An Attributive Logic of Set Descriptions and Set Operations

Suresh Manandhar
HCRC Language Technology Group
The University of Edinburgh
2 Buccleuch Place
Edinburgh EH8 9LW, UK
Internet: Suresh.Manandhar@ed.ac.uk

Abstract

This paper provides a model theoretic semantics to feature terms augmented with set descriptions. We provide constraints to specify HPSG style set descriptions, fixed cardinality set descriptions, set-membership constraints, restricted universal role quantifications, set union, intersection, subset and disjointness. A sound, complete and terminating consistency checking procedure is provided to determine the consistency of any given term in the logic. It is shown that determining consistency of terms is a NP-complete problem.

Subject Areas: feature logic, constraint-based grammars, HPSG

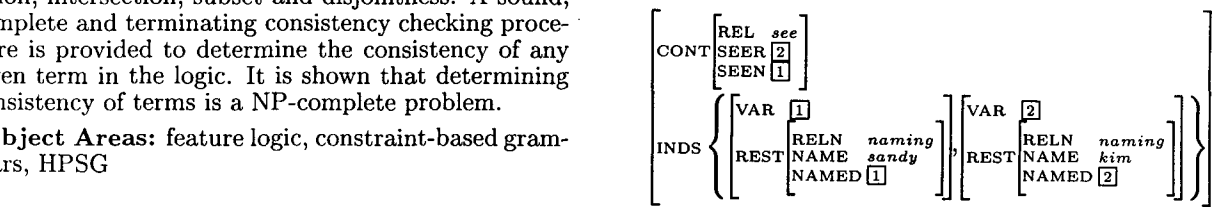
1 Introduction

Grammatical formalisms such as HPSG [Pollard and Sag, 1987] [Pollard and Sag, 1992] and LFG [Kaplan and Bresnan, 1982] employ feature descriptions [Kasper and Rounds, 1986] [Smolka, 1992] as the primary means for stating linguistic theories. However the descriptive machinery employed by these formalisms easily exceed the descriptive machinery available in feature logic [Smolka, 1992]. Furthermore the descriptive machinery employed by both HPSG and LFG is difficult (if not impossible) to state in feature based formalisms such as ALE [Carpenter, 1993], TFS [Zajac, 1992] and CUF [Dörre and Dorna, 1993] which augment feature logic with a type system. One such expressive device employed both within LFG [Kaplan and Bresnan, 1982] and HPSG but is unavailable in feature logic is that of set descriptions.

Although various researchers have studied set descriptions (with different semantics) [Rounds, 1988] [Pollard and Moshier, 1990] two issues remain unaddressed. Firstly there has not been any work on consistency checking techniques for feature terms augmented with set descriptions. Secondly, for applications within grammatical theories such as the HPSG formalism, set descriptions alone are not enough since descriptions involving set union are also needed. Thus to adequately address the knowledge representation needs of current linguistic theories one needs to provide set descriptions as well as mechanisms to manipulate these.

In the HPSG grammar formalism [Pollard and Sag, 1987], set descriptions are employed for the modelling of so called *semantic indices* ([Pollard and Sag, 1987] *pp.* 104). The attribute INDS in the example in (1) is a multi-valued attribute whose value models a set consisting of (at most) 2 objects. However multi-valued attributes cannot be described within feature logic [Kasper and Rounds, 1986] [Smolka, 1992].

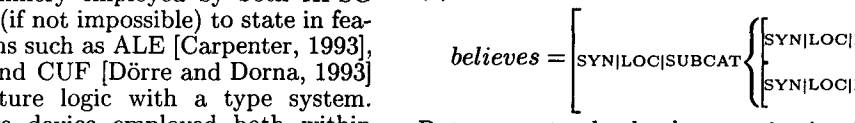
(1)



A further complication arises since to be able to deal with anaphoric dependencies we think that set memberships will be needed to resolve pronoun dependencies. Equally, set unions may be called for to incrementally construct discourse referents. Thus set-valued extension to feature logic is insufficient on its own.

Similarly, set valued subcategorisation frames (see (2)) has been considered as a possibility within the HPSG formalism.

(2)



But once set valued subcategorisation frames are employed, a set valued analog of the HPSG subcategorisation principle too is needed. In section 2 we show that the set valued analog of the subcategorisation principle can be adequately described by employing a disjoint union operation over set descriptions as available within the logic described in this paper.

2 The logic of Set descriptions

In this section we provide the semantics of feature terms augmented with set descriptions and various constraints over set descriptions. We assume an alphabet consisting of $x, y, z, \ldots \in \mathcal{V}$ the set of *variables*; $f, g, \ldots \in \mathcal{F}$ the set of *relation symbols*; $c_1, c_2, \ldots \in \mathcal{C}$ the set of *constant symbols*; $A, B, C, \ldots \in \mathcal{P}$ the set of primitive concept symbols and $a, b, \ldots \in \mathcal{At}$ the set of *atomic symbols*. Furthermore, we require that $\bot, \top \in \mathcal{P}$.

The syntax of our term language defined by the following BNF definition:
$P \longrightarrow x \mid a \mid c \mid C \mid \neg x \mid \neg a \mid \neg c \mid \neg C$

$S, T \longrightarrow$
P

$\mid f:T$	feature term
$\mid \exists f:T$	existential role quantification
$\mid \forall f:P$	universal role quantification
$\mid f:\{T_1,\ldots,T_n\}$	set description
$\mid f:\{T_1,\ldots,T_n\}_=$	fixed cardinality set description
$\mid f:g(x)\cup h(y)$	union
$\mid f:g(x)\cap h(y)$	intersection
$\mid f:\supseteq g(x)$	subset
$\mid f(x)\neq g(y)$	disjointness
$\mid S \sqcap T$	conjunction

where $S, T, T_1, \ldots, T_n$ are terms; a is an *atom*; c is a *constant*; C is a *primitive concept* and f is a *relation symbol.*

The interpretation of *relation symbols* and *atoms* is provided by an interpretation $\mathcal{I} = < \mathcal{U}^I, I >$ where $\mathcal{U}^I$ is an arbitrary non-empty set and I is an interpretation function that maps :

1. every relation symbol $f \in \mathcal{F}$ to a binary relation $f^I \subseteq \mathcal{U}^I \times \mathcal{U}^I$
2. every atom $a \in \mathcal{A}t$ to an element $a^I \in \mathcal{U}^I$

Notation:

- Let $f^I(e)$ denote the set $\{e' \mid (e,e') \in f^I\}$
- Let $f^I(e)\uparrow$ mean $f^I(e) = \emptyset$

$\mathcal{I}$ is required to satisfy the following properties :

1. if $a_1 \not\equiv a_2$ then $a_1^I \neq a_2^I$ (*distinctness*)
2. for any atom $a \in \mathcal{A}t$ and for any relation $f \in \mathcal{F}$ there exists no $e \in \mathcal{U}^I$ such that $(a,e) \in f^I$ (*atomicity*)

For a given interpretation $\mathcal{I}$ an **$\mathcal{I}$-assignment** α is a function that maps :

1. every variable $x \in \mathcal{V}$ to an element $\alpha(x) \in \mathcal{U}^I$
2. every constant $c \in \mathcal{C}$ to an element $\alpha(c) \in \mathcal{U}^I$ such that for distinct constants $c_1, c_2 : \alpha(c_1) \neq \alpha(c_2)$
3. every primitive concept $C \in \mathcal{P}$ to a subset $\alpha(C) \subseteq \mathcal{U}^I$ such that:
   - $\alpha(\bot) = \emptyset$
   - $\alpha(\top) = \mathcal{U}^I$

The interpretation of terms is provided by a denotation function $[\![.]\!]^{\mathcal{I},\alpha}$ that given an interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment α maps terms to subsets of $\mathcal{U}^I$.

The function $[\![.]\!]^{\mathcal{I},\alpha}$ is defined as follows :

$[\![x]\!]^{\mathcal{I},\alpha} = \{\alpha(x)\}$
$[\![a]\!]^{\mathcal{I},\alpha} = \{a^I\}$
$[\![c]\!]^{\mathcal{I},\alpha} = \{\alpha(c)\}$
$[\![C]\!]^{\mathcal{I},\alpha} = \alpha(C)$
$[\![f:T]\!]^{\mathcal{I},\alpha} =$
$\{e \in \mathcal{U}^I \mid \exists e' \in \mathcal{U}^I : f^I(e) = \{e'\} \ \wedge \ e' \in [\![T]\!]^{\mathcal{I},\alpha}\}$
$[\![\exists f:T]\!]^{\mathcal{I},\alpha} =$
$\{e \in \mathcal{U}^I \mid \exists e' \in \mathcal{U}^I : (e,e') \in f^I \ \wedge \ e' \in [\![T]\!]^{\mathcal{I},\alpha}\}$
$[\![\forall f:T]\!]^{\mathcal{I},\alpha} =$
$\{e \in \mathcal{U}^I \mid \forall e' \in \mathcal{U}^I : (e,e') \in f^I \Rightarrow e' \in [\![T]\!]^{\mathcal{I},\alpha}\}$
$[\![f:\{T_1,\ldots,T_n\}]\!]^{\mathcal{I},\alpha} =$
$\{e \in \mathcal{U}^I \mid \exists e_1,\ldots,\exists e_n \in \mathcal{U}^I :$
$f^I(e) = \{e_1,\ldots,e_n\} \wedge$
$e_1 \in [\![T_1]\!]^{\mathcal{I},\alpha} \wedge \ldots \wedge e_n \in [\![T_n]\!]^{\mathcal{I},\alpha}\}$
$[\![f:\{T_1,\ldots,T_n\}_=]\!]^{\mathcal{I},\alpha} =$
$\{e \in \mathcal{U}^I \mid \exists e_1,\ldots,\exists e_n \in \mathcal{U}^I :$
$|f^I(e)| = n \wedge f^I(e) = \{e_1,\ldots,e_n\} \wedge$
$e_1 \in [\![T_1]\!]^{\mathcal{I},\alpha} \wedge \ldots \wedge e_n \in [\![T_n]\!]^{\mathcal{I},\alpha}\}$
$[\![f:g(x)\cup h(y)]\!]^{\mathcal{I},\alpha} =$
$\{e \in \mathcal{U}^I \mid f^I(e) = g^I(\alpha(x)) \cup h^I(\alpha(y))\}$
$[\![f:g(x)\cap h(y)]\!]^{\mathcal{I},\alpha} =$
$\{e \in \mathcal{U}^I \mid f^I(e) = g^I(\alpha(x)) \cap h^I(\alpha(y))\}$
$[\![f:\supseteq g(x)]\!]^{\mathcal{I},\alpha} =$
$\{e \in \mathcal{U}^I \mid f^I(e) \supseteq g^I(\alpha(x))\}$
$[\![f(x)\neq g(y)]\!]^{\mathcal{I},\alpha} =$

- $\emptyset$ if $f^I(\alpha(x)) \cap g^I(\alpha(y)) \neq \emptyset$
- $\mathcal{U}^I$ if $f^I(\alpha(x)) \cap g^I(\alpha(y)) = \emptyset$

$[\![S \sqcap T]\!]^{\mathcal{I},\alpha} = [\![S]\!]^{\mathcal{I},\alpha} \cap [\![T]\!]^{\mathcal{I},\alpha}$
$[\![\neg T]\!]^{\mathcal{I},\alpha} = \mathcal{U}^I - [\![T]\!]^{\mathcal{I},\alpha}$

The above definitions fix the syntax and semantics of every term.

It follows from the above definitions that:
$f:T \ \equiv \ f:\{T\} \ \equiv f:\{T\}_=$

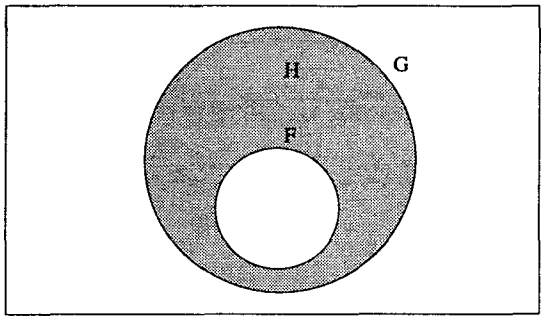


Figure 1

Although *disjoint union* is not a primitive in the logic it can easily be defined by employing set disjointness and set union operations:

$f:g(x) \uplus h(y) =_{def} \ g(x) \neq h(y) \sqcap f:g(x)\cup h(y)$

Thus disjoint set union is exactly like set union except that it additionally requires the sets denoted by $g(x)$ and $h(y)$ to be disjoint.

The set-valued description of the subcategorisation principle can now be stated as given in example (3).

(3) **Subcategorisation Principle**

$$\begin{bmatrix} \text{SYN|LOC } Y \\ \text{DTRS} \quad X \sqcap \left[\text{H-DTR|SYN|LOC|SUBCAT c-dtrs}(X) \uplus \text{subcat}(Y)\right] \end{bmatrix}$$

The description in (3) simply states that the subcat value of the H-DTR is the disjoint union of the subcat value of the mother and the values of C-DTRS. Note that the disjoint union operation is the right operation to be specified to split the set into two disjoint subsets. Employing just union operation would not work since

Decomposition rules

(DFeat) $\dfrac{x = F : T \wedge C_s}{x = F : y \wedge y = T \wedge C_s}$
if y is new and T is not a variable and F ranges over $\exists f, f$

(DForall) $\dfrac{x = \forall f : \bar{c} \wedge C_s}{x = \forall f : y \wedge y = \bar{c} \wedge C_s}$
if y is new and $\bar{c}$ ranges over a, c.

(DSet) $\dfrac{x = f : \{T_1, \ldots, T_n\} \wedge C_s}{x = f : \{x_1, \ldots, x_n\} \wedge x_1 = T_1 \wedge \ldots \wedge x_n = T_n \wedge C_s}$
if $x_1, \ldots, x_n$ are new and at least one of $T_i : 1 \leq i \leq n$ is not a variable

(DSetF) $\dfrac{x = f : \{T_1, \ldots, T_n\}_= \wedge C_s}{x = f : \{x_1, \ldots, x_n\} \wedge x = f : \{x_1, \ldots, x_n\}_= \wedge x_1 = T_1 \wedge \ldots \wedge x_n = T_n \wedge C_s}$
if $x_1, \ldots, x_n$ are new and at least one of $T_i : 1 \leq i \leq n$ is not a variable

(DConj) $\dfrac{x = S \sqcap T \wedge C_s}{x = S \wedge x = T \wedge C_s}$

Figure 2: Decomposition rules

it would permit repetition between members of the SUBCAT attribute and C-DTRS attribute.

Alternatively, we can assume that N is the only multi-valued relation symbol while both SUBCAT and C-DTRS are single-valued and then employ the intuitively appealing subcategorisation principle given in (4).

(4) **Subcategorisation Principle**

$$\left[\begin{array}{ll} \text{SYN|LOC|SUBCAT } Y & \\ \text{DTRS} & \left[\begin{array}{l} \text{H-DTR|SYN|LOC|SUBCAT|N } \mathrm{N}(X) \uplus \mathrm{N}(Y) \\ \text{C-DTRS } \quad X \end{array}\right] \end{array}\right]$$

With the availability of set operations, multi-valued structures can be incrementally built. For instance, by employing union operations, semantic indices can be incrementally constructed and by employing membership constraints on the set of semantic indices pronoun resolution may be carried out.

The set difference operation $f : g(y) - h(z)$ is not available from the constructs described so far. However, assume that we are given the term $x \sqcap f : g(y) - h(z)$ and it is known that $h^{\mathcal{I}}(\alpha(z)) \subseteq g^{\mathcal{I}}(\alpha(y))$ for every interpretation $\mathcal{I}, \alpha$ such that $[\![x \sqcap f : g(y) - h(z)]\!]^{\mathcal{I},\alpha} \neq \emptyset$. Then the term $x \sqcap f : g(y) - h(z)$ (assuming the obvious interpretation for the set difference operation) is consistent iff the term $y \sqcap g : f(x) \uplus h(z)$ is consistent. This is so since for sets G, F, H : G - F = H $\wedge$ F $\subseteq$ G *iff* G = F $\uplus$ H. See figure 1 for verification.

3 Consistency checking

To employ a term language for knowledge representation tasks or in constraint programming languages the minimal operation that needs to be supported is that of consistency checking of terms.

A term T is **consistent** if there exists an interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment α such that $[\![T]\!]^{\mathcal{I},\alpha} \neq \emptyset$.

In order to develop constraint solving algorithms for consistency testing of terms we follow the approaches in [Smolka, 1992] [Hollunder and Nutt, 1990].

A **containment constraint** is a constraint of the form $x = T$ where x is a variable and T is an term.

Constraint simplification rules - I

(SEquals) $\dfrac{x = y \wedge C_s}{x = y \wedge [x/y]C_s}$
if $x \not\equiv y$ and x occurs in C_s

(SConst) $\dfrac{x = \bar{c} \wedge y = \bar{c} \wedge C_s}{x = y \wedge x = \bar{c} \wedge C_s}$
where $\bar{c}$ ranges over a, c.

(SFeat) $\dfrac{x = f : y \wedge x = F : z \wedge C_s}{x = f : y \wedge y = z \wedge C_s}$
where F ranges over $f, \exists f, \forall f$

(SExists) $\dfrac{x = \exists f : y \wedge x = \forall f : z \wedge C_s}{x = f : y \wedge y = z \wedge C_s}$

(SForallE) $\dfrac{x = \forall f : \overline{C} \wedge x = \exists f : y \wedge C_s}{x = \forall f : \overline{C} \wedge x = \exists f : y \wedge y = \overline{C} \wedge C_s}$
if $\overline{C}$ ranges over $C, \neg C, \neg a, \neg c, \neg z$ and $C_s \not\models y = \overline{C}$.

Figure 3: Constraint simplification rules - I

In addition, for the purposes of consistency checking we need to introduce **disjunctive constraints** which are of the form $x = x_1 \sqcup \ldots \sqcup x_n$.

We say that an interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment α satisfies a constraint K written $\mathcal{I}, \alpha \models K$ *if*:

- $\mathcal{I}, \alpha \models x = T \iff \alpha(x) \in [\![T]\!]^{\mathcal{I},\alpha}$
- $\mathcal{I}, \alpha \models x = x_1 \sqcup \ldots \sqcup x_n \iff \alpha(x) = \alpha(x_i)$ for some $x_i : 1 \leq i \leq n$.

A **constraint system** C_s is a *conjunction* of constraints.

We say that an interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment α **satisfy** a constraint system C_s *iff* $\mathcal{I}, \alpha$ satisfies every constraint in C_s.

The following lemma demonstrates the usefulness of constraint systems for the purposes of consistency checking.

Lemma 1 *An term T is consistent iff there exists a variable x, an interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment α such that $\mathcal{I}, \alpha$ satisfies the constraint system $x = T$.*

Now we are ready to turn our attention to constraint solving rules that will allow us to determine the consistency of a given constraint system.

Constraint simplification rules - II

(SSetF) $\dfrac{x = F : y \wedge x = f : \{x_1, \ldots, x_n\} \wedge C_s}{x = f : y \wedge y = x_1 \wedge \ldots \wedge y = x_n \wedge C_s}$
where F ranges over $f, \forall f$

(SSet) $\dfrac{x = f : \{y\} \wedge C_s}{x = f : y \wedge C_s}$

(SDup) $\dfrac{x = f : \{x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n\} \wedge C_s}{x = f : \{x_1, \ldots, x_i, \ldots, \ldots, x_n\} \wedge C_s}$
if $x_i \equiv x_j$

(SForall) $\dfrac{x = \forall f : \overline{C} \wedge \ x = f : \{x_1, \ldots, x_n\} \wedge C_s}{x = f : \{x_1, \ldots, x_n\} \wedge x_1 = \overline{C} \wedge \ldots \wedge x_n = \overline{C} \wedge C_s}$
if $\overline{C}$ ranges over $C, \neg C, \neg a, \neg c, \neg z$ and
there exists $x_i : 1 \leq i \leq n$ such that $C_s \not\vdash x_i = \overline{C}$.

(SSetE) $\dfrac{x = \exists f : y \wedge x = f : \{x_1, \ldots, x_n\} \wedge C_s}{x = f : \{x_1, \ldots, x_n\} \wedge y = x_1 \sqcup \ldots \sqcup x_n \wedge C_s}$

(SSetSet) $\dfrac{x = f : \{x_1, \ldots, x_n\} \wedge x = f : \{y_1, \ldots, y_m\} \wedge C_s}{\begin{array}{l} x = f : \{x_1, \ldots, x_n\} \wedge \\ x_1 = y_1 \sqcup \ldots \sqcup y_m \wedge \ldots \wedge x_n = y_1 \sqcup \ldots \sqcup y_m \wedge \\ y_1 = x_1 \sqcup \ldots \sqcup x_n \wedge \ldots \wedge y_m = x_1 \sqcup \ldots \sqcup x_n \wedge C_s \end{array}}$
where $n \leq m$

(SDis) $\dfrac{x = x_1 \sqcup \ldots \sqcup x_n \wedge C_s}{x = x_1 \sqcup \ldots \sqcup x_n \wedge x = x_i \wedge C_s}$
if $1 \leq i \leq n$ and
there is no $x_j, 1 \leq j \leq n$ such that $C_s \vdash x = x_j$

Figure 4: Constraint simplification rules - II

We say that a constraint system C_s is **basic** if *none* of the *decomposition rules* (see figure 2) are applicable to C_s.

The purpose of the decomposition rules is to break down a complex constraint into possibly a number of simpler constraints upon which the constraint simplification rules (see figures 3, 4 and 5) can apply by possibly introducing new variables.

The first phase of consistency checking of a term T consists of exhaustively applying the decomposition rules to an initial constraint of the form $x = T$ (where x does not occur in T) until no rules are applicable. This transforms any given constraint system into *basic form*.

The constraint simplification rules (see figures 3, 4 and 5) either eliminate variable equalities of the form $x = y$ or generate them from existing constraints. However, they do not introduce new variables.

The constraint simplification rules given in figure 3 are the analog of the feature simplification rules provided in [Smolka, 1991]. The main difference being that our simplification rules have been modified to deal with relation symbols as opposed to just feature symbols.

The constraint simplification rules given in figure 4 simplify constraints involving set descriptions when they interact with other constraints such as feature constraints - rule (**SSetF**), singleton sets - rule (**SSet**), duplicate elements in a set - rule (**SDup**), universally quantified constraint - rule (**SForall**), another set description - rule (**SSetSet**). Rule (**SDis**) on the other hand simplifies disjunctive constraints. Amongst all the constraint simplification rules in figures 3 and 4 only rule (**SDis**) is non-deterministic and creates a n-ary choice point.

Rules (**SSet**) and (**SDup**) are redundant as completeness (see section below) is not affected by these rules. However these rules result in a simpler normal form.

The following syntactic notion of entailment is employed to render a slightly compact presentation of the constraint solving rules for dealing with set operations given in figure 5.

A constraint system C_s *syntactically entails* the (conjunction of) constraint(s) ϕ if $C_s \vdash \phi$ is derivable from the following deduction rules:

1. $\phi \wedge C_s \vdash \phi$
2. $C_s \vdash x = x$
3. $C_s \vdash x = y \longrightarrow C_s \vdash y = x$
4. $C_s \vdash x = y \wedge C_s \vdash y = z \longrightarrow C_s \vdash x = z$
5. $C_s \vdash x = \neg y \longrightarrow C_s \vdash y = \neg x$
6. $C_s \vdash x = f : y \longrightarrow C_s \vdash x = \exists f : y$
7. $C_s \vdash x = f : y \longrightarrow C_s \vdash x = \forall f : y$
8. $C_s \vdash x = f : \{\ldots, x_i, \ldots\} \longrightarrow C_s \vdash x = \exists f : x_i$

Note that the above definitions are an incomplete list of deduction rules. However $C_s \vdash \phi$ implies $C_s \models \phi$ where $\models$ is the semantic entailment relation defined as for predicate logic.

We write $C_s \not\vdash \phi$ if it is not the case that $C_s \vdash \phi$.

The constraint simplification rules given in figure 5 deal with constraints involving set operations. Rule ($\subseteq$) propagates g-values of y into f-values of x in the presence of the constraint $x = f :\supseteq g(y)$. Rule

Extended Constraint simplification rules

$$(\subseteq)\quad \frac{x = f :\supseteq g(y) \wedge C_s}{x = f :\supseteq g(y) \wedge x = \exists f : y_i \wedge C_s}$$
if:
- $C_s \not\vdash x = \exists f : y_i$ and
- $C_s \vdash y = \exists g : y_i$

$$(\cup Left)\quad \frac{x = f : g(y) \cup h(z) \wedge C_s}{x = f : g(y) \cup h(z) \wedge x = f :\supseteq g(y) \wedge C_s}$$
if $C_s \not\vdash x = f :\supseteq g(y)$

$$(\cup Right)\quad \frac{x = f : g(y) \cup h(z) \wedge C_s}{x = f : g(y) \cup h(z) \wedge x = f :\supseteq h(z) \wedge C_s}$$
if $C_s \not\vdash x = f :\supseteq h(z)$

$$(\cup Down)\quad \frac{x = f : g(y) \cup h(z) \wedge C_s}{x = f : g(y) \cup h(z) \wedge y = \exists g : x_i \mid z = \exists h : x_i \wedge C_s}$$
if:
- $C_s \not\vdash y = \exists g : x_i$ and
- $C_s \not\vdash z = \exists h : x_i$ and
- $C_s \vdash x = \exists f : x_i$

$$(\cap Down)\quad \frac{x = f : g(y) \cap h(z) \wedge C_s}{x = f : g(y) \cap h(z) \wedge y = \exists g : x_i \wedge z = \exists h : x_i \wedge C_s}$$
if:
- $(C_s \not\vdash y = \exists g : x_i$ or $C_s \not\vdash z = \exists h : x_i)$ and
- $C_s \vdash x = \exists f : x_i$

$$(\cap Up)\quad \frac{x = f : g(y) \cap h(z) \wedge C_s}{x = f : g(y) \cap h(z) \wedge x = \exists f : x_i \wedge C_s}$$
if:
- $C_s \not\vdash x = \exists f : x_i$ and
- $C_s \vdash y = \exists g : x_i$ and
- $C_s \vdash z = \exists h : x_i$

Figure 5: Constraint solving with set operations

($\cup Left$) (correspondingly Rule ($\cup Right$)) adds the constraint $x = f :\supseteq g(y)$ (correspondingly $x = f :\supseteq h(z)$) in the presence of the constraint $x = f : g(y) \cup h(z)$. Also in the presence of $x = f : g(y) \cup h(z)$ rule ($\cup Down$) non-deterministically propagates an f-value of x to either an g-value of y or an h-value of z (if neither already holds). The notation $y = \exists g : x_i \mid z = \exists h : x_i$ denotes a non-deterministic choice between $y = \exists g : x_i$ and $z = \exists h : x_i$. Rule ($\cap Down$) propagates an f-value of x both as a g-value of y and h-value of z in the presence of the constraint $x = f : g(y) \cap h(z)$. Finally, rule ($\cap Up$) propagates a common g-value of y and h-value of z as an f-value of x in the presence of the constraint $x = f : g(y) \cap h(z)$.

4 Invariance, Completeness and Termination

In this section we establish the main results of this paper - namely that our consistency checking procedure for set descriptions and set operations is invariant, complete and terminating. In other words, we have a decision procedure for determining the consistency of terms in our extended feature logic.

For the purpose of showing *invariance* of our rules we distinguish between *deterministic* and *non-deterministic* rules. Amongst all our rules only rule (**SDis**) given in figure 4 and rule ($\cup Down$) are non-deterministic while all the other rules are deterministic.

Theorem 2 (Invariance) *1. If a decomposition rule transforms C_s to C'_s then C_s is consistent iff C'_s is consistent.*

2. Let $\mathcal{I}, \alpha$ be any interpretation, assignment pair and let C_s be any constraint system.
- *If a deterministic simplification rule transforms C_s to C'_s then: $\mathcal{I}, \alpha \models C_s$ iff $\mathcal{I}, \alpha \models C'_s$*
- *If a non-deterministic simplification rule applies to C_s then there is at least one non-deterministic choice which transforms C_s to C'_s such that: $\mathcal{I}, \alpha \models C_s$ iff $\mathcal{I}, \alpha \models C'_s$*

A constraint system C_s is in **normal form** if no rules are applicable to C_s.

Let $succ(x, f)$ denote the set:
$succ(x, f) = \{y \mid C_s \vdash x = \exists f : y\}$

A constraint system C_s in normal form contains a **clash** if there exists a variable x in C_s such that *any* of the following conditions are satisfied :
1. $C_s \vdash x = a_1$ and $C_s \vdash x = a_2$ such that $a_1 \not\equiv a_2$
2. $C_s \vdash x = c_1$ and $C_s \vdash x = c_2$ such that $c_1 \not\equiv c_2$
3. $C_s \vdash x = \overline{S}$ and $C_s \vdash x = \neg\overline{S}$ where $\overline{S}$ ranges over x, a, c, C.
4. $C_s \vdash x = \exists f : y$ and $C_s \vdash x = a$
5. $C_s \vdash f(x) \neq g(y)$ and $succ(x, f) \cap succ(y, g) \neq \emptyset$
6. $C_s \vdash x = f : \{x_1, \ldots, x_n\}_=$ and $|succ(x, f)| < n$

If C_s does not contain a clash then C_s is called **clash-free**.

The constraint solving process can terminate as soon as a *clash-free* constraint system in normal form is found or alternatively all the choice points are exhausted.

The purpose of the *clash* definition is highlighted in the *completeness* theorem given below.

For a constraint system C_s in normal form an *equivalence relation* $\simeq$ on variables occurring in C_s is defined as follows:
$x \simeq y$ if $C_s \vdash x = y$

For a variable x we represent its equivalence class by $[x]$.

Theorem 3 (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 $\mathcal{I}$-assignment α which satisfies C_s.

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

We shall construct an interpretation $\mathcal{R} =< U^R, .^R >$

and a variable assignment α such that $\mathcal{R}, \alpha \models C_s$.

Let $\mathcal{U}^R = \mathcal{V} \cup \mathcal{A}t \cup \mathcal{C}$.

The assignment function α is defined as follows:

1. For every variable x in $\mathcal{V}$
   (a) if $C_s \vdash x = a$ then $\alpha(x) = a$
   (b) if the previous condition does not apply then $\alpha(x) = choose([x])$ where $choose([x])$ denotes a unique representative (chosen arbitrarily) from the equivalence class $[x]$.
2. For every constant c in $\mathcal{C}$:
   (a) if $C_s \vdash x = c$ then $\alpha(c) = \alpha(x)$
   (b) if c is a constant such that the previous condition does not apply then $\alpha(c) = c$
3. For every primitive concept C in $\mathcal{P}$:
   $\alpha(C) = \{\alpha(x) \mid C_s \vdash x = c\}$

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

- $f^R(x) = succ(x, f)$
- $a^R = a$

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.

Theorem 4 (Termination)
The consistency checking procedure terminates in a finite number of steps.

Proof Sketch: Termination is obvious if we observe the following properties:

1. Since decomposition rules breakdown terms into smaller ones these rules must terminate.
2. None of the simplification rules introduce new variables and hence there is an upper bound on the number of variables.
3. Every simplification rule does either of the following:
   (a) reduces the 'effective' number of variables.
   A variable x is considered to be *ineffective* if it occurs only once in C_s within the constraint $x = y$ such that rule (**SEquals**) does not apply. A variable that is not *ineffective* is considered to be *effective*.
   (b) adds a constraint of the form $x = \overline{C}$ where C ranges over $y, a, c, C, \neg y, \neg a, \neg c, \neg C$ which means there is an upper bound on the number of constraints of the form $x = \overline{C}$ that the simplification rules can add. This is so since the number of variables, atoms, constants and primitive concepts are bounded for every constraint system in basic form.
   (c) increases the size of $succ(x, f)$. But the size of $succ(x, f)$ is bounded by the number of variables in C_s which remains constant during the application of the simplification rules. Hence our constraint solving rules cannot indefinitely increase the size of $succ(x, f)$.

5 NP-completeness

In this section, we show that consistency checking of terms within the logic described in this paper is NP-complete. This result holds even if the terms involving set operations are excluded. We prove this result by providing a polynomial time translation of the well-known NP-complete problem of determining the satisfiability of propositional formulas [Garey and Johnson, 1979].

Theorem 5 (NP-Completeness) *Determining consistency of terms is NP-Complete.*

Proof: Let ϕ be any given propositional formula for which consistency is to be determined. We split our translation into two intuitive parts : *truth assignment* denoted by $\Delta(\phi)$ and *evaluation* denoted by $\tau(\phi)$.

Let $a, b, \ldots$ be the set of propositional variables occurring in ϕ. We translate every propositional variable a by a variable x_a in our logic. Let f be some relation symbol. Let $true, false$ be two atoms.

Furthermore, let $x_1, x_2, \ldots$ be a finite set of variables distinct from the ones introduced above.

We define the translation function $\Delta(\phi)$ by:

$$\begin{aligned}\Delta(\phi) = \quad & f : \{true, false\} \sqcap \\ & \exists f : x_a \sqcap \exists f : x_b \sqcap \ldots \sqcap \\ & \exists f : x_1 \sqcap \exists f : x_2 \sqcap \ldots\end{aligned}$$

The above description forces each of the variable $x_a, x_b, \ldots$ and each of the variables $x_1, x_2, \ldots$ to be either equivalent to *true* or *false*.

We define the evaluation function $\tau(\phi)$ by:

$\tau(a) = x_a$
$\tau(S \& T) = \tau(S) \sqcap \tau(T)$
$\tau(S \vee T) = x_i \sqcap \exists f : (f : \{\tau(S), \tau(T)\} \sqcap \exists f : x_i)$
where $x_i \in \{x_1, x_2, \ldots\}$ is a new variable
$\tau(\neg S) = x_i \sqcap \exists f : (\tau(S) \sqcap \neg x_i)$
where $x_i \in \{x_1, x_2, \ldots\}$ is a new variable

Intuitively speaking τ can be understood as follows. Evaluation of a propositional variable is just its value; evaluating a conjunction amounts to evaluating each of the conjuncts; evaluating a disjunction amounts to evaluating either of the disjuncts and finally evaluating a negation involves choosing something other than the value of the term.

Determining satisfiability of ϕ then amounts to determining the consistency of the following term:

$\exists f : \Delta(\phi) \sqcap \exists f : (true \sqcap \tau(\phi))$

Note that the term $true \sqcap \tau(\phi)$ forces the value of $\tau(\phi)$ to be *true*. This translation demonstrates that determining consistency of terms is NP-hard.

On the other hand, every deterministic completion of our constraint solving rules terminate in polynomial time since they do not generate new variables and the number of new constraints are polynomially bounded. This means determining consistency of terms is NP-easy. Hence, we conclude that determining consistency of terms is NP-complete.

6 Translation to Schönfinkel-Bernays class

The Schönfinkel-Bernays class (see [Lewis, 1980]) consists of function-free first-order formulae which have

the form:

$\exists x_1 \ldots x_n \forall y_1 \ldots y_m \delta$

In this section we show that the attributive logic developed in this paper can be encoded within the Schönfinkel-Bernays subclass of first-order formulae by extending the approach developed in [Johnson, 1991]. However formulae such as $\forall f : (\exists f : (\forall f : T))$ which involve an embedded existential quantification cannot be translated into the Schönfinkel-Bernays class. This means that an unrestricted variant of our logic which does not restrict the universal role quantification cannot be expressed within the Schönfinkel-Bernays class.

In order to put things more concretely, we provide a translation of every construct in our logic into the Schönfinkel-Bernays class.

Let T be any extended feature term. Let x be a variable *free* in T. Then T is consistent *iff* the formula $(x = T)^\delta$ is consistent where δ is a translation function from our extended feature logic into the Schönfinkel-Bernays class. Here we provide only the essential definitions of δ:

- $(x = a)^\delta = x = a$
- $(x = \neg a)^\delta = x \neq a$
- $(x = f : T)^\delta =$
  $f(x, y)\ \&\ (y = T)^\delta\ \&\ \forall y'(f(x, y') \rightarrow y = y')$
  where y is a new variable
- $(x = \exists f : T)^\delta = f(x, y)\ \&\ (y = T)^\delta$
  where y is a new variable
- $(x = \forall f : a)^\delta = \forall y(f(x, y) \rightarrow y = a)$
- $(x = \forall f : \neg a)^\delta = \forall y(f(x, y) \rightarrow y \neq a)$
- $(x = f : \{T_1, \ldots, T_n\})^\delta =$
  $f(x, x_1)\ \&\ \ldots\ \&\ f(x, x_n) \&$
  $\forall y(f(x, y) \rightarrow y = x_1 \vee \ldots \vee y = x_n) \&$
  $(x_1 = T_1)^\delta\ \&\ \ldots\ \&\ (x_1 = T_n)^\delta$
  where $x_1, \ldots, x_n$ are new variables
- $(x = f : g(y) \cup h(z))^\delta =$
  $\forall x_i(f(x, x_i) \rightarrow g(y, x_i) \vee h(z, x_i))\ \&$
  $\forall y_i(g(y, y_i) \rightarrow f(x, y_i))\ \&$
  $\forall z_i(h(z, z_i) \rightarrow f(x, z_i))$
- $(x = f : (y) \neq g(z))^\delta =$
  $\forall y_i z_j(f(y, y_i)\ \&\ g(z, z_i) \rightarrow y_i \neq z_i)$
- $(x = S \sqcap T)^\delta = (x = S)^\delta\ \&\ (x = T)^\delta$

These translation rules essentially mimic the decomposition rules given in figure 2.

Furthermore for every atom a and every feature f in T we need the following axiom:

- $\forall a x(\neg f(a, x))$

For every distinct atoms a, b in T we need the axiom:

- $a \neq b$

Taking into account the NP-completeness result established earlier this translation identifies a NP-complete subclass of formulae within the Schönfinkel-Bernays class which is suited for NL applications.

7 Related Work

Feature logics and concept languages such as KL-ONE are closely related family of languages [Nebel and Smolka, 1991]. The principal difference being that feature logics interpret attributive labels as functional binary relations while concept languages interpret them as just binary relations. However the integration of concept languages with feature logics has been problematic due to the fact the while path equations do not lead to increased computational complexity in feature logic the addition of role-value-maps (which are the relational analog of path equations) in concept languages causes undecidability [Schmidt-Schauß, 1989]. This blocks a straightforward integration of a variable-free concept language such as ALC [Schmidt-Schauß and Smolka, 1991] with a variable-free feature logic [Smolka, 1991].

In [Manandhar, 1993] the addition of variables, feature symbols and set descriptions to ALC is investigated providing an alternative method for integrating concept languages and feature logics. It is shown that set descriptions can be translated into the so called "number restrictions" available within concept languages such as BACK [von Luck *et al.*, 1987]. However, the propositionally complete languages ALV and ALS investigated in [Manandhar, 1993] are PSPACE-hard languages which do not support set operations.

The work described in this paper describes yet another unexplored dimension for concept languages - that of a restricted concept language with variables, feature symbols, set descriptions and set operations for which the consistency checking problem is within the complexity class NP.

8 Summary and Conclusions

In this paper we have provided an extended feature logic (excluding disjunctions and negations) with a range of constraints involving set descriptions. These constraints are set descriptions, fixed cardinality set descriptions, set-membership constraints, restricted universal role quantifications, set union, set intersection, subset and disjointness. We have given a model theoretic semantics to our extended logic which shows that a simple and elegant formalisation of set descriptions is possible if we add relational attributes to our logic as opposed to just functional attributes available in feature logic.

For realistic implementation of the logic described in this paper, further investigation is needed to develop concrete algorithms that are reasonably efficient in the average case. The consistency checking procedure described in this paper abstracts away from algorithmic considerations and clearly modest improvements to the basic algorithm suggested in this paper are feasible. However, a report on such improvements is beyond the scope of this paper.

For applications within constraint based grammar formalisms such as HPSG, minimally a type system [Carpenter, 1992] and/or a Horn-like extension [Höhfeld and Smolka, 1988] will be necessary.

We believe that the logic described in this paper provides both a better picture of the formal aspects of

current constraint based grammar formalisms which employ set descriptions and at the same time gives a basis for building knowledge representation tools in order to support grammar development within these formalisms.

9 Acknowledgments

The work described here has been carried out as part of the EC-funded project LRE-61-061 RGR (Reusability of Grammatical Resources). A longer version of the paper is available in [Erbach *et al.*, 1993]. The work described is a further development of the author's PhD thesis carried out at the Department of Artificial Intelligence, University of Edinburgh. I thank my supervisors Chris Mellish and Alan Smaill for their guidance. I have also benefited from comments by an anonymous reviewer and discussions with Chris Brew, Bob Carpenter, Jochen Dörre and Herbert Ruessink.

The Human Communication Research Centre (HCRC) is supported by the Economic and Social Research Council (UK).

References

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

[Carpenter, 1993] Bob Carpenter. ALE:Attribute Logic Engine Users Guide, Version β. Technical report, Carnegie Mellon University, Pittsburgh, PA 15213, 1993.

[Dörre and Dorna, 1993] Jochen Dörre and Michael Dorna. CUF: A Formalism for Linguistic Knowledge Representation. Dyana-2 deliverable, IMS, Stuttgart, Germany, August 1993.

[Erbach *et al.*, 1993] Gregor Erbach, Mark van der Kraan, Suresh Manandhar, M. Andrew Moshier, Herbert Ruessink, and Craig Thiersch. Specification of Datatypes. In *Deliverable D.B of LRE-61-061 "The Reusability of Grammatical Resources"*. 1993.

[Garey and Johnson, 1979] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[Höhfeld and Smolka, 1988] Markus Höhfeld and Gert Smolka. Definite relations over constraint languages. LILOG Report 53, IBM Deutschland, Stuttgart, Germany, October 1988.

[Hollunder and Nutt, 1990] B. Hollunder and W. Nutt. Subsumption Algorithms for Concept Languages. Research Report RR-90-04, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, 6600 Saarbrücken 11, Germany, 1990.

[Johnson, 1991] Mark Johnson. Features and Formulae. *Computational Linguistics*, 17(2):131–151, June 1991.

[Kaplan and Bresnan, 1982] Ronald M. Kaplan and Joan Bresnan. Lexical-Functional Grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173 – 281. MIT Press, Cambridge, Massachussets, 1982.

[Kasper and Rounds, 1986] Robert Kasper and William Rounds. A logical semantics for feature structures. In *24th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York*, pages 257–265, 1986.

[Lewis, 1980] Harry R. Lewis. Complexity Results for Classes of Quantificational Formulae. *Journal of Computer and System Sciences*, 21:317–353, 1980.

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

[Nebel and Smolka, 1991] Bernhard Nebel and Gert Smolka. Attributive description formalisms and the rest of the world. Research Report RR-91-15, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany, May 1991.

[Pollard and Moshier, 1990] Carl J. Pollard and M. Drew Moshier. Unifying partial descriptions of sets. In Philip P. Hanson, editor, *Information, Language and Cognition.* University of British Columbia Press, Vancouver, Canada, 1990. Vancouver Studies in Cognitive Science, no. 1.

[Pollard and Sag, 1987] Carl Pollard and Ivan Andrew Sag. *Information-Based Syntax and Semantics: Volume 1 Fundamentals*, volume 13 of *Lecture Notes.* Center for the Study of Language and Information, Stanford, CA, 1987.

[Pollard and Sag, 1992] Carl Pollard and Ivan Andrew Sag. *Head-driven Phrase Structure Grammar: Volume 2.* MIT Press, 1992. Forthcoming.

[Rounds, 1988] William C. Rounds. Set values for unification-based grammar formalisms and logic programming. Technical report, Center for the Study of Language and Information, Stanford, CA, 1988.

[Schmidt-Schauß and Smolka, 1991] Manfred Schmidt-Schauß and Gert Smolka. Attributive Concept Descriptions with Unions and Complements. *Artificial Intelligence*, 48:1–26, 1991. Also available as IWBS Report 68, IBM Germany, Scientific Center, IWBS, Stuttgart, Germary, June 1989.

[Schmidt-Schauß, 1989] Manfred Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In *First International Conference on Principles of Knowledge Representation and Reasoning, KR' 89, Toronto, Canada*, pages 421–431, May 1989.

[Smolka, 1991] Gert Smolka. A feature logic with subsorts. In Jürgen Wedekind and C. Rohrer (eds.), editors, *Unification in Grammar.* MIT Press, 1991. Also appeared as LILOG Report no. 33, IWBS, IBM Deutschland.

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

[von Luck *et al.*, 1987] K. von Luck, B. Nebel, C. Peltason, and A. Schmiedel. The Anatomy of the BACK System. KIT Report 41, Department of Computer Science, Technische Universität Berlin, Berlin, Germany, 1987.

[Zajac, 1992] Rémi Zajac. Inheritance and Constraint-Based Grammar Formalisms. *Computational Linguistics*, 18(2):159–182, 1992.