Conditional Descriptions in Functional Unification Grammar

Robert T. Kasper
USC/Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292 U.S.A.

Abstract

A grammatical description often applies to a linguistic object only when that object has certain features. Such conditional descriptions can be indirectly modeled in Kay's Functional Unification Grammar (FUG) using functional descriptions that are embedded within disjunctive alternatives. An extension to FUG is proposed that allows for a direct representation of conditional descriptions. This extension has been used to model the input conditions on the systems of systemic grammar. Conditional descriptions are formally defined in terms of logical implication and negation. This formal definition enables the use of conditional descriptions as a general notational extension to any of the unification-based grammar representation systems currently used in computational linguistics.

1 Introduction

Functional Unification Grammar [Kay79] (FUG) and other grammatical formalisms that use feature structures and unification provide a general basis for the declarative representation of natural language grammars. In order to utilize some of the computational tools available with unification grammars, we have developed a mapping from *systemic grammars* [Hall76] into FUG notation. This mapping has been used as the first step in creating a general parsing method for systemic grammars [Kas87a]. The experience of translating systemic grammars into FUG has shown several ways in which the notational resources of FUG may be improved. In particular, FUG has limited notational resources for expressing conditional information. In this paper we describe how FUG has been enhanced by the addition of conditional descriptions, building on research that has already been reported [Kas87a,Kas86,Kas87b].

Conditional information is stated explicitly in systemic grammars by the input conditions of systems that specify when a system must be used. Consider, for example, the two systems (MoodType and IndicativeType)[1] shown in Figure 1. The input condition for the MoodType system is the feature *Clause*, and the input condition for the IndicativeType system is the feature *Indicative*. Because the features of a systemic grammar are normally introduced by a unique system, these input conditions actually express a bidirectional type of logical implication:

1. If a constituent has the feature(s) specified by a system's input condition, then exactly one of the alternatives described by that system must also be valid for the constituent;
2. If a constituent has one of the feature alternatives described by a system, then it must also have the feature(s) specified by that system's input condition.

Thus the input condition of the *IndicativeType* system expresses the following implications:

1. If a clause has the feature *Indicative*, then it must also have exactly one of the alternatives from the *IndicativeType* system (either *Declarative* or *Interrogative*).
2. If a clause has one of the feature alternatives described by the *IndicativeType* system (either *Declarative* or *Interrogative*), then it must also have the feature *Indicative*.

While it is theoretically correct to regard the two directions of implication as exact converses of each other, there is a subtle difference between them. The consequent of the first type of implication is the description of the entire system, including systemic features and their realizations.[2] The antecedent of the second type of implication can be safely abbreviated by the systemic features without their realizations, because the presence of a systemic feature implies that its realizations also hold. We will return to this distinction when we provide a formal definition of conditional descriptions in Section 2.

For simple input conditions, the first type of implication can be expressed in FUG, as it was originally formulated by Kay [Kay79], by embedding the description of one system inside the description of another. For example, we can capture this implication for the IndicativeType system by embedding it within the description of the Indicative alternative of the

[1]This example is extracted from Nigel [Mann83], a large systemic grammar of English that has been developed in text generation research at USC/ISI.

[2]A realization is a statement of structural properties that are required by a feature, such as the statement that SUBJECT precedes FINITE for the feature declarative.

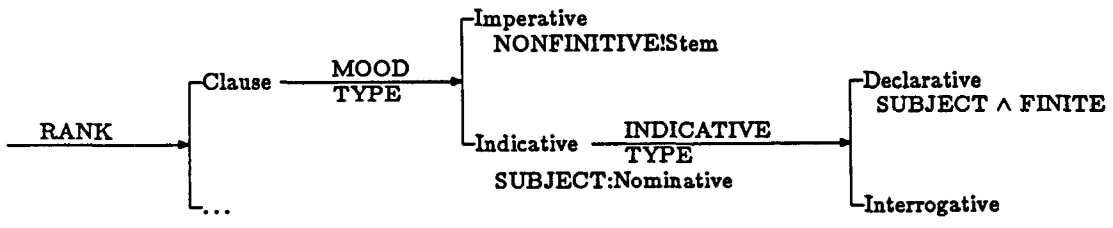

Figure 1: The MoodType and IndicativeType Systems

$$\left[\begin{array}{l}
\text{Rank} = \text{Clause}\\
\left\{\begin{array}{l}
\left[\begin{array}{l}\text{MoodType} = \text{Imperative}\\ \text{NONFINITIVE} = [\ \text{Form} = \text{Stem}\ ]\end{array}\right]\\
\left[\begin{array}{l}\text{MoodType} = \text{Indicative}\\ \text{SUBJECT} = [\ \text{Case} = \text{Nominative}\ ]\\ \left\{\begin{array}{l}\left[\begin{array}{l}\text{IndicativeType} = \text{Declarative}\\ \text{pattern} = (\ldots\ \text{SUBJECT FINITE}\ \ldots)\end{array}\right]\\ [\ \text{IndicativeType} = \text{Interrogative}\ ]\end{array}\right\}\end{array}\right]
\end{array}\right\}\\
\exists\ \text{IndicativeType} \longrightarrow [\ \text{MoodType} = \text{Indicative}\ ]
\end{array}\right]$$

Figure 2: The MoodType and IndicativeType Systems in FUG

MoodType system, as shown in Figure 2. Note that the second type of implication expressed by systemic input conditions has not been expressed by embedding one functional description inside another. To express the second type of implication, we have used a different notational device, called a feature existence condition; it will be defined in Section 2.4.

Not all systems have simple input conditions consisting of single features. Those input conditions which are complex boolean expressions over features cannot be expressed directly by embedding. Consider the BenefactiveVoice[3] system shown in Figure 3 as an example. Its input condition is the conjunction of two features, Agentive and Benefactive.

One way to express a system with a complex input condition in FUG is to use a disjunction with two alternatives, as shown in Figure 4. The first alternative corresponds to what happens when the BenefactiveVoice system is entered; the second alternative corresponds to what happens when the BenefactiveVoice system is not entered. The first alternative also includes the features of the input condition. The second alternative includes the features of the *negated* input condition. Notice that the input condition and its negation must both be stated explicitly, unlike in systemic notation. If the negation of the input condition was not included in the second alternative, it would be possible to use this alternative even when the input condition for the system holds. Thus the description of the system would not always be used when it should be. Note that this method of encoding systemic input conditions presupposes an adequate treatment of negated features.[4] A formal definition of negation will be developed in Section 2.3.

While it is formally possible to encode complex input conditions by disjunction and negation, such encoding is not altogether satisfactory. It should not be necessary to state the negated input condition explicitly, since it can always be derived automatically from the unnegated condition. It is also rather inefficient to mix the features of the input condition with the other features of the system. The features of the input condition contain exactly the information that is needed to choose between the two alternatives of the disjunction (i.e., to choose whether the system is entered or not). It would be more efficient and less verbose to have a notation in which the features of the input condition are distinguished from the other features of the system, and in which the negation of the input condition does not need to be stated explicitly. Therefore, we have developed an extension to FUG that uses a conditional operator ($\rightarrow$), as illustrated by the encoding of the BenefactiveVoice system shown in Figure 5. A description corresponding to the input condition appears to the left of the $\rightarrow$ symbol, and the description to be included when the input condition is satisfied appears to its right. A formal definition of what it means for a description to be satisfied will be given in Section 2.1.

[3]The BenefactiveVoice system is also extracted from the Nigel grammar [Mann83]. It describes the active and passive voice options that are possible in clauses that have both an agent and a beneficiary. The active/passive distinction is not primitive in systemic grammars of English. Instead, it is decomposed into several cases depending on which participant roles are present in the clause. In this case the subject of a passive clause may be conflated with either beneficiary or medium.

[4]Some negations of atomic features can be replaced by a finite disjunction of other possible values for that feature, but this technique only works effectively when the set of possible values is small and can be enumerated.

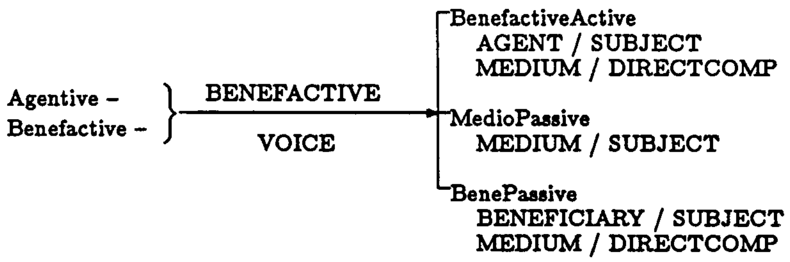

Figure 3: The BenefactiveVoice System.

$$\left[\begin{array}{l}
\text{Rank} = \text{Clause} \\
\left\{\begin{array}{l}
\left[\begin{array}{l}
\text{Agentivity} = \text{Agentive} \\
\text{Benefaction} = \text{Benefacitve} \\
\left\{\begin{array}{l}
\left[\begin{array}{l}\text{BenefactiveVoice} = \text{BenefactiveActive} \\ \text{AGENT} = \text{<SUBJECT>} \\ \text{MEDIUM} = \text{<DIRECTCOMP>}\end{array}\right] \\
\left[\begin{array}{l}\text{BenefactiveVoice} = \text{MedioPassive} \\ \text{MEDIUM} = \text{<SUBJECT>}\end{array}\right] \\
\left[\begin{array}{l}\text{BenefactiveVoice} = \text{BenePassive} \\ \text{BENEFICIARY} = \text{<SUBJECT>} \\ \text{MEDIUM} = \text{<DIRECTCOMP>}\end{array}\right]
\end{array}\right\}
\end{array}\right] \\
\left[\begin{array}{l}
\left\{\begin{array}{l}
\left[\text{Agentivity} = \text{NOT Agentive}\right] \\
\left[\text{Benefaction} = \text{NOT Benefactive}\right]
\end{array}\right\} \\
\text{BenefactiveVoice} = \text{NONE}
\end{array}\right]
\end{array}\right\}
\end{array}\right]$$

Figure 4: BenefactiveVoice system in FUG, using disjunction and negation.

$$\left[\begin{array}{l}
\text{Rank} = \text{Clause} \\
\left[\begin{array}{l}\text{Agentivity} = \text{Agentive} \\ \text{Benefaction} = \text{Benefactive}\end{array}\right] \rightarrow
\left\{\begin{array}{l}
\left[\begin{array}{l}\text{BenefactiveVoice} = \text{BenefactiveActive} \\ \text{AGENT} = \text{<SUBJECT>} \\ \text{MEDIUM} = \text{<DIRECTCOMP>}\end{array}\right] \\
\left[\begin{array}{l}\text{BenefactiveVoice} = \text{MedioPassive} \\ \text{MEDIUM} = \text{<SUBJECT>}\end{array}\right] \\
\left[\begin{array}{l}\text{BenefactiveVoice} = \text{BenePassive} \\ \text{BENEFICIARY} = \text{<SUBJECT>} \\ \text{MEDIUM} = \text{<DIRECTCOMP>}\end{array}\right]
\end{array}\right\} \\
\exists\ \text{BenefactiveVoice} \longrightarrow \left[\begin{array}{l}\text{Agentivity} = \text{Agentive} \\ \text{Benefaction} = \text{Benefactive}\end{array}\right]
\end{array}\right]$$

Figure 5: BenefactiveVoice system in extended FUG, using two conditional descriptions.

Note: In systemic notation curly braces represent conjunction and square braces represent disjunction, while in FUG curly braces represent disjunction and square braces represent conjunction.

NIL	denoting *no information*;
a	where $a \in A$, to describe atomic values;
$l : \phi$	where $l \in L$ and $\phi \in$ FDL, to describe structures in which the feature labeled by l has a value described by ϕ;
$[\![< p_1 >, \ldots, < p_n >]\!]$	where each $p_i \in L^*$, to describe an equivalence class of paths sharing a common value in a feature structure;
$\phi_1 \wedge \phi_2$ or $[\phi_1 \ldots \phi_n]$	where $\phi_i \in$ FDL, denoting conjunction;
$\phi_1 \vee \phi_2$ or $\{\phi_1 \ldots \phi_n\}$	where $\phi_i \in$ FDL, denoting disjunction.

Note: A and L are sets of symbols which are used to denote atomic values and feature labels, respectively.

Figure 6: Syntax of FDL Formulas.

2 Definitions

The feature description logic (FDL) of Kasper and Rounds [Kas86] provides a coherent framework to give a precise interpretation for conditional descriptions. As in previous work, we carefully observe the distinction between feature structures and their descriptions. Feature structures are represented by directed graphs (DGs), and descriptions of feature structures are represented by logical formulas. The syntax for formulas of FDL is given in Figure 6. We define several new types of formulas for conditional descriptions and negations, but the domain of feature structures remains DGs, as before.

2.1 Satisfaction and Compatibility

In order to understand how conditional descriptions are used, it is important to recognize two relations that may hold between a particular feature structure and a description: satisfaction and compatibility. Satisfaction implies compatibility, so there are three possible states that a particular structure may have with respect to a description: the structure may fully *satisfy* the description, the structure may be *incompatible* with the description, or the structure may be *compatible* with (but not satisfy) the description. To define these terms more precisely, consider the state of an arbitrary structure, $\mathcal{A}$, with respect to an atomic feature description, $f : v$:

$\mathcal{A}$ **satisfies** $f : v$ if f occurs in $\mathcal{A}$ with value v;

$\mathcal{A}$ **is incompatible with** $f : v$ if f occurs in $\mathcal{A}$ with value x, for some $x \neq v$;

$\mathcal{A}$ **is (merely) compatible with** $f : v$ if f does not occur in $\mathcal{A}$.

Because feature structures are used to represent partial information, it is possible for a structure that is merely compatible with a description to be extended (i.e., by adding a value for some previously nonexistent feature) so that it either satisfies or becomes incompatible with the description. Consider, for example, the structure ($\mathcal{A}_1$) shown in Figure 7, and the three descriptions:

$$subj : (person : 3 \wedge number : sing) \quad (1)$$

$$subj : (person : 1 \wedge number : sing) \quad (2)$$

$$subj : (case : nom \wedge number : sing) \quad (3)$$

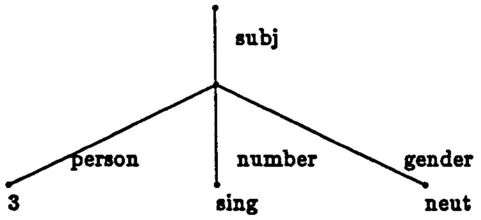

Figure 7: Example feature structure ($\mathcal{A}_1$).

Description (1) is *satisfied* by $\mathcal{A}_1$, because $\mathcal{A}_1$ is fully instantiated with all the required feature values. Description (2) is *incompatible* with $\mathcal{A}_1$, because $\mathcal{A}_1$ has a different value for the feature *subj : person*. Description (3) is *merely compatible* with $\mathcal{A}_1$ (but not satisfied by $\mathcal{A}_1$), because $\mathcal{A}_1$ has no value for the feature *subj : case*.

In the following definitions, the notation $\mathcal{A} \models \phi$ means that the structure $\mathcal{A}$ *satisfies* the description ϕ, and the notation $\mathcal{A} \sim \phi$ means that the structure $\mathcal{A}$ *is compatible with* the description ϕ.

Logical combinations of feature descriptions are evaluated with their usual semantics to determine whether they are satisfied by a structure. Thus, a conjunction is satisfied only when every conjunct is satisfied, and a disjunction is satisfied if any disjunct is satisfied. The formal semantics of the satisfaction relation has been specified in our previous work describing FDL [Kas86]. The semantics of the compatibility relation is given by the following conditions:

1. $\mathcal{A} \sim NIL$ always;
2. $\mathcal{A} \sim a \iff \mathcal{A}$ is the atomic structure a;
3. $\mathcal{A} \sim [\![< p_1 >, \ldots, < p_n >]\!] \iff$ all DGs in the set $\{\mathcal{A}/ < p_1 >, \ldots, \mathcal{A}/ < p_n >\}$ can be unified (any member of this set may be undefined; such members are equivalent to null DGs);
4. $\mathcal{A} \sim l : \phi \iff \mathcal{A}/l$ is undefined or $\mathcal{A}/l \sim \phi$;
5. $\mathcal{A} \sim \phi \vee \psi \iff \mathcal{A} \sim \phi$ or $\mathcal{A} \sim \psi$;
6. $\mathcal{A} \sim \phi \wedge \psi \iff \mathcal{A} \sim$ canonical form of $\phi \wedge \psi$.

Unlike satisfaction, the semantics of compatibility cannot be defined by simple induction over conjunctive formulas, because of a subtle interaction between path equivalences and nonexistent features. For example, consider whether $\mathcal{A}_1$, shown in Figure 7, is compatible with the description:

$$number : pl \wedge [\![< number >, < subj\ number >]\!].$$

$\mathcal{A}_1$ is compatible with *number* : *pl*, and $\mathcal{A}_1$ is also compatible with $[\![< number >, < subj\ number >]\!]$, but $\mathcal{A}_1$ is not compatible with the conjunction of these two descriptions, because it requires *subj* : *number* : *pl* and $\mathcal{A}_1$ has *sing* as the value of that feature.

Thus, in order to determine whether a structure is compatible with a conjunctive description, it is generally necessary to unify all conjuncts, putting the description into the canonical form described in [Kas87c]. This canonical form (i.e. the *feature-description* data structure) contains definite and indefinite components. The definite component contains no disjunction, and is represented by a DG structure that satisfies all non-disjunctive parts of a description. The indefinite component is a list of disjunctions. A structure is compatible with a description in canonical form if and only if it is unifiable with the definite component and it is compatible with each disjunction of the indefinite component.

2.2 Conditional Description

We augment FDL with a new type of formula to represent conditional descriptions, using the notation, $\alpha \rightarrow \beta$, and the standard interpretation given for material implication:

$$\mathcal{A} \models \alpha \rightarrow \beta \iff \mathcal{A} \models \neg\alpha \vee \beta. \qquad (4)$$

This interpretation of conditionals presupposes an interpretation of negation over feature descriptions, which is given below. To simplify the interpretation of negations, we exclude formulas containing path equivalences and path values from the antecedents of conditionals.

2.3 Negation

We use the classical interpretation of negation, where $\mathcal{A} \models \neg\phi \iff \mathcal{A} \not\models \phi$. Negated descriptions are defined for the following types of formulas:

1. $\mathcal{A} \models \neg a \iff \mathcal{A}$ is not the atom a;
2. $\mathcal{A} \models \neg(l : \phi) \iff \mathcal{A} \models l : \neg\phi$ or $\mathcal{A}/l$ is not defined;
3. $\mathcal{A} \models \neg(\phi \vee \psi) \iff \mathcal{A} \models \neg\phi \wedge \neg\psi$;
4. $\mathcal{A} \models \neg(\phi \wedge \psi) \iff \mathcal{A} \models \neg\phi \vee \neg\psi$.

Note that we have not defined negation for formulas containing path equivalences or path values. This restriction makes it possible to reduce all occurrences of negation to a boolean combination of a finite number of negative constraints on atomic values. While the classical interpretation of negation is not strictly monotonic with respect to the normal subsumption ordering on feature structures, the restricted type of negation proposed here does not suffer from the inefficiencies and order-dependent unification properties of general negation or intuitionistic negation [Mosh87,Per87]. The reason for this is that we have restricted negation so that all negative information can be specified as local constraints on single atomic feature values. Thus, these constraints only come into play when specific atomic values are proposed for a feature, and they can be checked as efficiently as positive atomic value constraints.

2.4 Feature Existence Conditions

A special type of conditional description is needed when the antecedent of a conditional is an existence predicate for a particular feature, and not a regular feature description. We call this type of conditional a *feature existence condition*, and use the notation:

$$\exists f \rightarrow \phi, \text{ where } \mathcal{A} \models \exists f \iff \mathcal{A}/f \text{ is defined.}$$

This use of $\exists f$ is essentially equivalent to the use of $f = ANY$ in Kay's FUG, where *ANY* is a place-holder for any substantive (i.e., non-NIL) value.

The primary effect of a feature existence condition, such as $\exists f \rightarrow \phi$, is that the consequent is asserted whenever a substantive value is introduced for a feature labeled by f. The treatment of feature existence conditions differs slightly from other conditional descriptions in the way that an unsatisfiable consequent is handled. In order to negate the antecedent of $\exists f \rightarrow \phi$, we need to state that f may never have any substantive value. This is accomplished by unifying a special atomic value, such as *NONE*, with the value of f. This special atomic value is incompatible with any other real value that might be proposed as a value for f.

Feature existence conditions are needed to model the second type of implication expressed by systemic input conditions – namely, when a constituent has one of the feature alternatives described by a system, it must also have the feature(s) specified by that system's input condition. Generally, a system named f with input condition α and alternatives described by β, can be represented by two conditional descriptions:

1. $\alpha \rightarrow \beta$;
2. $\exists f \rightarrow \alpha$.

For example, recall the BenfactiveVoice system, which is represented by the two conditionals shown in Figure 5.

It is important to note that feature existence conditions are used for systems with simple input conditions as well as for those with complex input conditions. The use of feature existence conditions is essential in both cases to encode the bidirectional dependency between systems that is implicit in a systemic network.

3 Unification with Conditional Descriptions

The unification operation, which is commonly used to combine feature structures (i.e., non-disjunctive, non-conditional DGs), can be generalized to define an operation for combining the information of two feature descriptions (i.e., formulas of FDL). In FDL, the unification of two descriptions is equivalent to their logical conjunction, as discussed in [Kas87b]. We have shown in previous work [Kas87c] how unification can be accomplished for disjunctive descriptions without expanding to disjunctive normal form.

This unification method factors descriptions into a canonical form consisting of definite and indefinite components. The definite component contains no disjunction, and is represented by a DG structure that satisfies all non-disjunctive parts of a description. The indefinite component of a description is a list of disjunctions. When two descriptions are unified, the first step is to unify their definite components. Then the indefinite components of each description are checked for compatibility with the resulting definite component. Disjuncts are eliminated from the description when they are inconsistent with definite information. When only one alternative of a disjunction remains, it is unified with the definite component of the description.

This section details how this unification method can be extended to handle conditional descriptions. Conditionals may be regarded as another type of indefinite information in the description of a feature structure. They are indefinite in the sense that they impose constraints that can be satisfied by several alternatives, depending on the values of features already present in a structure.

3.1 How to Satisfy a Conditional Description

The constraints imposed on a feature structure by a conditional description can usually be determined most efficiently by first examining the antecedent of the conditional, because it generally contains a smaller amount of information than the consequent. Examining the antecedent is often sufficient to determine whether the consequent is to be included or discarded.

Given a conditional description, $C = \alpha \rightarrow \beta$, we can define the constraints that it imposes on a feature structure ($\mathcal{A}$) as follows. When:

$\mathcal{A} \models \alpha$, then $\mathcal{A} \models \beta$;[5]

$\mathcal{A} \not\sim \alpha$, then C imposes no further constraint on $\mathcal{A}$, and can therefore be eliminated;

$\mathcal{A} \sim \alpha$, then check whether β is compatible with $\mathcal{A}$.
If compatible, then C must be retained in the description of $\mathcal{A}$.
If incompatible, then $\mathcal{A} \models \neg\alpha$ (and C can be eliminated).

These constraints follow directly from the interpretation (4) that we have given for conditional descriptions. These constraints are logically equivalent to those that would be imposed on $\mathcal{A}$ by the disjunction $\neg\alpha \vee \beta$, as required. However, the constraints of the conditional can often be imposed more efficiently than those of the equivalent disjunction, because examining the antecedent of the conditional carries the same cost as examining only one of the disjuncts. When the constraints of a disjunction are imposed, both of the disjuncts must be examined in all cases.

[5]Read this constraint as: "make sure that $\mathcal{A}$ satisfies β."

3.2 Extending the Unification Algorithm

The unification algorithm for disjunctive feature descriptions [Kas87c] can be extended to handle conditionals by recognizing two types of indefinite information in a description: disjunctions and conditionals. The extended *feature-description* data structure has the components:

definite: a DG structure;

disjunctions: a list of disjunctions;

conditionals: a list of conditional descriptions.

The part of the unification algorithm that checks the compatibility of indefinite components of a description with its definite component is defined by the function CHECK-INDEF, shown in Figure 8. This algorithm checks the disjunctions of a description before conditionals, but an equally correct version of this algorithm might check conditionals before disjunctions. In our application of parsing with a systemic grammar it is generally more efficient to check disjunctions first, but other applications might be made more efficient by varying this order.

4 Potential Refinements

Several topics merit further investigation regarding conditional descriptions. The implementation we describe has the constraints of conditionals and disjunctions imposed in an arbitrary order. Changing the order has no effect on the final result, but it is likely that the efficiency of unification could be improved by ordering the conditionals of a grammar in a deliberate way. Another way to improve the efficiency of unification with conditionals would involve indexing them by the features that they contain. Then a conditional would not need to be checked against a structure until some feature value of the structure might determine the manner in which it is satisfied. The amount of efficiency gained by such techniques clearly depends largely on the nature of the particular grammar being used in an application.

A slightly different type of conditional might be used as a way to speed up unification with binary disjunctive descriptions. If it is known that the values of a relatively small number of features can be used to discriminate between two alternative descriptions, then those features can be factored into a separate condition in a description such as

IF *condition* THEN alt_1 ELSE alt_2.

When the condition is satisfied by a structure, then alt_1 is selected. When the condition is incompatible with a structure, then alt_2 is selected. Otherwise both alternatives must remain under consideration. As it often requires a considerable amount of time to check which alternatives of a disjunction are applicable, this technique might offer a significant improvement in an application where large disjunctive descriptions are used.

Remember that we have restricted conditionals by requiring that their antecedents do not contain path equivalences.

```
Function CHECK-INDEF (desc) Returns feature-description:
     where desc is a feature-description.
Let $\mathcal{D}$ = desc.definite (a DG).
Let disjunctions = desc.disjunctions.
Let conditionals = desc.conditionals.
Let unchecked-parts = true.

While unchecked-parts, do:
     unchecked-parts := false.

     Check compatibility of disjunctions with $\mathcal{D}$ (omited, see [Kas87c]).

     Check compatibility of conditionals with $\mathcal{D}$:
     Let new-conditionals = $\emptyset$.
     For each $\alpha \rightarrow \beta$ in conditionals:
          test whether $\mathcal{D}$ satisfies or is compatible with $\alpha$:
               SATISFIES: $\mathcal{D}$ := UNIFY-DGS ($\mathcal{D}$, $\beta$.definite),
                    disjunctions := disjunctions $\cup$ $\beta$.disjunctions,
                    unchecked-parts := true;
               COMPATIBLE: If $\mathcal{D}$ is compatible with $\beta$,
                    then new-conditionals := new-conditionals $\cup$ $\{\alpha \rightarrow \beta\}$,
                    else let neg-ante = $\neg\alpha$,
                         $\mathcal{D}$ := UNIFY-DGS ($\mathcal{D}$, neg-ante.definite),
                         disjunctions := disjunctions $\cup$ neg-ante.disjunctions,
                         unchecked-parts := true;
               INCOMPATIBLE: this conditional imposes no further constraint.
          end (for loop).
     conditionals := new-conditionals.
     end (while loop).

Let nd = make feature-description with:
     nd.definite = $\mathcal{D}$, nd.disjunctions = disjunctions, nd.conditionals = conditionals.
Return (nd).
```

Figure 8: CHECK-INDEF: Algorithm for checking compatibility of indefinite parts of a feature-description with its definite component.

This restriction has been acceptable in our use of conditional descriptions to model systemic grammars. It is unclear whether a treatment of conditional descriptions without this restriction will be needed in other applications. If this restriction is lifted, then further work will be necessary to define the behavior of negation over path equivalences, and to handle such negations in a reasonably efficient manner.

5 Summary

We have shown how the notational resources of FUG can be extended to include descriptions of conditional information about feature structures. Conditional descriptions have been given a precise logical definition in terms of the feature description logic of Kasper and Rounds, and we have shown how a unification method for feature descriptions can be extended to use conditional descriptions. We have implemented this unification method and tested it in a parser for systemic grammars, using several hundred conditional descriptions. The definition of conditional descriptions and the unification method should be generally applicable as an extension to other unification-based grammar frameworks, as well as to FUG and the modeling of systemic grammars. In fact, the implementation described has been carried out by extending PATR-II [Shie84], a general representational framework for unification-based grammars.

While it is theoretically possible to represent the information of conditional descriptions indirectly using notational devices already present in Kay's FUG, there are practical advantages to representing conditional descriptions directly. The indirect encoding of conditional descriptions by disjunctions and negations entails approximately doubling the size of a description, adding many explicit nonexistence constraints on features (*NONE* values), and slowing the unification process. In our experiments, unification with conditional descriptions requires approximately 50% of the time required by unification with an indirect encoding of the same descriptions. By adding conditional descriptions as a notational resource to FUG, we have not changed the theoretical limits of what FUG can do, but we have developed a representation that is more perspicuous, less verbose, and computationally more efficient.

Acknowledgements

I would like to thank Bill Rounds for suggesting that it might be worthwhile to clarify ideas about conditional descriptions that were only partially formulated in my dissertation at the University of Michigan. Helpful comments on earlier versions of this paper were provided by Bill Mann, Ed Hovy and John Bateman.

This research was sponsored by the United States Air Force Office of Scientific Research under contract F49620-87-C-0005; the opinions expressed here are solely those of the author.

References

[Hall76] Gunther R. Kress, editor. *Halliday: System and Function in Language.* Oxford University Press, London, England, 1976.

[Kas87a] Robert T. Kasper. Systemic Grammar and Functional Unification Grammar. In J. Benson and W. Greaves, editors, *Systemic Functional Approaches to Discourse,* Norwood, New Jersey: Ablex (in press). Also available as USC/Information Sciences Institute, Technical Report RS-87-179, May 1987.

[Kas86] Robert T. Kasper and William C. Rounds. A Logical Semantics for Feature Structures. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics,* Columbia University, New York, NY, June 10-13, 1986.

[Kas87b] Robert T. Kasper. *Feature Structures: A Logical Theory with Application to Language Analysis.* PhD dissertation, University of Michigan, 1987.

[Kas87c] Robert T. Kasper. A Unification Method for Disjunctive Feature Descriptions. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics,* Stanford University, Stanford, CA, July 6-9, 1987.

[Kay79] Martin Kay. Functional Grammar. In *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society,* Berkeley Linguistics Society, Berkeley, California, February 17-19, 1979.

[Mann83] William C. Mann and Christian Matthiessen. Nigel: A Systemic Grammar for Text Generation. USC / Information Sciences Institute, RR-83-105. Also appears in R. Benson and J. Greaves, editors, *Systemic Perspectives on Discourse: Selected Papers Papers from the Ninth International Systemics Workshop,* Ablex, London, England, 1985.

[Mosh87] M. Drew Moshier and William C. Rounds. A Logic for Partially Specified Data Structures. In *Proceedings of the ACM Symposium on Principles of Programming Languages,* 1987.

[Per87] Fernando C.N. Pereira. Grammars and Logics of Partial Information. In *Proceedings of the International Conference on Logic Programming,* Melbourne, Australia, May 1987.

[Shie84] Stuart M. Shieber. The design of a computer language for linguistic information. In *Proceedings of the Tenth International Conference on Computational Linguistics: COLING 84,* Stanford University, Stanford, California, July 2-7, 1984.