In: Proc. of the ACL'93

ON THE DECIDABILITY OF FUNCTIONAL UNCERTAINTY*

Rolf Backofen
German Research Center for Artificial Intelligence (DFKI)
W-6600 Saarbrücken, Germany
backofen@dfki.uni-sb.de

Abstract

We show that feature logic extended by functional uncertainty is decidable, even if one admits cyclic descriptions. We present an algorithm, which solves feature descriptions containing functional uncertainty in two phases, both phases using a set of deterministic and non-deterministic rewrite rules. We then compare our algorithm with the one of Kaplan and Maxwell, that does not cover cyclic feature descriptions.

1 Introduction

Feature logic is the main device of unification grammars, the currently predominant paradigm in computational linguistics. More recently, feature descriptions have been proposed as a constraint system for logic programming (e.g. see [11]). They provide for partial descriptions of abstract objects by means of functional attributes called features.

Formalizations of feature logic have been proposed in various forms (for more details see [3] in this volume). We will follow the logical approach introduced by Smolka [9, 10], 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 either 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). The constraints of the form xfy can be generalized to constraints of the form xwy, where $w = f_1 \dots f_n$ is a finite feature path. This does not affect the computational properties.

In this paper we will be concerned with an extension to feature descriptions, which has been introduced as "functional uncertainty" by Kaplan and Zaenen [7] and Kaplan and Maxwell [5]. This formal device plays an important role in the framework of LFG in modelling so-called long distance dependencies and constituent coordination. For a detailed linguistic motivation see [7], [6] and [5]; a more general use of functional uncertainty can be found in [8].

Functional uncertainty consists of constraints of the form xLy, where L is a finite description of a regular language of feature paths. A constraint xLy holds if there is a path $w \in L$ such that xwy holds. Under 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 and Maxwell [5, 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 $s\ \mathit{topic}\ x \wedge s\ \mathit{comp}\ \mathit{obj}\ x$, $s\ \mathit{topic}\ x \wedge s\ \mathit{comp}\ \mathit{comp}\ \mathit{obj}\ x$ 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 $s\ \mathit{topic}\ x \wedge s\ \mathit{comp}^*\ \mathit{obj}\ x$.

Kaplan and Maxwell [5] have shown that consistency of feature descriptions is decidable, provided that a certain acyclicity condition is met. More recently, Baader et al. [1] have proven, that consistency is not decidable if we add negation. But it is an open problem whether consistency of feature descriptions without negation *and* without additional restrictions (such as acyclicity) is decidable. In the work presented here we show that it indeed is decidable.

2 The Method

We will first briefly describe the main part of solving the standard feature terms and then turn to their extension with functional uncertainty.

Consider a clause $\phi = xp_1y_1 \wedge xp_2y_2$ (from now on we will refer to pure conjunctive formulae as clauses). A standard method for solving feature terms would rewrite ϕ in order to achieve a solved form. This rewriting depends 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. This implies that ϕ is equivalent to $xp_1y_1 \wedge y_1 \doteq y_2$. If

*This work was supported by a research grant, ITW 9002 0, from the German Bundesministerium für Forschung und Technologie to the DFKI project DISCO. I would like to thank Jochen Dörre, Joachim Niehren and Ralf Treinen for reading draft version of this paper. For space limitations most of the proofs are omitted; they can be found in the complete paper [2]

p_1 is a prefix of p_2 and hence $p_2 = p_1 p'$, we can transform ϕ equivalently into the formulae $xp_1y_1 \wedge y_1p'y_2$. The reverse case is treated in a similar fashion. If neither prefix or equality holds between the paths, there is nothing to be done. By and large, clauses where this holds for every x and every pair of different constraints xp_1y and xp_2z are the solved forms in Smolka [9], which are consistent.

If we consider a clause of the form $\phi = xL_1y_1 \wedge xL_2y_2$, then we again have 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 we choose out of L_1 and L_2. Hence, we have to guess the relation between p_1 and p_2 before we can calculate the relation between y_1 and y_2. However, there is a problem with the original syntax, namely that it does not allow one to express any relation between the chosen paths (in a later section we will compare our algorithm to the one of Kaplan/Maxwell, thus showing where exactly the problem occurs in their syntax). Therefore, we extend the syntax by introducing so-called path variables (written $\alpha, \beta, \alpha', \ldots$), which are interpreted as feature paths (we will call the other variables first order variables). Hence, if we use the modified subterm relation $x\alpha y$ and a restriction constraint $\alpha \,\dot{\in}\, L$, a constraint xLy can equivalently be expressed as $x\alpha y \wedge \alpha \,\dot{\in}\, L$ (α new). The interpretation of $x\alpha y$ is done in two steps. Given a valuation $V_{\mathcal{P}}$ of the path variables as feature paths, a constraint $x\alpha y$ in ϕ is substituted by $xV_{\mathcal{P}}(\alpha)y$. This constraint is then interpreted using the valuation for the first order variables in the way such constraints are usually interpreted.

By using this extended (two-sorted) syntax we are now able to reason about the relations between different path variables. In doing so, we introduce additional constraints $\alpha \doteq \beta$ (equality), $\alpha \,\dot{\prec}\, \beta$ (prefix) and $\alpha \parallel \beta$ (divergence). Divergence holds if neither equality nor prefix holds. 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* iff for each pair of different constraint $x\alpha y_1$ and $x\beta y_2$ in ϕ there is a constraint $\alpha \parallel \beta$ in ϕ. We call this clauses pre-solved, since such clauses are not necessarily consistent. It may happen, that the divergence constraints together with the restrictions of form $\alpha \,\dot{\in}\, L$ are inconsistent (e.g. think of the clause $\alpha \,\dot{\in}\, f^+ \wedge \beta \,\dot{\in}\, ff^+ \wedge \alpha \parallel \beta$). But pre-solved clauses have the property, that if we find a valuation for the path variables, then the clause is consistent.

Our algorithm first transforms a clause into a set of pre-solved clauses, which is (seen as a disjunction) equivalent to the initial clause. In a second phase the pre-solved clauses are checked for consistency with respect to the path variables. In this paper we will concentrate on the first phase, since it is the more difficult one.

Before looking at the technical part we will illustrate the first phase. For the rest of the paper we will write clauses as sets of atomic constraints. Now consider the clause $\gamma = \{x\alpha y,\ \alpha_1 \,\dot{\in}\, L_1,\ x\beta z,\ \beta \,\dot{\in}\, L_2\}$. The first step is to guess the relation between the path variables α and β. Therefore, γ can be expressed equivalently by the set of clauses

$$\gamma_1 = \{\alpha \parallel \beta\} \cup \gamma \qquad \gamma_3 = \{\alpha \,\dot{\prec}\, \beta\} \cup \gamma$$
$$\gamma_2 = \{\alpha \doteq \beta\} \cup \gamma \qquad \gamma_4 = \{\beta \,\dot{\prec}\, \alpha\} \cup \gamma$$

The clause γ_1 is pre-solved. For the others we have to evaluate the relation between α and β, which is done as follows. For γ_2 we substitute β by α and z by y, 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 only interested in their valuation. Combining $\{\alpha \,\dot{\in}\, L_1, \alpha \,\dot{\in}\, L_2\}$ to $\{\alpha \,\dot{\in}\, (L_1 \cap L_2)\}$ then will give us the equivalent pre-solved clause $\gamma_2' = \{y \doteq z,\ x\alpha y,\ \alpha \,\dot{\in}\, (L_1 \cap L_2)\}$.

For γ_3 we know that the variable β can be split into two parts, one of them covered by α. We can use concatenation of path variables to express this, i.e. we can replace β by the term $\alpha{\cdot}\beta'$, where β' is new. Thus we get the clause

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

The only thing that we have to do additionally in order to achieve a pre-solved clause is to resolve the constraint $\alpha{\cdot}\beta' \,\dot{\in}\, L_2$. To do this we have to guess a so-called decomposition P, S of L_2 with $P{\cdot}S \subseteq L_2$ such that $\alpha \,\dot{\in}\, P$ and $\beta' \,\dot{\in}\, S$. 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 covering all possibilities. Finally, reducing $\{\alpha \,\dot{\in}\, L_1,\ \alpha \,\dot{\in}\, P\}$ to $\{\alpha \,\dot{\in}\, (L_1 \cap P)\}$ 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 $y\beta' z$. This implies that there again may be some path variables the relation of which is unknown. Hence, after reducing the terms of form $\alpha \doteq \beta$ or $\alpha \,\dot{\prec}\, \beta$ we may have to repeat the non-deterministic choice of relations between path variables. In the end, the only remaining constraints between path variables will be of the form $\alpha \parallel \beta$.

We have to consider some additional point, namely that the rules we present will (naturally) loop in some cases. Roughly speaking, one can say that this always occurs if a cycle in the graph coincides with a cycle in the regular language. To see this let us vary the above example and let γ now be the clause $\{x\alpha x,\ \alpha \,\dot{\in}\, f,\ x\beta z,\ \beta \,\dot{\in}\, f^*g\}$. Then a possible looping derivation could be

1. add $\alpha \,\dot{\prec}\, \beta$:
   $\{\alpha \,\dot{\prec}\, \beta,\ x\alpha x,\ \alpha \,\dot{\in}\, f,\ x\beta z,\ \beta \,\dot{\in}\, f^*g\}$
2. split β into $\alpha{\cdot}\beta'$:
   $\{x\alpha x,\ \alpha \,\dot{\in}\, f,\ x\beta' z,\ \alpha{\cdot}\beta' \,\dot{\in}\, f^*g\}$
3. decompose $\alpha{\cdot}\beta' \,\dot{\in}\, f^*g$:
   $\{x\alpha x,\ \alpha \,\dot{\in}\, f,\ x\beta' z,\ \alpha \,\dot{\in}\, f^*,\ \beta' \,\dot{\in}\, f^*g\}$

4. join α-restrictions:

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

However, we will proof that the rule system is quasi-terminating, which means that the rule system may cycle, but produces only finitely many different clauses (see [4]). This means that checking for cyclic derivations will give us an effective algorithm.

Quasi-termination 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 rules system produces only finitely many regular languages. In order to show that our rewrite system is complete, we also have to show that every solution can be found in a pre-solved clause.

3 Preliminaries

Our *signature* consists of a set of *sorts* $\mathcal{S}$ $(A, B, \ldots)$, *first order variables* $\mathcal{X}$ $(x, y, \ldots)$, *path variables* $\mathcal{P}$ $(\alpha, \beta, \ldots)$ and *features* $\mathcal{F}$ $(f, g, \ldots)$. We will assume a finite set of features and infinite sets of variables and sorts. A *path* is a finite string of features. 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. 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 $u = wfw_1 \wedge v = wgw_2$. Clearly, $\amalg$ 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* $(p, q, \ldots)$ is either a path variable α or a concatenation of path variables $\alpha \cdot \beta$. We will allow complex path terms only in divergence constraints and not in prefix or equality constraints. Hence, the set of atomic constraints is given by

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

We exclude empty paths in subterm agreement since $x\epsilon y$ is equivalent to $x \doteq y$. Therefore, we require $f_1 \cdot \ldots \cdot f_n \in \mathcal{F}^+$ and $L \subseteq \mathcal{F}^+$.

A *clause* is a finite set of atomic constraint 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 $\alpha \cdot \beta \dot{\in} L$ or a constraint $\alpha \cdot \beta\ \dot{\amalg}\ q$.[1] Constraints of the form $p \dot{\in} L$, $p\ \dot{\amalg}\ q$, $\alpha \dot{\prec} \beta$ and $\alpha \doteq \beta$ will be called *path constraints*.

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 unary, disjoint predicates (hence $A^{\mathcal{I}} \cap B^{\mathcal{I}} = \emptyset$ for $A \neq B$). A *valuation* is a pair $(V_{\mathcal{X}}, V_{\mathcal{P}})$, where $V_{\mathcal{X}}$ is a standard first order valuation of the variables in X and $V_{\mathcal{P}}$ is a function $V_{\mathcal{P}} : \mathcal{P} \rightarrow \mathcal{F}^+$. 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:

$$\begin{array}{lll} (V_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{I}} Ax & :\Longleftrightarrow & V_{\mathcal{X}}(x) \in A^{\mathcal{I}} \\ (V_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{I}} x \doteq y & :\Longleftrightarrow & V_{\mathcal{X}}(x) = V_{\mathcal{X}}(y) \\ (V_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{I}} x\ p\ y & :\Longleftrightarrow & V_{\mathcal{X}}(x)\ F_1^{\mathcal{I}} \circ \ldots \circ F_n^{\mathcal{I}}\ V_{\mathcal{X}}(y) \\ (V_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{I}} x\alpha y & :\Longleftrightarrow & (V_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{I}} x\ V_{\mathcal{P}}(\alpha)\ y \\ (V_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{I}} p \dot{\in} L & :\Longleftrightarrow & V_{\mathcal{P}}(p) \in L \\ (V_{\mathcal{X}}, V_{\mathcal{P}}) \models_{\mathcal{I}} p\ \dot{\diamond}\ q & :\Longleftrightarrow & V_{\mathcal{P}}(p) \diamond V_{\mathcal{P}}(q) \\ & & \text{for } \dot{\diamond} \in \{\dot{\amalg}, \dot{\prec}, \doteq\}, \end{array}$$

where p is the path $f_1 \cdot \ldots \cdot f_n$ and $F_i^{\mathcal{I}}$ are the interpretations of f_i in $\mathcal{I}$.

For a set $\xi \subseteq \mathcal{X}$ we define $=_\xi$ to be the following relation on first order valuation:

$$V_{\mathcal{X}} =_\xi V'_{\mathcal{X}} \quad \text{iff} \quad \forall x \in \xi :\ V_{\mathcal{X}}(x) = V'_{\mathcal{X}}(x).$$

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 $V_{\mathcal{X}} =_{\mathcal{X} \cap \vartheta} V'_{\mathcal{X}}$, $V_{\mathcal{P}} =_{\mathcal{P} \cap \vartheta} V'_{\mathcal{P}}$ 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 will call X-solutions just solutions and write $[\![\phi]\!]^{\mathcal{I}}$ instead of $[\![\phi]\!]^{\mathcal{I}}_{\mathcal{X}}$.

For checking satisfiability we will use transformation rules. A rule R is *ϑ-sound* $\phi \rightarrow_R \gamma \Rightarrow [\![\phi]\!]^{\mathcal{I}}_{\vartheta} \supseteq [\![\gamma]\!]^{\mathcal{I}}_{\vartheta}$ for every interpretation $\mathcal{I}$. R is called *ϑ-preserving* if $\phi \rightarrow_R \gamma \Rightarrow [\![\phi]\!]^{\mathcal{I}}_{\vartheta} \subseteq [\![\gamma]\!]^{\mathcal{I}}_{\vartheta}$. R is *globally ϑ-preserving* if $[\![\phi]\!]^{\mathcal{I}}_{\vartheta} \subseteq \bigcup_{\phi \rightarrow_R \gamma} [\![\gamma]\!]^{\mathcal{I}}_{\vartheta}$.

4 The First Phase

4.1 A Set of Rules

Recall that we have switched from the original syntax to a (two-sorted) syntax by translating constraints xLy into $\{x\alpha y, \alpha \dot{\in} L\}$, where α is new. The result of the translation constitutes a special class of clauses, namely the class of prime clauses, which will be defined below. Hence, it suffices to show decidability of consistency of prime clauses. They are the input clauses for the first phase.

Let ϕ be some clause and x, y be different 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, i.e. 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

1. $x \doteq y$ appears in ϕ iff ϕ eliminates y,
2. For every path variable α used in ϕ there is *at most* one constraint $x\alpha y \in \phi$.

[1] We will not differentiate between $p\ \dot{\amalg}\ q$ and $q\ \dot{\amalg}\ p$.

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

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

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

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

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

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

$$\text{(DClash1)}\ \frac{\{\alpha{\cdot}\beta \mathrel{\dot\shortparallel} \alpha\} \cup \psi}{\bot}$$

$$\text{(DClash2)}\ \frac{\{\alpha \mathrel{\dot\shortparallel} \alpha\} \cup \psi}{\bot}$$

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

Figure 1: Simplification rules. Note that (Pre) does not introduce a new variable.

A basic clause ϕ is called *prime* if ϕ does not contain an atomic constraint of the form $p \mathrel{\dot\shortparallel} q$, $\alpha \mathrel{\dot\prec} \beta$ or $\alpha \doteq \beta$. Every clause ϕ in the original Kaplan/Maxwell syntax can be translated into a prime clause γ such that ϕ is consistent iff γ is consistent.

Now let's turn to the output clauses of the first step. A basic clause is said to be *pre-solved* if 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 $L = L'$. Furthermore, $\alpha \mathrel{\dot\in} \emptyset$ is not in ϕ.
3. $\alpha{\cdot}\beta$, $\alpha \doteq \beta$ or $\alpha \mathrel{\dot\prec} \beta$ are not contained in ϕ.
4. $\alpha \mathrel{\dot\shortparallel} \beta \in \phi$ iff $\alpha \neq \beta$, $x\alpha y \in \phi$ and $x\beta z \in \phi$.

Lemma 4.1 *A pre-solved clause ϕ is consistent 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.*

Now let's turn to the rule system. As we have explained informally, the first rule adds non-deterministicly 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. Furthermore, we will consider only the constraints $\alpha \doteq \beta$, $\alpha \mathrel{\dot\shortparallel} \beta$ and $\alpha \mathrel{\dot\prec} \beta$ and not additionally the constraint $\alpha \mathrel{\dot\succ} \beta$.

For better readability we will use pseudo-code for describing this rule (using the usual don't care/don't know distinction for non-determinism):

(PathRel)
  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** $\alpha \neq \beta$ and $\alpha \mathrel{\dot\shortparallel} \beta \notin \phi$
      add $\alpha \mathrel{\dot\diamond_\beta} \beta$ with $\dot\diamond_\beta \in \{\doteq, \dot\prec, \dot\shortparallel\}$ (don't know)

"don't care non-determinism" means that one is free to choose an arbitrary alternative at this choose point, whereas "don't know" means that one has to consider every alternative in parallel (i.e. for every alternative of the don't care non-determinism a clause ϕ is equivalent to the set of all don't know alternatives that can be generated by applying the rule to ϕ). Note that the order of rule application is another example for don't care non-determinism in our rule system.

Although we have restricted the relations $\dot\diamond_\beta$ to $\{\doteq, \dot\prec, \dot\shortparallel\}$, 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 $V_{\mathcal{P}}(\alpha)$ is prefix 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 $\alpha \mathrel{\dot\shortparallel} \beta \notin \phi$ 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 whose interpretation is prefix minimal. Therefore, the restriction $\dot\diamond_\beta \in \{\doteq, \dot\prec, \dot\shortparallel\}$ is satisfied.

We have defined (PathRel) in a very special way. The reason for this is that only by using this special definition we can maintain the condition that concatenation of path variables is restricted to binary concatenation. E.g. assume that we would have added both $\beta_1 \mathrel{\dot\prec} \alpha$ and $\alpha \mathrel{\dot\prec} \beta_2$ to a clause γ. Then first splitting up the variable α into $\beta_1{\cdot}\alpha'$ and then β_2 into $\alpha{\cdot}\beta_2'$ 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 occur.

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

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

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

where $P, S, L \subseteq F^+$ and Λ is a finite set of reg. languages with $L, P, S \in \Lambda$. L must contain a word w with $|w| > 1$.

The clash rule is needed since we require regular languages not to contain the empty path. The remaining rules are listed in Figure 1.

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

For ($\mathrm{LangDec}_\Lambda$) to be globally preserving we need to find a suitable pair P, S in Λ for every possible valuation of α and β. Therefore, we require Λ to satisfy

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

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.* Surely, we will not find a *finite* Λ that is closed under decomposition and ϕ-closed for arbitrary ϕ. But the next lemma states some weaker condition that suffices. We say that γ is a *$(\phi, \mathcal{R}_\Lambda)$-derivative* if γ is derivable from ϕ by using only rules from $\mathcal{R}_\Lambda$. If $\mathcal{R}_\Lambda$ is clear from the context, we will just say that γ is a ϕ-derivative.

Lemma 4.2

1. *If Λ is ϕ-closed and closed under intersection, then Λ is γ-closed for all $(\phi, \mathcal{R}_\Lambda)$-derivatives γ.*
2. *For every prime clause ϕ there is a* finite *Λ such that Λ is ϕ-closed and closed under intersection and decomposition.*

The proof of this lemma (containing the construction of the set Λ) can be found in the appendix.

4.2 Completeness and Quasi-Termination

The rule system serves for an algorithm to transform a prime clause into an equivalent set of pre-solved clauses. The rules are applied in arbitrary order until a pre-solved clause has been derived. If one of the non-deterministic rules is applied, a clause is substituted by a whole set of clauses, one for each of the don't know alternatives. Since the rule system is quasi-terminating, we may encounter cycles during the application of the rules. In this case we skip the corresponding alternative, since every pre-solved clause that can be produced via a cyclic derivation can also be produced via a derivation that does not contain a cycle.

Theorem 4.3 *Let ϕ be a prime clause. If Λ is ϕ-closed, closed under intersection and decomposition, then $[\![\phi]\!]^{\mathcal{I}} = \cup_{\gamma \in \Phi}\ [\![\gamma]\!]^{\mathcal{I}}$ for every interpretation $\mathcal{I}$, where Φ is the set of pre-solved $(\phi, \mathcal{R}_\Lambda)$-derivatives. The set Φ is finite and effectively computable.*

To prove this theorem we have to show that the rule system is sound and complete. Sound means, that we do not add new solutions during the processing, whereas complete means that we find all solutions in the set of pre-solved derivatives.

For the completeness it normally suffices to show that (1) every rule preserves (or globally preserves) the initial solutions and (2) the pre-solved clauses are exactly the $\mathcal{R}_\Lambda$-irreducible clause (i.e. if a clause is not pre-solved, then one rule applies). But in our case this is not sufficient as the rule system is quasi-terminating. A prime clause ϕ may have a solution $V_\mathcal{X}$ which is a solution of all $(\phi, \mathcal{R}_\Lambda)$-derivatives in some cyclic derivation, but can not be found in any pre-solved $(\phi, \mathcal{R}_\Lambda)$-derivative. We have to show that this cannot happen. Since this part of the proof is unusual, we will explain the main idea (see the appendix for a more detailed outline of the proofs).

Let ϕ be some (consistent) prime clause and let $V_\mathcal{X} \in [\![\phi]\!]^{\mathcal{I}}$ for some $\mathcal{I}$. Then there exists a path valuation $V_\mathcal{P}$ such that $(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} \phi$. We will find a pre-solved ϕ-derivative that has $V_\mathcal{X}$ as a solution by imposing an additional control that depends on $V_\mathcal{P}$. This control will guarantee (1) finiteness of derivations, (2) that each derivation ends with a pre-solved clause, (3) the initial solution is a solution of every clause that is derivable under this control. Since the (Pre) rule does not preserve the initial path valuation $V_\mathcal{P}$ (recall that the variable β is substituted by the term $\alpha{\cdot}\beta$), we have to change the path valuation $V_\mathcal{P}$ every time (Pre) is applied. It is important to notice that this control is only used for proof purposes and not part of the algorithm. For the algorithm it suffices to encounter *all* pre-solved ϕ-derivatives.

To understand this control, we will compare derivations in our syntax to derivations in standard feature logic. Recall that we have a two-level interpretation. A constraint $x\alpha y$ is valid under $V_\mathcal{X}$ and $V_\mathcal{P}$ if $xV_\mathcal{P}(\alpha)y$ is valid under $V_\mathcal{X}$. Hence, for each clause ϕ and each valuation $V_\mathcal{X}, V_\mathcal{P}$ with ϕ valid under $V_\mathcal{X}$ and $V_\mathcal{P}$ there is a clause $\phi_{V_\mathcal{P}}$ in standard feature logic syntax (not containing functional uncertainty) such that $\phi_{V_\mathcal{P}}$ is valid under $V_\mathcal{X}$. E.g. for the clause $\{x\alpha x,\ \alpha \dot{\in} f,\ x\beta z,\ \beta \dot{\in} f^*g\}$ and a path valuation $V_\mathcal{P}$ with $V_\mathcal{P}(\alpha) = f$ and $V_\mathcal{P}(\beta) = g$ the clause $\phi_{V_\mathcal{P}}$ is $\{xfx, xgy\}$. The control we have mentioned requires (by and large) that only those rewrite rules will be applied, that are compatible to the clause $\phi_{V_\mathcal{P}}$ and thus preserve $V_\mathcal{X}$. If one of the rules (Eq) or (Pre) is applied, we also have to rewrite $\phi_{V_\mathcal{P}}$. Taking the above example, we are only allowed to add $\alpha \mathbin{\dot{\amalg}} \beta$ to ϕ (using (PathRel)), since $\phi_{V_\mathcal{P}}$ is already in pre-solved form.

Now let's vary the example and let $V_\mathcal{P}$ be a path valuation with $V_\mathcal{P}(\alpha) = f$ and $V_\mathcal{P}(\beta) = ffg$. Then we have to add $\alpha \mathrel{\dot{\prec}} \beta$ in the first step, since this relation holds between α and β. The next step is to apply (Pre) on $\alpha \mathrel{\dot{\prec}} \beta$. Here we have to rewrite both ϕ and $\phi_{V_\mathcal{P}}$. Hence, the new clauses ϕ' and $\phi'_{V_\mathcal{P}}$ are $\{x\alpha x,\ \alpha \dot{\in} f,\ x\beta z,\ \alpha{\cdot}\beta \dot{\in} f^*g\}$ and $\{x\,f\,x, x\,fg\,y\}$ respectively. Note that the constraint $x\,ffg\,y$ has been reduced to $x\,fg\,y$ by the application of (Pre). Since infinite derivations must infinitely often use (Pre), this control guarantees that we find a pre-solved clause that has $V_\mathcal{X}$ as a solution.

5 The Second Phase

In the second phase we have to check consistency of pre-solved clauses. As we have mentioned, a pre-solved clause is consistent if we find some appropriate path valuation. This means that we have to check the consistency of divergence constraints of the form $\alpha_1 \mathbin{\dot{\amalg}} \alpha_2$ together with path restrictions

$\alpha_1 \dot\in L_1$ and $\alpha_2 \dot\in L_2$. A constraint $\alpha_1 \,\dot{\|}\, \alpha_2$ is valid under some valuation $V_{\mathcal{P}}$ if there are (possibly empty) words w, w_1, w_2 and features $f \neq g$ such that $V_{\mathcal{P}}(\alpha_1) = wfw_1$ and $V_{\mathcal{P}}(\alpha_2) = wgw_2$. This definition could directly be used for a rewrite rule that solves a single divergence constraint, which gives us

$$\text{(Solv)}\ \frac{\{\alpha_1 \,\dot{\|}\, \alpha_2\} \cup \psi}{\{\alpha_1' \dot\in f\mathcal{F}^*, \alpha_2' \dot\in g\mathcal{F}^*\} \cup \psi'}\quad f \neq g,\ \beta, \alpha_{1,2} \text{ new}$$

where $\psi' = \psi[\alpha_1 \leftarrow \beta{\cdot}\alpha_1', \alpha_2 \leftarrow \beta{\cdot}\alpha_2']$. By the application of this rule we will get constraints of the form $\beta{\cdot}\alpha_1' \dot\in L_1$ and $\beta{\cdot}\alpha_2' \dot\in L_2$. Decomposing these restriction constraints and joining the corresponding path restrictions for β and $\alpha_{1,2}'$ will result in $\{\beta \dot\in (P_1 \cap P_2),\ \alpha_1' \dot\in (f\mathcal{F}^* \cap S_1),\ \alpha_2' \dot\in (g\mathcal{F}^* \cap S_2)\}$ with $P_1{\cdot}S_1 \subseteq L_1$ and $P_2{\cdot}S_2 \subseteq L_2$, which completes the consistency check.

Additionally, one has to consider the effects of introducing the path terms $\beta{\cdot}\alpha_i'$. The main part of this task is to resolve constraints of the form $\beta{\cdot}\alpha_i' \,\dot{\|}\, \alpha$. There are two possibilities: Either α has also β as an prefix, in which case we have to add $\beta \dot\prec \alpha$; or β is not a prefix of α, which means that we have to add $\alpha \,\dot{\|}\, \beta$. After doing this, the introduced prefix constraints have to be evaluated using (Pre). (In the appendix we present a solution which is more appropriate for proofing termination).

6 Kaplan and Maxwell's Method

We are now able to compare our method with the one used by Kaplan and Maxwell. In our method, the non-deterministic addition of path relation and the evaluation of these relations are done at different times. The evaluation of the introduced constraints $\alpha \doteq \beta$ and $\alpha \dot\prec \beta$ are done after (PathRel) in the first phase of the algorithm, whereas the evaluation of the divergence constraints is done in a separate second phase.

In Kaplan and Maxwell's algorithm all these steps are combined into one single rule. Roughly, they substitute a clause $\{xL_1y, xL_2z,\} \cup \psi$ non-deterministicly by one of the following clauses:[2]

$$\begin{array}{ll}
\{\ x\,(L_1 \cap L_2)\,y,\ x \doteq y\ \} \cup \psi & \\
\{\ x\,(L_1 \cap P)\,y,\ y\,S\,z\ \} \cup \psi & P{\cdot}S \subseteq L_2 \\
\{\ x\,(L_2 \cap P)\,z,\ z\,S\,y\ \} \cup \psi & P{\cdot}S \subseteq L_1 \\
\{\ x\,(P_1 \cap P_2)\,u,\ u\,(f{\cdot}S_1)\,y,\ u\,(g{\cdot}S_2)\,z\ \} \cup \psi & \text{with} \\
\qquad P_1{\cdot}f{\cdot}S_1 \subseteq L_1,\ P_2{\cdot}g{\cdot}S_2 \subseteq L_2,\ f \neq g,\ u \text{ new} &
\end{array}$$

Recall that $\{xL_1y, xL_2z\}$ is expressed in our syntax by the clause $\gamma = \{x\alpha y, \alpha \dot\in L_1, x\beta z, \beta \dot\in L_2\}$, which is the example we have used on page . The first three cases correspond exactly to the result of the derivations that have been described for γ_2, γ_3 and γ_4. By and large, the last case is achieved if we first add $\alpha \,\dot{\|}\, \beta$ to γ and then turn over to the second phase as described in the last section.

The problem with Kaplan/Maxwell's algorithm is that one has to introduce a new variable u in the last case, since there is no other possibility to express divergence. If their rule system is applied to a cyclic description, it will not terminate as the last part introduces new variables. Hence it cannot be used for an algorithm in case of cyclic descriptions.

The delaying of the evaluation of divergence constraint may not only be useful when applied to cyclic feature descriptions. As Kaplan and Maxwell pointed out, it is in general useful to postpone the consistency check for functional uncertainty. With the algorithm we have described it is also possible to delay single parts of the evaluation of constraints containing functional uncertainty.

[2] This is not the way their algorithm was originally described in [5] as they use a slightly different syntax. Furthermore, they don't use non-deterministic rules, but use a single rule that produces a disjunction. However, the way we describe their method seems to be more appropriate in comparing both approaches.

Appendix

Proof of Lemma 4.2. The first claim is easy to prove. For the second claim let $\{L_1, \ldots, L_n\} \subseteq P(\mathcal{F}^+)$ be the set of regular languages used in ϕ and let $\mathcal{A}_i = (Q_{\mathcal{A}_i}, i_{\mathcal{A}_i}, \sigma_{\mathcal{A}_i}, Fin_{\mathcal{A}_i})$ be finite, deterministic automatons such that $\mathcal{A}_i$ recognizes L_i. For each $\mathcal{A}_i$ we define $\mathrm{dec}(\mathcal{A}_i)$ to be the set

$$\mathrm{dec}(\mathcal{A}_i) = \{\overline{L_p^q} \mid p, q \in Q_{\mathcal{A}_i}\},$$

where $\overline{L_p^q} = \{w \in \mathcal{F}^+ \mid \sigma^*_{\mathcal{A}_i}(p, w) = q\}$. It is easy to show that $\mathrm{dec}(\mathcal{A}_i)$ is a set of regular languages that contains L_i and is closed under decomposition. Hence, the set $\Lambda_0 = \bigcup_{i=1}^n \mathrm{dec}\,(\mathcal{A}_i)$ contains each L_i and is closed under decomposition. Let $\Lambda = \mathrm{fi}\,(\Lambda_0)$ 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.

We will prove that Λ is also closed under decomposition. Given some $L \in \Lambda$ and a word $w = w_1w_2 \in L$, we have to find an appropriate decomposition P, S in Λ. Since each L in Λ can be written as a finite intersection $L = \bigcap_{k=1}^m L_{i_k}$ where L_{i_k} is in Λ_0, we know that $w = w_1w_2$ is in L_{i_k} for $1..m$. As Λ_0 is closed under decomposition, there are languages P_{i_k} and S_{i_k} for $k = 1..m$ with $w_1 \in P_{i_k}$, $w_2 \in S_{i_k}$ and $P_{i_k}{\cdot}S_{i_k} \subseteq L_{i_k}$. Let $P = \bigcap_{k=1}^m P_{i_k}$ and $S = \bigcap_{k=1}^m S_{i_k}$. 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_1w_2. $\square$

A.1 Phase I: Soundness, Completeness and Quasi-Termination

Proposition A.1 *The rule* (PathRel) *is $\mathcal{X} \cup \mathcal{V}$-sound and globally $\mathcal{X} \cup \mathcal{V}$-preserving. If Λ is closed under decomposition, then* (LangDec$_\Lambda$) *is $\mathcal{X} \cup \mathcal{V}$-sound and globally $\mathcal{X} \cup \mathcal{V}$-preserving. The* (Pre) *rule is $\mathcal{X}$-sound and $\mathcal{X}$-preserving. All other rules are $\mathcal{X} \cup \mathcal{V}$-sound and $\mathcal{X} \cup \mathcal{V}$-preserving.*

Next we will prove some syntactic properties of the clauses derivable by the rule system. For the rest of the paper we will call clauses that are derivable from prime clauses *admissible*.

Proposition A.2 *Every admissible clause is basic. If $\alpha \dot{\prec} \beta$, $\alpha \doteq \beta$ or $\alpha \,\dot{\|}\, \beta$ 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.*

Note that (by this proposition) (Pre) (resp. (Eq)) can always be applied if a constraint $\alpha \dot{\prec} \beta$ (resp. $\alpha \doteq \beta$) 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 rule to some prefix or path equality constraint will not change any other prefix or path equality constraint contained in the same clause. This is a direct consequence of the way (PathRel) was defined.

Lemma A.3 *Given two admissible clauses γ, γ' with $\gamma \to_r \gamma'$ and r different from* (PathRel). *Then $\alpha \doteq \beta \in \gamma'$ (resp. $\alpha \dot{\prec} \beta \in \gamma'$) implies $\alpha \doteq \beta \in \gamma$ (resp. $\alpha \dot{\prec} \beta \in \gamma$). Furthermore, if $\alpha{\cdot}\beta$ is contained in γ', then either $\alpha{\cdot}\beta$ or $\alpha \dot{\prec} \beta$ is contained in γ.*

Note that this lemma implies that new path equality or prefix constraints are only introduced by (PathRel). We can derive from this lemma some syntactic properties of admissible clauses which are needed for proving completeness and quasi-termination.

Lemma A.4 *If ϕ is an admissible clause, then*

1. *If $\alpha \dot{\prec} \beta$ is contained in ϕ, then there is no other prefix or equality constraint in ϕ involving β. Furthermore, neither $\beta{\cdot}\beta'$ nor $\beta'{\cdot}\beta$ is contained in ϕ.*
2. *if $\alpha{\cdot}\beta \,\dot{\|}\, \beta'$ is in ϕ, then either β' equals α or ϕ contains a constraint of form $\alpha \,\dot{\|}\, \beta'$, $\alpha \doteq \beta'$ or $\alpha \dot{\prec} \beta'$.*

The first property will guarantee that concatenation does not occur in prefix or equality constraints and that the length of path concatenation is restricted to 2. The second property ensures that a constraint $\alpha{\cdot}\beta \,\dot{\|}\, \beta'$ is always reducible.

Theorem A.5 *For every finite Λ the rule system $\mathcal{R}_\Lambda$ is quasi-terminating.*

Proof. The rule system produces only finitely many different clauses since the rules introduce no additional variables or sort symbols and the set of used languages is finite. Additionally, the length of concatenation is restricted to 2. □

Lemma A.6 *There are no infinite derivations using only finitely many instances of* (Pre).

Since the rule system is quasi-terminating, the completeness proof consists of two parts. In the first part we will proof that pre-solved clauses are just the irreducible clauses. In the second part we will show that one finds for each solution $V_\mathcal{X}$ of a prime clause ϕ a pre-solved ϕ-derivative γ such that $V_\mathcal{X}$ is also a solution of γ.

Theorem A.7 (Completeness I) *Given an admissible clause $\phi \neq \bot$ such that ϕ is not in pre-solved form. If Λ is ϕ-closed and closed under decomposition, then ϕ is $\mathcal{R}_\Lambda$-reducible.*

Theorem A.8 (Completeness II) *For every prime clause ϕ and for every Λ that is ϕ-closed, closed under decomposition and intersection we have*

$$[\![\phi]\!]^\mathcal{I} \subseteq \bigcup_{\gamma \in \text{pre-solved}(\phi, R_\Lambda)} [\![\gamma]\!]^\mathcal{I}$$

where pre-solved(ϕ, R_Λ) *is the set of pre-solved (ϕ, R_Λ)-derivatives.*

Proof (Sketch) We have to show, that for each prime clause ϕ and each $V_\mathcal{X}, V_\mathcal{P}, \mathcal{I}$ with $(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} \phi$ there is a pre-solved $(\phi, \mathcal{R}_\Lambda)$-derivative γ such that $V_\mathcal{X} \in [\![\gamma]\!]^\mathcal{I}$. We will do this by controlling derivation using the valuation $(V_\mathcal{X}, V_\mathcal{P})$. The control will guarantee finiteness of derivations and will maintain the first completeness property, namely that the irreducible clauses are exactly the pre-solved clauses.

We allow only those instances of the non-deterministic rules (PathRel) and (LangDec$_\Lambda$), which preserve exactly the valuation $(V_\mathcal{X}, V_\mathcal{P})$. That means if $(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} \phi$ and $\phi \to_r \gamma$ for one of these rules, then $(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} \gamma$ must hold. Note that the control depends only on $V_\mathcal{P}$. E.g. for the clause $\phi = \{x\alpha y, \alpha \dot{\in} L_1, x\beta z, \beta \dot{\in} L_2\}$ and arbitray $\mathcal{I}$, $V_\mathcal{X}$ this means that if $V_\mathcal{P}(\alpha) = f$, $V_\mathcal{P}(\beta) = g$ and $(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} \phi$, the rule (PathRel) can transform ϕ only into $\{\alpha \,\dot{\|}\, \beta\} \cup \phi$.

If $V_\mathcal{P}$ satisfies $V_\mathcal{P}(\alpha) \not\preceq V_\mathcal{P}(\beta)$ for α different from β with $x\alpha y \in \phi$ and $x\beta z \in \phi$, we cannot add any prefix constraint using this control. Hence, (Pre) cannot be applied, which implies (by lemma A.6) that in this case there is no infinite controlled derivation. We will call such path valuations prefix-free with respect to ϕ.

If $V_\mathcal{P}$ is not prefix-free, then (Pre) will be applied during the derivations. In this case we have to change the path valuation, since (Pre) is not $\mathcal{P}$-preserving. If $(V_\mathcal{X}, V_\mathcal{P}) \models_\mathcal{I} \phi = \{\alpha \dot{\prec} \beta\} \cup \psi$ and we apply (Pre) on $\alpha \dot{\prec} \beta$ yielding γ, then the valuation $V'_\mathcal{P}$ with $V_\mathcal{P}(\beta) = V_\mathcal{P}(\alpha) V'_\mathcal{P}(\beta)$ and $V_\mathcal{P}(\alpha) = V'_\mathcal{P}(\alpha)$ for $\alpha \neq \beta$ will satisfy $(V_\mathcal{X}, V'_\mathcal{P}) \models_\mathcal{I} \gamma$. We will use $V'_\mathcal{P}$ for controlling the further derivations.

If we change the path valuation in this way, there will again be only finite derivations. To see this, note that every time (Pre) is applied and the path valuation is changed, the valuation of one variable is shortened by a non-empty path. As the number of variables used in clauses does not increase, this shortening can only be done finitely many times. This implies, that (Pre) can only finitely often be applied under this control. Hence (by lemma A.6), there are again only finite controlled derivations. □

A.2 Consistency of Pre-Solved Clauses

We will first do 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 $u' \parallel_0 v'$.

We will reformulate the reduction of divergence constraints in order to avoid constraints of form $\alpha{\cdot}\beta \mathrel{\dot{\parallel}} \beta'$. Handling such constraints would make the termination proof somewhat complicated. For the reformulation we use a special property of pre-solved clauses, namely that $\alpha \mathrel{\dot{\parallel}} \beta$ is in a pre-solved clause ϕ iff $x\alpha y$ and $x\beta z$ is in ϕ. Hence, if $\alpha \mathrel{\dot{\parallel}} \beta$ and $\beta \mathrel{\dot{\parallel}} \delta$ is in ϕ, then $\alpha \mathrel{\dot{\parallel}} \delta$ is also in ϕ. This implies, that we can write ϕ_P as $\dot{\parallel}(A_1) \uplus \ldots \uplus \dot{\parallel}(A_n) \uplus \psi$, where $\dot{\parallel}(A)$ is a syntactic sugar for

$$\dot{\parallel}(A) = \{\alpha \mathrel{\dot{\parallel}} \alpha' \mid \alpha \neq \alpha' \wedge \alpha, \alpha' \in A\},$$

$A_1, \ldots, A_n$ are disjoint sets of path variables and ψ does not contain divergence constraints. Note that for every $A_i = \{\alpha_1, \ldots, \alpha_n\}$ there are variables $x, y_1, \ldots, y_n$ such that $\{x\alpha_1 y_1, \ldots, x\alpha_n y_n\} \subseteq \phi$. Now given such that a constraint $\dot{\parallel}(A)$, we assume that a whole set of path variables $A_1 \subseteq A$ diverges with the same prefix β. That means we can replace $\dot{\parallel}(A_1) \subseteq \dot{\parallel}(A)$ by

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

where β is new, $A_1' = \{\alpha_1', \ldots, \alpha_n'\}$ is a disjoint copy of $A_1 = \{\alpha_1, \ldots, \alpha_n\}$ and $A \doteq \beta{\cdot}A_1'$ is an abbreviation for the clause $\{\alpha_1 \doteq \beta{\cdot}\alpha_1', \ldots, \alpha_n \doteq \beta{\cdot}\alpha_n'\}$. $\dot{\parallel}_0(A)$ is defined similar to $\dot{\parallel}(A)$. Assuming additionally that the common prefix β is maximal implies that $\beta \mathrel{\dot{\parallel}} \alpha$ holds for $\alpha \in (A - A_1)$. If we also consider the effects of $A_1 = \beta{\cdot}A_1'$ on the subterm agreements in ϕ that involves variables of A_1, then we result in the following rule:

$$(\mathrm{Red}_1)\quad \frac{xA_1Y_1 \cup \dot{\parallel}(A) \cup \psi}{\{x\beta z\} \cup zA_1'Y_1 \cup \dot{\parallel}_0(A_1') \cup \dot{\parallel}(\{\beta\} \cup A_2) \cup \psi'}$$

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 $\{x\alpha_1 y_1, \ldots, x\alpha_n y_n\}$. 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 $\alpha{\cdot}\beta \mathrel{\dot{\parallel}} \beta'$. The rules

$$(\mathrm{Red}_2)\qquad \frac{\dot{\parallel}(A) \cup \psi}{\dot{\parallel}_0(A) \cup \psi}$$

$$(\mathrm{Solv})\quad \frac{\dot{\parallel}_0(A) \cup \psi}{\{\alpha \in f_\alpha{\cdot}F^* \mid \alpha \in A\} \ \cup\ \psi}\ f_\alpha \neq f_{\alpha'} \text{ for } \alpha \neq \alpha'$$

together with the rules ($\mathrm{LangDec}_\Lambda$), (Join) and (Empty) completes the rule system $\mathcal{R}_\Lambda^{\mathrm{Solv}}$. ($\mathrm{Red}_2$) is needed as path variables always denote non-empty paths. We will view (Red_1) and (Red_2) as one single rule (Reduce).

A clause ϕ is said to be *solved* if (1) $\alpha{\cdot}\beta \mathrel{\dot{\in}} L$ and $\alpha \mathrel{\dot{\in}} \emptyset$ is not in ϕ_P; (2) $\alpha \mathrel{\dot{\in}} L_1$ in ϕ_P and $\alpha \mathrel{\dot{\in}} L_2$ in ϕ_P implies $L_1 = L_2$; (3) ϕ does not contain constraints of form $\alpha \mathrel{\dot{\parallel}} \beta$, $\alpha \mathrel{\dot{\parallel}_0} \beta$, $\alpha \mathrel{\dot{\prec}} \beta$, or $\alpha \doteq \beta$; and (4) for every $\{x\alpha y, x\beta z\} \subseteq \phi$ with $\alpha \neq \beta$ there are features $f \neq g$ with $\{\alpha \mathrel{\dot{\in}} fL_1, \beta \mathrel{\dot{\in}} gL_2\} \subseteq \phi$. It is easy to see that every solved clause is consistent. Note that every solved clause is also prime.

Lemma A.9 *The rules* (Reduce) $=$ (Red_1) $+$ (Red_2) *and* (Solv) *are $\mathcal{X}$-sound and globally $\mathcal{X}$-preserving. Furthermore, $\mathcal{R}_\Lambda^{\mathrm{Solv}}$ is terminating.*

Lemma A.10 *Let ϕ be a pre-solved clause. If Λ is ϕ-closed, closed under intersection and decomposition, then a $(\phi, \mathcal{R}_\Lambda^{\mathrm{Solv}})$-derivative different from $\perp$ is irreducible if and only if it is solved.*

Finally we can combine both phases of the algorithm.

Theorem A.11 *Consistency of prime clauses is decidable.*

References

[1] F. Baader, H.-J. Bürckert, B. Nebel, W. Nutt, and G. Smolka. On the expressivity of feature logics with negation, functional uncertainity, and sort equations. Research Report RR-91-01, DFKI, 1991.

[2] R. Backofen. Regular path expressions in feature logic. Research Report RR-93-17, DFKI, 1993.

[3] R. Backofen and G. Smolka. A complete and recursive feature theory. In *Proc. of the 31 th ACL*, 1993. this volume.

[4] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

[5] R. M. Kaplan and J. T. Maxwell III. An algorithm for functional uncertainty. In *Proc. of the 12th COLING*, pages 297–302, Budapest, Hungary, 1988.

[6] R. M. Kaplan and A. Zaenen. Functional uncertainty and functional precedence in continental west germanic. In H. Trost, editor, *4. Österreichische Artificial-Intelligence-Tagung: Wiener Workshop - Wissensbasierte Sprachverarbeitung*, pages 114–123. Springer, Berlin, Heidelberg, 1988.

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

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

[9] G. Smolka. A feature logic with subsorts. LILOG-Report 33, IBM Deutschland, Stuttgart, 1988.

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

[11] G. Smolka and R. Treinen. Records for logic programming. In *Proceedings of the 1992 Joint International Conference and Symposium on Logic Programming*, pages 240–254, Washington, DC, 1992.