
An Attributive Logic of Set Descriptions and Set Operations: Extended Report

Suresh Manandhar

1	Introduction and Background	39
2	The logic of Set descriptions	41
3	Consistency checking	44
4	Invariance, Completeness and Termination	49
5	Canonical set unifiers	53
6	NP-completeness	56
7	Translation to Schönfinkel-Bernays class	57
8	Summary and Conclusions	58
	References	59

Abstract

Set descriptions are regarded as an important datatype within unification based linguistic theories. Although several formalisations of set descriptions exists consistency checking methods for set descriptions have not been developed. This report attempts to rectify this situation in several ways. We begin by providing 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 and restricted universal role quantifications. This repertoire of constraints on set descriptions is extended by allowing constraints such as 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 consistency checking with set descriptions is a NP-complete problem.

1 Introduction and Background

Grammatical formalisms such as HPSG (Pollard & Sag, 1987) (Pollard & Sag, 1994) and LFG (Kaplan & Bresnan, 1982) employ feature descriptions (Kasper & 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 & Dorna, 1993) which augment feature logic with a type system. One such expressive device employed both within LFG (Kaplan, 1989)

and HPSG but is unavailable in feature logic is that of set-valued feature descriptions or set descriptions for short.

Although various researchers have studied set descriptions (with different semantics) (Rounds, 1988) (Pollard & 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 needed within HPSG. 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 & Sag, 1987), set descriptions are employed for the modelling of so called *semantic indices* ((Pollard & Sag, 1987) pp. 104). The following attribute-value matrix notation is intended to represent the semantics of the sentence *Kim sees Sandy*.

$$(1) \quad \left[\begin{array}{l} \text{CONT} \left[\begin{array}{l} \text{REL}_{see} \\ \text{SEER} \boxed{2} \\ \text{SEEN} \boxed{1} \end{array} \right] \\ \text{INDS} \left\{ \left[\begin{array}{l} \text{VAR} \boxed{1} \\ \text{REST} \left[\begin{array}{l} \text{RELN}_{naming} \\ \text{NAME}_{sandy} \\ \text{NAMED} \boxed{1} \end{array} \right] \end{array} \right], \left[\begin{array}{l} \text{VAR} \boxed{2} \\ \text{REST} \left[\begin{array}{l} \text{RELN}_{naming} \\ \text{NAME}_{kim} \\ \text{NAMED} \boxed{2} \end{array} \right] \end{array} \right] \right\} \end{array} \right]$$

The value of the CONT label represents the semantics of the sentence which roughly means *see(kim, sandy)*. The value of the INDS label is a set known as the *semantic indices* which denotes the objects involved in the *seeing* situation. In this case these objects are *Kim* and *Sandy*.

The boxed numbers $\boxed{1}$ and $\boxed{2}$ represent variables and serve as co-reference markers. Semantic indices in HPSG are intended to serve as possible *pronoun antecedents*.

The example in (1) can be easily be encoded by employing set descriptions as shown in example (2).

$$(2) \quad \begin{array}{l} cont : (\quad rel : see \quad \sqcap \\ \quad \quad seer : y \quad \quad \sqcap \\ \quad \quad seen : x \quad \quad) \sqcap \\ inds : \{ \quad var : x \quad \sqcap \\ \quad \quad rest : (\quad reln : naming \quad \sqcap \\ \quad \quad \quad \quad name : sandy \quad \sqcap \\ \quad \quad \quad \quad named : x \quad \quad \quad) , \\ \quad \quad var : y \quad \sqcap \\ \quad \quad rest : (\quad reln : naming \quad \sqcap \\ \quad \quad \quad \quad name : kim \quad \quad \sqcap \\ \quad \quad \quad \quad named : y \quad \quad \quad) \} \end{array}$$

To be able to deal with anaphoric dependencies we posit that minimally mechanisms to state

set memberships to resolve pronoun dependencies and set unions to incrementally construct discourse referents will be necessary.

Similarly, set valued subcategorisation frames (see (3)) have been considered as a possibility within the HPSG formalism.

$$(3) \textit{believes} = \begin{array}{l} \textit{syn} : \textit{loc} : \textit{subcat} : \{ \textit{syn} : \textit{loc} : (\textit{head} : n \sqcap \textit{subcat} : \{ \}), \\ \textit{syn} : \textit{loc} : (\textit{head} : v \sqcap \textit{subcat} : \{ \}) \} \end{array}$$

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 $x, y, z, \dots \in \mathcal{V}$ the set of *variables*; $f, g, \dots \in \mathcal{F}$ the set of *relation symbols*; $c_1, c_2, \dots \in \mathcal{C}$ the set of *constant symbols*; $A, B, C, \dots \in \mathcal{P}$ the set of primitive concept symbols and $a, b, \dots \in \mathcal{At}$ the set of *atomic symbols*. Furthermore, we require that $\perp, \top \in \mathcal{P}$.

The syntax of our term language is defined by the following BNF definition:

$$\begin{array}{l} P \perp \rightarrow x \mid a \mid c \mid C \mid \neg x \mid \neg a \mid \neg c \mid \neg C \\ S, T \perp \rightarrow \\ \quad P \\ \quad \mid f : T \quad \text{feature term} \\ \quad \mid \exists f : T \quad \text{existential role quantification} \\ \quad \mid \forall f : P \quad \text{universal role quantification} \\ \quad \mid f : \{T_1, \dots, T_n\} \quad \text{set description} \\ \quad \mid f : \{T_1, \dots, T_n\}_= \quad \text{fixed cardinality set description} \\ \quad \mid f : g(x) \cup h(y) \quad \text{union} \\ \quad \mid f : g(x) \cap h(y) \quad \text{intersection} \\ \quad \mid f : \supseteq g(x) \quad \text{subset} \\ \quad \mid f(x) \neq g(y) \quad \text{disjointness} \\ \quad \mid S \sqcap T \quad \text{conjunction} \end{array}$$

where S, T 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 $\mathcal{I} = \langle \mathcal{U}^I, I \rangle$ 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 $a \in \mathcal{At}$ 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 $a \in \mathcal{At}$ 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:
 - $\alpha(\perp) = \emptyset$
 - $\alpha(\top) = \mathcal{U}^I$

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

The function $\llbracket \cdot \rrbracket^{\mathcal{I}, \alpha}$ is defined as follows :

$$\llbracket x \rrbracket^{\mathcal{I}, \alpha} = \{\alpha(x)\}$$

$$\llbracket a \rrbracket^{\mathcal{I}, \alpha} = \{a^I\}$$

$$\llbracket c \rrbracket^{\mathcal{I}, \alpha} = \{\alpha(c)\}$$

$$\llbracket C \rrbracket^{\mathcal{I}, \alpha} = \alpha(C)$$

$$\llbracket f : T \rrbracket^{\mathcal{I}, \alpha} = \{e \in \mathcal{U}^I \mid \exists e' \in \mathcal{U}^I : f^I(e) = \{e'\} \wedge e' \in \llbracket T \rrbracket^{\mathcal{I}, \alpha}\}$$

$$\llbracket \exists f : T \rrbracket^{\mathcal{I}, \alpha} = \{e \in \mathcal{U}^I \mid \exists e' \in \mathcal{U}^I : (e, e') \in f^I \wedge e' \in \llbracket T \rrbracket^{\mathcal{I}, \alpha}\}$$

$$\llbracket \forall f : T \rrbracket^{\mathcal{I}, \alpha} = \{e \in \mathcal{U}^I \mid \forall e' \in \mathcal{U}^I : (e, e') \in f^I \Rightarrow e' \in \llbracket T \rrbracket^{\mathcal{I}, \alpha}\}$$

$$\begin{aligned} \llbracket f : \{T_1, \dots, T_n\} \rrbracket^{\mathcal{I}, \alpha} = \\ \{e \in \mathcal{U}^I \mid \exists e_1, \dots, \exists e_n \in \mathcal{U}^I : \\ f^I(e) = \{e_1, \dots, e_n\} \wedge \\ e_1 \in \llbracket T_1 \rrbracket^{\mathcal{I}, \alpha} \wedge \dots \wedge e_n \in \llbracket T_n \rrbracket^{\mathcal{I}, \alpha}\} \end{aligned}$$

$$\begin{aligned} \llbracket f : \{T_1, \dots, T_n\} = \rrbracket^{I, \alpha} &= \\ &\{e \in \mathcal{U}^I \mid \exists e_1, \dots, \exists e_n \in \mathcal{U}^I : \\ &\quad |f^I(e)| = n \wedge f^I(e) = \{e_1, \dots, e_n\} \wedge \\ &\quad e_1 \in \llbracket T_1 \rrbracket^{I, \alpha} \wedge \dots \wedge e_n \in \llbracket T_n \rrbracket^{I, \alpha}\} \\ \llbracket f : g(x) \cup h(y) \rrbracket^{I, \alpha} &= \\ &\{e \in \mathcal{U}^I \mid f^I(e) = g^I(\alpha(x)) \cup h^I(\alpha(y))\} \\ \llbracket f : g(x) \cap h(y) \rrbracket^{I, \alpha} &= \\ &\{e \in \mathcal{U}^I \mid f^I(e) = g^I(\alpha(x)) \cap h^I(\alpha(y))\} \\ \llbracket f : \supseteq g(x) \rrbracket^{I, \alpha} &= \\ &\{e \in \mathcal{U}^I \mid f^I(e) \supseteq g^I(\alpha(x))\} \\ \llbracket f(x) \neq g(y) \rrbracket^{I, \alpha} &= \\ &\bullet \emptyset \text{ if } f^I(\alpha(x)) \cap g^I(\alpha(y)) \neq \emptyset \\ &\bullet \mathcal{U}^I \text{ if } f^I(\alpha(x)) \cap g^I(\alpha(y)) = \emptyset \\ \llbracket S \sqcap T \rrbracket^{I, \alpha} &= \llbracket S \rrbracket^{I, \alpha} \cap \llbracket T \rrbracket^{I, \alpha} \\ \llbracket \neg T \rrbracket^{I, \alpha} &= \mathcal{U}^I \perp \llbracket T \rrbracket^{I, \alpha} \end{aligned}$$

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

It follows from the above definitions that:

$$f : T \equiv f : \{T\} \equiv f : \{T\} =$$

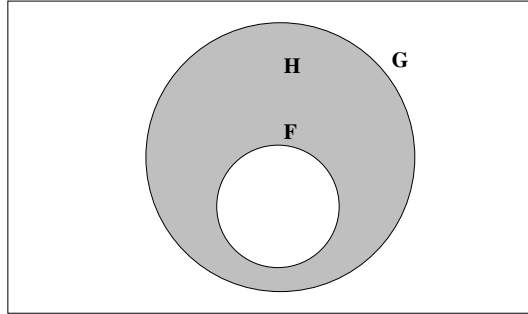


Figure 1: Set Difference

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 (4).

(4) **Subcategorisation Principle**

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

The description in (4) 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 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 (5).

(5) **Subcategorisation Principle**

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

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 $f : g(y) \perp h(z)$ is not available from the constructs described so far. However, assume that we are given the term $x \sqcap f : g(y) \perp h(z)$ and it is known that $h^{\mathcal{I}}(\alpha(z)) \subseteq g^{\mathcal{I}}(\alpha(y))$ for every interpretation \mathcal{I}, α such that $\llbracket x \sqcap f : g(y) \perp h(z) \rrbracket^{\mathcal{I}, \alpha} \neq \emptyset$. Then the term $x \sqcap f : g(y) \perp h(z)$ (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 $\llbracket T \rrbracket^{\mathcal{I}, \alpha} \neq \emptyset$.

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

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

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

Decomposition rules	
(DFeat)	$\frac{x = F : T \wedge C_s}{x = F : y \wedge y = T \wedge C_s}$ if y is new and T is not a variable and F ranges over $\exists f, f$
(DForall)	$\frac{x = \forall f : \bar{c} \wedge C_s}{x = \forall f : y \wedge y = \bar{c} \wedge C_s}$ if y is new and \bar{c} ranges over a, c .
(DSet)	$\frac{x = f : \{T_1, \dots, T_n\} \wedge C_s}{x = f : \{x_1, \dots, x_n\} \wedge x_1 = T_1 \wedge \dots \wedge x_n = T_n \wedge C_s}$ if x_1, \dots, x_n are new and at least one of $T_i : 1 \leq i \leq n$ is not a variable
(DSetF)	$\frac{x = f : \{T_1, \dots, T_n\} = \wedge C_s}{x = f : \{x_1, \dots, x_n\} \wedge x = f : \{x_1, \dots, x_n\} = \wedge x_1 = T_1 \wedge \dots \wedge x_n = T_n \wedge C_s}$ if x_1, \dots, x_n are new and at least one of $T_i : 1 \leq i \leq n$ is not a variable
(DConj)	$\frac{x = S \sqcap T \wedge C_s}{x = S \wedge x = T \wedge C_s}$

Figure 2: Decomposition rules

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

- $\mathcal{I}, \alpha \models x = T \iff \alpha(x) \in \llbracket T \rrbracket^{\mathcal{I}, \alpha}$
- $\mathcal{I}, \alpha \models x = x_1 \sqcup \dots \sqcup x_n \iff \alpha(x) = \alpha(x_i)$ 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}, α 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}, α 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.

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*.

Constraint simplification rules - I	
(SEquals)	$\frac{x = y \wedge C_s}{x = y \wedge [x/y]C_s}$ if $x \neq y$ and x occurs in C_s
(SConst)	$\frac{x = \bar{c} \wedge y = \bar{c} \wedge C_s}{x = y \wedge x = \bar{c} \wedge C_s}$ where \bar{c} ranges over a, c .
(SFeat)	$\frac{x = f : y \wedge x = F : z \wedge C_s}{x = f : y \wedge y = z \wedge C_s}$ where F ranges over $f, \exists f, \forall f$
(SExists)	$\frac{x = \exists f : y \wedge x = \forall f : z \wedge C_s}{x = f : y \wedge y = z \wedge C_s}$
(SForallE)	$\frac{x = \forall f : \bar{C} \wedge x = \exists f : y \wedge C_s}{x = \forall f : \bar{C} \wedge x = \exists f : y \wedge y = \bar{C} \wedge C_s}$ if \bar{C} ranges over $C, \neg C, \neg a, \neg c, \neg z$ and $C_s \not\vdash y = \bar{C}$.

Figure 3: Constraint simplification rules - I

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 (3), singleton sets - rule (3), duplicate elements in a set - rule (3), universally quantified constraint - rule (3), another set description - rule (3). Rule (3) on the other hand simplifies disjunctive constraints. Amongst all the constraint simplification rules in figures 3 and 4 only rule (3) is non-deterministic and creates a n -ary choice point.

Rules (3) and (3) 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. $C_s \vdash x = y \perp \rightarrow C_s \vdash y = x$

Constraint simplification rules - II	
(SSetF)	$\frac{x = F : y \wedge x = f : \{x_1, \dots, x_n\} \wedge C_s}{x = f : y \wedge y = x_1 \wedge \dots \wedge y = x_n \wedge C_s}$ where F ranges over $f, \forall f$
(SSet)	$\frac{x = f : \{y\} \wedge C_s}{x = f : y \wedge C_s}$
(SDup)	$\frac{x = f : \{x_1, \dots, x_i, \dots, x_j, \dots, x_n\} \wedge C_s}{x = f : \{x_1, \dots, x_i, \dots, \dots, x_n\} \wedge C_s}$ if $x_i \equiv x_j$
(SForall)	$\frac{x = \forall f : \overline{C} \wedge x = f : \{x_1, \dots, x_n\} \wedge C_s}{x = f : \{x_1, \dots, x_n\} \wedge x_1 = \overline{C} \wedge \dots \wedge x_n = \overline{C} \wedge C_s}$ 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)	$\frac{x = \exists f : y \wedge x = f : \{x_1, \dots, x_n\} \wedge C_s}{x = f : \{x_1, \dots, x_n\} \wedge y = x_1 \sqcup \dots \sqcup x_n \wedge C_s}$
(SSetSet)	$\frac{x = f : \{x_1, \dots, x_n\} \wedge x = f : \{y_1, \dots, y_m\} \wedge C_s}{x = f : \{x_1, \dots, x_n\} \wedge x_1 = y_1 \sqcup \dots \sqcup y_m \wedge \dots \wedge x_n = y_1 \sqcup \dots \sqcup y_m \wedge y_1 = x_1 \sqcup \dots \sqcup x_n \wedge \dots \wedge y_m = x_1 \sqcup \dots \sqcup x_n \wedge C_s}$ where $n \leq m$
(SDis)	$\frac{x = x_1 \sqcup \dots \sqcup x_n \wedge C_s}{x = x_1 \sqcup \dots \sqcup x_n \wedge x = x_i \wedge C_s}$ 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

4. $C_s \vdash x = y \wedge C_s \vdash y = z \perp \rightarrow C_s \vdash x = z$
5. $C_s \vdash x = \neg y \perp \rightarrow C_s \vdash y = \neg x$
6. $C_s \vdash x = f : y \perp \rightarrow C_s \vdash x = \exists f : y$
7. $C_s \vdash x = f : y \perp \rightarrow C_s \vdash x = \forall f : y$
8. $C_s \vdash x = f : \{\dots, x_i, \dots\} \perp \rightarrow C_s \vdash x = \exists f : x_i$

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 (3) propagates g -values of y into f -values of x in the presence of the constraint $x = f : \supseteq g(y)$. Rule (3) (correspondingly Rule (3)) 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

Extended Constraint simplification rules

$$(\subseteq) \frac{x = f : \supseteq g(y) \wedge C_s}{x = f : \supseteq g(y) \wedge x = \exists f : y_i \wedge C_s}$$

if:

- $C_s \not\vdash x = \exists f : y_i$ and
- $C_s \vdash y = \exists g : y_i$

$$(\cup Left) \frac{x = f : g(y) \cup h(z) \wedge C_s}{x = f : g(y) \cup h(z) \wedge x = f : \supseteq g(y) \wedge C_s}$$

if $C_s \not\vdash x = f : \supseteq g(y)$

$$(\cup Right) \frac{x = f : g(y) \cup h(z) \wedge C_s}{x = f : g(y) \cup h(z) \wedge x = f : \supseteq h(z) \wedge C_s}$$

if $C_s \not\vdash x = f : \supseteq h(z)$ $(\cup Down)$

$$\frac{x = f : g(y) \cup h(z) \wedge C_s}{x = f : g(y) \cup h(z) \wedge y = \exists g : x_i \mid z = \exists h : x_i \wedge C_s}$$

if:

- $C_s \not\vdash y = \exists g : x_i$ and
- $C_s \not\vdash z = \exists h : x_i$ and
- $C_s \vdash x = \exists f : x_i$

 $(\cap Down)$

$$\frac{x = f : g(y) \cap h(z) \wedge C_s}{x = f : g(y) \cap h(z) \wedge y = \exists g : x_i \wedge z = \exists h : x_i \wedge C_s}$$

if:

- $(C_s \not\vdash y = \exists g : x_i$ or $C_s \not\vdash z = \exists h : x_i)$ and
- $C_s \vdash x = \exists f : x_i$

$$(\cap Up) \frac{x = f : g(y) \cap h(z) \wedge C_s}{x = f : g(y) \cap h(z) \wedge x = \exists f : x_i \wedge C_s}$$

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

in the presence of $x = f : g(y) \cup h(z)$ rule (3) 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 (3) 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 (3) 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 report - 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 satisfiability 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 (3) given in figure 4 is non-deterministic while all the other rules are deterministic.

Theorem 1 (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}, α 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 $\mathcal{I}, \alpha \models C'_s$*

It is easy to verify the invariance claim for most of the rules. However it is bit difficult to see that the theorem holds for rule 3. Hence we show that the theorem holds for this rule too.

Lemma 2 *Rule (3) is invariant.*

Proof: Let C'_s be derived from C_s by applying rule (3).

We shall show that for any interpretation \mathcal{I} and assignment α : $\mathcal{I}, \alpha \models C_s \iff \mathcal{I}, \alpha \models C'_s$.

Assume that there exists an interpretation \mathcal{I} and assignment α such that $\mathcal{I}, \alpha \models C_s$ then:
 $\mathcal{I}, \alpha \models x = f : \{x_1, \dots, x_n\} \wedge x = f : \{y_1, \dots, y_m\}$.

From the definition of satisfaction of containment constraints and the semantics of set descriptions it follows that:

$$f^{\mathcal{I}}(\alpha(x)) = \{\alpha(x_1), \dots, \alpha(x_n)\} \text{ and } f^{\mathcal{I}}(\alpha(x)) = \{\alpha(y_1), \dots, \alpha(y_m)\}.$$

This means that for:

- (6) a. for each $x_i : 1 \leq i \leq n$ there exists $y_j : 1 \leq j \leq m$ such that $\alpha(x_i) = y_j$
 b. for each $y_j : 1 \leq j \leq m$ there exists $x_i : 1 \leq i \leq n$ such that $\alpha(x_i) = y_j$

But, this is exactly what the following constraints express:

$$(7) \quad \begin{aligned} x_1 = y_1 \sqcup \dots \sqcup y_m \wedge \dots \wedge x_n = y_1 \sqcup \dots \sqcup y_m \wedge \\ y_1 = x_1 \sqcup \dots \sqcup x_n \wedge \dots \wedge y_m = x_1 \sqcup \dots \sqcup x_n \wedge C_s \\ \text{where } n \leq m \end{aligned}$$

Thus, $\mathcal{I}, \alpha \models C'_s$.

Conversely, if $\mathcal{I}, \alpha \models C'_s$ then we need to show that: $\mathcal{I}, \alpha \models C_s$. For this, since $\mathcal{I}, \alpha \models x = f : \{x_1, \dots, x_n\}$ already holds, it is enough to show that: $\mathcal{I}, \alpha \models x = f : \{y_1, \dots, y_m\}$.

Since \mathcal{I}, α satisfies the constraints given in (7) it follows that \mathcal{I}, α satisfies the condition given in (6). But, this means: $\mathcal{I}, \alpha \models x = f : \{y_1, \dots, y_m\}$.

Hence, we have the theorem.

We next turn our attention to the completeness theorem for our consistency checking procedure.

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

Let $\text{succ}(x, f)$ denote the set:

$$\text{succ}(x, f) = \{y \mid C_s \vdash x = \exists f : y\}$$

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 $a_1 \neq a_2$
2. $C_s \vdash x = c_1$ and $C_s \vdash x = c_2$ such that $c_1 \neq c_2$
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 $\text{succ}(x, f) \cap \text{succ}(y, g) \neq \emptyset$
6. $C_s \vdash x = f : \{x_1, \dots, x_n\} =$ and $|\text{succ}(x, f)| < n$

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:

$$x \simeq y \text{ if } C_s \vdash x = y$$

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

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 \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 $\mathcal{R} = \langle \mathcal{U}^R, \cdot^R \rangle$ 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 $\alpha(x) = \text{choose}([x])$ where $\text{choose}([x])$ 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 \cdot^R is defined as follows:

- $f^R(x) = \text{succ}(x, f)$
- $a^R = a$

We now demonstrate that for every constraint K in C_s :

$\mathcal{R}, \alpha \models K$.

1. if $x = a$ in C_s then we know from the definition of clash that $C_s \not\vdash x = b$ where b is distinct from a . Furthermore, we know that $\text{succ}(x, f) = \emptyset$ for every f .
Hence $C_s \models x = a$.
2. if $x = c_1$ in C_s then we know from the definition of clash that $C_s \not\vdash x = c_2$ where c_2 is distinct from c_1 .
Hence $\mathcal{R}, \alpha \models x = c_1$.
3. if $x = C$ in C_s then $\alpha(x) \in \alpha(C)$ by construction.
Hence, $\mathcal{R}, \alpha \models x = C$.

4. if $x = \neg\overline{C}$ in C_s where \overline{C} ranges over a, c, C then since C_s is clash-free we know that $x = \overline{C}$ not in C_s . This means that $\alpha(x) \notin \alpha(C)$.

Hence, $\mathcal{R}, \alpha \models x = \neg\overline{C}$

5. if $x = \exists f : y$ in C_s then we know that $y \in \text{succ}(x, f)$. This means $(\alpha(x), \alpha(y)) \in f^R$. Hence, $\mathcal{R}, \alpha \models x = \exists f : y$.

6. if $x = f : y$ in C_s then $\text{succ}(x, f) = \{y\}$ since our constraint solving rules eliminate $x = \exists f : z, x = f : z, x = f : \{x_1, \dots, x_n\}, x = \forall f : z$.

Hence $\mathcal{R}, \alpha \models x = f : y$.

7. if $x = \forall f : y$ in C_s then $\text{succ}(x, f) = \emptyset$ since otherwise our constraint solving rules eliminate $x = \forall f : y$ in the presence of $x = \exists f : z, x = f : z$ or $x = f : \{x_1, \dots, x_n\}$.

Hence $\mathcal{R}, \alpha \models x = \forall f : y$.

8. if $x = \forall f : \overline{C}$ in C_s where \overline{C} is one of $C, \neg C, \neg a, \neg c$ then for every variable y in $\text{succ}(x, f) : C_s \vdash y = \overline{C}$.

This means that $\mathcal{R}, \alpha \models x = \forall f : \overline{C}$.

Note that constraints of the form $x = \forall f : \overline{c}$ where \overline{c} is one of a, c is not present in the normal form since they are eliminated by the decomposition rule (3).

9. if $x = f : \{x_1, \dots, x_n\}$ in C_s then $\text{succ}(x, f) = \{x_1, \dots, x_n\}$ since our constraint solving rules eliminate $x = \exists f : z, x = f : z, x = f : \{y_1, \dots, y_n\}, x = \forall f : z$.

Hence $\mathcal{R}, \alpha \models x = f : \{x_1, \dots, x_n\}$.

10. if $x = f : \{x_1, \dots, x_n\}_=$ in C_s then it follows that $x = f : \{x_1, \dots, x_n\}$ in C_s (decomposition rule (3)). From the previous step we know that $\mathcal{R}, \alpha \models x = f : \{x_1, \dots, x_n\}$.

It remains to verify that $\text{succ}(x, f) = n$. Assume that $\text{succ}(x, f) < n$ then it follows from the definition of clash that C_s contains a clash. By hypothesis C_s is clash-free hence $\text{succ}(x, f) = n$.

Hence $\mathcal{R}, \alpha \models x = f : \{x_1, \dots, x_n\}_=$.

11. if $x = x_1 \sqcup \dots \sqcup x_n$ in C_s then we know that $C_s \vdash x = x_i$ where $1 \leq i \leq n$. This means that $\mathcal{R}, \alpha \models x = x_1 \sqcup \dots \sqcup x_n$.

12. if $x = f : \supseteq g(y)$ in C_s then from rule (3) we know that $\text{succ}(y, g) \subseteq \text{succ}(x, f)$.

13. if $x = f : g(y) \cup h(z)$ in C_s then from rules (3) and (3) we know that both $x = f : \supseteq g(y)$ in C_s and $x = f : \supseteq h(z)$ in C_s .

\hookrightarrow From the previous step it follows that both $\text{succ}(y, g) \subseteq \text{succ}(x, f)$ and $\text{succ}(z, h) \subseteq \text{succ}(x, f)$. Hence, $\text{succ}(y, g) \cup \text{succ}(z, h) \subseteq \text{succ}(x, f)$.

\hookrightarrow From rule (3) it follows that $\text{succ}(x, f) \subseteq \text{succ}(y, g) \cup \text{succ}(z, h)$.

Hence, $\mathcal{R}, \alpha \models x = f : g(y) \cup h(z)$.

14. if $x = f : g(y) \cap h(z)$ in C_s then from rule (3) $\text{succ}(x, f) \subseteq \text{succ}(y, g) \cap \text{succ}(z, h)$.

\hookrightarrow From rule (3) we know that $\text{succ}(y, g) \cap \text{succ}(z, h) \subseteq \text{succ}(x, f)$.

Hence, $\mathcal{R}, \alpha \models x = f : g(y) \cap h(z)$.

15. if $f(x) \neq g(y)$ in C_s then since C_s is clash-free we know that $\text{succ}(x, f) \cap \text{succ}(y, g) = \emptyset$.
Hence, $C_s \models f(x) \neq g(y)$.

Since, our interpretation \mathcal{R}, α satisfy every constraint K in $C_s : \mathcal{R}, \alpha \models C_s$.

Hence we have the theorem.

Finally, our completeness theorem shows that rules (3) and (3) are not essential since our proof is not dependent upon these rules. However, these rules result in a simpler constraint system and hence have been included.

Theorem 3 (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 (3) 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 $\text{succ}(x, f)$. But the size of $\text{succ}(x, f)$ 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 $\text{succ}(x, f)$.

5 Canonical set unifiers

Some basic optimisations are possible for the combination of rule 3 and rule 3 so that only the *maximal* solutions are enumerated. In particular, we do not want solutions that are contained in other solutions to be enumerated. We shall refer to the set of equations:

$$\begin{aligned} x_1 &= y_{k1} \wedge \dots \wedge x_n = y_{kn} \wedge \\ y_1 &= x_{k1} \wedge \dots \wedge y_m = x_{km} \wedge C_s \end{aligned}$$

where $n \leq m$

generated by the combination of rule 3 and rule 3 as the *set unifier* to the constraint:

$$x = f : \{x_1, \dots, x_n\} \wedge x = f : \{y_1, \dots, y_m\}.$$

Note that from lemma 2 we know that the addition of every set unifier results in an invariant transformation of the original constraint system.

The problem with the combination of rule 3 and rule 3 is that it enumerates solutions that are included in other solutions. We say that a set unifier ϕ is included in ψ , written $\phi \sqsubseteq \psi$, if for every interpretation \mathcal{I} and variable assignment α : $\mathcal{I}, \alpha \models \phi \Rightarrow \mathcal{I}, \alpha \models \psi$. For instance, the set of equations:

- $x_1 = y_1, x_2 = y_2, y_1 = x_2, y_2 = x_2$

is included in the set of equations:

- $x_1 = y_1, x_2 = y_2, y_1 = x_1, y_2 = x_2$

We say that a set unifier ϕ is *maximal* if for every other set unifier ψ : $\phi \sqsubseteq \psi \Rightarrow \psi \sqsubseteq \phi$. For a given pair of set descriptions there would normally be multiple maximal set unifiers. Thus for the constraints:

$$f : \{x_1, x_2\} \wedge x = f : \{y_1, y_2\}$$

the set unifiers:

- $x_1 = y_1, x_2 = y_2, y_1 = x_2, y_2 = x_2$

and:

- $x_1 = y_2, x_2 = y_1, y_1 = x_2, y_2 = x_1$

are both maximal where as the set unifier:

- $x_1 = y_1, x_2 = y_2, y_1 = x_1, y_2 = x_2$

is included in the each of the previous two.

The complete set of maximal set unifiers can be characterised by the following modified rule 5 given below. We use the notation $\{x_1, x_2, \dots, x_n\} = \{y_1, y_2, \dots, y_m\}$ to mean the set of equations:

- $x_1 = x_2 = \dots = x_n = y_1 = y_2 = \dots = y_m$.

$$\text{(SSetSetM)} \quad \frac{x = f : \{x_1, \dots, x_n\} \wedge x = f : \{y_1, \dots, y_m\} \wedge C_s}{\begin{array}{l} x = f : \{x_1, \dots, x_n\} \wedge \\ Xs_1 = Ys_1, \dots, Xs_k = Ys_k \wedge \\ C_s \end{array}} \text{ (non-deterministically)}$$

where

1. each of Xs_1, \dots, Xs_k is a non-empty partition of the set $\{x_1, \dots, x_n\}$.
2. each of Ys_1, \dots, Ys_k is a non-empty partition of the set $\{y_1, \dots, y_m\}$.
3. each equation $Xs_i = Ys_i$ is one of the following forms:
 - (a) $\{x_i\} = Ys_i$ or
 - (b) $Xs_i = \{y_i\}$ such that Xs_i is of size at least 2

Since each $x_i \in \{x_1, \dots, x_n\}$ and each $y_j \in \{y_1, \dots, y_m\}$ appears at exactly once in one of $Xs_i = Ys_i$, every solution generated by our original rule 3 is also included in at least one of the solutions generated by our modified rule 5. Furthermore this condition is enough to demonstrate that rule 5 too is invariant in at least one of its non-deterministic transitions.

One further optimisation is possible with rule 5. This has to do with the case when there are common elements between sets $\{x_1, \dots, x_n\}$ and $\{y_1, \dots, y_m\}$. Consider for instance the constraints:

$$\bullet x_0 = f : \{x, y, z\} \wedge x_0 = f : \{x, y, w\}$$

This has the following maximal solutions and every other solution is included in one of these:

1. $\{z\} = \{w\}, \{y\} = \{y\}, \{x\} = \{x\}$
2. $\{y\} = \{y, z\}, \{x\} = \{x, w\}$
3. $\{y\} = \{y, w\}, \{x\} = \{x, z\}$

This means that not all the solutions (15 in this case) generated by 5 are maximal. This is so because every other solution generated by 5 has more than one equation of the form $Xs_i = Ys_i$ and $Xs_j = Ys_j$ such that either x or y is included in both the equations making them equivalent to $Xs_i = Ys_i = Xs_j = Ys_j$ which in turn makes the solution non-maximal.

This can be rectified by adding 2 further conditions in rule 5. Our final modified rule 5 is given below:

$$(\mathbf{SSetSetM2}) \frac{x = f : \{x_1, \dots, x_n\} \wedge x = f : \{y_1, \dots, y_m\} \wedge C_s}{\begin{array}{l} x = f : \{x_1, \dots, x_n\} \wedge \\ Xs_1 = Ys_1, \dots, Xs_k = Ys_k \wedge \\ C_s \end{array}} \text{ (non-deterministically)}$$

where

1. each of Xs_1, \dots, Xs_k is a non-empty partition of the set $\{x_1, \dots, x_n\}$.
2. each of Ys_1, \dots, Ys_k is a non-empty partition of the set $\{y_1, \dots, y_m\}$.
3. each equation $Xs_i = Ys_i$ is one of the following forms:
 - (a) $\{x_i\} = Ys_i$ or
 - (b) $Xs_i = \{y_i\}$ such that Xs_i is of size at least 2
4. for each equation of the form $\{x_i\} = Ys_i$ if $x_i \in \{y_1, \dots, y_m\}$ then $x_i \in Ys_i$
5. for each equation of the form $Xs_i = \{y_i\}$ if $y_i \in \{x_1, \dots, x_n\}$ then $y_i \in Xs_i$

Again it can be verified that every solution generated by rule 5 is included in at least one solution generated by rule 5.

This completes our description of a constraint solving rule for enumerating the complete set of maximal set unifiers.

6 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 & Johnson, 1979).

Theorem 4 (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, \dots 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, \dots be a finite set of variables distinct from the ones introduced above.

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

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

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

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

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

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.

7 Translation to Schönfinkel-Bernays class

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

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

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
- $(x = \forall f : a)^\delta = \forall y (f(x, y) \rightarrow y = a)$

- $(x = \forall f : \neg a)^\delta = \forall y (f(x, y) \rightarrow y \neq a)$
- $(x = f : \{T_1, \dots, T_n\})^\delta =$
 $f(x, x_1) \ \& \ \dots \ \& \ f(x, x_n) \ \&$
 $\forall y (f(x, y) \rightarrow y = x_1 \vee \dots \vee y = x_n) \ \&$
 $(x_1 = T_1)^\delta \ \& \ \dots \ \& \ (x_n = T_n)^\delta$
 where x_1, \dots, x_n are new variables
- $(x = f : g(y) \cup h(z))^\delta =$
 $\forall x_i (f(x, x_i) \rightarrow g(y, x_i) \vee h(z, x_i)) \ \&$
 $\forall y_i (g(y, y_i) \rightarrow f(x, y_i)) \ \&$
 $\forall z_i (h(z, z_i) \rightarrow f(x, z_i))$
- $(x = f : (y) \neq g(z))^\delta =$
 $\forall y_i z_j (f(y, y_i) \ \& \ g(z, z_j) \rightarrow y_i \neq z_j)$
- $(x = S \sqcap T)^\delta = (x = S)^\delta \ \& \ (x = T)^\delta$

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:

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

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.

8 Summary and Conclusions

In this report 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.

Based on the abstract algorithm described in this report a Prolog implementation of the complete logic has been implemented and shows a reasonably efficient behaviour.

We believe that the logic described in this paper provides a rigorous formalisation of set descriptions employed within constraint-based grammatical theories especially HPSG. Furthermore our formalisation focusses on consistency checking methods which so far had not been addressed within computational linguistics. The work described in this report is a necessary prerequisite for building knowledge representation frameworks that support set descriptions.

References

- Carpenter, B. (1993). ALE:Attribute Logic Engine Users Guide. Tech. rep. version β , Carnegie Mellon University, Pittsburgh, PA 15213.
- Dörre, J. & Dorna, M. (1993). CUF: A Formalism for Linguistic Knowledge Representation. Dyana-2 deliverable, IMS, Stuttgart, Germany.
- Garey, M. R. & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.
- Hollunder, B. & Nutt, W. (1990). Subsumption Algorithms for Concept Languages. Research Report RR-90-04, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, 6600 Saarbrücken 11, Germany.
- Johnson, M. (1991). Features and Formulae. *Computational Linguistics*, 17(2), 131–151.
- Kaplan, R. M. (1989). The Formal Architecture of Lexical-Functional Grammar. In *Proceedings of ROCLING II*, pp. 3 – 18 Taipei, Republic of China.
- Kaplan, R. M. & Bresnan, J. (1982). Lexical-Functional Grammar: A Formal System for Grammatical Representation. In Bresnan, J. (Ed.), *The Mental Representation of Grammatical Relations*, pp. 173 – 281. MIT Press, Cambridge, Massachusetts.
- Kasper, R. & Rounds, W. (1986). A Logical Semantics for Feature Structures. In *24th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York*, pp. 257–265.
- Lewis, H. R. (1980). Complexity Results for Classes of Quantificational Formulae. *Journal of Computer and System Sciences*, 21, 317–353.
- Manandhar, S. (1994). An Attributive Logic of Set Descriptions and Set Operations. In *32nd Annual Meeting of the Association for Computational Linguistics, Los Cruces, New Mexico*, pp. 255–262.
- Pollard, C. J. & Moshier, M. D. (1990). Unifying Partial Descriptions of Sets. In Hanson, P. P. (Ed.), *Information, Language and Cognition*. University of British Columbia Press, Vancouver, Canada. Vancouver Studies in Cognitive Science, no. 1.
- Pollard, C. & Sag, I. A. (1987). *Information-Based Syntax and Semantics: Volume 1 Fundamentals*, Vol. 13 of *Lecture Notes*. Center for the Study of Language and Information, Stanford, CA.
- Pollard, C. & Sag, I. A. (1994). *Head-driven Phrase Structure Grammar*. Chicago: University of Chicago Press and Stanford: CSLI Publications.
- Rounds, W. C. (1988). Set Values for Unification-Based Grammar Formalisms and Logic Programming. Tech. rep. CSLI-88-129, Center for the Study of Language and Information, Stanford, CA.
- Smolka, G. (1991). A Feature Logic with Subsorts. In Wedekind, J. & (eds.), C. R. (Eds.), *Unification in Grammar*. MIT Press. Also appeared as LILOG Report no. 33, IWBS, IBM Deutschland.

Smolka, G. (1992). Feature Constraint Logics for Unification Grammars. *Journal of Logic Programming*, 12, 51–87.

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