_Λ, Λ must be a *finite* set of regular languages. The clash rule is needed since we require that regular languages do not contain the empty path.

We use Λ in (LangDec_Λ) as a global restriction, which means that for every Λ we get a different rule (LangDec_Λ) (and hence a different rule system $\mathcal{R}_\Lambda$). This is done as the rule system is quasi-terminating. By restricting (LangDec_Λ) we can guarantee that only finitely many regular languages are produced.

For (LangDec_Λ) to be globally preserving we need to find, for every possible valuation of α and β, a suitable pair P, S in Λ. Therefore, we require Λ to satisfy

$$\begin{array}{l}\forall L \in \Lambda,\ \forall w_1, w_2 \neq \epsilon : \\ \quad [w_1 w_2 \in L \Rightarrow \exists P, S \in \Lambda : (P{\cdot}S \subseteq L \wedge w_1 \in P \wedge w_2 \in S)].\end{array}$$

We will call Λ **closed under decomposition** if it satisfies this condition. Additionally, we have to ensure that $L \in \Lambda$ for every L that is contained in some clause ϕ. We will call such a set Λ **ϕ-closed**.

The remaining rules are listed in figure 1. Note that we have not considered clauses containing subterm agreement 1, since we have excluded this kind of constraints in the definition of a basic clause. This is possible because a constraint $x\ f_1 \ldots f_n\ y$ can be expressed by the equivalent clause $\{x\alpha y,\ \alpha \,\dot{\in}\, f_1 \ldots f_n\}$, where α is a new variable (see also the proof of proposition 5.1).

The (Pre) rule needs some additional explanation. One might expect (Pre) to be of the form

$$\text{(Pre')} \qquad \frac{\{\alpha \dot{\prec} \beta,\ x\alpha y,\ x\beta z\} \cup \psi}{\{x\alpha y,\ y\beta' z\} \cup \psi[\beta \leftarrow \alpha{\cdot}\beta']} \qquad \beta' \text{ new.}$$

$$\text{(Eq)}\quad \frac{\{\alpha \doteq \beta,\ x\alpha y,\ x\beta z\} \cup \psi}{\{y \doteq z,\ x\alpha y\} \cup \psi[\beta \leftarrow \alpha, z \leftarrow y]}$$

$$\text{(Join)}\quad \frac{\{\alpha \mathrel{\dot{\in}} L,\ \alpha \mathrel{\dot{\in}} L'\} \cup \psi}{\{\alpha \mathrel{\dot{\in}} (L \cap L')\} \cup \psi} \quad L \neq L'$$

$$\text{(Div1)}\quad \frac{\{\alpha \mathrel{\dot{\amalg}} \beta'\} \cup \{\alpha{\cdot}\beta \mathrel{\dot{\amalg}} \beta'\} \cup \psi}{\{\alpha \mathrel{\dot{\amalg}} \beta'\} \cup \psi}$$

$$\text{(Div2)}\quad \frac{\{\alpha{\cdot}\beta \mathrel{\dot{\amalg}} \alpha{\cdot}\beta'\} \cup \psi}{\{\beta \mathrel{\dot{\amalg}} \beta'\} \cup \psi}$$

$$\text{(DClash1)}\quad \frac{\{\alpha{\cdot}\beta \mathrel{\dot{\amalg}} \alpha\} \cup \psi}{\perp}$$

$$\text{(DClash2)}\quad \frac{\{\alpha \mathrel{\dot{\amalg}} \alpha\} \cup \psi}{\perp}$$

$$\text{(Empty)}\quad \frac{\{\alpha \mathrel{\dot{\in}} \emptyset\} \cup \psi}{\perp}$$

$$\text{(SClash)}\quad \frac{\{Ax,\ Bx\} \cup \psi}{\perp} \quad A \neq B$$

$$\text{(Pre)}\quad \frac{\{\alpha \mathrel{\dot{\prec}} \beta,\ x\alpha y,\ x\beta z\} \cup \psi}{\{x\alpha y,\ y\beta z\} \cup \psi[\beta \leftarrow \alpha{\cdot}\beta]} \quad \alpha \neq \beta$$

Figure 1. Simplification rules

But as we have mentioned, we have to define our rules in a way such that no additional variables are introduced. This is not satisfied by the rule (Pre'). For solving this problem note that β is not used in the result of applying (Pre'). Hence, we can substitute β' by β, which has the effect that no new variable is needed. This leads to the definition of (Pre) as presented in figure 1.

The following proposition and lemma will show that the definition of (LangDec$_\Lambda$) is meaningful.

PROPOSITION 6.1. *If Λ is ϕ-closed and closed under intersection, then Λ is γ-closed for all $\mathcal{R}_\Lambda$-derivatives γ of ϕ.*

PROOF. We will prove this lemma by induction over the length of derivations. We use the term $\mathrm{reg}(\gamma)$ to denote the set of regular languages used in γ. Then $\mathcal{R}_\Lambda$ is γ-closed if $\mathrm{reg}(\gamma) \subseteq \Lambda$.

Let γ be some $\mathcal{R}_\Lambda$-derivative of ϕ. For the base step $\gamma = \phi$ the lemma holds trivially. For the induction step let γ satisfy the induction hypotheses $\mathrm{reg}(\gamma) \subseteq \Lambda$ and let $r \in \mathcal{R}_\Lambda$ be a rule such that $\gamma \rightarrow_r \gamma'$.

If r is some clash rule, then $\mathrm{reg}(\gamma') = \emptyset$.

If r is not a clash rule and not in (LangDec$_\Lambda$) or (Join), then $\mathrm{reg}(\gamma') = \mathrm{reg}(\gamma)$ and therefore $\mathrm{reg}(\gamma') \subseteq \Lambda$ by induction hypotheses. If $r \in$ (LangDec$_\Lambda$), then r adds only regular languages $P, S \in \Lambda$.

Now let

$$r' = \frac{\{\alpha \mathrel{\dot\in} L,\ \alpha \mathrel{\dot\in} L'\} \cup \psi}{\{\alpha \mathrel{\dot\in} (L \cap L')\} \cup \psi} \in (\text{Join}).$$

By induction hypotheses we know that $L, L' \in \Lambda$. But then $(L \cap L') \in \Lambda$ since Λ is closed under intersection. $\square$

LEMMA 6.2. *For every prime clause ϕ there is a* finite Λ *such that Λ is ϕ-closed, closed under intersection and decomposition.*

PROOF. We define a deterministic automaton $\mathcal{A}$ over $\mathcal{F}$ to be a tuple $(Q_\mathcal{A}, i_\mathcal{A}, \delta_\mathcal{A}, Fin_\mathcal{A})$, where

1 $Q_\mathcal{A}$ is a finite set of states,
2 $i_\mathcal{A} \in Q_\mathcal{A}$ is the initial state,
3 $\delta_\mathcal{A} : Q_\mathcal{A} \times \mathcal{F} \to Q_\mathcal{A}$ is a transition function,
4 and $Fin_\mathcal{A} \subseteq Q_\mathcal{A}$ are the final states.

With $\delta^*_\mathcal{A}$ we mean the unique extension of $\delta_\mathcal{A}$ to $\mathcal{F}^*$. The regular language that is accepted by an automaton $\mathcal{A}$ is defined as

$$L(\mathcal{A}) = \{w \mid \delta^*_\mathcal{A}(i_\mathcal{A}, w) \in Fin_\mathcal{A}\}.$$

Let $\text{reg}(\phi) = \{L_1, \ldots, L_n\} \subseteq P(\mathcal{F}^+)$ be the set of regular languages used in ϕ and let $\mathcal{A}_i = (Q_{\mathcal{A}_i}, i_{\mathcal{A}_i}, \delta_{\mathcal{A}_i}, Fin_{\mathcal{A}_i})$ be finite, deterministic automatons such that $\mathcal{A}_i$ accepts L_i. For each $\mathcal{A}_i$ we define $\text{dec}(\mathcal{A}_i)$ to be the set

$$\text{dec}(\mathcal{A}_i) = \{\overline{L^q_p} \mid p, q \in Q_{\mathcal{A}_i}\},$$

where $\overline{L^q_p} = \{w \in \mathcal{F}^+ \mid \delta^*_{\mathcal{A}_i}(p, w) = q\}$.

Of course, each $\text{dec}(\mathcal{A}_i)$ is finite and contains L_i. Furthermore, it is also closed under decomposition. The complete set of decompositions for a language $\overline{L^q_p} \in \text{dec}(\mathcal{A}_i)$ consists of the languages

$$P = \overline{L^s_p} \text{ and } S = \overline{L^q_s} \quad \text{for } s \in Q_{\mathcal{A}_i}.$$

We define Λ_0 to be $\bigcup_{i=1}^n \text{dec}(\mathcal{A}_i)$. Λ_0 contains each $L_i \in \text{reg}(\phi)$ and is closed under decomposition. Now let

$$\Lambda = \text{fi}\,(\Lambda_0)$$

be the least set that contains Λ_0 and is closed under intersection. Then Λ is finite and ϕ-closed, since it contains each $L_i \in \text{reg}(\phi)$.

We will prove that Λ is also closed under decomposition. Given some $L \in \Lambda$ and a path $w = w_1 w_2 \in L$, we have to find an appropriate decomposition P, S in Λ. Since each L in Λ can be written as a finite intersection

$$L = \bigcap_{k=1}^{m} L_k$$

with L_k in Λ_0, we know that $w = w_1 w_2$ is in L_k for $1..m$. As Λ_0 is closed under decomposition, there are languages P_k and S_k for $k = 1..m$ with $w_1 \in P_k$, $w_2 \in S_k$ and $P_k{\cdot}S_k \subseteq L_k$. Let $P = \bigcap_{k=1}^{m} P_k$ and $S = \bigcap_{k=1}^{m} S_k$. Clearly, $w_1 \in P$, $w_2 \in S$ and $P{\cdot}S \subseteq L$. Furthermore, $P, S \in \Lambda$ as Λ is closed under intersection. This implies that P, S is an appropriate decomposition for $w_1 w_2$. $\square$

Before proceeding to some properties of the rule system, we present some sample derivations. We will start with the prime clause

$$\phi = \left\{ x\alpha_1 y_1,\ x\alpha_2 y_2,\ x\alpha_3 y_3,\ \alpha_1 \mathrel{\dot{\in}} (f \cup g),\ \alpha_2 \mathrel{\dot{\in}} (f^* \cup h^*),\ \alpha_3 \mathrel{\dot{\in}} h(f \cup g) \right\}.$$

To find an appropriate set of rules $\mathcal{R}_\Lambda$, we must generate a ϕ-closed set of languages Λ, which is closed under decomposition. By the construction used in the proof of the last lemma we obtain the following set of languages:

$$\Lambda = \{(f \cup g),\ (f^+ \cup h^+),\ h(f \cup g),\ h\}.$$

For the sake of clarity, we will say that we apply (PathRel) *on* some variable α if we apply an instance of (PathRel) of the form

$$\frac{\{x\alpha y\} \cup \psi}{\{\ldots\}}.$$

Figure 2 is an example of an $\mathcal{R}_\Lambda$-derivation which transforms ϕ into a pre-solved clause. We use the frames to highlight the corresponding parts of a clause which have been modified by the last rule application. The empty frame $\boxed{}$ denotes the deletion of a constraint. Note that we have removed the constraint $\alpha_2 \mathrel{\dot{\prec}} \alpha_3$ in the fourth clause using the (Pre) rule, $\alpha_1 \mathrel{\dot{\parallel}} \alpha_2{\cdot}\alpha_3$ in the fifth clause using the (Div1) rule, and the constraint $\alpha_2 \mathrel{\dot{\in}} (f^+ \cup h^+)$ in the last clause.

Next we want to examine two clashing $\mathcal{R}_\Lambda$-derivations. The first one (see figure 3) shows that $\alpha_2 \mathrel{\dot{\prec}} \alpha_1$ cannot hold. The second clashing derivation (see figure 4) shows that α_3 cannot be the prefix of α_2.

Finally, here is the complete list of pre-solved clauses different from $\perp$ that are derivable from ϕ using $\mathcal{R}_\Lambda$:

$$\left\{ \begin{array}{c} \alpha_1 \mathrel{\dot{\parallel}} \alpha_2,\ \alpha_1 \mathrel{\dot{\parallel}} \alpha_3,\ \alpha_2 \mathrel{\dot{\parallel}} \alpha_3,\ x\alpha_1 y_1,\ x\alpha_2 y_2,\ x\alpha_3 y_3, \\ \alpha_1 \mathrel{\dot{\in}} (f \cup g),\ \alpha_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \alpha_3 \mathrel{\dot{\in}} h(f \cup g) \end{array} \right\}$$

$$\left\{ \alpha_1 \mathrel{\dot{\parallel}} \alpha_2,\ x\alpha_1 y_1,\ x\alpha_2 y_2,\ y_2\alpha_3 y_3,\ \alpha_1 \mathrel{\dot{\in}} (f \cup g),\ \alpha_2 \mathrel{\dot{\in}} h,\ \alpha_3 \mathrel{\dot{\in}} (f \cup g) \right\}$$

$$\left\{ \alpha_1 \mathrel{\dot{\parallel}} \alpha_3,\ x\alpha_1 y_1,\ y_1\alpha_2 y_2,\ x\alpha_3 y_3,\ \alpha_1 \mathrel{\dot{\in}} f,\ \alpha_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \alpha_3 \mathrel{\dot{\in}} (f \cup g) \right\}$$

$$\left\{ \alpha_1 \mathrel{\dot{\parallel}} \alpha_3,\ y_1 \doteq y_2,\ x\alpha_1 y_1,\ x\alpha_3 y_3,\ \alpha_1 \mathrel{\dot{\in}} f,\ \alpha_3 \mathrel{\dot{\in}} (f \cup g) \right\}$$

6.2. Some Properties of the Rule System

For the rest of the paper we will call clauses that are derivable from prime clauses **admissible**.

$$\left\{x\alpha_1 y_1,\ x\alpha_2 y_2,\ x\alpha_3 y_3,\ \alpha_1 \mathrel{\dot{\in}} (f \cup g),\ \alpha_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \alpha_3 \mathrel{\dot{\in}} h(f \cup g)\right\}$$

$$\downarrow \quad \text{(PathRel) on } \alpha_1$$

$$\left\{\begin{array}{c} \boxed{\alpha_1 \mathrel{\dot{\parallel}} \alpha_2,\ \alpha_1 \mathrel{\dot{\parallel}} \alpha_3},\ x\alpha_1 y_1,\ x\alpha_2 y_2,\ x\alpha_3 y_3, \\ \alpha_1 \mathrel{\dot{\in}} (f \cup g),\ \alpha_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \alpha_3 \mathrel{\dot{\in}} h(f \cup g) \end{array}\right\}$$

$$\downarrow \quad \text{(PathRel) on } \alpha_2$$

$$\left\{\begin{array}{c} \boxed{\alpha_2 \mathrel{\dot{\prec}} \alpha_3},\ \alpha_1 \mathrel{\dot{\parallel}} \alpha_2,\ \alpha_1 \mathrel{\dot{\parallel}} \alpha_3,\ x\alpha_1 y_1,\ x\alpha_2 y_2,\ x\alpha_3 y_3, \\ \alpha_1 \mathrel{\dot{\in}} (f \cup g),\ \alpha_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \alpha_3 \mathrel{\dot{\in}} h(f \cup g) \end{array}\right\}$$

$$\downarrow \quad \text{(Pre)}$$

$$\left\{\begin{array}{c} \boxed{},\ \alpha_1 \mathrel{\dot{\parallel}} \alpha_2,\ \boxed{\alpha_1 \mathrel{\dot{\parallel}} \alpha_2 \cdot \alpha_3},\ x\alpha_1 y_1,\ x\alpha_2 y_2,\ \boxed{y_2 \alpha_3 y_3}, \\ \alpha_1 \mathrel{\dot{\in}} (f \cup g),\ \alpha_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \boxed{\alpha_2 \cdot \alpha_3 \mathrel{\dot{\in}} h(f \cup g)} \end{array}\right\}$$

$$\downarrow \quad \text{(Div1)}$$

$$\left\{\begin{array}{c} \alpha_1 \mathrel{\dot{\parallel}} \alpha_2,\ \boxed{},\ x\alpha_1 y_1,\ x\alpha_2 y_2,\ y_2 \alpha_3 y_3, \\ \alpha_1 \mathrel{\dot{\in}} (f \cup g),\ \alpha_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \alpha_2 \cdot \alpha_3 \mathrel{\dot{\in}} h(f \cup g) \end{array}\right\}$$

$$\downarrow \quad (\text{LangDec}_\Lambda)$$

$$\left\{\begin{array}{c} \alpha_1 \mathrel{\dot{\parallel}} \alpha_2,\ x\alpha_1 y_1,\ x\alpha_2 y_2,\ y_2 \alpha_3 y_3, \\ \alpha_1 \mathrel{\dot{\in}} (f \cup g),\ \alpha_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \boxed{\alpha_2 \mathrel{\dot{\in}} h,\ \alpha_3 \mathrel{\dot{\in}} (f \cup g)} \end{array}\right\}$$

$$\downarrow \quad \text{(Join)}$$

$$\{\alpha_1 \mathrel{\dot{\parallel}} \alpha_2,\ x\alpha_1 y_1,\ x\alpha_2 y_2,\ y_2 \alpha_3 y_3,\ \alpha_1 \mathrel{\dot{\in}} (f \cup g),\ \boxed{},\ \alpha_2 \mathrel{\dot{\in}} h,\ \alpha_3 \mathrel{\dot{\in}} (f \cup g)\}$$

Figure 2. A successful $\mathcal{R}_\Lambda$-derivation

$$\left\{x\alpha_1 y_1,\ x\alpha_2 y_2,\ x\alpha_3 y_3,\ \alpha_1 \mathrel{\dot{\in}} (f \cup g),\ \alpha_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \alpha_3 \mathrel{\dot{\in}} h(f \cup g)\right\}$$

$$\downarrow \quad \text{(PathRel) on } \alpha_2$$

$$\left\{\begin{array}{c} \boxed{\alpha_2 \mathrel{\dot{\prec}} \alpha_1,\ \alpha_2 \mathrel{\dot{\parallel}} \alpha_3},\ x\alpha_1 y_1,\ x\alpha_2 y_2,\ x\alpha_3 y_3, \\ \alpha_1 \mathrel{\dot{\in}} (f \cup g),\ \alpha_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \alpha_3 \mathrel{\dot{\in}} h(f \cup g) \end{array}\right\}$$

$$\downarrow \quad \text{(Pre)}$$

$$\left\{\begin{array}{c} \boxed{},\ \alpha_2 \mathrel{\dot{\parallel}} \alpha_3,\ \boxed{y_2 \alpha_1 y_1},\ x\alpha_2 y_2,\ x\alpha_3 y_3, \\ \boxed{\alpha_2 \cdot \alpha_1 \mathrel{\dot{\in}} (f \cup g)},\ \alpha_2 \mathrel{\dot{\in}} (f^+ \cup h^+),\ \alpha_3 \mathrel{\dot{\in}} h(f \cup g) \end{array}\right\}$$

$$\downarrow \quad \text{(DecClash)}$$

$$\bot$$

Figure 3. A clashing $\mathcal{R}_\Lambda$-derivation

$$\left\{x\alpha_1 y_1,\ x\alpha_2 y_2,\ x\alpha_3 y_3,\ \alpha_1 \dot\in (f \cup g),\ \alpha_2 \dot\in (f^+ \cup h^+),\ \alpha_3 \dot\in h(f \cup g)\right\}$$

$$\downarrow \quad \text{(PathRel) on } \alpha_3$$

$$\left\{\begin{array}{c} \boxed{\alpha_3 \mathbin{\dot{\amalg}} \alpha_1,\ \alpha_3 \mathbin{\dot\prec} \alpha_2},\ x\alpha_1 y_1,\ x\alpha_2 y_2,\ x\alpha_3 y_3, \\ \alpha_1 \dot\in (f \cup g),\ \alpha_2 \dot\in (f^+ \cup h^+),\ \alpha_3 \dot\in h(f \cup g) \end{array}\right\}$$

$$\downarrow \quad \text{(Pre)}$$

$$\left\{\begin{array}{c} \alpha_3 \mathbin{\dot{\amalg}} \alpha_1,\ \boxed{},\ x\alpha_1 y_1,\ \boxed{y_3\alpha_2 y_2},\ x\alpha_3 y_3, \\ \alpha_1 \dot\in (f \cup g),\ \boxed{\alpha_3{\cdot}\alpha_2 \dot\in (f^+ \cup h^+)},\ \alpha_3 \dot\in h(f \cup g) \end{array}\right\}$$

$$\downarrow \quad (\text{LangDec}_\Lambda)$$

$$\left\{\begin{array}{c} \alpha_3 \mathbin{\dot{\amalg}} \alpha_1,\ x\alpha_1 y_1,\ y_3\alpha_2 y_2,\ x\alpha_3 y_3, \\ \alpha_1 \dot\in (f \cup g),\ \boxed{\alpha_2 \dot\in (f^+ \cup h^+),\ \alpha_3 \dot\in (f^+ \cup h^+)},\ \alpha_3 \dot\in h(f \cup g) \end{array}\right\}$$

$$\downarrow \quad \text{(Join)}$$

$$\left\{\begin{array}{c} \alpha_3 \mathbin{\dot{\amalg}} \alpha_1,\ x\alpha_1 y_1,\ y_3\alpha_2 y_2,\ x\alpha_3 y_3, \\ \alpha_1 \dot\in (f \cup g),\ \alpha_2 \dot\in (f^+ \cup h^+),\ \boxed{\alpha_3 \dot\in \emptyset} \end{array}\right\}$$

$$\downarrow \quad \text{(Empty)}$$

$$\perp$$

Figure 4. Another clashing $\mathcal{R}_\Lambda$-derivation

LEMMA 6.3.

1 *Every admissible clause is basic.*
2 *If $\alpha \mathbin{\dot\prec} \beta$, $\alpha \doteq \beta$ or $\alpha \mathbin{\dot{\amalg}} \beta$ is contained in some admissible clause ϕ, then there is a variable x such that $x\alpha y$ and $x\beta z$ is in ϕ.*

PROOF. The proof of the first claim is left to the reader. The second claim will be proved by induction over the length of derivations. For prime clauses the claim holds trivially. For the induction hypotheses assume that we have proven the claim for every admissible clause ϕ that is derivable from a prime clause in n steps and let $\phi \rightarrow_r \phi'$. If r is different from (Pre), (PathRel), (Eq) or (Div2), there is nothing to prove. Thus we have the following cases:

$r \in$ (PathRel): the claim holds by definition of (PathRel).
$r \in$ (Eq): the claim is invariant under substitution of one variable β by another variable α if both $x\alpha y$ and $x\beta z$ are contained in ϕ.
$r \in$ (Pre): then $\phi = \{\alpha \mathbin{\dot\prec} \beta,\ x\alpha y,\ x\beta z\} \cup \psi$ and $\phi' = \{x\alpha y,\ y\beta z\} \cup \psi[\beta \leftarrow \alpha{\cdot}\beta]$. The only subterm agreement constraint that is changed is $x\beta z$. But as β is substituted by $\alpha{\cdot}\beta$, ϕ' does not contain any path equality or prefix constraints involving β.
$r \in$ (Div2): then $\phi = \{\alpha{\cdot}\beta \mathbin{\dot{\amalg}} \alpha{\cdot}\beta'\} \cup \psi$ and $\phi' = \{\beta \mathbin{\dot{\amalg}} \beta'\} \cup \psi$. We will prove below that if $\alpha{\cdot}\beta$ is contained in some admissible clause γ, then there are variables x, y, z such that $x\alpha y$ and $y\beta z$ are contained in γ. This will complete the proof, since

then $\alpha{\cdot}\beta \mathrel{\dot{\parallel}} \alpha{\cdot}\beta'$ in ϕ implies that there are variables x, y, z and x', y', z' with $\{x\alpha y, y\beta z, x'\alpha y', y'\beta' z'\} \subseteq \phi$. But as ϕ is admissible, it is also basic by the first claim. Hence, x equals x' and y equals y'. Therefore, both $y\beta z$ and $y\beta' z'$ are in ϕ and in ϕ'.

Thus it remains to show that if $\alpha{\cdot}\beta$ is used in some admissible clause γ, then there are variables x, y, z such that $x\alpha y$ and $y\beta z$ are in γ. Let ϕ be an admissible clause for which this holds, and let $\gamma \rightarrow_r \gamma'$. The only rules we have to consider are (Eq) and (Pre). For (Eq) note that the claim is invariant under consistent variable renaming. If $r \in$ (Pre), then we have to check the path term $\alpha{\cdot}\beta$ that is introduced by r. But by definition of (Pre) the clause γ' must contain both $x\alpha y$ and $y\beta z$. $\square$

This lemma implies that (Eq) can always be applied if a constraint $\alpha \doteq \beta$ is contained in some admissible clause. The next lemma will show that different applications of (Pre) or (Eq) will not interact. This means the application of one of these rules to some prefix or path equality constraint will not change any other prefix or path equality constraint contained in the same clause.

Lemma 6.4. *Given some prime clause ϕ and a derivation*

$$\phi = \phi_0 \rightarrow_{r_0} \phi_1 \cdots \phi_{n-1} \rightarrow_{r_{n-1}} \phi_n = \gamma$$

that contains an application of (PathRel). *Then $\alpha \doteq \beta \in \gamma$ (resp. $\alpha \mathrel{\dot{\prec}} \beta \in \gamma$) implies $\alpha \doteq \beta \in \phi_i$ (resp. $\alpha \mathrel{\dot{\prec}} \beta \in \phi_i$) for $i > k$, where k is the number of the last application of* (PathRel). *Furthermore, if $\alpha{\cdot}\beta$ is contained in γ, then either $\alpha{\cdot}\beta$ or $\alpha \mathrel{\dot{\prec}} \beta$ is contained in ϕ_i for $i > k$.*

Proof. We will use induction over length of derivations. Assume that we have proven the lemma for admissible clauses γ that are derivable in n steps and let $\gamma \rightarrow_r \gamma'$ with $r \notin$ (PathRel). If r is different from (Eq) or (Pre), then there is nothing to prove. If $r \in$ (Eq), then a constraint $\alpha \mathrel{\dot{\prec}} \beta$ or $\alpha \doteq \beta$ in γ' can be missing in γ if and only if γ contains a constraint $\alpha \doteq \beta'$ or $\alpha \mathrel{\dot{\prec}} \beta'$ (resp. $\beta' \doteq \beta$ or $\beta' \mathrel{\dot{\prec}} \beta$) and r is of the form

$$\frac{\{\beta \doteq \beta', \ldots\} \cup \psi}{\ldots} \text{ with } \beta' \neq \beta \qquad \left(\text{resp. } \frac{\{\alpha \doteq \beta', \ldots\} \cup \psi}{\ldots} \text{ with } \beta' \neq \alpha\right).$$

Hence, γ must contain at least two prefix or path equality constraints, the left sides of which are different. By induction hypotheses these path equality or prefix constraints must have been introduced by the last application of (PathRel). But this contradicts to the definition of (PathRel). A similar argument can be given for the part of the lemma concerning path terms of form $\alpha{\cdot}\beta$.

If r is in (Pre), then we have to check only the second claim of the lemma, namely that $\alpha{\cdot}\beta$ contained in γ' implies that either $\alpha \mathrel{\dot{\prec}} \beta$ is in γ or $\alpha{\cdot}\beta$ is used in γ. For the all path terms in γ' that are not introduced by this application of (Pre) this holds trivially. For the path term $\alpha{\cdot}\beta$ that is introduced, this is guaranteed by the application condition of (Pre), namely that γ must contain $\alpha \mathrel{\dot{\prec}} \beta$. $\square$

We can derive from this lemma certain syntactic properties of admissible clauses which are needed for proving completeness and quasi-termination.

COROLLARY 6.5. *If $\alpha \mathrel{\dot{\prec}} \beta$ is contained in an admissible clause ϕ, then α is different from β. Furthermore, there is no other prefix or equality constraint in ϕ involving β and neither $\beta{\cdot}\beta'$ nor $\beta'{\cdot}\beta$ is in ϕ.*

Note that by lemma 6.3 together with this corollary, the rule (Pre) is always applicable if a constraint $\alpha \mathrel{\dot{\prec}} \beta$ is contained in an admissible clause. Furthermore, an application of (Pre) causes no violation of the restrictions that we have imposed on the syntax. This means that concatenation does not occur in prefix or path equality constraints; and concatenation of path variables is restricted to binary concatenation.

LEMMA 6.6. *If $\alpha{\cdot}\beta \mathrel{\dot{\|}} \beta'$ is contained in an admissible clause ϕ with α different from β', then ϕ contains a constraint of form $\alpha \mathrel{\dot{\|}} \beta'$, $\alpha \doteq \beta'$ or $\alpha \mathrel{\dot{\prec}} \beta'$.*

PROOF. We will prove a stronger result, namely that if $\{\alpha \mathrel{\dot{\prec}} \beta, \beta \mathrel{\dot{\|}} \beta'\} \subseteq \phi$ or $\{\alpha{\cdot}\beta \mathrel{\dot{\|}} \beta'\} \subseteq \phi$, then ϕ contains a constraint of form $\alpha \mathrel{\dot{\|}} \beta'$, $\alpha \doteq \beta'$ or $\alpha \mathrel{\dot{\prec}} \beta'$. We will prove this by induction over length of derivations. Assume that we have proven the claim for every admissible clause ϕ that is derivable in n steps from a prime clause and let $\phi \rightarrow_r \phi'$. Again we have to check only the rules (Pre), (PathRel), (Eq) or (Div2):

$r \in$ (PathRel): we have to check only constraints $\beta \mathrel{\dot{\|}} \beta'$ that are already in ϕ. By lemma 6.3 we know that if $\beta \mathrel{\dot{\|}} \beta'$ is in ϕ, then there is a variable x with both $x\beta y$ and $x\beta' z$ in ϕ. Hence, if (PathRel) adds the constraint $\alpha \mathrel{\dot{\prec}} \beta$, it must by definition also add a constraint $\alpha \mathrel{\dot{\|}} \beta'$, $\alpha \doteq \beta'$ or $\alpha \mathrel{\dot{\prec}} \beta'$.

$r \in$ (Eq): the claim is invariant under consistent variable renaming.

$r \in$ (Pre): then $\phi = \{\alpha \mathrel{\dot{\prec}} \beta,\ x\alpha y,\ x\beta z\} \cup \psi$ and $\phi' = \{x\alpha y,\ y\beta z\} \cup \psi[\beta \leftarrow \alpha{\cdot}\beta]$. The only case that we have to check is that ϕ contains a constraint $\beta \mathrel{\dot{\|}} \beta'$. Then ϕ' contains $\alpha{\cdot}\beta \mathrel{\dot{\|}} \beta'$. By induction hypotheses ϕ must contain a constraint c of form $\alpha \mathrel{\dot{\|}} \beta'$, $\alpha \doteq \beta'$ or $\alpha \mathrel{\dot{\prec}} \beta'$. Since (Pre) does not change c, this must holds also for ϕ'.

$r \in$ (Div2): then $\phi = \{\alpha{\cdot}\beta \mathrel{\dot{\|}} \alpha{\cdot}\beta'\} \cup \psi$ and $\phi' = \{\beta \mathrel{\dot{\|}} \beta'\} \cup \psi$. The only new divergence constraint that comes in is $\beta \mathrel{\dot{\|}} \beta'$. But as ϕ contains both $\alpha{\cdot}\beta$ and $\alpha{\cdot}\beta'$, it may not contain $\alpha \mathrel{\dot{\prec}} \beta$ or $\alpha \mathrel{\dot{\prec}} \beta'$ by corollary 6.5. Hence, ϕ' does not contain such a constraint.

□

This lemma ensures that a constraint $\alpha{\cdot}\beta \mathrel{\dot{\|}} \beta'$ is always reducible. If β' equals α, then we could apply (DClash1). If $\alpha \mathrel{\dot{\|}} \beta'$ is in ϕ, we can apply (Div1). If $\alpha \doteq \beta'$ is in ϕ we can apply (Eq) followed by (DClash1). If $\phi = \{\alpha \dot{\prec} \beta', \alpha{\cdot}\beta \dot{\amalg} \beta'\} \cup \psi$, then we can apply (Pre) yielding $\{\alpha{\cdot}\beta \dot{\amalg} \alpha{\cdot}\beta'\} \cup \psi'$, where we can apply (Div2).

6.3. Soundness and Completeness

Proposition 6.7. *The rules* (Eq), (Div1,2), (SClash), (Join), (Empty) and (DClash1,2) *are $\mathcal{X} \cup \mathcal{P}$-sound and $\mathcal{X} \cup \mathcal{P}$-preserving.*

Proposition 6.8. *The rule* (Pre) *is $\mathcal{X}$-sound and $\mathcal{X}$-preserving.*

For (Pre) we can even characterize pairs of path valuations which preserve the $\mathcal{X}$-solutions.

Proposition 6.9. *Let $\phi = \{\alpha \dot{\prec} \beta, x\alpha y, x\beta z\} \cup \psi$ and γ be the result of applying (Pre) to ϕ. Given a pair of path valuations $V_\mathcal{P}, V'_\mathcal{P}$ with*

$$V_\mathcal{P} =_{\mathcal{P}-\{\beta\}} V'_\mathcal{P} \quad \textit{and} \quad V_\mathcal{P}(\beta) = V_\mathcal{P}(\alpha)V'_\mathcal{P}(\beta) = V'_\mathcal{P}(\alpha)V'_\mathcal{P}(\beta),$$

then for each interpretation $\mathcal{I}$ and for each first order valuation $V_\mathcal{X}$

$$(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} \phi \iff (V_\mathcal{X}, V'_\mathcal{P}) \models_\mathcal{I} \gamma.$$

Proposition 6.10. *If Λ is closed under decomposition, then* (LangDec$_\Lambda$) *is $\mathcal{X}\cup\mathcal{P}$-sound and globally $\mathcal{X}\cup\mathcal{P}$-preserving. Furthermore,* (PathRel) *is $\mathcal{X}\cup\mathcal{P}$-sound and globally $\mathcal{X}\cup\mathcal{P}$-preserving.*

Finally, we have to prove that the rules are complete. This means that given an input clause ϕ, for every solution $V_\mathcal{X}$ of ϕ in some interpretation $\mathcal{I}$ there is a pre-solved clause γ derivable from ϕ such that $V_\mathcal{X}$ is a solution of γ. If the rule system is terminating, then for completeness one has to prove that the pre-solved clauses are just the irreducible clauses.

In our case this is not enough since the rule system can loop. Therefore, we have to prove explicitly that each solution of a given prime clause ϕ can be found in some pre-solved ϕ-derivative. We define Irred$(\phi, \mathcal{R}_\Lambda)$ to be the set all $\mathcal{R}_\Lambda$-derivatives of ϕ which are $\mathcal{R}_\Lambda$-irreducible, and Pre-Solved$(\phi, \mathcal{R}_\Lambda)$ to be the set of all pre-solved clauses which are derivable from ϕ. A set of rules $\mathcal{R}_\Lambda$ is said to be **ϕ-complete** w.r.t. to a set of variables ϑ if

1 Irred(ϕ, R_Λ) = Pre-Solved(ϕ, R_Λ),
2 for every interpretation $\mathcal{I}$

$$[\![\phi]\!]^\mathcal{I}_\vartheta \subseteq \bigcup_{\gamma \in \text{Pre-Solved}(\phi, R_\Lambda)} [\![\gamma]\!]^\mathcal{I}_\vartheta.$$

We will show that for every prime clause ϕ there is a set of regular languages Λ such that $\mathcal{R}_\Lambda$ is ϕ-complete w.r.t the first order variables $\mathcal{X}$.

THEOREM 6.11. (COMPLETENESS I) *Given a prime clause ϕ. If Λ is a set of regular languages that is ϕ-closed, closed under intersection and closed under decomposition, then every $\mathcal{R}_\Lambda$-derivative γ of ϕ that is not pre-solved is $\mathcal{R}_\Lambda$-reducible.*

PROOF. Let γ be a $\mathcal{R}_\Lambda$-derivative of ϕ that is not pre-solved. We will check all conditions that are stated in the definition on page 430.

If one of the conditions 1–3 is not satisfied by γ, then one of the rules (SClash), (Join) or (Empty) will apply.

Now let's check the conditions 4 and 5:

γ **contains a constraint** $\alpha{\cdot}\beta \,\dot{\in}\, L$**.** As Λ is ϕ-closed, we know that Λ is also γ-closed by lemma 6.1. Therefore we can apply (LangDec$_\Lambda$) or (DecClash).

γ **contains a constraint** $\alpha{\cdot}\beta \,\dot{\parallel}\, \alpha'{\cdot}\beta'$**.** By lemma 6.4 we know that in every $\mathcal{R}_\Lambda$-derivation for γ the last application of (PathRel) must have introduced the constraints $\alpha \dot{\prec} \beta$ and $\alpha' \dot{\prec} \beta'$. By the definition of (PathRel) this implies that α equals α'. Hence, we can apply (Div2).

γ **contains a constraint** $\alpha{\cdot}\beta \,\dot{\parallel}\, \beta'$**.** If β' equals α, then we can directly apply (DClash1). Otherwise, there is by lemma 6.6 a constraint $\alpha \doteq \beta'$, $\alpha \dot{\prec} \beta'$ or $\alpha \,\dot{\parallel}\, \beta'$ in γ. If $\alpha \doteq \beta'$ is in γ, we can apply (Eq) by lemma 6.3. This will result in the substitution of β' by α. The remaining constraint $\alpha{\cdot}\beta \,\dot{\parallel}\, \alpha$ can be reduced using (DClash1). If $\alpha \dot{\prec} \beta'$ is in γ, then we can apply (Pre) by lemma 6.3 and corollary 6.5. We will obtain the constraint $\alpha{\cdot}\beta \,\dot{\parallel}\, \alpha{\cdot}\beta'$, which can be reduced using (Div2). The last case is that $\alpha \,\dot{\parallel}\, \beta'$ is in γ, where we can apply (Div1).

γ **contains a constraint** $\alpha \doteq \beta$**.** Then (Eq) is applicable by lemma 6.3.

γ **contains a constraint** $\alpha \dot{\prec} \beta$**.** Then (Pre) is applicable by lemma 6.3 and corollary 6.5.

The remaining case is that γ does not satisfy the last condition of a pre-solved clause, namely that $\alpha \,\dot{\parallel}\, \beta$ with $\alpha \neq \beta$ in γ if and only if $x\alpha y$ and $x\beta y$ in γ. Given the above, we can now assume that γ does not contain a path concatenation or a prefix or path equality constraint.

There are three possibilities for γ to violate the last condition. The first is that γ contains a constraint of the form $\alpha \,\dot{\parallel}\, \alpha$. Then (DClash2) is applicable. The second is that there is a constraint $\alpha \,\dot{\parallel}\, \beta$ with $x\alpha y \in \gamma$ and $x'\beta y' \in \gamma$ such that x is different from x'. But this is excluded by lemma 6.3.

The last case is that there are different path variables α and β such that $x\alpha y$ and $x\beta z$ are in γ but $\alpha \,\dot{\parallel}\, \beta$ is not. As γ contains no concatenation and no path equality or prefix constraints, the rule (PathRel) is applicable. □

Next we have to establish the second condition for ϕ-completeness, namely that for every interpretation $\mathcal{I}$ and for every solution $V_{\mathcal{X}}$ of ϕ there is a pre-solved ϕ-derivative γ with $V_{\mathcal{X}} \in [\![\gamma]\!]^{\mathcal{I}}$. This property is needed since our rule system can loop. Let us recall an example of a looping derivation in order to explain the main idea involved in the second part of the completeness proof. In contrast with our first example of a looping derivation (see page 425), we will omit the path restrictions, since they are not needed for what we want to demonstrate. Let ϕ be the clause

$$\phi = \{x\alpha x, x\beta y\}.$$

A looping derivation can consist of an application of (PathRel) yielding the clause $\phi_1 = \{\alpha \mathrel{\dot{\prec}} \beta, x\alpha x, x\beta y\}$, followed by an application of (Pre) on γ yielding $\phi_2 = \phi$.[†] Clearly, the cause of the looping derivation is the rule (Pre). We will later prove that, indeed, every infinite derivation must use the (Pre) rule infinitely often.

To prove the second completeness condition we restrict the set of allowed derivations of a prime clause ϕ to those depending on some arbitrary but fixed valuation $(V_{\mathcal{X}}, V_{\mathcal{P}})$ with $(V_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{I}} \phi$. This control will guarantee that

1 $V_{\mathcal{X}}$ is a solution of every clause in the derivation,
2 under this control, all derivations are finite.

We will additionally show that even under this control the irreducible clauses are just the pre-solved clauses. Hence this control will give us, for every clause ϕ and every initial solution $V_{\mathcal{X}}$, a pre-solved ϕ-derivative that has $V_{\mathcal{X}}$ as an solution.

We will add this further control only on the non-deterministic rules (PathRel) and (LangDec$_{\Lambda}$), thus restricting the set of instances of these rules that may be applied. We allow only those instances which preserve the valuation $(V_{\mathcal{X}}, V_{\mathcal{P}})$. Using our above example, if $V_{\mathcal{P}}$ satisfies

$$V_{\mathcal{P}}(\alpha) = f \quad \text{and} \quad V_{\mathcal{P}}(\beta) = g$$

we may apply only that instance of (PathRel) which transforms ϕ into $\phi_1 = \{\alpha \mathrel{\dot{\parallel}} \beta, x\alpha x, x\beta y\}$. Since the choice of the instances depends only on the path valuation, we will call such restricted derivations $V_{\mathcal{P}}$-strict.

It is easy to see that the above restriction will always enforce finiteness of derivations if the initial path valuation $V_{\mathcal{P}}$ satisfies

$$V_{\mathcal{P}}(\alpha) \not\prec V_{\mathcal{P}}(\beta) \quad \text{where} \quad \alpha \neq \beta \wedge x\alpha y \in \phi \wedge x\beta z \in \phi.$$

One might say that in this case $V_{\mathcal{P}}$ is prefix-free with respect to ϕ.

For initial path valuations which are not prefix-free we must have a closer look at the (Pre) rule, since this rule is the cause of looping derivations. Since the (Pre) rule is not $\mathcal{P}$-preserving, it may happen that the clause γ resulting of an application of (Pre) is not

[†] The first example of a looping derivation on page 425 shows that the situation is no different if we add path restrictions.

valid under the initial valuation $(V_{\mathcal{X}}, V_{\mathcal{P}})$. But as (Pre) is $\mathcal{X}$-preserving, we know that there is a $V'_{\mathcal{P}}$ such that $(V_{\mathcal{X}}, V'_{\mathcal{P}}) \models_{\mathcal{I}} \gamma$.

Hence, in a $V_{\mathcal{P}}$-strict derivation we can keep the initial valuation of the first order variables $V_{\mathcal{X}}$, but we must change the path valuation every time the (Pre) rule is applied. Since application of (Pre) is the cause of looping derivations, this implies that we can obtain finiteness of $V_{\mathcal{P}}$-strict derivations if we guarantee that after a finite number of (Pre) applications the initial path valuation has been transformed into a prefix-free path valuation.

We will again turn to our example to clarify this. If the initial path valuation $V_{\mathcal{P}}$ for ϕ is of the form

$$V_{\mathcal{P}}(\alpha) = f \quad \text{and} \quad V_{\mathcal{P}}(\beta) = fffg,$$

the first rule in a $V_{\mathcal{P}}$-strict ϕ-derivation could be an application of (PathRel) transforming $\phi = \phi_0$ into $\phi_1 = \{\alpha \dot{\prec} \beta, x\alpha x, x\beta y\}$. Now we are able to apply (Pre), which implies that we have to change $V_{\mathcal{P}}$. Using proposition 6.9 we can use the following $V'_{\mathcal{P}}$:

$$V'_{\mathcal{P}}(\alpha) = f \quad \text{and} \quad V'_{\mathcal{P}}(\beta) = ffg.$$

Proposition 6.9 guarantees that this can be done without loosing $\mathcal{X}$-preservingness. Note that we have shortened $V_{\mathcal{P}}(\beta)$ by f. Now we could iterate this twice more before ending up with a prefix-free path valuation.

After these remarks we can turn to the technical part.

THEOREM 6.12. (COMPLETENESS-II) *Let ϕ be a prime clause, let Λ be a set of regular languages which is ϕ-closed, closed under intersection and decomposition. Then $\mathcal{R}_\Lambda$ is ϕ-complete w.r.t. the first order variables $\mathcal{X}$.*

First we need an additional lemma.

LEMMA 6.13. *There are no infinite derivations using only finitely many instances of (Pre).*

PROOF. Assume there is such a derivation. Then there exists an infinite sub-derivation not using any instance of (Pre). Let ϕ be the starting point of such a derivation. Let γ be some clause. Then we define the following functions on γ:

$$\begin{array}{rcl} \Theta_1(\gamma) & = & \text{number of concatenations in } \gamma \\ \Theta_2(\gamma) & = & \text{number of different path variables in } \gamma \\ \Delta^\phi(\gamma) & = & \text{number of constraints } \alpha \mathbin{\dot\diamond} \beta \text{ with } \dot\diamond \in \{\doteq, \dot\prec, \dot\Pi\}, \\ & & \alpha, \beta \in \mathcal{V}ars_{\mathcal{P}}(\phi) \text{ and } \alpha \mathbin{\dot\diamond} \beta \ \underline{\text{not in}}\ \gamma \\ \Pi(\gamma) & = & \text{total number of constraints in } \gamma \end{array}$$

We define $\Theta(\gamma)$ to be the tuple $\langle \Theta_1(\gamma), \Theta_2(\gamma) \rangle$. Using the functions Θ, Δ^ϕ and Π we can construct a partial order on clauses by defining $\gamma <_\phi \gamma'$ iff

$$(\Theta(\gamma) < \Theta(\gamma'))$$

	Θ_1	Θ_2	Δ^ϕ	Π
(PathRel)	$=$	$=$	$<$	
(Eq)	$=$	$<$		
(LangDec$_\Lambda$)	$<$	$=$		
(Join)	$=$	$=$	$=$	$<$
(Div1)	$<$	$=$		
(Div2)	$<$	$=$		

Table 1. Monotonicity of the rules w.r.t the measure functions.

or

$$(\Theta(\gamma) = \Theta(\gamma')) \quad \wedge \quad (\Delta^\phi(\gamma) < \Delta^\phi(\gamma'))$$

or

$$(\Theta(\gamma) = \Theta(\gamma')) \quad \wedge \quad (\Delta^\phi(\gamma) = \Delta^\phi(\gamma')) \quad \wedge \quad (\Pi(\gamma) = \Pi(\gamma')).$$

Here $<$ is the lexicographic ordering on tuples for $\Theta(\gamma)$ and elsewhere the usual numeric comparison. It is easy to check, that $<_\phi$ defines a well-founded, partial ordering on clauses.

Let γ be some derivation of ϕ. Now $\mathcal{V}ars_{\mathcal{P}}(\gamma) \subseteq \mathcal{V}ars_{\mathcal{P}}(\phi)$ holds, which is important for the value of Δ^ϕ. In table 1 we have summarized for every non-clash rule other than (Pre) the variation of $\Theta(\gamma)$, $\Delta^\phi(\gamma)$ and $\Pi(\gamma)^\dagger$. The clash rules are not considered because they automaticly terminate every derivation. The table shows that for every rule r $\gamma \rightarrow_r \gamma'$ implies $\gamma' <_\phi \gamma$. Because $<_\phi$ is a well-founded ordering and therefore cannot have infinite descending chains, this contradicts our assumption that there is a infinite derivation not using (Pre). □

Corollary 6.14. *There are no infinite derivations using only finitely many instances of* (PathRel).

Proof. By the above lemma we know that there are no infinite derivations without infinite use of (Pre). But (Pre) removes the constraints $\alpha \dot{\prec} \beta$, the existence of which is an application condition for (Pre). But additional constraints of form $\alpha \dot{\prec} \beta$ are only introduced by (PathRel). □

Proof of theorem 6.12 (Completeness II). The first condition for ϕ-completeness was proved in theorem 6.11 (Completeness I). For the second, let $\mathcal{I}$ be some interpretation and $(V_{\mathcal{X}}, V_{\mathcal{P}})$ be a valuation with $(V_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{I}} \phi$. We have to show that there is a $\mathcal{R}_\Lambda$-derivative γ of ϕ which is pre-solved and satisfies $\exists V'_{\mathcal{P}} : (V_{\mathcal{X}}, V'_{\mathcal{P}}) \models_{\mathcal{I}} \gamma$. This will be done by defining $V_{\mathcal{P}}$-strict derivations, which will always end up in a pre-solved clause. As we

† If a rule decreases the Θ-value, the clause resulting from applying this rule is smaller than the input clause w.r.t $<_\phi$ independently of the effects of the rule on the Δ^ϕ-part. Therefore, we omit the corresponding Δ^ϕ-entries in this case; and similarly for the Π-part.

have mentioned, we have to redefine the path valuation every time (Pre) is applied. This leads to the following definition: a derivation

$$\phi = \phi_0 \rightarrow_{r_0} \phi_1 \cdots \phi_n \rightarrow_{r_n} \phi_{n+1} \cdots$$

is called $V_{\mathcal{P}}$-strict if there is a family of path valuations $(V_{\mathcal{P}}^i)$ such that

1 $V_{\mathcal{P}}^0 = V_{\mathcal{P}}$;
2 for each i the proposition $(V_{\mathcal{X}}, V_{\mathcal{P}}^i) \models_{\mathcal{I}} \phi_i$ holds; and
3 for each i

$r_i \notin$ (Pre) implies $V_{\mathcal{P}}^i = V_{\mathcal{P}}^{i+1}$ and
$r_i = \frac{\{\alpha \dot{\prec} \beta,\ \cdots\ \} \cup \psi}{\cdots} \in$ (Pre) implies

$$V_{\mathcal{P}}^i =_{\mathcal{P}-\{\beta\}} V_{\mathcal{P}}^{i+1} \qquad \text{and} \qquad V_{\mathcal{P}}^i(\beta) = V_{\mathcal{P}}^{i+1}(\alpha)V_{\mathcal{P}}^{i+1}(\beta).$$

Now for every $V_{\mathcal{P}}$-strict $(\phi, \mathcal{R}_\Lambda)$-derivation

$$\phi = \phi_0 \rightarrow_{r_0} \phi_1 \cdots \phi_{n-1} \rightarrow_{r_{n-1}} \phi_n$$

where ϕ_n is not pre-solved, there is a $V_{\mathcal{P}}$-strict continuation, as the following argumentation shows. If ϕ_n is not pre-solved, then there is (by theorem 6.11) a rule which is applicable. We have to show that there is an applicable rule instance such that a corresponding $V_{\mathcal{P}}^{n+1}$ can be found.

If the applicable rule is different from (Pre), then we know that there is an appropriate path valuation $V_{\mathcal{P}}^{n+1}$, as all rules different from (Pre) are either $\mathcal{X} \cup \mathcal{P}$-preserving or globally $\mathcal{X} \cup \mathcal{P}$-preserving. If (Pre) is applicable, then proposition 6.9 shows that we can find an appropriate $V_{\mathcal{P}}^{n+1}$.

Next we must show that there is no infinite $V_{\mathcal{P}}$-strict $(\phi, \mathcal{R}_\Lambda)$-derivation, which finally proves the lemma. This is done by introducing a norm on path valuations. For a path valuation $V_{\mathcal{P}}$ we define $|V_{\mathcal{P}}|_\phi$ to be:

$$|V_{\mathcal{P}}|_\phi = \sum_{\alpha \in \mathcal{V}ars_{\mathcal{P}}(\phi)} |V_{\mathcal{P}}(\alpha)|.$$

Now let

$$\phi_i \rightarrow_{r_i} \phi_{i+1}$$

be a step in some $V_{\mathcal{P}}$-strict $(\phi, \mathcal{R}_\Lambda)$-derivation and let $V_{\mathcal{P}}^i, V_{\mathcal{P}}^{i+1}$ be the corresponding path valuations. If $r_i \notin$ (Pre) we know that $V_{\mathcal{P}}^i = V_{\mathcal{P}}^{i+1}$ and hence $|V_{\mathcal{P}}^i|_\phi = |V_{\mathcal{P}}^{i+1}|_\phi$. If $r_i \in$ (Pre) we know by the third condition of $V_{\mathcal{P}}$-strictness that there are α and β such that

$$V_{\mathcal{P}}^i =_{\mathcal{P}-\{\beta\}} V_{\mathcal{P}}^{i+1} \qquad \text{and} \qquad V_{\mathcal{P}}^i(\beta) = V_{\mathcal{P}}^{i+1}(\alpha)V_{\mathcal{P}}^{i+1}(\beta).$$

As $\mathcal{V}ars_{\mathcal{P}}(\phi_{i+1}) \subseteq \mathcal{V}ars_{\mathcal{P}}(\phi_i) \subseteq \mathcal{V}ars_{\mathcal{P}}(\phi)$ this implies $|V_{\mathcal{P}}^{i+1}|_\phi < |V_{\mathcal{P}}^i|_\phi$.

As there are no infinite derivations without infinite use of (Pre) this proves that there are no infinite $V_{\mathcal{P}}$-strict derivations. □

6.4. Quasi-Termination

LEMMA 6.15. *Let ϕ be a prime clause and Λ be a finite ϕ-closed set of regular languages. Then the set of all $\mathcal{R}_\Lambda$-derivatives of ϕ is finite.*

PROOF. We will first consider the sets $\mathcal{C}$ which contains every atomic constraint that occur in at least one $\mathcal{R}_\Lambda$-derivative of ϕ. $\mathcal{C}$ could be seen as the union of all $\mathcal{R}_\Lambda$-derivatives of ϕ. We will show that $\mathcal{C}$ is finite. As every $\mathcal{R}_\Lambda$-derivative of ϕ is a subset of $\mathcal{C}$ this will prove the lemma.

First we know that no rule adds new variables. This implies that there are at most $n_1 = |\mathcal{V}ars_{\mathcal{P}}(\phi)| + |\mathcal{V}ars_{\mathcal{P}}(\phi)|^2$ many different path terms. By lemma 6.1 we know that Λ is γ-closed for every $\mathcal{R}_\Lambda$-derivative γ of ϕ, which implies that at most $|\Lambda|$ different regular languages are used in the $\mathcal{R}_\Lambda$-derivatives of ϕ.

Therefore $\mathcal{C}$ contains at most $|\mathcal{V}ars_{\mathcal{X}}(\phi)|^2$ node agreements, $|\mathcal{V}ars_{\mathcal{X}}(\phi)| * |\mathcal{V}ars_{\mathcal{P}}(\phi)| * |\mathcal{V}ars_{\mathcal{X}}(\phi)|$ subterm agreements, n_1^2 path divergence constraints, $|\mathcal{V}ars_{\mathcal{P}}(\phi)|^2$ prefix and equality constraints and $n_1 * |\Lambda|$ path restriction constraints. Since no rule adds new sort symbols we know that $\mathcal{C}$ contains at most $n_2 * |\mathcal{V}ars_{\mathcal{X}}(\phi)|$ different node restrictions, where n_2 is the number of sort symbols in ϕ. $\square$

THEOREM 6.16. *For every prime clause ϕ there exists a set of regular languages Λ such that $\mathcal{R}_\Lambda$ is ϕ-complete w.r.t. $\mathcal{X}$ and the set* Pre-Solved$(\phi, \mathcal{R}_\Lambda)$ *is finite and computable.*

PROOF. Let reg(ϕ) be the set of regular languages used in ϕ. By lemma 6.2 there must be a finite Λ such that Λ is ϕ-closed, closed under intersection and decomposition. Note that the construction of the set Λ given in lemma 6.2 is effective. Then $\mathcal{R}_\Lambda$ is ϕ-complete w.r.t. $\mathcal{X}$ by theorem 6.12. By lemma 6.15 we know that Pre-Solved$(\phi, \mathcal{R}_\Lambda)$ must be finite. Hence, it suffices to prove that the set Pre-Solved$(\phi, \mathcal{R}_\Lambda)$ is computable.

To do this we will consider loop-free derivations. A derivation is called loop-free if it is not of the form

$$\phi_0 \rightarrow_{r_1} \ldots \rightarrow_{r_i} \phi_i \ldots \rightarrow_{r_k} \phi_k \ldots,$$

where $\phi_i = \phi_k$. In order to generate the set of derivatives (or a subset of them) it is enough to consider loop-free derivations. This is because for every pair γ, γ' every γ-derivation which yields γ' and is not loop-free can be replaced with a shorter derivation by removing some loop. Iterating this step finally yields a loop-free γ-derivation for γ'.

Furthermore, the set of all loop-free $(\phi, \mathcal{R}_\Lambda)$-derivations must be finite since $\mathcal{R}_\Lambda$ can only generate finitely many $\mathcal{R}_\Lambda$-derivatives of ϕ by lemma 6.15, and there are only finitely many rules of $\mathcal{R}_\Lambda$ applicable on every $\mathcal{R}_\Lambda$-derivative of ϕ. But as we have mentioned we need to consider only the loop-free derivations, which shows that Pre-Solved$(\phi, \mathcal{R}_\Lambda)$ is computable. $\square$

COROLLARY 6.17. *For every prime clause ϕ there exists a finite and computable set of pre-solved clauses Γ such that Γ is equivalent to ϕ.*

PROOF. Follows from the last theorem and the fact, that every rule is at least $V_{\mathcal{X}}$-sound. $\square$

7. The Second Phase: Satisfiability of Pre-Solved Clauses

In this section we present a rule system that transforms each pre-solved clause into an equivalent set of solved clauses (interpreted as a disjunction), each of which is different from $\perp$ and is thus satisfiable by lemma 5.3. A pre-solved clause is satisfiable if and only if the corresponding set is non-empty.

We will first make a minor redefinition of divergence. We say that two paths u, v are **directly diverging** (written $u \parallel_0 v$) if there are features $f \neq g$ such that $u \in f\mathcal{F}^*$ and $v \in g\mathcal{F}^*$. Then $u \parallel v$ holds if there are a possible empty prefix w and paths u', v' such that $u = wu'$ and $v = wv'$ and $u' \parallel_0 v'$. Using this definition of divergence and the additional atomic constraint

$$\alpha \, \dot{\parallel}_0 \, \beta \qquad \textbf{direct divergence},$$

we can (non-deterministically) transform a clause $\phi = \{\alpha_1 \, \dot{\parallel} \, \alpha_2\} \cup \psi$ into either $\{\alpha_1 \, \dot{\parallel}_0 \, \alpha_2\} \cup \psi$ or $\{\alpha_1 \doteq \beta{\cdot}\alpha_1', \ \alpha_2 \doteq \beta{\cdot}\alpha_2', \ \alpha_1' \, \dot{\parallel}_0 \, \alpha_2'\} \cup \psi$.† By the definition of $\parallel_0$ we can reduce (non-deterministically) the constraints of form $\alpha_1 \, \dot{\parallel}_0 \, \alpha_2$ into $\{\alpha_1 \, \dot{\in} \, f\mathcal{F}^*, \alpha_2 \, \dot{\in} \, g\mathcal{F}^*\}$ with $f \neq g$. The aim is to process all divergence constraints this way in order to achieve a solved clause.

But we have to reformulate the reduction of divergence constraints. The reason is that we have to evaluate constraints of the form $\alpha_1 \doteq \beta{\cdot}\alpha_1'$. This can produce constraints of the forms $\alpha{\cdot}\beta \, \dot{\in} \, L$ and $\alpha{\cdot}\beta \, \dot{\parallel} \, \beta'$. The second is problematic as we must guess the relation between α and β'. This complicates the termination proof.

We will avoid this problem by using a special property of pre-solved clauses, namely that $\alpha \, \dot{\parallel} \, \beta$ is in a pre-solved clause ϕ iff $x\alpha y$ and $x\beta z$ are in ϕ. Hence, if $\alpha \, \dot{\parallel} \, \beta$ and $\beta \, \dot{\parallel} \, \delta$ are in ϕ, then $\alpha \, \dot{\parallel} \, \delta$ is also in ϕ. This implies that we can write ϕ as $\dot{\parallel}(A_1) \uplus \ldots \uplus \dot{\parallel}(A_n) \uplus \psi$, where $\dot{\parallel}(A)$ abbreviates

$$\{\alpha \, \dot{\parallel} \, \alpha' \mid \alpha \neq \alpha' \ \wedge \ \alpha, \alpha' \in A\},$$

$A_1, \ldots, A_n$ are disjoint sets of path variables and ψ contains no divergence constraints. Now given such a constraint $\dot{\parallel}(A)$, and some valuation $V_{\mathcal{P}}$ with $V_{\mathcal{P}} \models \dot{\parallel}(A)$, suppose that a whole set of path variables $A_1 \subseteq A$ diverge under $V_{\mathcal{P}}$ with the same prefix, i.e., there is a path p such that

$$\forall \alpha \in A_1 : V_{\mathcal{P}}(\alpha) = pp_\alpha \quad \text{and} \quad \forall \alpha, \alpha' \in A_1 : \alpha \neq \alpha' \rightarrow p_\alpha \parallel_0 p_{\alpha'}.$$

Then we can replace the constraints set $\dot{\parallel}(A_1) \subseteq \dot{\parallel}(A)$ by

$$A_1 \doteq \beta{\cdot}A_1' \ \cup \ \dot{\parallel}_0(A_1')$$

under this valuation, since every valuation $V_{\mathcal{P}}'$ with $V_{\mathcal{P}}' \models (A_1 \doteq \beta{\cdot}A_1' \ \cup \ \dot{\parallel}_0(A_1'))$ satisfies

† The first case is needed because we do not allow values of path variables to be empty paths.

$V_{\mathcal{P}} =_{A_1} V'_{\mathcal{P}}$. Here β is new, $A'_1 = \{\alpha'_1, \ldots, \alpha'_n\}$ is a fresh copy of $A_1 = \{\alpha_1, \ldots, \alpha_n\}$ and $A \doteq \beta{\cdot}A'_1$ abbreviates the clause $\{\alpha_1 \doteq \beta{\cdot}\alpha'_1, \ldots, \alpha_n \doteq \beta{\cdot}\alpha'_n\}$. $\dot{\amalg}_0(A)$ is defined similarly to $\dot{\amalg}(A)$. If $V_{\mathcal{P}}$ additionally satisfies

$$\forall \alpha \in (A - A_1) : p \amalg V_{\mathcal{P}}(\alpha),$$

then we can replace the set of constraints $\dot{\amalg}(A)$ by

$$\dot{\amalg}(\{\beta\} \cup (A - A_1)) \;\cup\; A_1 \doteq \beta{\cdot}A'_1 \;\cup\; \dot{\amalg}_0(A'_1)$$

under the valuation $V_{\mathcal{P}}$.

Now for every path valuation $V_{\mathcal{P}}$ with $V_{\mathcal{P}} \models \dot{\amalg}(A)$ there must be a set A_1 with the properties stated above. We can find an appropriate A_1 by taking a path p which is maximal in

$$\{p \mid \exists \alpha, \alpha' \in A : \alpha \neq \alpha' \wedge p \prec V_{\mathcal{P}}(\alpha) \wedge p \prec V_{\mathcal{P}}(\alpha')\},$$

and defining A_1 as $\{\alpha \in A \mid p \prec V_{\mathcal{P}}(\alpha)\}$.

This finally leads to the following non-deterministic rule, where we also consider the effects of $A_1 \doteq \beta{\cdot}A'_1$ on the subterm agreements in ϕ:

$$(\text{Reduce}_1) \qquad \frac{xA_1Y_1 \cup \dot{\amalg}(A) \cup \psi}{\{x\beta z\} \cup zA'_1Y_1 \cup \dot{\amalg}_0(A'_1) \cup \dot{\amalg}(\{\beta\} \cup A_2) \;\cup\; \psi'}$$

where $\psi' = \psi[\alpha_1 \leftarrow \beta{\cdot}\alpha'_1, \ldots, \alpha_n \leftarrow \beta{\cdot}\alpha'_n]$, $A_1 \uplus A_2 = A$, $|A_1| > 1$ and z, β new. A'_1 is a disjoint copy of A_1. xA_1Y_1 is short for $\{x\alpha_1y_1, \ldots, x\alpha_ny_n\}$. ψ may not contain constraints of form $\delta{\cdot}\delta' \mathrel{\dot\in} L$ in ψ.

Note that we have avoided constraints of the form $\alpha{\cdot}\beta \;\dot{\amalg}\; \beta'$. We also employ the non-deterministic rules

$$(\text{Reduce}_2) \qquad \frac{\dot{\amalg}(A) \cup \psi}{\dot{\amalg}_0(A) \cup \psi}$$

$$(\text{Solv}) \qquad \frac{\dot{\amalg}_0(A) \cup \psi}{\{\alpha \mathrel{\dot\in} f_\alpha{\cdot}F^* \mid \alpha \in A\} \;\cup\; \psi} \qquad f_\alpha \neq f_{\alpha'} \text{ for } \alpha \neq \alpha'.$$

(Reduce$_2$) is needed because path variables always denote non-empty paths. We will view (Reduce$_1$) and (Reduce$_2$) as a single rule (Reduce). Note that (Solv) is the only rule where we need finiteness of the set of features $\mathcal{F}$.[†]

[†] If we had decided to use an infinite set of features, then we would have had to consider the case that there is a constraint $\alpha \;\dot{\amalg}_0\; \beta \in \phi$ with $\phi \models \alpha \mathrel{\dot\in} F\mathcal{F}^*$ for an infinite subset F of $\mathcal{F}$. Although this constraint is always satisfiable independent of what constraints are contained in ϕ for β, we have the problem that there is no finite set of solved clauses equivalent to ϕ. Hence, an infinite set of features would cause technical overhead. Since the proof provides no great additional insight when using an infinite set of features, we have decided to restrict ourself to a finite set of features.

To complete our rule system, we need the rules (LangDec$_\Lambda$), (DecClash), (Join) and (Empty). Since we will show that the rule system is terminating, we can replace (LangDec$_\Lambda$) by a simpler version, namely

$$(\text{LangDec}_{\text{dfun}}) \quad \frac{\{\alpha{\cdot}\beta \mathrel{\dot\in} L\} \cup \psi}{\{\alpha \mathrel{\dot\in} P\} \cup \{\beta \mathrel{\dot\in} S\} \cup \psi} \qquad P{\cdot}S \subseteq L, \quad (P,S) \in \text{dfun}(L)$$

L must contain a path w with $|w| > 1$.

Here dfun : $\mathcal{P}(\mathcal{F}^+) \to \mathcal{P}(\mathcal{F}^+) \times \mathcal{P}(\mathcal{F}^+)$ is a decomposition function that assigns to each regular language L a finite set of decompositions. dfun is called **decomposition complete** if for every regular language L and every path $w = w_1 w_2 \in L$ there is a pair (P, S) in dfun(L) with $w_1 \in P$ and $w_2 \in S$. The complete set of rules is denoted $\mathcal{R}^{\text{Solv}}_{\text{dfun}}$.

We illustrate the rule system $\mathcal{R}^{\text{Solv}}_{\text{dfun}}$ using one of the pre-solved clauses listed in section 6.1 on page 437, namely

$$\phi = \left\{ \begin{array}{c} \alpha_1 \mathrel{\dot{\amalg}} \alpha_2,\ \alpha_1 \mathrel{\dot{\amalg}} \alpha_3,\ \alpha_2 \mathrel{\dot{\amalg}} \alpha_3,\ x\alpha_1 y_1,\ x\alpha_2 y_2,\ x\alpha_3 y_3, \\ \alpha_1 \mathrel{\dot\in} (f \cup g),\ \alpha_2 \mathrel{\dot\in} (f^+ \cup h^+),\ \alpha_3 \mathrel{\dot\in} h(f \cup g) \end{array} \right\}$$

Using the notation introduced in this section, we can write ϕ as

$$\dot{\amalg}(\{\alpha_1, \alpha_2, \alpha_3\}) \cup \left\{ \begin{array}{c} x\alpha_1 y_1,\ x\alpha_2 y_2,\ x\alpha_3 y_3,\ \alpha_1 \mathrel{\dot\in} (f \cup g), \\ \alpha_2 \mathrel{\dot\in} (f^+ \cup h^+),\ \alpha_3 \mathrel{\dot\in} h(f \cup g) \end{array} \right\}$$

A successful $\mathcal{R}^{\text{Solv}}_{\text{dfun}}$-derivation transforming ϕ into a solved clause appears in figure 5. In the derivation we assume a function dfun with

$$\text{dfun}(f^+ \cup h^+) = \{(f^+ \cup h^+,\ f^+ \cup h^+)\}$$

and

$$\text{dfun}(h(f \cup g)) = \{(h,\ f \cup g)\}.$$

After the explanation of the rule system we can commence the technical part. Since we have added constraints of the form $\alpha \mathrel{\dot{\amalg}_0} \beta$, we have to extend condition 5 in the definition of a solved clause as presented on page 432. We require solved clauses not to contain constraints of the form $\alpha \mathrel{\dot{\amalg}_0} \beta$.

A clause ϕ is called **partitioned** if the set of divergence constraints of ϕ is of the form $\dot{\amalg}(A_1) \uplus \ldots \uplus \dot{\amalg}(A_k) \uplus \dot{\amalg}_0(A_{k+1}) \uplus \ldots \uplus \dot{\amalg}_0(A_n)$, where the A_i are disjoint.

PROPOSITION 7.1. *There exists a decomposition function* dfun *that is decomposition complete.*

PROOF. See proof of lemma 6.2 for the construction of such a function. □

PROPOSITION 7.2. *Let ϕ be a pre-solved clause and let γ be a $\mathcal{R}^{\text{Solv}}_{\text{dfun}}$-derivative of ϕ. Then γ is partitioned. Furthermore, for every pair of variables α, β such that $\alpha \neq \beta$, $x\alpha y \in \gamma$ and $x\beta z \in \gamma$ we have $\gamma \models \alpha \mathrel{\dot{\amalg}} \beta$.*

$$
\begin{array}{c}
\phi \\
\downarrow \text{(Reduce}_1\text{) with } A_1 = \{\alpha_2, \alpha_3\} \\
\left\{ \begin{array}{c} \alpha_1 \,\dot{\amalg}\, \beta,\ \alpha'_2 \,\dot{\amalg}_0\, \alpha'_3,\ x\alpha_1 y_1,\ x\beta z,\ z\alpha'_2 y_2,\ z\alpha'_3 y_3,\ \alpha_1 \,\dot{\in}\, (f \cup g), \\ \beta\cdot\alpha'_2 \,\dot{\in}\, (f^+ \cup h^+),\ \beta\cdot\alpha'_3 \,\dot{\in}\, h(f \cup g) \end{array} \right\} \\
\downarrow 2\times \text{(LangDec}_{\text{dfun}}\text{)} \\
\left\{ \begin{array}{c} \alpha_1 \,\dot{\amalg}\, \beta,\ \alpha'_2 \,\dot{\amalg}_0\, \alpha'_3,\ x\alpha_1 y_1,\ x\beta z,\ z\alpha'_2 y_2,\ z\alpha'_3 y_3,\ \alpha_1 \,\dot{\in}\, (f \cup g), \\ \beta \,\dot{\in}\, (f^+ \cup h^+),\ \alpha'_2 \,\dot{\in}\, (f^+ \cup h^+),\ \beta \,\dot{\in}\, h,\ \alpha'_3 \,\dot{\in}\, (f \cup g) \end{array} \right\} \\
\downarrow \text{(Solv)} \\
\left\{ \begin{array}{c} \alpha_1 \,\dot{\amalg}\, \beta,\ x\alpha_1 y_1,\ x\beta z,\ z\alpha'_2 y_2,\ z\alpha'_3 y_3,\ \alpha_1 \,\dot{\in}\, (f \cup g), \\ \alpha'_2 \,\dot{\in}\, h\mathcal{F}^*,\ \alpha'_3 \,\dot{\in}\, g\mathcal{F}^*,\ \beta \,\dot{\in}\, (f^+ \cup h^+),\ \alpha'_2 \,\dot{\in}\, (f^+ \cup h^+),\ \beta \,\dot{\in}\, h,\ \alpha'_3 \,\dot{\in}\, (f \cup g) \end{array} \right\} \\
\downarrow 3\times \text{(Join)} \\
\left\{ \begin{array}{c} \alpha_1 \,\dot{\amalg}\, \beta,\ x\alpha_1 y_1,\ x\beta z,\ z\alpha'_2 y_2,\ z\alpha'_3 y_3,\ \alpha_1 \,\dot{\in}\, (f \cup g), \\ \beta \,\dot{\in}\, h,\ \alpha'_2 \,\dot{\in}\, h^+,\ \alpha'_3 \,\dot{\in}\, g \end{array} \right\} \\
\downarrow \text{(Reduce}_2\text{)} \\
\left\{ \begin{array}{c} \alpha_1 \,\dot{\amalg}_0\, \beta,\ x\alpha_1 y_1,\ x\beta z,\ z\alpha'_2 y_2,\ z\alpha'_3 y_3,\ \alpha_1 \,\dot{\in}\, (f \cup g), \\ \beta \,\dot{\in}\, h,\ \alpha'_2 \,\dot{\in}\, f^+,\ \alpha'_3 \,\dot{\in}\, g \end{array} \right\} \\
\downarrow \text{(Solv)} \\
\left\{ \begin{array}{c} x\alpha_1 y_1,\ x\beta z,\ z\alpha'_2 y_2,\ z\alpha'_3 y_3,\ \alpha_1 \,\dot{\in}\, (f \cup g), \\ \alpha_1 \,\dot{\in}\, f\mathcal{F}^*,\ \beta \,\dot{\in}\, h\mathcal{F}^*,\ \beta \,\dot{\in}\, h,\ \alpha'_2 \,\dot{\in}\, f^+,\ \alpha'_3 \,\dot{\in}\, g \end{array} \right\} \\
\downarrow 2\times \text{(Join)} \\
\left\{ \begin{array}{c} x\alpha_1 y_1,\ x\beta z,\ z\alpha'_2 y_2,\ z\alpha'_3 y_3, \\ \alpha_1 \,\dot{\in}\, f,\ \beta \,\dot{\in}\, h,\ \alpha'_2 \,\dot{\in}\, f^+,\ \alpha'_3 \,\dot{\in}\, g \end{array} \right\}
\end{array}
$$

Figure 5. A successful $\mathcal{R}^{\mathsf{Solv}}_{\text{dfun}}$-derivation

PROPOSITION 7.3. *For every partitioned clause ϕ the rule* (Reduce) = (Reduce$_1$) + (Reduce$_2$) *is $\mathcal{V}ars_{\mathcal{X}}(\phi)$-sound and globally $\mathcal{V}ars_{\mathcal{X}}(\phi)$-preserving. The rule* (Solv) *is $V_{\mathcal{X}} \cup V_{\mathcal{P}}$-sound and $V_{\mathcal{X}} \cup V_{\mathcal{P}}$-preserving. If* dfun *is decomposition complete, then* (LangDec$_{\text{dfun}}$) *is $V_{\mathcal{X}} \cup V_{\mathcal{P}}$-sound and $V_{\mathcal{X}} \cup V_{\mathcal{P}}$-preserving.*

LEMMA 7.4. $\mathcal{R}^{\mathsf{Solv}}_{\text{dfun}}$ *is terminating.*

PROOF. For (Solv), (Join), (LangDec), (DecClash) and (Empty) it is trivial to see that there are no infinite derivations using only these rules. Furthermore, there are no derivations which use (Reduce) infinitely often, since during every application of (Reduce) at least one divergence constraint is removed (note that $|A_1| > 1$ is an application condition of (Reduce$_1$)). Hence, there are no infinite $\mathcal{R}^{\mathsf{Solv}}_{\text{dfun}}$-derivations. □

LEMMA 7.5. *Let ϕ be a pre-solved clause. If* dfun *is decomposition complete, then a $\mathcal{R}^{\mathsf{Solv}}_{\text{dfun}}$-derivative of ϕ is $\mathcal{R}^{\mathsf{Solv}}_{\text{dfun}}$-irreducible if and only if it is solved.*

PROOF. Let γ be a $\mathcal{R}^{\mathsf{Solv}}_{\text{dfun}}$-derivative of ϕ. We have to show that if γ is not solved, then one of the rules applies. We will check all conditions that are stated in the definition on page 432.

Condition 1 is satisfied by every $\mathcal{R}_{\mathrm{dfun}}^{\mathrm{Solv}}$-derivative of ϕ since ϕ is pre-solved and we do not add or change any sort restriction constraint. If one of the conditions 2 or 3 is not satisfied, then one of the rules (Join) or (Empty) will apply. Condition 6 is satisfied by every $\mathcal{R}_{\mathrm{dfun}}^{\mathrm{Solv}}$-derivative of ϕ by proposition 7.2. Now let's check the conditions 4 and 5:

γ **contains a constraint** $\alpha{\cdot}\beta \mathrel{\dot\in} L$**.** (LangDec$_{\mathrm{dfun}}$) or (DecClash) is applicable.

γ **contains a constraint** $\alpha \mathrel{\dot{\amalg}_0} \beta$**.** Then γ is of the form $\dot{\amalg}_0(A) \cup \psi$ by proposition 7.2, which implies that (Solv) is applicable.

γ **contains a constraint** $\alpha \mathrel{\dot{\amalg}} \beta$**.** By proposition 7.2 we know that in this case γ is of the form $\dot{\amalg}(A) \cup \psi$. Given the above we can assume that (Reduce) is applicable.

□

LEMMA 7.6. *For every pre-solved clause ϕ there is a finite and effectively computable set of solved clauses Γ such that for every $\mathcal{I}$*

$$[\![\phi]\!]^{\mathcal{I}}_{\mathcal{V}ars_{\mathcal{X}}(\phi)} = \bigcup_{\gamma\in\Gamma} [\![\gamma]\!]^{\mathcal{I}}_{\mathcal{V}ars_{\mathcal{X}}(\phi)}.$$

PROOF. Follows from propositions 7.1, 7.2 and 7.3 and lemmas 7.4 and 7.5. □

COROLLARY 7.7. *Satisfiability of pre-solved clauses is decidable.*

Finally, we are able to combine both phases.

THEOREM 7.8. *Satisfiability of prime clauses is decidable.*

PROOF. Follows from the corollaries 6.17 and 7.7. □

8. Conclusion

We have shown that the pure existential fragment of feature logic extended by regular path expressions is decidable. The main prerequisite for achieving this result was to switch from the original, unsorted syntax to a two-sorted syntax. For each clause in the original syntax we get an equivalent clause in the new syntax by translating a regular path expression xLy into $\{x\alpha y, \alpha \mathrel{\dot\in} L\}$ with α new.

The result of the translation constitutes a special class of clauses: the class of prime clauses. The main restriction imposed on prime clauses is that for each path variable

α there is *at most* one constraint $x\alpha y$ contained in a clause. For prime clauses we have presented an algorithm that transforms a clause into an equivalent set of pre-solved clauses. In a second phase pre-solved clauses are checked for satisfiability by transforming them into an equivalent set of solved clauses. Since every solved clause is prime, the result may be reused for later computation.

Our syntax is more expressive than the original one. Although restriction to prime clauses was sufficient for our purposes, it may be interesting to examine whether decidability can be preserved in the absence of the restriction.

Finally, we mention a possible extension of the work in this paper, namely the generalization of features (which are binary, functional relations) to arbitrary relations. If there are no additional axioms for the relations, then every conjunction of positive constraints is satisfiable. To see this, let $\phi = \{xRy,\ xRz\}$ be some formula where R is a binary relation. Now Skolemization yields the formula $\phi' = \{c_xRc_y,\ c_xRc_z\}$, where ϕ is satisfiable if ϕ' is. But ϕ' is valid in the Herbrand interpretation of ϕ'.

But the use of regular languages of binary relations has also been considered in the context of terminological logics, which have their roots in the knowledge representation formalism KL-ONE (see Brachman & Schmolze (1985); for a comparison of feature logic and terminological logics see Nebel & Smolka (1991)). Baader (1990) has consider an extension of the terminological language $\mathcal{ALC}$ of Schmidt-Schauß & Smolka (1991), which uses regular languages of roles (binary relations) instead of single roles to build up concept terms. He has shown, that testing satisfiability of this extension is decidable. In contrast with our problem, there are cycles in the languages but no cycles in the graph, since these cycles (i.e., cyclic concept definitions) can be compiled out using a technique called "internalization". As we have pointed out in the example on page 425, the combination of both types of cycle makes our problem hard. Thus, while Baader can split the problem into independent subproblems, we cannot use a similar technique because of the structure of our problem (to be more concrete, the technique of "internalization" has been used in (Baader *et al.* 1993) to prove undecidability of functional uncertainty with negation).

Acknowledgements

I would like to thank Jochen Dörre, Joachim Niehren, Stephen Spackman and Ralf Treinen for helpful discussions and reading draft versions of the paper. In particular, I am grateful to Joachim Niehren for his comments on an earlier draft. Furthermore, I want to thank two anonymous referees for valuable comments.

References

Aït-Kaci, H., Nasr, R. (1986). Login: A logic programming language with built-in inheritance. *The Journal of Logic Programming*, 3:185–215.

Aït-Kaci, H., Podelski, A. (1991). Towards a meaning of LIFE. In *Proc. of the PLILP'91*, Springer LNCS vol. 528, pages 255–274. Springer-Verlag.

Aït-Kaci, H. (1986). An algebraic semantics approach to the effective resolution of type equations. *Theoretical Computer Science*, 45:293–351.

Aït-Kaci, H., Lincoln, P., Nasr, R. (1987). Le Fun: Logic, equations, and functions. In *Proceedings of the 1987 Symposium on Logic Programming*, pages 17–23. IEEE Computer Society.

Aït-Kaci, H., Podelski, A., Smolka, G. (1992). A feature-based constraint system for logic programming with entailment. In *Fifth Generation Computer Systems 1992*, pages 1012–1021, Tokyo, Japan. Institute for New Generation Computer Technology.

Baader, F. (1990). Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. Research Report RR-90-13, DFKI, Postfach 2080, 6750 Kaiserslautern, Germany.

Baader, F., Bürckert, H.-J., Nebel, B., Nutt, W., Smolka, G. (1993). On the expressivity of feature logics with negation, functional uncertainty, and sort equations. *Journal of Logic, Language and Information*, 2:1–18.

Backofen, R., Smolka, G. (1993). A complete and recursive feature theory. In *Proc. of the 31st ACL*, pages 193–200, Columbus, Ohio. acl. Full version has appeared as Research Report RR-92-30, DFKI, Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany, and will appear in Theoretical Computer Science.

Backofen, R. (1993). On the decidability of functional uncertainty. In *Proc. of the 31 th ACL*, pages 201–208, Columbus, Ohio. Association for Computational Linguistics.

Backofen, R. (1993). Regular path expressions in feature logic. In Kirchner, C., editor, *Proc. of the RTA'93*, pages 121–135, Montreal, Canada.

Brachman, R. J., Schmolze, J. G. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216.

Dershowitz, N. (1987). Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116.

Johnson, M. (1988). *Attribute-Value Logic and the Theory of Grammar*, volume 16 of *CSLI Lecture Notes*. CSLI.

Johnson, M. (1991). Logic and feature structures. In *Proceedings of IJCAI-91*, Sydney, Australia.

Kaplan, R. M., Bresnan, J. (1982). Lexical-Functional Grammar: A formal system for grammatical representation. In Bresnan, J., editor, *The Mental Representation of Grammatical Relations*, pages 173–381. MIT Press, Cambridge (MA).

Kaplan, R. M., Maxwell III, J. T. (1988). An algorithm for functional uncertainty. In *Proceedings of the 12th International Conference on Computational Linguistics*, pages 297–302, Budapest, Hungary.

Kaplan, R. M., Zaenen, A. (1988). Long-distance dependencies, constituent structure, and functional uncertainty. In Baltin, M., Kroch, A., editors, *Alternative Conceptions of Phrase Structure*. University of Chicago Press, Chicago.

Kasper, R. T., Rounds, W. C. (1986). A logical semantics for feature structures. In *Proceedings of the 24th Annual Meeting of the ACL, Columbia University*, pages 257–265, New York, N.Y.

Keller, B. (1991). Feature logics, infinitary descriptions and the logical treatment of grammar. Cognitive Science Research Report 205, Univerity of Sussex, School of Cognitive and Computing Sciences.

Nebel, B., Smolka, G. (1991). Attributive description formalisms ... and the rest of the world. In Herzog, O., Rollinger, C.-R., editors, *Text Understanding in LILOG: Integrating Computational Linguistics and Artificial Intelligence*, volume 546 of *Lectures Notes in Artificial Intelligence*, pages 439–452. Springer-Verlag, Berlin, Germany.

Rounds, W. C., Kasper, R. (1986). A complete logical calculus for record structures representing linguistic information. In *Proc. of the Symposium on Logic in Computer Sciences*, pages 38–43, Cambridge (MA). IEEE Computer Society.

Schmidt-Schauß, M., Smolka, G. (1991). Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26.

Shieber, S. M. (1986). *An Introduction to Unification-Based Approaches to Grammar*, volume 4 of *CSLI Lecture Notes*. Stanford University, Stanford (CA).

Smolka, G., Treinen, R. (1992). Records for logic programming. In Apt, K., editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 240–254, Washington, USA. The MIT Press. Full version has appeared as Research Report RR-92-23, DFKI, Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany; will also appear in Journal of Logic Programming.

Smolka, G. (1988). A feature logic with subsorts. LILOG-Report 33, IWBS, IBM Deutschland, Stuttgart.

Smolka, G. (1992). Feature constraint logics for unification grammars. *Journal of Logic Programming*, 12:51–87.