Controlling Functional Uncertainty

Rolf Backofen[1]

Abstract. There have been two different methods for checking the satisfiability of feature descriptions that use the functional uncertainty device, namely [3] and [2]. Although only the one in [2] solves the satisfiability problem completely, both methods have their merits. But it may happen that in one single description, there are parts where the first method is more appropriate, and other parts where the second should be applied. In this paper, we present a common framework that allows one to combine both methods. This is done by presenting a set of rules for simplifying feature descriptions. The different methods are described as different controls on this rule set, where a control specifies in which order the different rules must be applied.

1 Introduction

This paper is concerned with an extension to feature descriptions, which has been introduced as "functional uncertainty" in [5, 3]. This formal device plays an important role in the framework of LFG in modeling so-called long distance dependencies and constituent coordination. For a detailed linguistic motivation see [5, 4, 3, 6]. Functional uncertainty consists of constraints of the form xLy, where L is regular expression. xLy is interpreted as $\bigvee\{xwy \mid w \in L\}$. Since this disjunction may be infinite, functional uncertainty gives additional expressivity. Let us recall an example from [3] and consider the topicalized sentence *Mary John telephoned yesterday.* Using s as a variable denoting the whole sentence, the LFG-like clause $s\ topic\ x \wedge s\ obj\ x$ 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. Using functional uncertainty, it is possible to have a finite presentation of this infinite specification, namely the clause $s\ topic\ x \wedge s\ comp^*\ obj\ x$.

It was shown in [3] that consistency of feature descriptions is decidable, provided that a certain acyclicity condition is met. More recently, [2] has shown that the satisfiability problem is decidable without additional conditions. Both algorithms have their merits. The one in [2] solves the satisfiability problem using an extended syntax, which makes it possible to avoid the computational explosion that causes the undecidability in the cyclic case. But there are cases where the additional syntax causes some overhead. In these cases, one would like to switch to the method used in [3], where this overhead is avoided. On the other hand, the algorithm in [3], which is used in the implementation of the LFG system, cannot be extended to the cyclic case.

In this paper, we present a new algorithm that allows one to combine both methods under a common framework. We use the extended syntax as proposed in [2] and present a new set of rewrite rules. The different methods used in [2] and [3] can then be described as different control on this rule set, where a control specifies the order of rule application. Thus, it is now possible to compare both algorithms and their effects. In [2], this was not possible since the set of rules presented there was tailored for the purpose of proving decidability. As an extension, we present a control which allows the flexibility to switch between both methods. This flexibility is needed since none of the methods is optimal for all parts of a clause. Which one is best depends on the regular languages used in the corresponding part.

In Section 2, we present some needed preliminaries. In Section 3, we introduce the input clauses and two different output clauses of our algorithm. In Section 4 and 5 we present the rule system and some of its basic properties. Equipped with these tools, we turn to the most interesting part in Section 6, where we define three different controls for the given set of rules and compare their properties.

2 Preliminaries

Our *signature* consists of a set of *sorts* $\mathcal{S}$ $(A, B, \ldots)$, *first-order variables* $\mathcal{X}$ $(x, y, \ldots)$, *path variables* $\mathcal{P}$ $(\mu, \nu, \ldots)$, and *features* F $(f, g, \ldots)$. We 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 is a common, possibly empty prefix w of u, v and paths w_1, w_2 such that $u = wfw_1 \wedge v = wgw_2$ Clearly, $\amalg$ is a symmetric relation. Furthermore, for any pair of paths u and v, then exactly one of the relations $u = v$, $u \prec v$, $u \succ v$, or $u \amalg v$ holds.

A *simple path term* $(s, t, \ldots)$ is either a feature or a path variable. A *path term* $(p, q, \ldots)$ is either a simple path term

[1] Deutsches Forschungszentrum für Künstliche Intelligenz GmbH.

ECAI 96. *12th European Conference on Artificial Intelligence*
Edited by W. Wahlster
Published in 1996 by John Wiley & Sons, Ltd.

or a concatenation of two path terms $p{\circ}q$ (called a *complex path term*). The set of constraints is given by

Ax	*sort restriction*	$p \dot{\amalg} q$	*divergence*[2]
$x \doteq y$	*agreement*	$p \dot{\prec} q$	*prefix*
$x[p]y$	*subterm agreement*	$p \doteq q$	*path equality*
		$p \dot{\in} L$	*path restriction*

We exclude empty paths in subterm agreement since $x\epsilon y$ is equivalent to $x \doteq y$, and use $p \dot{\succ} q$ as a synonym for $q \dot{\prec} p$. A *clause* is a finite set of constraints denoting their conjunction.

An *interpretation* $\mathcal{I}$ is a standard first-order structure, where every feature $f \in \mathrm{F}$ 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 $(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}})$, where $\alpha_{\mathcal{X}}$ is a standard first-order valuation of the variables in X and $\alpha_{\mathcal{P}}$ is a function $\alpha_{\mathcal{P}} : \mathcal{P} \to \mathrm{F}^+$. We define $\alpha_{\mathcal{P}}(f) = f$ for every feature $f \in \mathrm{F}$, and $\alpha_{\mathcal{P}}(p{\circ}q)$ to be the path $\alpha_{\mathcal{P}}(p)\alpha_{\mathcal{P}}(q)$. Validity for sort restrictions and agreement constraints is defined as usual. The other constraints are valid in an interpretation $\mathcal{I}$ under a valuation $(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}})$ iff

$$
\begin{aligned}
(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}}) \models_{\mathcal{I}} x[p]y \;&:\Longleftrightarrow\; \alpha(p) = f_1 \cdots f_n \text{ and} \\
&\qquad (\alpha_{\mathcal{X}}(x), \alpha_{\mathcal{X}}(y)) \in F_1^{\mathcal{I}} \bullet \ldots \bullet F_n^{\mathcal{I}} \\
(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}}) \models_{\mathcal{I}} p \dot{\in} L \;&:\Longleftrightarrow\; \alpha_{\mathcal{P}}(p) \in L \\
(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}}) \models_{\mathcal{I}} p \,\dot{\diamond}\, q \;&:\Longleftrightarrow\; \alpha_{\mathcal{P}}(p) \diamond \alpha_{\mathcal{P}}(q) \text{ for } \dot{\diamond} \in \{\dot{\amalg}, \dot{\prec}, \doteq\}.
\end{aligned}
$$

where $\bullet$ denotes binary concatenation of relations. Note that the validity of a path constraint depends only on the path valuation. The set of all $\mathcal{X}$-solutions of a clause ϕ in some interpretation $\mathcal{I}$ is the set of all valuations $\alpha_{\mathcal{X}}$ such that there is a path valuation $\alpha_{\mathcal{P}}$ with $(\alpha_{\mathcal{X}}, \alpha_{\mathcal{P}}) \models_{\mathcal{I}} \phi$.

3 Prime, Pre-Solved, and Solved Clauses

In this section, we define the input and output clauses for both phases of the algorithm. In the following, we consider only those clauses ϕ such that for every distinct pair of variables x, y, $x \doteq y$ is in ϕ if and only if $x \neq y$ and x occurs only once in ϕ. A clause ϕ is called *prime* iff

Pr1. every path term in ϕ is simple,

Pr2. for every path variable μ used in ϕ there is *at most* one constraint $x[\mu]y \in \phi$, and

Pr3. ϕ has no constraints of the forms $s \dot{\amalg} t$, $s \dot{\prec} t$, or $s \doteq t$.

Kaplan/Maxwell [3] formulated the satisfiability problem for functional uncertainty in an unsorted syntax. Essentially, this syntax consists of the atomic constraints Ax, $x\,f\,y$ and $x \doteq y$ together with the additional constraint xLy. Constraints of this form are interpreted as $xLy = \bigvee\{xwy \mid w \in L\}$. A clause ϕ in Kaplan/Maxwell Syntax can be translated into an clause in our syntax by replacing every constraint xLy by $x[\mu]y \wedge \mu \dot{\in} L$, where μ is a new variable. The resulting clause will have the same $\mathcal{X}$-solutions. The resulting clauses are prime clauses and hence our input clauses. A clause is called *simplified* iff

Si1. $Ax \in \phi$ and $Bx \in \phi$ implies $A = B$,

Si2. $p \dot{\in} L \in \phi$ and $p \dot{\in} L' \in \phi$ implies $L = L'$,

Si3. $p \dot{\in} \emptyset$ is not in ϕ,

Si4. $f \dot{\in} L$ implies that f is an element of denotation of L,

Si5. $x[f]y \in \phi$ and $x[f]z \in \phi$ implies $y = z$,

Si6. ϕ contains no constraint of the form $s \doteq t$ or $s \dot{\prec} t$,

Si7. every path term in ϕ is simple.

A simplified clause is called *pre-solved* iff

Ps1. $s \dot{\amalg} t \in \phi$ if and only $s \neq t$, either s or t is a path variable, and there is an x such that $\{x[s]y, x[t]z\} \subseteq \phi$.

Pre-solved clauses are not consistent per se, since it might be that a divergence constraint contradicts some of the path restrictions. E.g., the pre-solved clause $x[\mu]y \wedge x[\nu]z \wedge \mu \dot{\amalg} \nu \wedge \mu \dot{\in} f^+ \wedge \nu \dot{\in} (ff)^+$ is inconsistent. A clause ϕ is called *solved* if it is either $\perp$, or it is simplified and satisfies

So1. ϕ contains no constraint of form $p \dot{\amalg} q$, and

So2. if $x[\mu]y$ is in ϕ, then there is no $x[s]z$ with $s \neq \mu$ in ϕ.

Lemma 1 *Let ϕ be a pre-solved clause different from $\perp$. Then ϕ is satisfiable iff there is a path valuation $\alpha_{\mathcal{P}}$ with $\alpha_{\mathcal{P}} \models \phi_p$, where ϕ_p is the set of constraints in ϕ of the forms $s \dot{\amalg} t$ or $s \dot{\in} \phi$. Furthermore, every solved clause different from $\perp$ is satisfiable.*

4 Simplification Rules

The first set of rules, $\mathcal{R}^{\mathrm{DFun}}_{\mathrm{simpl}}$, is displayed in Figure 1 and allows one to simplify a clause satisfying certain restrictions that will be captured under the notion of a admissible clause. Most of the rules are deterministic, i.e., replacing a clause with the result of applying one of these rules yields a clause having the same $\mathcal{X}$-solutions. The rules (RelD) and ($\mathrm{Dec}_{\mathrm{DFun}}$) are non-deterministic rules, which implies that we have to replace a clause by the disjunction of all possible applications of the corresponding rule. Thus, applying (RelD) to a clause of the form $\mu{\circ}\mu \dot{\amalg} \nu \wedge \ldots$ yields the disjunction

$$(\mu{\circ}\mu' \dot{\amalg} \nu \wedge \mu \dot{\amalg} \nu \wedge \ldots) \;\vee\; (\mu{\circ}\mu' \dot{\amalg} \nu \wedge \mu \dot{\prec} \nu \wedge \ldots)$$

The rule set is indexed by the decomposition function DFun used in ($\mathrm{Dec}_{\mathrm{DFun}}$). The simplest version of DFun just decomposes a regular language L into a set of pairs (P, S) with the property that there is a state q in the minimal automaton $\mathcal{A}$ for L with $P = \{w \neq \epsilon \mid \delta_{\mathcal{A}}(q_{\mathrm{in}}, w) = q\}$ and $S = \{w \neq \epsilon \mid \delta_{\mathcal{A}}(q, w) \in \mathrm{Fin}_{\mathcal{A}}\}$. Here, q_{in} is the initial state, $\mathrm{Fin}_{\mathcal{A}}$ is the set of final states and $\delta_{\mathcal{A}}$ the transition function of $\mathcal{A}$. This decomposition function is sufficient for the case of non-cyclic clauses. For cyclic clauses, we have to use a different decomposition function (as will explained later). In any case, in order to preserve all solutions of a clause the decomposition function has to satisfy

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

The simplification does not handle arbitrary clauses. E.g., we handle only those prefix and equality constraints $s \dot{\prec} t$ and

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

$$\text{(Join)}\quad \frac{p\,\dot{\in}\,L \wedge p\,\dot{\in}\,L' \wedge \psi}{p\,\dot{\in}\,(L\cap L') \wedge \psi}\quad L\neq L' \qquad \text{(Empty)}\quad \frac{p\,\dot{\in}\,\emptyset \wedge \psi}{\bot}$$

$$\text{(FClash)}\quad \frac{f\,\dot{\in}\,L \wedge \psi}{\bot}\quad f\notin L \qquad \text{(SClash)}\quad \frac{Ax \wedge Bx \wedge \psi}{\bot}\quad A\neq B$$

$$\text{(DClash1)}\quad \frac{s\,\dot{\amalg}\,s \wedge \psi}{\bot} \qquad \text{(DClash2)}\quad \frac{s{\circ}\mu\,\dot{\amalg}\,s \wedge \psi}{\bot}$$

$$\text{(Div1)}\quad \frac{s{\circ}\mu\,\dot{\amalg}\,s{\circ}\nu \wedge \psi}{\mu\,\dot{\amalg}\,\nu \wedge \psi} \qquad \text{(Div2)}\quad \frac{s{\circ}\mu\,\dot{\amalg}\,\nu \wedge s\,\dot{\amalg}\,\nu \wedge \psi}{s\,\dot{\amalg}\,\nu \wedge \psi}$$

$$\text{(DivInst)}\quad \frac{s{\circ}\mu\,\dot{\amalg}\,g \wedge \psi}{s\,\dot{\amalg}\,g \wedge \psi} \qquad \text{(Triv1)}\quad \frac{f\,\dot{\amalg}\,g \wedge \psi}{\psi}\quad f\neq g$$

$$\text{(RelD)}\quad \frac{s{\circ}\mu\,\dot{\amalg}\,\nu \wedge \psi}{s\{\dot{\amalg},\dot{\prec}\}\nu \wedge s{\circ}\mu\,\dot{\amalg}\,\nu \wedge \psi}\ \text{if } s,\nu \text{ are unrelated} \qquad \text{(Triv2)}\quad \frac{f{\circ}s\,\dot{\amalg}\,g{\circ}t \wedge \psi}{\psi}\quad f\neq g$$

$$\text{(Eq1)}\quad \frac{x[f]y \wedge x[f]z \wedge \psi}{z \doteq y \wedge x[f]y \wedge \psi[z\leftarrow y]}$$

$$\text{(Eq2)}\quad \frac{\mu \doteq s \wedge x[s]y \wedge x[\mu]z \wedge \psi}{z \doteq y \wedge x[s]y \wedge \psi[\mu\leftarrow s, z\leftarrow y]}$$

$$\text{(Pre)}\quad \frac{s\,\dot{\prec}\,\mu \wedge x[s]y \wedge x[\mu]z \wedge \psi}{x[s]y \wedge y[\mu]z \wedge \psi[\mu\leftarrow s{\circ}\mu]}\quad s\neq\mu$$

$$\text{(DecFeat)}\quad \frac{f{\circ}\mu\,\dot{\in}\,L \wedge \psi}{\mu\,\dot{\in}\,f^{-1}L \wedge \psi}$$

$$\text{(DecClash)}\quad \frac{\mu{\circ}\nu\,\dot{\in}\,L \wedge \psi}{\bot}\quad \text{if } \forall w\in L: |w|=1$$

$$\text{(Dec}_{\text{DFun}}\text{)}\quad \frac{\mu{\circ}\nu\,\dot{\in}\,L \wedge \psi}{\mu\,\dot{\in}\,P \wedge \nu\,\dot{\in}\,S \wedge \psi}\quad (P,S)\in \text{DFun}(L)$$

Figure 1. The simplification rules $\mathcal{R}^{\text{DFun}}_{\text{simpl}}$. DFun associates to every regular expression L a set of decompositions (P, S) with $P{\circ}S \subseteq L$

$s \doteq t$ in a clause ϕ with the property that there is a variable x and variables y, z such that $x[s]y$ and $x[t]z$ is in ϕ. Furthermore, the rules cannot reduce divergence constraints of the form $s{\circ}s'\,\dot{\amalg}\,t{\circ}t'$ with $s \neq t$, and the control imposed on our rewrite rules carefully avoid such constraints. The reason is that for decomposing the complex path terms in $s{\circ}s'\,\dot{\amalg}\,t{\circ}t'$, we might be forced to introduce complex path terms that have a length greater than 2, which we must avoid to achieve a quasi-terminating rewrite system.

We now define the restriction imposed on derivable clauses. Given a clause ϕ, we define the *outgoing edges* of a first-order variable x in ϕ as

$$\text{outgoing}_\phi(x) \quad := \{s \mid \text{there is } z \text{ with } x[s]z \in \phi\}$$

We say that a variable x in ϕ is *tagged* if there is a prefix constraint $s\,\dot{\prec}\,\mu$ in ϕ with $\{s,\mu\} \subseteq \text{outgoing}_\phi(x)$. A clause is is called *admissible* if ϕ contains no complex path terms in prefix or path equality constraints and

- Ad1. for every path variable $\mu \in \mathcal{V}_\mathcal{P}(\phi)$, there is *exactly* one constraint $x[\mu]y \in \phi$,
- Ad2. for every path constraint of the forms $s\{\doteq, \dot{\prec}, \dot{\amalg}\}t$ in ϕ, there exists a variable x such that $\{s, t\} \subseteq \text{outgoing}_\phi(x)$,
- Ad3. if ϕ contains a prefix constraint, then ϕ contains no path equality constraint,
- Ad4. if ϕ contains *at most* one tagged variable,
- Ad5. if ϕ contains two different prefix constraints $s\,\dot{\prec}\,\mu$ and $t\,\dot{\prec}\,\nu$, then either $s = t$, or s and t are different features,
- Ad6. ϕ contains no trivial constraints of the form $s\,\dot{\prec}\,s$, $s\,\dot{\prec}\,f$, $f \doteq g$, or $f \doteq f$.

The last condition just lists constraints which either are inconsistent or superfluous. We could also get rid of these constraints using some appropriate rewrite rules, but we think that it is more efficient to avoid these constraints. Note that every prime clause is admissible. A clause is called *basic* if is derivable using $\mathcal{R}^{\text{DFun}}_{\text{simpl}}$ from an admissible ϕ that contains no complex path terms.

Proposition 2 *Every basic clause is admissible.*

The tedious part of the proof of this proposition are the rules (Pre), (Div1), and (RelD), since one has to check whether the new introduced constraints satisfy the conditions Ad2 and Ad5. For this purpose, one has to record exactly all possible effects that the introduction of complex path terms in the (Pre) rule can have on admissible clauses. E.g., it is guaranteed by the definition of (Pre) that if a basic ϕ contains a complex path term $\alpha{\circ}\nu$, then there are variables x, y, z such that $x[\mu]y$ and $y[\nu]z$ are in ϕ. This, together with condition Ad1, implies that if ϕ contains a constraint of the form $\mu{\circ}\nu\,\dot{\amalg}\,\mu{\circ}\nu'$ in ϕ, then there are variables x, y, z, z' such that $\{x[\mu]y, y[\nu]z, y[\nu']z'\} \subseteq \phi$. Hence, we know for the new relation introduced in Div1, the condition Ad2 is satisfied.

Lemma 3 (Termination,Completeness) *A basic clause is irreducible w.r.t. $\mathcal{R}^{\text{DFun}}_{\text{simpl}}$ iff ϕ is simplified. Furthermore, for every admissible clause ϕ there are no infinite derivations starting with ϕ, and ϕ has the same $\mathcal{X}$-solutions as the set of simplified clauses derivable from ϕ.*

Proof (Sketch) We consider only the claim of termination. Here, the (RelD) rule is the most difficult part since it introduces a new relation. All other rules reduces either the number of variables, constraints or complex path terms. To show that (RelD) terminates, it is necessary to know that there is exactly one variable x in ϕ such that for all constraints of the form $s{\circ}\mu\,\dot{\amalg}\,\nu$ in ϕ, both s and ν are in $\text{outgoing}_\phi(x)$. But this is an immediate consequence of the fact that there is at most one tagged variable in ϕ (Condition Ad4) together with Condition Ad2. Hence, (RelD) only adds constraints between unrelated simple terms that are in $\text{outgoing}_\phi(x)$, where x is the tagged variable of ϕ. Since there are only finitely many possible relations, and since both the (Pre) rule and the (Div2) rule do not increase the number of unrelated simple terms in outgoing$_\phi(x)$, (RelD) cannot cause non-termination. This leads to the following termination ordering. Let x be the tagged variable in ϕ, and let $\Theta_{\mathrm{simpl}}(\phi)$ be the quadruple

$$(\quad \#\text{unrelated terms in outgoing}_\phi(x),\ \#\text{constraints}, \ \#\text{complex path terms in } \phi,\ \#\text{variables})$$

Then for every $r \in \mathcal{R}^{\mathrm{DFun}}_{\mathrm{simpl}}$, if ϕ' is the result of applying r to a basic clause ϕ, then $\Theta_{\mathrm{simpl}}(\phi) >_4 \Theta_{\mathrm{simpl}}(\phi')$, where $>_4$ is the lexicographic greater ordering on quadruples. □

5 Generating pre-solved and solved clauses

As we have explained in the introduction, one of the main tools for solving prime clauses is to "guess" the different relations between path variables, and to check this relation for consistency with the rest of the clause afterwards. Clearly, one has to "guess" all possible relations, which implies that the rules for introducing this relation must be non-deterministic. We have already encountered one rule for non-deterministically introducing relations between simple path terms, namely the rule (RelD). The other two rules are listed below and form the rule set $\mathcal{R}_{\mathrm{pre}}$.

$$\text{(Relate1)} \quad \frac{x[\mu]y \wedge x[\nu]z \wedge \psi}{\mu\{\doteq, \dot{\succ}, \dot{\prec}, \dot{\amalg}\}\nu \wedge x[\mu]y \wedge x[\nu]z \wedge \psi}$$

μ, ν unrelated in ψ, (RelD) not applicable

$$\text{(Relate2)} \quad \frac{x[f]y \wedge x[\mu]z \wedge \psi}{f\{\doteq, \dot{\prec}, \dot{\amalg}\}\mu \wedge x[f]y \wedge x[\mu]z \wedge \psi}$$

f, μ unrelated in ψ, (RelD) not applicable

Using the following set of rules $\mathcal{R}_{\mathrm{solve}}$, we can transform a pre-solved clause into an equivalent set of solved clauses.

$$\text{(Inst)} \quad \frac{\mu \,\dot{\amalg}\, f \wedge \psi}{g \,\dot{\prec}\, \mu \wedge \mu \,\dot{\amalg}\, f \wedge \psi} \qquad f \neq g$$

$$\text{(Intro)} \quad \frac{f \,\dot{\prec}\, \mu \wedge x[\mu]y \wedge \phi}{f \,\dot{\prec}\, \mu \wedge x[\mu]y \wedge x[f]y' \wedge \phi} \qquad \text{if } \forall z : x[f]z \notin \phi$$

$$\text{(Solv1)} \quad \frac{\mu \,\dot{\amalg}\, \nu \wedge \psi}{f \,\dot{\prec}\, \mu \wedge g \,\dot{\prec}\, \nu \wedge \mu \,\dot{\amalg}\, \nu \wedge \psi} \qquad f \neq g$$

$$\text{(Solv2)} \quad \frac{\mu \,\dot{\amalg}\, \nu \wedge x[\mu]y \wedge x[\nu]z \wedge \psi}{f \,\dot{\prec}\, \mu \wedge g \,\dot{\prec}\, \nu \wedge \mu \,\dot{\amalg}\, \nu \wedge x[\delta]u \wedge u[\nu]y \wedge u[\nu]y \wedge \psi_{\mathrm{subs}}}$$

where $\psi_{\mathrm{subs}} = \psi[\mu \leftarrow \delta \circ \mu, \nu \leftarrow \delta \circ \nu]$, $f \neq g$ and δ, u are new variables

The two rules (Solv1) and (Solv1) together will be seen as one complex, non-deterministic rule called (Solve). The (Solve) directly expands a divergence constraints into its definition, thus solving a single divergence constraint. The (Solv1) rules reflects the case that two paths diverge with an empty prefix while (Solv2) reflects the case that the common prefix is not empty. Since the valuations always associates non-empty paths to path variables, we have to distinguish these cases. Note that (Intro) is the only deterministic rule, and that all of the other rules are non-deterministic.

Proposition 4 *If a simplified clause is not pre-solved, then one of* (Relate1) *or* (Relate2) *is applicable. Furthermore, a clause is pre-solved if none of the rules in* $\mathcal{R}^{\mathrm{DFun}}_{\mathrm{simpl}} \cup \mathcal{R}_{\mathrm{pre}}$ *is applicable, and solved if none of the rules in* $\mathcal{R}^{\mathrm{DFun}}_{\mathrm{simpl}} \cup \mathcal{R}_{\mathrm{pre}} \cup \mathcal{R}_{\mathrm{solve}}$ *is applicable.*

6 Controlling rule application

In this section, we present different possible controls over the set of rules given by $\mathcal{R}^{\mathrm{DFun}} = \mathcal{R}^{\mathrm{DFun}}_{\mathrm{simpl}} \cup \mathcal{R}_{\mathrm{pre}} \cup \mathcal{R}_{\mathrm{solve}}$. A *control* is a partial order $<^{\mathrm{con}}$ on $\mathcal{R}^{\mathrm{DFun}}$. A derivation $\phi_1 \rightarrow_{r_1} \phi_2 \ldots$ is *licensed* by a control $<^{\mathrm{con}}$ iff for every step $\phi_i \rightarrow_{r_i} \phi_{i+1}$, no rule instance r with $r <^{\mathrm{con}} r_i$ is applicable. We use $<^{\mathrm{con}}$-derivative and $<^{\mathrm{con}}$-derivation in the obvious way.

If we would apply the rules without any control, then not only is termination not guaranteed, but we may even produce a clause that is not admissible. E.g., consider the clause $x[\mu]y \wedge x[\nu]z \wedge x[\nu']z'$. Then applying (Relate1) twice may produce the clause $\mu \,\dot{\prec}\, \nu \wedge \nu \,\dot{\prec}\, \nu' \wedge x[\mu]y \wedge x[\nu]z \wedge x[\nu']z'$, which is not admissible since it does not fulfill condition Ad5. Hence, our minimal control $<^{\mathrm{basic}}$ guarantees that the simplification rules are applied before one of the rules in $\mathcal{R}_{\mathrm{pre}} \cup \mathcal{R}_{\mathrm{solve}}$ are applied, i.e.,

$$\forall r \in \mathcal{R}^{\mathrm{DFun}}_{\mathrm{simpl}}, \forall r' \in \mathcal{R}_{\mathrm{pre}} \cup \mathcal{R}_{\mathrm{solve}} : r <^{\mathrm{basic}} r'.$$

Proposition 5 *If ϕ is derivable with $\mathcal{R}^{\mathrm{DFun}}$ from a prime clause using the control $<^{\mathrm{basic}}$, then ϕ is admissible.*

Proof (Sketch) This follows from the fact that if ϕ is an admissible clause that contains no complex path terms (which prime clauses are), then it is basic and therefore admissible due to Proposition 2. Furthermore, it can be simplified due to Lemma 3. Hence, according to the control $<^{\mathrm{basic}}$, we can apply a rule in $\mathcal{R}_{\mathrm{pre}} \cup \mathcal{R}_{\mathrm{solve}}$ if and only if the corresponding rule is simplified. And it is easy to check that applying a rule $\mathcal{R}_{\mathrm{pre}} \cup \mathcal{R}_{\mathrm{solve}}$ to a simplified clause yields an admissible clause that contains no complex path terms. □

If for every prime clause ϕ there are no infinite derivations using $\mathcal{R}^{\mathrm{DFun}}$, then we know that we could transform every prime clause ϕ into an equivalent set of solved clauses. But this is not the case. Consider e.g. the clause

$$x[\mu]x \wedge x[f]y \wedge \mu \,\dot{\in}\, f^+ \wedge Ax \wedge By.$$

Then applying (Relate2) to introduce a constraint $f \,\dot{\prec}\, \mu$ followed by an application of (Pre) and (DecFeat) yields the same clause again. The reason for the loop is that we have a cyclic description of the form $x[\mu]x$. But we can show that, similar to Kaplan/Maxwell's Algorithm, $\mathcal{R}^{\mathrm{DFun}}$ is terminating under $<^{\mathrm{basic}}$ if no cyclic descriptions are encountered.

Theorem 6 *Let ϕ be a prime clause such that no $<^{\mathrm{basic}}$-derivative of ϕ contains a cycle. Then there is no infinite $<^{\mathrm{basic}}$-derivation. Furthermore, ϕ has the same $\mathcal{X}$-solutions as the set of solved clauses derivable from ϕ.*

Hence, the control $<^{\text{basic}}$ can be used if one does not want to handle cyclic structures. Note that one can easily recognize whether the algorithm runs in a loop using an occurs check (i.e., by checking whether one visits some variable twice). In this case, one can either stop (without knowing anything about the satisfiability), or switch to the more complex control $<^{\text{quasi}}$ that at least guarantees quasi-termination. A rewrite system is quasi-terminating, if it may loop, but produces only finitely many different clauses. Given a quasi-terminating rewrite system, an algorithm using this system must record the previously calculated clauses and stop, if one clause is produced for the second time. This is expensive, but necessary if you want to handle cyclic structures. $<^{\text{quasi}}$ is the control extending $<^{\text{basic}}$ with the property that

$$\forall r \in \mathcal{R}_{\text{pre}}, \forall r' \in \mathcal{R}_{\text{solve}} : r <^{\text{quasi}} r'.$$

Since in $<^{\text{quasi}}$ the rules in $\mathcal{R}_{\text{pre}}$ are applied first, we know that every clause is first transformed into a set of pre-solved clauses, which are then solved using $\mathcal{R}_{\text{solve}}$. By an adaptation of [2] we get the following theorem.

A necessary condition for this theorem is that for every prime clause ϕ, the set of all regular languages introduced in some $<^{\text{quasi}}$-derivative of ϕ by (Dec_{DFun}), (DecFeat), or (Join) is finite. Clearly, there are only finitely many different regular languages produced by (DecFeat) or (Join), but (Dec_{DFun}) may be a problem. [2] shows how an appropriate decomposition function can be found for a given prime clause ϕ.

Theorem 7 *There exists a decomposition function* DFun *such that for every prime clause ϕ there are only finitely many $<^{\text{quasi}}$-derivatives. Furthermore, ϕ has the same $\mathcal{X}$-solutions as the set of $<^{\text{quasi}}$-derivatives that are solved.*

Next, we want to show that we can simulate Kaplan/Maxwell's algorithm. The idea is that one can associate with every application of one of the $\mathcal{R}_{\text{pre}}$ rules a corresponding rule in Kaplan/Maxwell's algorithm. But in their algorithm, there is no syntactic equivalent for the prefix, divergence, and path equality constraints. The path equality and prefix constraints are not a problem, since they will be removed under $<^{\text{basic}}$ before the next rule in $\mathcal{R}_{\text{pre}}$ is applied. But the divergence constraints may survive. This is not the case if we apply the rules in $\mathcal{R}_{\text{solve}}$ before applying a rule in $\mathcal{R}_{\text{pre}}$. Hence, we can define the control $<^{\text{KM}}$ extending $<^{\text{basic}}$ by

$$\forall r \in \mathcal{R}_{\text{solve}}, \forall r' \in \mathcal{R}_{\text{pre}} : r <^{\text{KM}} r'.$$

Note that this control orders the rules in $\mathcal{R}_{\text{pre}}$ and $\mathcal{R}_{\text{solve}}$ in exactly the other direction than the control $<^{\text{quasi}}$. The control $<^{\text{KM}}$ has the property that there are no divergence constraints in the derivable, simplified clauses, which implies that we can translate them back into the Kaplan/Maxwell syntax. Furthermore, the only rule for handling the divergence constraints that is needed under this control is (Triv1), i.e., the handling of divergence constraints under this control is trivial.

Theorem 8 *Let ϕ be a prime clause and let ϕ' be a $<^{\text{KM}}$-derivative. Then a rule in $\mathcal{R}_{\text{pre}}$ is applicable if and only if ϕ' is simplified and contains no divergence constraints. Furthermore, we can associate with every $<^{\text{KM}}$-derivation D a corresponding derivation D' in Kaplan/Maxwell's algorithm such that the sequence of translations of clauses in D' is exactly the sequence of simplified clauses in D, and vice versa.*

Now what's left? This are the $<^{\text{basic}}$-derivations that are neither $<^{\text{quasi}}$-derivations nor $<^{\text{KM}}$-derivations. The question arise whether there is any use for such derivations, and there are. The reason simply is that it depends on the used regular languages whether for a specific divergence constraint, it is more useful to solve this divergence constraint immediately (as it is done under the $<^{\text{KM}}$ control), or whether it is better to delay this solving (as in the $<^{\text{quasi}}$ control) hoping that this might be superfluous since other rules may detect a simple inconsistence. Consider a generalization of the example given in the introduction using regular languages of the form $comp^{+}\{grel_1, \ldots, grel_n\}$, where $grel_1, \ldots, grel_n$ are grammatical relations such as direct object, indirect object and so on. Now let ϕ be a clause of the form

$$\begin{aligned} x[\mu]y \wedge x[\nu]z \wedge \mu \mathbin{\dot{\amalg}} \nu \wedge\ & \mu \mathrel{\dot{\in}} comp^{+}\{grel_1, \ldots, grel_n\} \\ & \wedge \nu \mathrel{\dot{\in}} comp^{+}\{grel_1, \ldots, grel_n\} \wedge \psi \end{aligned}$$

Then we know that μ is of the form $\delta \circ f \circ \mu'$ and ν is of the form $\delta \circ g \circ \nu'$ such that the common prefix δ is in $comp^{+}$, $f \in \{comp, grel_1, \ldots, grel_n\}$ and $g \in \{comp, grel_1, \ldots, grel_n\} - \{f\}$. Hence, there are $(n+1) \times n$ different possibilities that μ and ν diverge. Solving the divergence constraint immediately as forced by the $<^{\text{KM}}$ control, would produce a disjunction of $(n+1) \times n$ clauses. This makes sense for $n = 1$ (as in the case of $comp^{+}subj$) since it reduces the overhead for keeping the divergence constraint. But should be delayed in the case where n is greater than 1. Using the control $<^{\text{basic}}$, one has the flexibility to do so, and Theorem 6 guarantees that the algorithm terminates in the case of non-cyclic descriptions.

REFERENCES

[1] Franz Baader, Hans-Jürgen Bürckert, Bernhard Nebel, Werner Nutt, and Gert Smolka. On the expressivity of feature logics with negation, functional uncertainty, and sort equations. *Journal of Logic, Language and Information*, 2:1–18, 1993.

[2] Rolf Backofen. Regular path expressions in feature logic. *Journal of Symbolic Computation*, 17:412–455, 1994.

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

[4] 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.

[5] Ronald M. Kaplan and Annie 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.

[6] Bill 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.