A THREE-VALUED INTERPRETATION OF NEGATION IN FEATURE STRUCTURE DESCRIPTIONS

Anuj Dawar
Dept. of Comp. and Info. Science
University of Pennsylvania
Philadelphia, PA 19104

K. Vijay-Shanker
Dept. of Comp. and Info. Science
University of Delaware
Newark, DE 19716

April 20, 1989

ABSTRACT

Feature structures are informational elements that have been used in several linguistic theories and in computational systems for natural-language processing. A logical calculus has been developed and used as a description language for feature structures. In the present work, a framework in three-valued logic is suggested for defining the semantics of a feature structure description language, allowing for a more complete set of logical operators. In particular, the semantics of the negation and implication operators are examined. Various proposed interpretations of negation and implication are compared within the suggested framework. One particular interpretation of the description language with a negation operator is described and its computational aspects studied.

1 Introduction and Background

A number of linguistic theories and computational approaches to parsing natural language have employed the notion of associating informational elements, called feature structures, consisting of features and their values, with phrases. Rounds and Kasper [KR86, RK86] developed a logical calculus that serves as a description language for these structures.

Several researchers have expressed a need for extending this logic to include the operators of negation and implication. Various interpretations have been suggested that define a semantics for these operators (see Section 1.2), but none has gained universal acceptance. In [Per87], Pereira set forth certain properties that any such interpretation should satisfy.

In this paper we present an extended logical calculus, with a semantics in three-valued logic (based on Kleene's three-valued logic [Kle52]), that includes an interpretation of negation motivated by the approach given by Karttunen [Kar84]. We show that our logic meets the conditions stated by Pereira. We also show that the three-valued framework is powerful enough to express most of the proposed definitions of negation and implication. It therefore makes it possible to compare these different approaches.

1.1 Rounds-Kasper Logic

In [Kas87] and [RK86], Rounds and Kasper introduced a logical formalism to describe feature structures with disjunctive specification. The language is a form of modal propositional logic (with modal operator ":").

In order to define the semantics of this language, feature structures are formally defined in terms of *acyclic finite automata*. These are finite-state automata whose transition graphs are acyclic. The formal definition may be found in [RK86].

A fundamental property of the semantics is that the set of automata satisfying a given formula is *upward-closed* under the operation of subsumption. This is important, because we consider a formula to be only a partial description of a feature structure. The property is stated in the following theorem [RK86]:

Theorem 1.1 *$\mathcal{A} \sqsubseteq \mathcal{B}$ if and only if for every formula, ϕ, if $\mathcal{A} \models \phi$ then $\mathcal{B} \models \phi$.*

1.2 The Problem of Adding Negation

Several researchers in the area have suggested that the logic described above should be extended to include *negation* and *implication*.

Karttunen [Kar84] provides examples of feature structures where a negation operator might be useful. For instance, the most natural way to represent the *number* and *person* attributes of a verb such as *sleep* would be to say that it is *not* third person singular rather than expressing it as a disjunction of the other five possibilities. Karttunen also suggests an implementation technique to handle negative information.

Johnson [Joh87], defined an Attribute Value Logic (AVL), similar to the Rounds-Kasper Logic, that included a classical form of negation. Kasper [Kas88] discusses an interpretation of negation and implication in an implementation of Functional Unification Grammar [Kay79] that includes conditionals. Kasper's semantics is classical, but his unification procedure uses notions similar to those of three-valued logic[1].

One aspect of the classical approach is that the property of upward-closure under subsumption is lost. Thus the evaluation of negation may not be freely interleaved with unification [2].

In [Kas88], Kasper localized the effects of negation by disallowing path expressions within the scope of a negation. This restriction may not be linguistically warranted as can be seen by the following example from Pereira [Per87] which expresses the semantic constraint that the subject and object of a clause cannot be coreferential unless the object is a reflexive pronoun:

$$obj : type : reflexive \vee \neg(subj : ref \approx obj : ref)$$

Moshier and Rounds [MR87] proposed an intuitionistic interpretation of negation that preserves upward-closure. They replace the notion of ***satisfaction*** with one of model-theoretic ***forcing*** as described in Fitting [Fit69]. They also provide a complete proof system for their logic. The satisfiability problem for this logic was shown to be PSPACE-complete.

1.3 Outline of this Paper

In the following section we will present our proposed solution in a three-valued framework, for defining the semantics of feature structure descriptions including negation[3]. This solution is a formalization of the notion of negation in Karttunen [Kar84]. In Section 3 we will show that the framework of three-valued logic is flexible enough to express most of the different interpretations of negation mentioned above. In Section 4 we will show that the satisfiability problem for the logic we propose is NP-complete.

[1] see Section 3.4

[2] see Pereira [Per87] p.1006

[3] We shall concentrate only on the problem of extending the logic to include the negation operator, and later in Section 3.4 discuss implication.

2 Feature Structure Descriptions with Negation

We will now present our extended version of the Rounds-Kasper logic including negation. We do this by giving the semantics of the logic in a three-valued setting. This provides an interpretation of negation that is intuitively appealing, formally simple and computationally no harder than the original Rounds-Kasper logic.

With each formula we associate the set (*Tset*) of automata that ***satisfy*** the formula, a set (***Fset***) of automata that ***contradict*** it and a set (*Uset*) of automata which neither satisfy nor contradict it[4]. Different interpretations of negation are obtained by varying definitions of what constitutes "contradiction." In the semantics we will define, we choose a definition in which contradiction is equivalent to ***essential incompatibility*** [5]. We will define the *Tset* and the ***Fset*** so that they are upward-closed with respect to subsumption for all formulae. Thus, we avoid the problems associated with the classical interpretation of negation. In our logic, negation is defined so that an automaton $\mathcal{A}$ satisfies $\neg\phi$ if and only if it contradicts ϕ.

2.1 The Syntax

The symbols in the descriptive language, other than the connectives $:, \vee, \wedge, \neg$ and $\approx$ are taken from two primitive domains: *Atoms (A)*, and *Labels (L)*.

The set of well formed formulae (W), is given by: NIL; TOP; a; $l : \phi$; $\phi \wedge \psi$; $\phi \vee \psi$; $\neg\phi$ and $p_1 \approx p_2$, where $a \in A$; $l \in L$; $\phi, \psi \in W$ and $p_1, p_2 \in L^*$.

2.2 The Semantics

Formally, the semantics is defined over the domain of partial functions from acyclic finite automata[6] to boolean values.

Definition 2.1 *An acyclic finite automaton is a 7-tuple* $\mathcal{A} = < Q, \Sigma, \Gamma, \delta, q_0, F, \lambda >$, *where:*

1. *Q is a non-empty finite set (of states),*
2. *Σ is a countable set (the alphabet),*

[4] A similar notion was used by Kasper [Kas88], who introduces the notion of *compatibility*. We shall compare this approach with ours in greater detail in Section 3.4.

[5] In general, a feature structure is incompatible with a formula if the information it contains is inconsistent with that in the formula. We will distinguish two kinds of incompatibility. A feature structure is *essentially* incompatible with a formula if the information in it contradicts the information in the formula. It is *trivially* incompatible with the formula if the inconsistency is due to an excess of information within the formula itself.

[6] In this paper we will not consider cyclic feature structures

3. *Γ is a countable set (the output alphabet),*

4. *$\delta : Q \times \Sigma \rightarrow Q$ is a finite partial function (the transition function),*

5. *$q_0 \in Q$ (the initial state),*

6. *$F \subseteq Q$ (the set of final states),*

7. *$\lambda : F \rightarrow \Gamma$ is a total function (the output function),*

8. *the directed graph (Q, E) is acyclic, where pEq iff for some $l \in \Sigma$, $\delta(p, l) = q$,*

9. *for every $q \in Q$, there exists a directed path from q_0 to q in (Q, E), and*

10. *for every $q \in F$, $\delta(q, l)$ is not defined for any l.*

A formula ϕ over the set of labels L and the set of atoms A is characterized by a *partial* function:

$$\mathcal{F}_\phi : \{\mathcal{A} | \mathcal{A} =< Q, L, A, \delta, q_0, F, \lambda >\} \rightarrow \{True, False\}$$

$\mathcal{F}_\phi(\mathcal{A})$ is $True$ iff $\mathcal{A}$ satisfies ϕ. It is $False$ if $\mathcal{A}$ contradicts ϕ [7] and is undefined otherwise. The formal definition is given below.

Definition 2.2 *For any formula ϕ, the partial function $\mathcal{F}_\phi$ over the set of acyclic finite automata, $\mathcal{A} =< Q, L, A, \delta, q_0, F, \lambda >$, is defined as follows:*

1. *if $\phi = NIL$ then*

   $\mathcal{F}_\phi(\mathcal{A}) = True$ *for all* $\mathcal{A}$;

2. *if $\phi = TOP$ then*

   $\mathcal{F}_\phi(\mathcal{A}) = False$ *for all* $\mathcal{A}$;

3. *if $\phi = a$ for some $a \in A$ then*

   $\mathcal{F}_\phi(\mathcal{A}) = True$
   if $\mathcal{A}$ is atomic and $\lambda(q_0) = a$
   $\mathcal{F}_\phi(\mathcal{A}) = False$
   if $\mathcal{A}$ is atomic and $\lambda(q_0) = b$ for some b, $b \neq a$ (see Note 2.)
   $\mathcal{F}_\phi(\mathcal{A})$ *is undefined otherwise*;

4. *if $\phi = l : \phi_1$ for some $l \in L$ and $\phi_1 \in W$ then*

   $\mathcal{F}_\phi(\mathcal{A}) = \mathcal{F}_{\phi_1}(\mathcal{A}/l)$ *if $\mathcal{A}/l$ is defined. (see Note 3.)*
   $\mathcal{F}_\phi(\mathcal{A})$ *is undefined otherwise*;

5. *if $\phi = \phi_1 \wedge \phi_2$ for some $\phi_1, \phi_2 \in W$ then*

   $\mathcal{F}_\phi(\mathcal{A}) = True$
   if $\mathcal{F}_{\phi_1}(\mathcal{A}) = True$ and $\mathcal{F}_{\phi_2}(\mathcal{A}) = True$
   $\mathcal{F}_\phi(\mathcal{A}) = False$
   if $\mathcal{F}_{\phi_1}(\mathcal{A}) = False$ or $\mathcal{F}_{\phi_2}(\mathcal{A}) = False$
   $\mathcal{F}_\phi(\mathcal{A})$ *is undefined otherwise* ;

6. *if $\phi = \phi_1 \vee \phi_2$ for some $\phi_1, \phi_2 \in W$ then*

   $\mathcal{F}_\phi(\mathcal{A}) = True$
   if $\mathcal{F}_{\phi_1}(\mathcal{A}) = True$ or $\mathcal{F}_{\phi_2}(\mathcal{A}) = True$
   $\mathcal{F}_\phi(\mathcal{A}) = False$
   if $\mathcal{F}_{\phi_1}(\mathcal{A}) = False$ and $\mathcal{F}_{\phi_2}(\mathcal{A}) = False$
   $\mathcal{F}_\phi(\mathcal{A})$ *is undefined otherwise* ;

7. *if $\phi = \neg\phi_1$ for some $\phi_1 \in W$ then*

   $\mathcal{F}_\phi(\mathcal{A}) = True$ *if* $\mathcal{F}_{\phi_1}(\mathcal{A}) = False$
   $\mathcal{F}_\phi(\mathcal{A}) = False$ *if* $\mathcal{F}_{\phi_1}(\mathcal{A}) = True$
   $\mathcal{F}_\phi(\mathcal{A})$ *is undefined otherwise* ;

8. *if $\phi = p_1 \approx p_2$ for some $p_1, p_2 \in L^*$ then*

   $\mathcal{F}_\phi(\mathcal{A}) = True$
   if $\delta(q_0, p_1)$ and $\delta(q_0, p_2)$ are defined and $\delta(q_0, p_1) = \delta(q_0, p_2)$
   $\mathcal{F}_\phi(\mathcal{A}) = False$
   if $\mathcal{A}/p_1$ and $\mathcal{A}/p_2$ are both defined and are not unifiable
   $\mathcal{F}_\phi(\mathcal{A})$ *is undefined otherwise (see Note 4.).*

Notes:

1. We have not included an implication operator in the formal language, since we find that defining implication in terms of negation and disjunction (i.e $\phi \Rightarrow \psi \equiv \neg\phi \vee \psi$) yields a semantics for implication that corresponds exactly to our intuitive understanding of implication.

2. As one would expect, an atomic formula is satisfied by the corresponding atomic feature structure. On the other hand, only atomic feature structures are defined as contradicting an atomic formula. Though a complex feature structure is clearly incompatible with an atomic formula we do not view it as being *essentially* incompatible with it. An interpretation of negation that defines a complex feature structure as contradicting a (and hence satisfying $\neg a$) is also possible. However, our definition is motivated by the linguistic intention of the negation operator as given by Karttunen [Kar84]. Thus, for instance, we require that an automaton satisfying the formula *case* : $\neg$*dative* have an atomic value for the *case* feature.

3. In *4.* above, we state that: $\mathcal{F}_\phi(\mathcal{A}) = \mathcal{F}_{\phi_1}(\mathcal{A}/l)$ if $\mathcal{A}/l$ is defined. When $\mathcal{A}/l$ is defined, $\mathcal{F}_{\phi_1}(\mathcal{A}/l)$ may still

[7] and therefore it satisfies the formula $\neg\phi$

be *True*, *False* or undefined. In any of these cases, $\mathcal{F}_\phi(\mathcal{A}) = \mathcal{F}_{\phi_1}(\mathcal{A}/l)$[8]. $\mathcal{F}_\phi(\mathcal{A})$ is not defined if $\mathcal{A}/l$ is not defined. Not only is this condition required to preserve upward-closure, it is also linguistically motivated.

Here again, we could have said that a formula of the form $l : \phi_1$ is contradicted by any atomic feature structure, but we have chosen not to do so for the reasons outlined in the previous note.

4. We have chosen to state that the set of automata that are incompatible with the formula $p_1 \approx p_2$ is *not* the set of automata for which $\delta(q_0, p_1)$ and $\delta(q_0, p_2)$ are defined and $\delta(q_0, p_1) \neq \delta(q_0, p_2)$, since such an automaton could subsume one in which $\delta(q_0, p_1) = \delta(q_0, p_2)$. Thus, we would lose the property of upward-closure under subsumption. However, an automaton, $\mathcal{A}$, in which $\delta(q_0, p_1)$ and $\delta(q_0, p_2)$ are defined and $\mathcal{A}/p_1$ is not unifiable[9] with $\mathcal{A}/p_2$ cannot subsume one in which $\delta(q_0, p_1) = \delta(q_0, p_2)$.

2.2.1 Upward-Closure

As has been stated before, the set of automata that satisfy a given formula in the logic defined above is upward-closed under subsumption. This property is formally stated below.

Theorem 2.1 *Given a formula ϕ and two acyclic finite automata $\mathcal{A}$ and $\mathcal{B}$, if $\mathcal{F}_\phi(\mathcal{A})$ is defined and $\mathcal{A} \sqsubseteq \mathcal{B}$ then $\mathcal{F}_\phi(\mathcal{B})$ is defined and $\mathcal{F}_\phi(\mathcal{B}) = \mathcal{F}_\phi(\mathcal{A})$.*

Proof:
The proof is by induction on the structure of the formula. The details may be found in Dawar [Daw88].

2.3 Examples

We now take a look at the examples mentioned earlier and see how they are interpreted in the logic just defined. The first example expressed the *agreement* attribute of the verb *sleep* by the following formula:

$$agreement : \neg(person : third \wedge number : singular) \quad (1)$$

This formula is satisfied by any structure that has an *agreement* feature which, in turn, either has a *person* feature with a value other than *third* or a *number* feature with a value other than *singular*. Thus, for instance, the following two structures satisfy the given formula:

$$\begin{bmatrix} cat : VP \\ agreement : \begin{bmatrix} person : second \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} agreement : \begin{bmatrix} person : third \\ number : plural \end{bmatrix} \end{bmatrix}$$

On the other hand, for a structure to contradict formula(1) it must have an *agreement* feature defined for both *person* and *number* with values *third* and *singular* respectively. All other automata would have an undefined truth value for formula(1).

Turning to the other example mentioned earlier, the formula:

$$obj : type : reflexive \vee \neg(subj : ref \approx obj : ref) \quad (2)$$

is satisfied by the first two of the following structures, but is contradicted by the third (here co-index boxes are used to indicate co-reference or path-equivalence).

$$\begin{bmatrix} obj : \begin{bmatrix} type : reflexive \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} obj : \begin{bmatrix} ref : \boxed{1} \end{bmatrix} \\ subj : \begin{bmatrix} ref : \boxed{1} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} obj : \begin{bmatrix} ref : \boxed{1} \\ type : reflexive \end{bmatrix} \\ subj : \begin{bmatrix} ref : \boxed{1} \end{bmatrix} \end{bmatrix}$$

3 Comparison with Other Interpretations of Negation

As we have stated before, the semantics for negation described in the previous section is motivated by the discussion of negation in Karttunen [Kar84], and that it is closely related to the interpretation of Kasper [Kas88]. In this section, we take a look at the interpretations of negation that have been suggested and how they may be related to interpretations in a three-valued framework.

3.1 Classical Negation

By classical negation, we mean an interpretation in which an automaton $\mathcal{A}$ satisfies a formula $\neg\phi$ if and only if it does not satisfy ϕ. This is, of course, a two-valued logic. Such an interpretation is used by Johnson in his Attribute-Value Language [Joh87]. We can express it in our framework by making $\mathcal{F}_\phi$ a total function such that wherever $\mathcal{F}_\phi(\mathcal{A})$ was undefined, it is now defined to be *False*.

Returning to our earlier example, we can observe that for formula(1) the structure

$$\begin{bmatrix} agreement : \begin{bmatrix} person : third \end{bmatrix} \end{bmatrix}$$

has a truth value of *false* in the classical semantics but has an undefined truth value in the semantics we define. This illustrates the problem of non-monotonicity in the classical semantics since this structure does subsume one that satisfies formula (1).

[8]Equality here is strong equality (*i.e.* if $\mathcal{F}_{\phi_1}(\mathcal{A}/l)$ is undefined then so is $\mathcal{F}_\phi(\mathcal{A})$.)

[9]Two automata are not unifiable if and only if they do not have a least upper bound

3.2 Intuitionistic Logic

In [MR87], Moshier and Rounds describe an extension of the Rounds-Kasper logic, including an implication operator and hence, by extension, negation. The semantics is based on intuitionistic techniques. The notion of *satisfying* is replaced by one of *forcing*. Given a set of automata $\mathcal{K}$, a formula ϕ, and $\mathcal{A}$ such that $\mathcal{A} \in \mathcal{K}$, $\mathcal{A}$ *forces in* $\mathcal{K}$ $\neg\phi$ ($\mathcal{A} \vdash_{\mathcal{K}} \neg\phi$) if and only if for all $\mathcal{B} \in \mathcal{K}$ such that $\mathcal{A} \sqsubseteq \mathcal{B}$, $\mathcal{B}$ does not force ϕ in $\mathcal{K}$. Thus, in order to find if a formula, ϕ, is satisfiable, we have to find a set $\mathcal{K}$ and an automaton $\mathcal{A}$ such that *forces in* $\mathcal{K}$ ϕ.

Moshier and Rounds consider a version in which forcing is always done with respect to the set of all automata, *i.e.* $\mathcal{K}^*$. This means that the set of feature structures that satisfy $\neg\phi$ is the largest *upward-closed* set of feature structures that do not satisfy ϕ (*i.e.* the set of feature structures *incompatible* with ϕ). We can capture this in the three-valued framework described above by modifying the definition of $\mathcal{F}_\phi$ to make it *False* for all automata that are incompatible (trivially or essentially) with ϕ (we call this new function $\mathcal{F}'_\phi$). The definition of $\mathcal{F}'_\phi$ differs from that of $\mathcal{F}_\phi$ in the following cases:

- $\phi = a$

  $\mathcal{F}_\phi(\mathcal{A}) = True$
  if $\mathcal{A}$ is atomic and $\lambda(q_0) = a$
  $\mathcal{F}_\phi(\mathcal{A}) = False$ otherwise

- $\phi = l : \phi_1$

  $\mathcal{F}'_\phi(\mathcal{A}) = True$
  if $\mathcal{F}_\phi(\mathcal{A}) = True$
  $\mathcal{F}'_\phi(\mathcal{A}) = False$
  if $\mathcal{A}/l$ is defined and
  $\forall\mathcal{B}(\mathcal{A}/l \sqsubseteq \mathcal{B} \Rightarrow \mathcal{F}'_{\phi_1}(\mathcal{B}) = False)$
  $\mathcal{F}'_\phi(\mathcal{A})$ is undefined otherwise.

- $\phi = \phi_1 \wedge \phi_2$

  $\mathcal{F}'_\phi(\mathcal{A}) = True$
  if $\mathcal{F}_{\phi_1}(\mathcal{A}) = True$ *and* $\mathcal{F}_{\phi_2}(\mathcal{A}) = True$
  $\mathcal{F}'_\phi(\mathcal{A}) = False$
  if $\forall\mathcal{B}(\mathcal{A} \sqsubseteq \mathcal{B} \Rightarrow$
  $\mathcal{F}'_{\phi_1}(\mathcal{B}) \neq True$ *or* $\mathcal{F}'_{\phi_2}(\mathcal{B}) \neq True)$
  $\mathcal{F}_\phi(\mathcal{A})$ is undefined otherwise ;

- $\phi = \phi_1 \vee \phi_2$

  $\mathcal{F}'_\phi(\mathcal{A}) = True$
  if $\mathcal{F}_{\phi_1}(\mathcal{A}) = True$ *or* $\mathcal{F}_{\phi_2}(\mathcal{A}) = True$
  $\mathcal{F}'_\phi(\mathcal{A}) = False$
  if $\forall\mathcal{B}(\mathcal{A} \sqsubseteq \mathcal{B} \Rightarrow$
  $\mathcal{F}'_{\phi_1}(\mathcal{B}) \neq True$ *and* $\mathcal{F}'_{\phi_2}(\mathcal{B}) \neq True)$
  $\mathcal{F}_\phi(\mathcal{A})$ is undefined otherwise ;

- $\phi = p_1 \approx p_2$

  $\mathcal{F}'_\phi(\mathcal{A}) = True$
  if $\delta(q_0, p_1)$ and $\delta(q_0, p_2)$ are defined
  and $\delta(q_0, p_1) = \delta(q_0, p_2)$
  $\mathcal{F}'_\phi(\mathcal{A}) = False$
  if $\mathcal{A}/p_1$ and $\mathcal{A}/p_2$ are both defined
  and are not unifiable or if $\mathcal{A}$ is atomic
  $\mathcal{F}'_\phi(\mathcal{A})$ is undefined otherwise .

In the other cases, the definition of $\mathcal{F}'_\phi$ parallels that of $\mathcal{F}_\phi$.

To illustrate the difference between $\mathcal{F}'_\phi$ and $\mathcal{F}_\phi$, we define the following (somewhat contrived) formula:

$$\phi = (l_1 : a \vee l_2 : a) \wedge l_2 : b$$

We also define the automaton

$$\mathcal{A} = [l_1 : b]$$

We can now observe that $\mathcal{F}_\phi(\mathcal{A})$ is undefined but $\mathcal{F}'_\phi(\mathcal{A}) = False$. To see how this arises, note that in either system, the truth value of $\mathcal{A}$ is undefined with respect to each of the conjuncts of ϕ. This is so because $\mathcal{A}$ can certainly be extended to satisfy either one of the conjuncts, just as it can be extended to contradict either one of them. But, for $\mathcal{F}_\phi(\mathcal{A})$ to be *False*, $\mathcal{A}$ must have a truth value of *False* for one of the conjuncts and therefore $\mathcal{F}_\phi(\mathcal{A})$ is undefined. On the other hand, since $\mathcal{A}$ can never be extended to satisfy both conjuncts of ϕ simultaneously, it can never be extended to satisfy ϕ. Hence $\mathcal{A}$ is certainly incompatible with ϕ, but because this incompatibility is a result of the excess of information in the formula itself, we say that it is only *trivially* incompatible with ϕ.

To see more clearly what is going on in the above example, consider the formula $\neg\phi$ and apply distributivity and DeMorgan's law (which is a valid equivalence in the logic described in the previous section, but not in the intuitionistic logic of this section) which gives us:

$$\neg\phi = (\neg l_1 : a \wedge \neg l_2 : a) \vee \neg l_2 : b$$

We can now see why we do not wish $\mathcal{A}$ to satisfy $\neg\phi$, which would be the case if $\mathcal{F}_\phi(\mathcal{A})$ were *False*.

One justification given for the use of forcing sets other than $\mathcal{K}^*$ is the interpretation of formulae such as $\neg h : NIL$. It is argued that since $h : NIL$ denotes all feature structures that have a feature labeled h, $\neg h : NIL$ should denote those structures that do not have such a feature. However, the formula $\neg h : NIL$ is unsatisfiable both in the interpretation given in the last section as well as in the $\mathcal{K}^*$ version of intuitionistic logic. It is our opinion that the use of negation to assert the non-existence of features is an operation distinct from the use of negation to describe values and should be described by a distinct operator. The present work attempts to deal only with the latter notion of negation. The authors expect to present in a forthcoming paper a simple extension to the current semantics that will deal with issues of existence of features.

3.3 Karttunen's Implementation of Negation

As mentioned earlier, our approach was motivated by Karttunen's implementation as described in [Kar84]. In the unification algorithm given, negative constraints are attached to feature structures or automata (which themselves do not have any negative values). When the feature structure is extended to have enough information to determine whether it satisfies or falsifies[10] the formula then the constraints may be dropped. We feel that our definition of the *Uset* elegantly captures the notion of associating constraints with automata that do not have sufficient information to determine whether they satisfy or contradict a given formula.

3.4 Kasper's Interpretation of Negation and Conditionals

As mentioned earlier, Kasper [Kas88] used the operations of negation and implication in extending Functional Unification Grammar. Though the semantics defined for these operators is a classical one, for the purposes of the algorithm Kasper identified three classes of automata associated with any formula: those that *satisfy* it, those that are *incompatible* with it and those that are *merely compatible* with it. We can observe that these are closely related to our *Tset*, *Fset* and *Uset* respectively. For instance, Kasper states that an automaton $\mathcal{A}$ satisfies a formula $f : v$ if it is defined for f with value v; it is incompatible with $f : v$ if it is defined for f with value x ($x \neq v$) and it is merely compatible with $f : v$ if it is not defined for f. In three-valued logic, we incorporate these notions into the formal semantics, thus providing a formal basis for the unification procedure given by Kasper. Our logic also gives a more uniform treatment to the negation operator since we have removed the restriction that disallowed path equivalences in the scope of a negation.

4 Computational Issues

In this section, we will discuss some computational aspects related to determining whether a formula is satisfiable or not. We will show that the satisfiability problem is NP-complete, which is not surprising considering that the problem is NP-complete for the logic not involving negation (Rounds-Kasper logic).

The NP-hardness of this problem is trivially shown if we observe that for any formula, ϕ, without negation, $Tset(\phi)$ is exactly the set of automata that satisfy ϕ according to the definition of satisfaction given by Rounds and Kasper [KR86, RK86] in their original logic. Since the satisfiability problem in that logic is NP-complete, the given problem is NP-hard.

In order to see that the given problem is in NP, we observe that a simple nondeterministic algorithm[11] can be given that is linear in the length of the input formula ϕ and that returns a minimal automaton which satisfies ϕ, provided it is satisfiable. To see this, note that the size (in terms of the number of states) of a minimal automaton satisfying ϕ is linear in the length of ϕ and verifying whether a given automaton satisfies ϕ is a problem linear in the length of ϕ and the size of the automaton. The details of the algorithm can be found in Dawar [Daw88].

5 Conclusions

A logical formalism with a complete set of logical operators has come to be accepted as a means of describing feature structures. While the intended semantics of most of these operators is well understood, the negation and implication operators have raised some problems, leading to a variety of approaches in their interpretation. In the present work, we have presented an interpretation that combines the following advantages: it is formally simple as well as uniform (it places no special restriction on the negation operator); it is motivated by the linguistic applications of feature structures; it takes into account the partial nature of feature structures by preserving the property of monotonicity under unification and it is computationally no harder than the Rounds-Kasper logic. More significantly, perhaps, we have shown that most existing interpretations of negation can also be expressed within three-valued logic. This framework therefore provides a means for comparing and evaluating various interpretations.

References

[Daw88] Anuj Dawar. *The Semantics of Negation in Feature Structure Descriptions.* Master's thesis, University of Delaware, 1988.

[Fit69] Melvin Fitting. *Intuitionistic Logic and Model Theoretic Forcing.* North-Holland, Amsterdam, 1969.

[Joh87] Mark Johnson. *Attribute Value Logic and the Theory of Grammar.* PhD thesis, Stanford University, August 1987.

[Kar84] Lauri Karttunen. Features and values. In *Proceedings of the Tenth International Conference on Computational Linguistics*, July 1984.

[10] It is not clear whether falsification is equivalent to incompatibility or only essential incompatibility, but from the examples involving case and agreement, we believe that only essential incompatibility is intended.

[11] this algorithm assumes that the set of atoms is finite.

[Kas87] Robert T. Kasper. *Feature Structures: A Logical Theory with Application to Language Analysis.* PhD thesis, University of Michigan, 1987.

[Kas88] Robert T. Kasper. Conditional descriptions in Functional Unification Grammar. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 233–240, June 1988.

[Kay79] M. Kay. Functional grammar. In *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society*, 1979.

[Kle52] S. C. Kleene. *Introduction to Metamathematics.* Van Nostrand, New York, 1952.

[KR86] 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*, 1986.

[MR87] M. Drew Moshier and William C. Rounds. A logic for partially specified data structures. In *ACM Symposium on the Principles of Programming Languages*, pages 156–167, ACM, 1987.

[Per87] Fernando C. N. Pereira. Grammars and logics of partial information. In Jean-Louis Lassez, editor, *Proceedings of the 4th International Conference on Logic Programming*, pages 989–1013, May 1987.

[RK86] William C. Rounds and Robert T. Kasper. A complete logical calculus for record structures representing linguistic information. In *IEEE Symposium on Logic in Computer Science*, pages 34–43, IEEE Computer Society, June 1986.