Deterministic Consistency Checking of LP Constraints

Suresh Manandhar
Language Technology Group
Human Communication Research Centre
University of Edinburgh, Scotland
email: Suresh.Manandhar@ed.ac.uk

Abstract

We provide a constraint based computational model of linear precedence as employed in the HPSG grammar formalism. An extended feature logic which adds a wide range of constraints involving precedence is described. A sound, complete and terminating deterministic constraint solving procedure is given. Deterministic computational model is achieved by weakening the logic such that it is sufficient for linguistic applications involving word-order.

Subject areas: feature logic, constraint based grammars

1 Introduction

Within HPSG (Pollard and Sag, 1987) (Pollard and Sag, 1994) the *constituent ordering principle* given in (1) is intended to express the relation between the value of the PHON attribute and the DTRS attribute which contains the hierarchical structure of the derivation.

(1) $phrasal\_sign = \begin{bmatrix} \text{PHON } order\_constituent(\boxed{1}) \\ \text{DTRS } \boxed{1} \end{bmatrix}$

(2) Linear Precedence Constraint 1 (LP1):
$HEAD[LEX+] < []$

However, it is not entirely clear how *order_constituent* is supposed to interpret various linear precedence statements such as LP1.

1.1 Reape's approach

The idea taken in Reape's approach (Reape, 1993) is to suggest that word-order is enforced between locally definable word order domains which are ordered sequences of constituents. Word order domains in Reape's approach are totally ordered sequences. A domain union operation as given in (3) is then employed to construct word order domains locally within a HPSG derivation step.

(3) $\bigcirc(\eta, \eta, \eta)$.
$\bigcirc(x \circ \sigma_1, \sigma_2, x \circ \sigma_3) \leftrightarrow \bigcirc(\sigma_1, \sigma_2, \sigma_3)$.
$\bigcirc(\sigma_1, x \circ \sigma_2, x \circ \sigma_3) \leftrightarrow \bigcirc(\sigma_1, \sigma_2, \sigma_3)$.

If A is the string $< a, b >$ and B is the string $< c, d >$, their domain union C given by $\bigcirc(A, B, C)$ will produce all the sequences in which a precedes b and c precedes d *i.e.* the following sequences:

$< a, b, c, d >$ $< a, c, b, d >$
$< a, c, d, b >$ $< c, d, a, b >$
$< c, d, a, b >$ $< c, a, b, d >$

However in this system to encode the property that $\{x, y, z\}$ is a domain in which the ordering is arbitrary (*i.e.* free) then one needs the following disjunctive statements:

$< x, y, z > \sqcup < x, z, y > \sqcup$
$< y, x, z > \sqcup < y, z, x > \sqcup$
$< z, x, y > \sqcup < z, y, x >$

It is simply not possible to be agnostic about the relative ordering of sequence elements within Reape's system.

We identify two deficiencies in Reape's approach namely:

- System is non-deterministic (generate and test paradigm)
- Not possible to be agnostic about order

This is so since domain union is a *non-deterministic* operation and secondly underspecification of ordering within elements of a domain is not permitted.

In the following sections we describe a constraint language for specifying LP constraints that overcomes both these deficiencies. Additionally our constraint language provides a broad range of constraints for specifying linear precedence that go well beyond what is available within current typed feature formalisms. Our approach is in the spirit of Reape's approach but improves upon it.

Furthermore, a sound, complete and terminating consistency checking procedure is described.

Our constraint solving rules are *deterministic* and *incremental.* Hence these do not introduce costly choice-points. These constraint solving rules can be employed for building an efficient implementation. This is an important requirement for practical systems. Indeed we have successfully extended the ProFIT typed feature formalism (Erbach, 1995) with the constructs described in this paper.

2 Outline of an alternative approach

To motivate our approach we start with an example on scrambling in German subordinate clauses.

(4) daß er einen Mann in der Straße laufen
that he a man in the street walking
sah.
saw.
that he saw a man walking in the street.

(5) daß er in der Straße einen Mann laufen sah.

(6) daß einen Mann er in der Straße laufen sah.

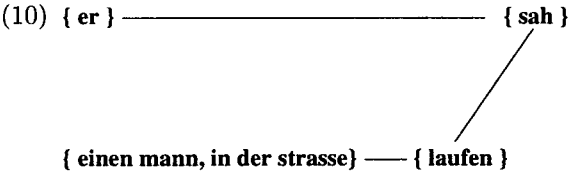
(7) daß einen Mann in der Straße er laufen sah.

(8) daß in der Straße er einen Mann laufen sah.

(9) daß in der Straße einen Mann er laufen sah.

The above data can be captured precisely if we can state that *sah* requires both its verbal argument *laufen* and its NP argument *er* to precede it. Similarly, *laufen* would require both its arguments *einen Mann* and *in der Straße* to precede it. This is illustrated schematically in (10) below.

(10)



Our idea is to employ a specification such as the one given in (11) which is a partial specification of the lexical entry for the verb *sah.* The specification can be thought of as a formal specification of the intuitive description given in (12).

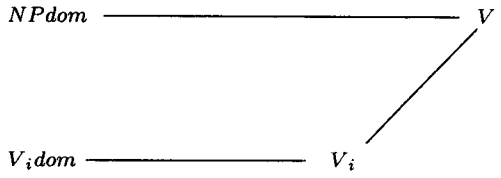
(11)

$$\left(\begin{array}{l} V \sqcap \quad phon: \ < sah > \ \sqcap \\ \qquad field: \ Field \sqcap \\ \qquad syn: (\ cat: v \sqcap \\ \qquad\qquad subcat: \{\ NP \ \sqcap dom: NPdom, \\ \qquad\qquad\qquad\qquad V_i \ \sqcap dom: V_i dom\} \sqcap \\ \qquad\qquad dom :\supseteq NPdom \sqcap \\ \qquad\qquad dom :\supseteq V_i dom\ ) \sqcap \\ \quad V_i dom < _{dom} \ \{V\} \sqcap \\ \quad NPdom < _{dom} \ \{V_i\} \sqcap \\ \quad V_i < V \end{array}\right)$$

For space reasons, our treatment is necessarily somewhat superficial since we do not take into account other interacting phenomena such as *fronting* or *extraposition.*

The definition in (11) does not make specific assumption about whether a context-free backbone is employed or not. However, if a CFG backbone is employed then we assume that the value of the *subcat* attribute is treated as an *unordered* sequence (*i.e.* a set) as defined in (11).

(12)



The essential idea is to use set-valued descriptions to model word-order domains. In particular subset constraints (Manandhar, 1994) are employed to construct larger domains from smaller ones. Thus in example (11) the domain of the verb is constructed by including the domains of the subcategorised arguments (enforced by the constraints $dom :\supseteq NPdom \sqcap dom :\supseteq V_i Dom$). Note that in this example the verb itself is not part of its own domain. The binary constraint $V_i < V$ enforces precedence ordering between the *signs* V_i and V. The constraint $V_i dom <_{dom} \{V\}$ ensures that every element of the set $V_i Dom$ precedes the sign V. In other words, the set $V_i Dom$ is in the *domain precedence* relation with the singleton $\{V\}$.

However there are strong constraints on ordering in the middle field. For instance, when pronomial complements are involved then not all permutations are acceptable. Examples such as (13) are considered ungrammatical.

(13) *daß in der Straße ihn er laufen sah.

According to Uszkoreit (Uszkoreit, 1985), ordering of arguments in the middle field is governed by the following set of LP constraints given in (14) which are to be interpreted disjunctively.

(14)
$$\begin{array}{l} PPRN: + \ < \ PPRN: - \\ TR: agent \ < \ TR: theme \\ TR: agent \ < \ TR: goal \\ TR: goal \ < \ TR: theme \\ FOCUS: - \ < \ FOCUS: + \end{array}$$

The LP constraint in (14) states that for every pair of constituents in the middle field at least one of the conditions should apply otherwise the sentence is considered ungrammatical. A related but more elaborate LP rule mechanism is considered in (Steinberger, 1994).

To approximate this complex LP constraint employing the kind of logical machinery described in this paper, we can use a description such as the one given in (15). The definition given in (15) extends the description given in (11).

(15) $syn : dom : MF \sqcap$
$\exists x \exists y$ *if* $x \in MF \wedge y \in MF \wedge x < y$
then
if $x = pprn : + \wedge y = pprn : -$
then $\top$
else
if $x = tr : agent \wedge y = tr : theme$
then $\top$
else
if $x = tr : agent \wedge y = tr : goal$
then $\top$
else
if $x = tr : goal \wedge y = tr : theme$
then $\top$
else
$x = focus : - \wedge y = focus : +$

The definition in (15) can be understood as follows. The feature constraint $syn : dom : MF$ coinstantiates the middle field domain to the variable MF. To keep the example simple, we assume that the whole domain is in the middle field and we ignore *fronting* or *extraposition*. A more complex condition would be needed to handle these.

The rest of the definition in (15) ensures that for every pair of elements x and y such that x and y are both members of MF and x *precedes* y at least one of the LP constraints hold. If every LP constraint is violated then an inconsistency results. The constraints in (15) is a weaker representation of the disjunctive specification given in (16).

(16) $\exists x \exists y$ *if* $(x \in MF \wedge y \in MF \wedge x < y)$
then
$$\bigvee \left\{ \begin{array}{l} x = pprn : + \wedge y = pprn : - \\ x = tr : agent \wedge y = tr : theme \\ x = tr : agent \wedge y = tr : goal \\ x = tr : goal \wedge y = tr : theme \\ x = focus : - \wedge y = focus : + \end{array} \right\}$$

The description in (16) non-deterministically requires that at least one of the LP constraints hold. On the other hand, the description in (15) *waits* until either one of the LP constraints is satisfied (in which case it succeeds) or all the LP constraints are violated (in which case it fails). Thus the description in (15) can be solved deterministically.

Thus (15) should rule out the ungrammatical example in (13) if the assumptions regarding *focus* are made as in (17).

(17) *daß	in der Straße	ihn	er laufen sah.
	pprn:-	*focus:-*	
	th:theme	*pprn:+*	
		tr:agent	

Note that it is not necessary to know whether the PP *in der Straße* is focussed to rule out (17) since the fact that the pronoun *ihn* is *focus:-* is enough to trigger the inconsistency.

3 Some generic LP constraints

As suggested by the example in (11), in general we would want support within typed feature formalisms for at least the following kinds of LP constraints.

1. $\text{Sign}_1 < \text{Sign}_2$
2. $\text{Dom}_1 <_{dom} \text{Dom}_2$
(Dom_1 and Dom_2 are set-valued)
3. Dom_1 *is included in* Dom_2

The constraint $\text{Sign}_1 < \text{Sign}_2$ states that Sign_1 *precedes* Sign_2. The constraint $\text{Dom}_1 <_{dom} \text{Dom}_2$ states that every element of the set described by Dom_1 precedes every element of the set described by Dom_2. Constraints such as Dom_1 *is included in* Dom_2 essentially builds larger domains from smaller ones and can be thought of as achieving the same effect as Reape's domain union operation. Note crucially that within our approach the specification of precedence constraints (such as $\text{Sign}_1 < \text{Sign}_2$ and $\text{Dom}_1 <_{dom} \text{Dom}_2$) is independent of the domain building constraint (*i.e.* the constraint Dom_1 *is included in* Dom_2). This we believe is a generalisation of Reape's approach.

Other constraints such as the following involving *immediate precedence* and *first element of a domain* are of lesser importance. However, these could be of the form:

1. Sign_1 *immediately-precedes* Sign_2
2. *First daughter of* Dom_1 is Sign_1

To be able to state descriptions such as in (15), we also want to introduce *guarded* (or *conditional*) LP constraints such the following:

1. *if* Sign_1 is NP[acc] $\wedge$ Sign_2 is NP[dat]
then $\text{Sign}_1 < \text{Sign}_2$
(Guards on Feature constraints)
2. *if* $\text{Sign}_1 < \text{Sign}_2$ *then* .....
(Guards on precedence constraints)
3. $\exists x \exists y$ (*if* x:NP[acc] $\in$ Dom $\wedge$
y:NP[dat] $\in$ Dom
then x < y)
(Guards on set members)

Guarded constraints can be thought of as *conditional constraints* whose execution depends on the presence of other constraints. The condition part

G of a guarded constraint *if* G *then* S *else* T is known as a *guard*. The consequent S is executed if the current set of constraints *entail* the guard G. The consequent T is executed if the current set of constraints *disentail* the guard G. If the current set of constraints neither entail nor disentail G then the execution of the whole guarded constraint is *blocked* until more information is available.

The application of guarded constraints within computational linguistics has not been well explored. However, the *Horn extended feature structures* described in (Hegner, 1991) can be thought of as adding guards to feature structures. On the other hand, within logic programming guarded logic programming languages have a longer history originating with *committed-choice languages* (Ueda, 1985) and popularised by the *concurrent constraint programming* paradigm due to *Saraswat* (Saraswat and Rinard, 1990) (Saraswat, 1993).

For space reasons, we do not cover the logic of guarded feature constraints, guards on set membership constraints and guards on precedence constraints. Guarded feature constraints have been extensively studied in (Aït-Kaci et al., 1992) (Smolka and Treinen, 1994) (Aït-Kaci and Podelski, 1994).

4 A feature logic with LP constraints

In this section we provide formal definitions for the syntax and semantics of an extended feature logic that directly supports linear precedence constraints as logical primitives. The logic described in this paper is a further development of the one described in (Manandhar, 1993).

The syntax of the constraint language is defined by the following BNF definitions.

Syntax

Let $\mathcal{F}$ be the set of relation symbols and let $\mathcal{P}$ be the set of irreflexive relation symbols. We shall require that $\mathcal{F}$ and $\mathcal{P}$ are disjoint.

$\phi, \psi \longrightarrow$	$x = f : y$	feature constraint
	$x = \exists f : y$	set-membership
	$x = \exists p^+ : y$	transitive closure
	$x = \exists p^* : y$	reflex-trans closure
	$x = f :\supseteq g(y)$	subset inclusion
	$x = [f\ p\ 1]y$	first daughter
	$f(x) : p^+ : g(y)$	domain precedence
	$f(x) : p^* : g(y)$	domain prec. equals
	$\phi\ \&\ \psi$	conjunction

where $f \in \mathcal{F}$ and $p \in \mathcal{P}$

The constraint $x = f : y$ specifies that y is the *only* f-value of x. The constraint $x = \exists f : y$ states that y is one of the f-values of x.

The constraint $x = \exists p^+ : y$ just says that x is related to y via the transitive closure of p. The precedence constraint such as Sign_1 *precedes* Sign_2 is intended to be captured by the constraint $\text{Sign}_1 = \exists p^+ : \text{Sign}_2$ where p denotes the (user chosen) *immediate precedence* relation.

Similarly, $x = \exists p^* : y$ states that x is related to y via the transitive, reflexive closure of p. This constraint is similar to the constraint $x = \exists p^+ : y$ except that it permits x and y to be equal.

The constraints $f(x) : p^+ : g(y)$ and $f(x) : p^* : g(y)$ are intended to enforce precedence between two word-ordering domains. The constraint $f(x) : p^+ : g(y)$ states that every f-value of x *precedes* (*i.e.* is in the p^+ relation with) every g-value of y. The constraint $f(x) : p^* : g(y)$ is analogous.

The constraint $x = [f\ p\ 1]y$ states that y is the *first daughter* amongst the f-values of x (*i.e.* is in the p^* relation with every f-value of x).

Since our language supports both feature constraints and set-membership constraints the conventional semantics for feature logic (Smolka, 1992) needs to be extended. The essential difference being that we interpret every feature/relation as a binary relation on the domain of interpretation. Feature constraints then require that they behave functionally on the variable upon which the constraint is expressed.

A precise semantics of our constraint language is given next.

Semantics

An interpretation structure $\mathcal{I} =< \mathcal{U}^I, .^I >$ is a structure such that:

- $\mathcal{U}^I$ is an arbitrary non-empty set
- $.^I$ is an interpretation function which maps:
  - every relation $f \in \mathcal{F}$ to a binary relation: $f^I \subseteq \mathcal{U}^I \times \mathcal{U}^I$
  - every relation $p \in \mathcal{P}$ to a binary relation: $p^I \subseteq \mathcal{U}^I \times \mathcal{U}^I$ with the added condition that $(p^I)^+$ is irreflexive

A variable assignment α is a function $\alpha : \mathcal{V} \longrightarrow \mathcal{U}^I$.

We shall write $f^I(e)$ to mean the set:

$$f^I(e) = \{e' \in \mathcal{U}^I \mid (e, e') \in f^I\}$$

We say that an interpretation $\mathcal{I}$ and a variable assignment α satisfies a constraint ϕ written $\mathcal{I}, \alpha \models \phi$ if the following conditions are satisfied:

$$
\begin{array}{lll}
\mathcal{I},\alpha \models \phi \,\&\, \psi & \Longleftrightarrow & \mathcal{I},\alpha \models \phi \wedge \mathcal{I},\alpha \models \psi \\
\mathcal{I},\alpha \models x = f : y & \Longleftrightarrow & f^I(\alpha(x)) = \{\alpha(y)\} \\
\mathcal{I},\alpha \models x = \exists f : y & \Longleftrightarrow & (\alpha(x),\alpha(y)) \in f^I \\
\mathcal{I},\alpha \models x = \exists p^+ : y & \Longleftrightarrow & (\alpha(x),\alpha(y)) \in (p^I)^+ \\
\mathcal{I},\alpha \models x = \exists p^* : y & \Longleftrightarrow & (\alpha(x),\alpha(y)) \in (p^I)^* \\
\mathcal{I},\alpha \models x = f :\supseteq g(y) & \Longleftrightarrow & f^I(\alpha(x)) \supseteq g^I(\alpha(y)) \\
\mathcal{I},\alpha \models x = [f\ p\ 1]y & \Longleftrightarrow & \alpha(y) \in f^I(\alpha(x)) \wedge \\
 & & \forall e \in \mathcal{U}^I \\
 & & (e \in f^I(\alpha(x)) \Rightarrow \\
 & & (\alpha(y), e) \in (p^I)^*) \\
\mathcal{I},\alpha \models f(x) : p^+ : g(y) & \Longleftrightarrow & \forall e_1, e_2 \in \mathcal{U}^I \\
 & & ((e_1 \in f^I(\alpha(x)) \wedge \\
 & & e_2 \in g^I(\alpha(y))) \\
 & & \Rightarrow (e_1, e_2) \in (p^I)^+) \\
\mathcal{I},\alpha \models f(x) : p^* : g(y) & \Longleftrightarrow & \forall e_1, e_2 \in \mathcal{U}^I \\
 & & ((e_1 \in f^I(\alpha(x)) \wedge \\
 & & e_2 \in g^I(\alpha(y))) \\
 & & \Rightarrow (e_1, e_2) \in (p^I)^*)
\end{array}
$$

Given the above semantics, it turns out that the *first-daughter* constraint can be defined in terms of other constraints in the logic. Let f_p_1 be a distinct relation symbol then we can equivalently define the first-daughter constraint by:

- $x = [f\ p\ 1]y \approx x = f\_p\_1 : y \wedge$
  $x = \exists f : y \wedge f\_p\_1(x) : p^* : f(x)$

The translation states that y (which is the f_p_1-value of x) precedes or is equal to every f-value of x and y is a f-value of x. For this to work, we require that the feature symbol f_p_1 appears only in the translation of the constraint $x = [f\ p\ 1]y$.

4.1 Two Restrictions

The logic we have described comes with 2 limitations which at first glance appears to be somewhat severe, namely:

- **NO** *atomic values*
- **NO** *precedence as a feature*

This is so because it turns out that adding both functional precedence and atoms in general leads to a non-deterministic constraint solving procedure. To illustrate this, consider the following constraints:

$$x = f : y \wedge y = a \wedge x = \exists f^* : z$$

where a is assumed to be an *atom*.

The above constraints state that y is the f-value of x and y is the atom a and z is related to x by the reflexive-transitive closure of f.

Determining consistency of such constraints in general involves solving for the following disjunctive choices of constraints.

$$x = z \text{ or } y = z$$

$$\text{(Equals)}\ \frac{x = y \wedge C_s}{x = y \wedge [x/y]C_s} \quad \text{if } x \neq y \text{ and } x \text{ occurs in } C_s$$

$$\text{(Feat)}\ \frac{x = f : y \wedge x = f : z \wedge C_s}{x = f : y \wedge y = z \wedge C_s}$$

$$\text{(FeatExists)}\ \frac{x = f : y \wedge x = \exists f : z \wedge C_s}{x = f : y \wedge x = \exists f : z \wedge y = z \wedge C_s}$$

$$\text{(Subset)}\ \frac{x = f :\supseteq g(y) \wedge y = G : z \wedge C_s}{x = \exists f : y \wedge x = f :\supseteq g(y) \wedge y = G : z \wedge C_s} \quad \text{if } x = \exists f : y \notin C_s \text{ where } G \text{ ranges over } g, \exists g$$

Figure 1: Constraint Solving - I

However for practical reasons we want to eliminate any form of backtracking since this is very likely to be expensive for implemented systems. On the other hand, we certainly cannot prohibit atoms since they are crucially required in grammar specification. But disallowing functional precedence is less problematic from a grammar development perspective.

4.2 Imposing the restriction

We note that precedence can be restricted to non-atomic types such as HPSG *signs* without compromising the grammar in any way. We then need to ensure that precedence constraints never have to consider atoms as their values. This can be easily achieved within current typed feature formalisms by employing *appropriateness conditions* (Carpenter, 1992).

An *appropriateness condition* just states that a given feature (in our case a relation) can only be defined on certain (appropriate) types. The assumption we make is that precedence is specified in such a way that is appropriate only for non-atomic types. This restriction can be imposed by the system (*i.e.* a typed feature formalism) itself.

5 Constraint Solving

We are now ready to consider consistency checking rules for our constraint language. To simplify the presentation we have split up the rules into two groups given in figure 1 and figure 2.

The constraint solving rules given in figure 1 deal with constraints involving *features*, *set-memberships*, *subset* and *first daughter*. Rules (Equals) and (Feat) are the usual feature logic rules (Smolka, 1992) that deal with equality and features. By $[x/y]C_s$ we mean replacing every occurrence of x with y in C_s. Rule (FeatEx-

(TransConj) $\dfrac{x = \exists p^* : y \wedge x = \exists p^+ : y \wedge C_s}{x = \exists p^+ : y \wedge C_s}$

(TransClos) $\dfrac{x = \exists R_1 : y \wedge y = \exists R_2 : z \wedge C_s}{x = \exists (R_1 \times R_2) : z \wedge}$
$x = \exists R_1 : y \wedge y = \exists R_2 : z \wedge C_s$
if $x = \exists p^+ : z \notin C_s \wedge$
$x = \exists (R_1 \times R_2) : z \notin C_s$
where $R_1 \times R_2$ is computed from:

$\times$	p^*	p^+
p^*	p^*	p^+
p^+	p^+	p^+

(Cycle) $\dfrac{x = \exists p^* : y \wedge y = \exists p^* : x \wedge C_s}{x = y \wedge C_s}$

(DomPrec) $\dfrac{f(x) : R : g(y) \wedge x = \exists f : x_1 \wedge \quad y = \exists g : y_1 \wedge C_s}{x_1 = \exists R : y_1 \wedge f(x) : R : g(y) \wedge \quad x = \exists f : x_1 \wedge y = \exists g : y_1 \wedge C_s}$
if $x_1 = \exists p^+ : y_1 \notin C_s \wedge$
$x_1 = \exists R : y_1 \notin C_s$
where R ranges over p^+, p^*

Figure 2: Constraint Solving - II

ists) deals with the interaction of feature and set-membership constraint. Rule (Subset) deals with subset constraints and adds a new constraint $x = \exists f : y$ in the presence of the subset constraint $x = f :\supseteq g(y)$ and the constraint $y = G : z$ (where G ranges over $g, \exists g$).

The constraint solving rules given in figure 2 deal with constraints involving the *precedes* and the *precedes or equal to* relations and *domain precedence*. Rule (TransConj) eliminates the weaker constraint $x = \exists p^* : y$ when both $x = \exists p^* : y \wedge x = \exists p^+ : y$ hold. Rule (TransClos) effectively computes the transitive closure of the precedence relation one-step at a time. Rule (Cycle) detects cyclic relations that are consistent, namely, when x *precedes or equals* y and *vice versa* then $x = y$ is asserted. Finally rule (DomPrec) propagates constraints involving domain precedence.

We say that a set of constraints are in **normal form** if *no* constraint solving rules are applicable to it. We say that a set of constraints in normal form contains a **clash** if it contains constraints of the form:

$$x = \exists p^+ : x$$

In the following sections we show that our constraint solving rules are sound and every **clash-free** constraint system in normal form is consistent.

5.1 Soundness, Completeness and Termination

Theorem 1 (Soundness) *Let $\mathcal{I}, \alpha$ be any interpretation, assignment pair and let C_s be any set of constraints. If a constraint solving rule transforms C_s to C'_s then:*

$$\mathcal{I}, \alpha \models C_s \text{ iff } \mathcal{I}, \alpha \models C'_s$$

Proof Sketch: The soundness claim can be verified by checking that every rule indeed preserves the interpretation of every variable and every relation symbol.

Let $succ(x, f)$ and $succ(x, p)$ and denote the sets:

- $succ(x, f) =$
  $\{y \mid x = \exists f : y \in C_s \vee \ x = f : y \in C_s\}$
- $succ(x, p) = \{y \mid$
  $x = \exists R : y \in C_s \wedge$
  $\neg \exists z : (x = \exists R_1 : z \wedge z = \exists R_2 : y) \in C_s\}$
  where $R, R_1, R_2 \in \{p^+, p^*\}$

Theorem 2 (Completeness) *A constraint system C_s in normal form is consistent iff C_s is clash-free.*

Proof Sketch: For the first part, let C_s be a constraint system containing a clash then it is clear from the definition of clash that there is no interpretation $\mathcal{I}$ and variable assignment α which satisfies C_s.

Let C_s be a clash-free constraint system in normal form.

We shall construct an interpretation $\mathcal{R} = < \mathcal{U}^R, .^R >$ and a variable assignment α such that $\mathcal{R}, \alpha \models C_s$.

Let $\mathcal{U}^R = \mathcal{V}$.

The assignment function α is defined as follows:

- if x does not occur in C_s then $\alpha(x) = x$
- if x is such that x occurs exactly once in $x = y \in C_s$ then $\alpha(x) = x$
- if $x = y \in C_s$ then $\alpha(y) = \alpha(x)$

Note that for constraints in normal form: if $x = y \in C_s$ then either x is identical to y or x occurs just once in C_s (in the constraint $x = y$). Otherwise Rule (Equals) is applicable.

The interpretation function $.^R$ is defined as follows:

- $f^R(\alpha(x)) = succ(\alpha(x), f)$
- $p^R(\alpha(x)) = succ(\alpha(x), p)$

It can be shown by a case by case analysis that for every constraint K in C_s:
$\mathcal{R}, \alpha \models K$.

Hence we have the theorem.

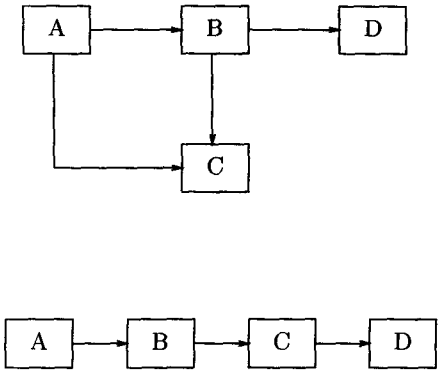


Figure 3: Linearisation of precedence ordered DAGs

Theorem 3 (Termination)
The consistency checking procedure terminates in a finite number of steps.

Proof Sketch: The termination claim can be easily verified if we first exclude rules (Subset), (TransClos) and (DomPrec) from consideration. Then for the remainder of the rules termination is obvious since these rules only simplify existing constraints. For these rules:

1. Rule (Subset) increases the size of $succ(x, f)$ but since none of our rules introduces new variables this is terminating.
2. Rules (TransClos) and (DomPrec) asserts a relation R between pairs of variables x, y. However, none of these rules apply once $x = \exists p^+ : y$ is known. Furthermore, if $x = \exists p^+ : y$ is known it is never simplified to the weaker $x = \exists p^* : y$. This means that these rules converge.

6 Linearisation of precedence ordered DAGs

The models generated by the completeness theorem interpret (the map of) every precedence relation p as a *directed acyclic graph* (DAG) as depicted in figure 3. However sentences in natural languages are always totally ordered (*i.e.* they are strings of words). This then raises the question:

Is it possible to generate linearised models?

For the logic that we have described this is always possible. We only provide a graphical argument given in figure 3 to illustrate that this is indeed possible.

The question that arises is then:

What happens when we add immediate precedence?

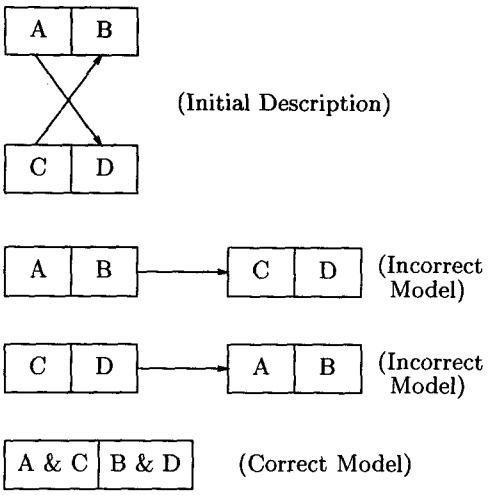


Figure 4: Difficulty in guaranteeing linearisable models with immediate precedence

6.1 Problem with immediate precedence

However if we add immediate precedence to our logic then it is not clear whether we can guarantee linearisable models. This is highlighted in figure 4.

As illustrated in this figure consistency checking of constraints involving both linear precedence and immediate precedence with a semantics that requires linearised models is not trivial. So we do not explore this scenario in this paper.

However, it is possible to add immediate precedence and extend the constraint solving rules described in this paper in such a way that it is sound and complete with respect to the current semantics described in this paper (which does not insist on linearised models).

7 Handling immediate precedence

In this section, we provide additional constraint solving rules for handling immediate precedence. The basic idea is to treat immediate precedence as a functional relation whose inverse too is functional.

In effect what we add to our logic is both precedence as a feature and a new constraint for representing the inverse functional precedence.

This is summarised by:

- *Represent x immediately precedes y* by : $x = p : y \wedge y = p^{-1} : x$
- Semantics: $\mathcal{I}, \alpha \models y = p^{-1} : x \iff (p^I)^{-1}(\alpha(y)) = \{\alpha(x)\}$

The additional rules given in figure below are all that is needed to handle immediate precedence.

$$\text{(FeatExists)}\ \frac{x = p : y \wedge C_s}{x = p : y \wedge x = \exists p : y \wedge C_s} \quad \text{if } x = \exists p : y \notin C_s$$

$$\text{(ExistsTrans)}\ \frac{x = \exists p : y \wedge C_s}{x = \exists p : y \wedge x = \exists p^{+} : y \wedge C_s} \quad \text{if } x = \exists p^{+} : y \notin C_s$$

$$\text{(InvIntro)}\ \frac{x = p^{-1} : y \wedge C_s}{y = \exists p : x \wedge x = p^{-1} : y \wedge C_s} \quad \text{if } y = \exists p : x \notin C_s$$

$$\text{(InvExists)}\ \frac{x = p^{-1} : y \wedge z = \exists p : x \wedge C_s}{y = z \wedge x = p^{-1} : y \wedge y = \exists p : x \wedge C_s} \quad \text{if } y \neq z$$

8 Conclusions

We have shown that the logic of linear precedence can be handled elegantly and deterministically by adding new logical primitives to feature logic. Although, theoretically speaking, our logic comes with some restrictions these have no practical consequences whatsoever. Our implementation of the logic as an extension to the ProFIT typed feature formalism shows that a reasonably efficient implementation is feasible. Some further work is necessary to determine the computational complexity of our constraint solving procedure. However, we believe that it is polynomial.

The logic presented in this paper generalises the approach taken in (Reape, 1993). Our approach demonstrates that it is not necessary to employ a non-deterministic operation such as *domain union* to manipulate domains. Instead precedence constraints are directly embedded in feature logic and a deterministic constraint solving procedure is provided. A wide range of constraints involving precedence is provided directly in feature logic ranging from constraints expressing *precedence between variables*, *precedence between domains* to *guards on precedence constraints*.

9 Acknowledgments

This work was supported by The Commission of the European Communities through the project LRE-61-061 "Reusable Grammatical Resources", where the logic described in this paper has been implemented. Thanks to Wojciech Skut for developing sample grammars to test the implementation and for working on the interface to ProFIT. Thanks to Gregor Erbach for demoing the extended system dubbed CL-ONE. Thanks to Herbert Ruessink and Craig Thiersch for using and providing feedback on the implementation. And thanks to Ralf Steinberger for providing useful comments on an earlier draft.

References

Hassan Aït-Kaci and Andreas Podelski. 1994. Functions as Passive Constraints in LIFE. *ACM Transactions on Programming Languages and Systems*, 16(4):1–40, July.

Hassan Aït-Kaci, Gert Smolka, and R. Treinen. 1992. A feature-based constraint system for logic programming with entailment. Research report, DFKI, Saarbrücken, Germany.

Bob Carpenter. 1992. *The Logic of Typed Feature Structures.* Cambridge University Press.

Gregor Erbach. 1995. ProFIT: Prolog with Features, Inheritance and Templates. In *Seventh Conference of the EACL* (This Vol.), Dublin, Ireland, March.

S. Hegner. 1991. Horn extended feature structures: fast unification with negation and limited disjunction. In *Fifth Conference of the EACL*, pages 33–38, Berlin, Germany.

Suresh Manandhar. 1993. *Relational Extensions to Feature Logic: Applications to Constraint Based Grammars.* Ph.D. thesis, Department of Artificial Intelligence, University of Edinburgh.

Suresh Manandhar. 1994. An Attributive Logic of Set Descriptions and Set Operations. In *32nd Annual Meeting of the ACL*, pages 255–262, Las Cruces, New Mexico.

Carl Pollard and Ivan Andrew Sag. 1987. *Information-Based Syntax and Semantics: Volume 1 Fundamentals*, volume 13 of *Lecture Notes.* CSLI, Stanford, CA.

Carl Pollard and Ivan Andrew Sag. 1994. *Head-driven Phrase Structure Grammar.* Chicago: University of Chicago Press and Stanford: CSLI Publications.

Mike Reape. 1993. Getting Things in Order. In Wietske Sijtsma and Arthur van Horck, editors, *Discontinuous Constituency.* Berlin: Mouton de Gruyte.

V. Saraswat and M. Rinard. 1990. Concurrent Constraint Programming. In *Proceedings of the 7th ACM Symposium on the Principles of Programming Languages*, pages 232–245, San Francisco, CA, January.

Vijay Saraswat. 1993. *Concurrent Constraint Programming.* MIT Press.

Gert Smolka and Ralf Treinen. 1994. Records for logic programming. *Journal of Logic Programming*, 18(3):229–258, April.

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

Ralf Steinberger. 1994. Treating 'Free Word Order' in Machine Translation. In *Proceedings of COLING 1994, Vol. I*, pages 69–75, Kyoto, Japan.

K. Ueda. 1985. Guarded Horn Clauses. Technical Report TR-103, ICOT, Japan.

Hans Uszkoreit. 1985. Constraints on order. Technical Note 364, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, October.