

# Linguistic Understanding with Speech Acts<sup>1</sup>

Donald G. Thaxton  
Graduate Student, Department of Computer Science  
North Carolina Agricultural and Technical State University  
Greensboro, NC 27411  
[dgt@ncat.edu](mailto:dgt@ncat.edu)  
Dr. Albert Esterline, advisor

## *Abstract*

*We describe a prototype scheduler implemented in the logic programming language LIFE that tracks the effects of speech acts. A speech act consists of an illocutionary force (indicated by a performative verb) and a propositional content. The effects of speech acts of interest here are obligations, permissions, prohibitions, and assertions. We have developed a context free grammar using LIFE definite clause grammar rules. To structure the values of attributes, we use feature structures. These record-like structures represent limited information. Feature structures are also used to record the effects of speech acts.*

## 1. Introduction

We present a prototype scheduler implemented in the logic programming language LIFE that tracks the effects of speech acts. Speaking a language is performing speech acts, such as making statements, giving commands, asking questions, and making promises [Se69]. There are assertive, directive, commissive, permissive, prohibitive and declarative speech acts [Si91]. Assertives are statements of fact. Directives are commands, requests, or suggestions. Commissives commit the speaker to a course of action or an obligation. Permissives issue permissions. Prohibitives take them away or enforce a prohibition. Different illocutionary forces, determined by the performative verb, distinguish these acts. Besides an illocutionary force, a speech act has a propositional content.

We have used LIFE's definite clause grammar to develop a parser for speech acts entered by the users (or agents). Using feature structures, we construct internal representations of obligations or commitments (the effects of directive and commissive speech acts), permissions, prohibitions, and assertions that agents have made. A feature structure groups together individual features, each of which associates a name with one or more values. Feature structures allow partial information to be represented. The performative scheduler detects incompatibilities that arise because the effects of speech acts conflict. It also flags cases where an obligation or prohibition is respected or violated and where an assertion proves to be true or false. The performative scheduler also supports queries about the obligations, permissions, and prohibitions that hold of an agent. Finally, it maintains a history of the outcomes of these effects: which have been respected or violated, proved true or false.

This prototype scheduler has been kept simple to facilitate implementation and analysis. The only performative verbs are *tell* and *command* (directives), *promise* (commissive), *allow* and *let* (permissives), *forbid* (prohibitive), and *tell* (assertive). A speech act here establishes an effect that relates only to one specific time. For example, we allow a commissive speech act by which Don promises Al to meet him in the lab at 13:00, thereby establishing an obligation on the part of Don.

---

<sup>1</sup> This research was supported by grant NAG 5-4102, "Formal Foundations of Agents," from NASA Goddard Space Flight Center.

We do not allow Don, for example, to promise to meet Al in the lab whenever Al is in the lab. Finally, the domain of discourse is restricted to a microworld (described below) with limited numbers of agents, places, objects, and actions.

The next section provides background on feature structures and LIFE. Section 3 describes the performative scheduler, and Section 4 outlines our speech-act parser. Section 5 concludes with a summary, the significance of this work, and suggestions for future work.

## 2. Background

A feature structure is a data structure that groups together individual features, each of which associates a name with one or more values [Ca92]. It can be represented by a directed finite labeled attribute-value graph in which the value of a feature either is undefined or is another feature structure. Feature structures employ an inheritance network of concepts defined in terms of features and values. Two feature structures can have identical type and identical values for all their features without themselves being identical. We thus distinguish between type identity and token identity. Two feature structures may be tokens of the same type (have identical type and identical values of their features) without thereby being the same feature structure (token). Use of token identity introduces *intensionality* into feature structure systems. The intensional nature of feature structures is largely due to how substructures may be accessed by sequences of features, which also allows feature structures to be cyclic. Feature structures thus allow us to capture semantic properties that transcend first-order terms, although they are sufficiently similar to first-order terms to allow the usual logical semantics and unification procedures to be extended in well-defined ways. Unification of feature structures simultaneously determines the consistency of two items of partial information and, if they are consistent, combines them into one item.

LIFE (Logic, Inheritance, Functions and Equations) [Ai94] is the programming languages used for our application. LIFE is a declarative logic-based language, whose syntax and resolution are extensions of those of Prolog. LIFE allows one to formulate efficient programs easily, concisely and naturally through the addition of functions, feature structures (called  $\psi$ -terms in LIFE) and inheritance. The syntax of a  $\psi$ -term is  $r_s(f_1=>v_1, \dots, f_n=>v_n)$ , where  $r_s$  is the *root sort* of the term and  $f_i=>v_i, 1 \leq i \leq n$ , shows value  $v_i$  for feature  $f_i$ .

## 3. Performative Scheduler

The performative scheduler stores on file the remaining schedule and the history of the success and failure of the participants. One file stores the schedule as a single  $\psi$ -term with features *assertions* (whose value is a list of assertions), *obligations* (a list of obligations), *permissions* (a list of permissions), and *prohibitions* (a list of prohibitions). The assertions, obligations, permissions, and prohibitions are all effects of speech acts and are represented by  $\psi$ -terms. The sort hierarchy of the root sorts for these effects is the sort *effect*, as depicted in Figure 1. Every effect has a *time* feature whose value is the time embedded in the content, for example, when an agent is obligated to do something. One child of the sort *effect* is

assertion. An assertion is the effect of an assertive speech act. It has (in addition to a time feature) features `prop` (the proposition asserted) and `source` (the utterer). It also has a feature `value`, used only in the history, with possible values

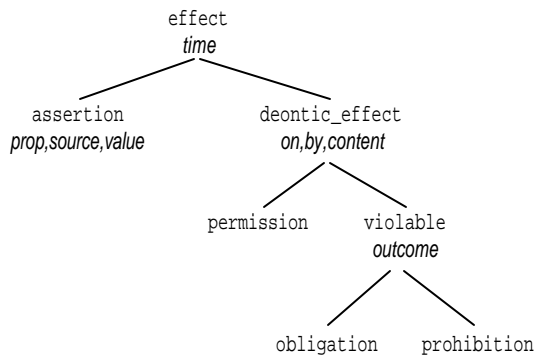


Figure 1. Root sort hierarchy for effects

true and false to indicate the truth-value found for the asserted proposition. The other child of the sort `effect` is `deontic_effect` the ancestor of the effects of speech acts involving the deontic notions of obligation, permission, and prohibition. (These notions are formalized in deontic logic, which has been used in computer science and elsewhere when one distinguishes what is actual from what is ideal.)

The features of `deontic_effect` are `on` (the agent subject to the obligation, prohibition, or permission), `by` (the agent, for example, to which the `on` agent is obligated), and `content` (the propositional content of the effect). One child of `deontic_effect` is `permission`, with no additional features. The other child is `violable`, whose children are `obligation` and `prohibition`, and which has the additional feature `outcome`, used only in the history, where it records whether the obligation or prohibition was violated or respected (discharged). The possible values of the `outcome` feature are `violated` and `respected`. These can be parameterized with the agent asserting what violates or respects the effect.

Input to the performative scheduler is of three forms: speech acts associated with agents, statements of fact about the microworld, and queries. The microworld has two agents (Don and Al), two places (the office and the lab), and two objects (a book and a disk) that one agent might give to another. There are two sorts of actions: one agent meets the other at a specified place at a specified time, and one agent gives a specified object to the other at a specified place at a specified time. A speech act is entered in the form

*⟨Agent⟩ : I ⟨Performative verb⟩ ⟨Agent⟩ ⟨Clause⟩ ⟨Time⟩.*

An example of a speech act is

*Don : I tell Al to meet Don in the lab at 13:30 at 11:20.*

Note that pronouns other than “I” are not used, and that two times are mentioned: the first is part of the `⟨Clause⟩` category hence associated with the content of the speech act, while the second is the time of utterance of the speech act. The parser returns the abstract syntax tree and the illocutionary force for this speech act, from which the performative scheduler constructs an `effect` feature structure, which it attempts to insert into the schedule.

When the performative scheduler attempts to insert the effect of a speech act into the schedule, it must check earlier entries for conflicts. Space limitations preclude discussion of the rules for detecting such conflicts or how such conflicts are handled. An assertion at odds with an obligation or prohibition expresses a violation of the obligation or prohibition. An assertion can also express that an obligation or prohibition has been respected. In either case, the obligation or prohibition is moved from the schedule to the history, with the value of its `outcome` feature set to

violated(A) or respected(A), where A is the agent making the assertion. Since an assertion may be contradicted by a later assertion, effects that have been relegated to the history in this way may later be reinstated in the schedule.

Input to the performative scheduler also includes statements of fact. Such a statement consists simply of a clause, giving a content of the same form as that of a speech act, and a time. Statements of fact model events in our. We assume that statements of fact never. A statement of fact, like an assertion, may express violation or respect of an obligation or prohibition. With statements of fact, however, a deontic effect is irrevocably relegated to the history, and the value of the `outcome` feature lacks the agent argument. An assertion is relegated to the history with `true` or `false` for its `value` feature when a statement of fact, respectively, confirms or conflicts with it. The notion of the current time is supplied by the time entered with a statement of fact. For the performative scheduler to keep pace with the current time, a degenerate form of a statement of fact is allowed in which only the time is entered. In a more realistic setting, the time would be read from a clock. As time passes, an agent may fail to meet an obligation scheduled for a particular time. This is treated as a violation, and the obligation is accordingly relegated to the history. Indeed, as the current time advances beyond the time of any effect, that effect is relegated to the history. A prohibition that has not be violated is given `respected` as the value of its `outcome` feature, a permission is relegated without modification, and an assertion that has been neither conflicted not confirmed receives no value for its `value` feature.

The information the performative scheduler supplies includes information about conflicts, violations, and cases where deontic effects are respected. There is also a query facility, which currently uses a simple, stylized grammar requiring only a trivial parser. A user may ask for all the deontic effects for the remainder of the schedule, all the deontic effects for a certain window in time, the time of a given deontic effect, or the next obligation for a given agent.

## 4. The Parser

Space restrictions allow us to present only a fragment of our definite clause grammar for speech acts. Our scanner returns a list of tokens, which is the input for the parser. The general form of a grammar rule in LIFE, as in Prolog, is

```
cat --> chcat1, ..., chcatn.
```

where `cat` is a parent syntactic category and `chcat1, ..., chcatn` are its child categories. The preprocessor adds two additional arguments to the head (`cat`) and to the subgoals (`chcat1, ..., chcatn`); in LIFE, these are the values of the features `dcg_in` and `dcg_out`. Thus, the above is translated into something of the form

```
cat(dcg_in => I, dcg_out => O) :-
    chcat1(dcg_in=>I, dcg_out=>M1), chcat2(dcg_in=>M1, dcg_out=>M2),
    ..., chcatn(dcg_in => Mn-1, dcg_out => O).
```

That is, the input to the parent category is the input to the first child category and the output of the parent category is the output from the last category. The output of the  $i^{\text{th}}$  child category is the input to the  $(i+1)^{\text{st}}$  child category,  $1 \leq i \leq n-1$ . So each child category consumes an initial segment of the token list input to it that corresponds to the tokens constituting a realization of that category. The parent category consumes all the tokens consumed by its children. A subgoal

of the form [  $\tau$  ] consumes a token  $\tau$  appearing at the head of the token list when that subgoal is encountered. In both Prolog and LIFE, arguments may be used with the head and subgoals of a definite clause; in the translated clauses, these appear as additional arguments.

We use one additional argument, which is a feature structure. There are different root sorts (arranged in an inheritance hierarchy) for the value of this feature, depending on the category. Every feature structure produced has an `ast` feature, whose value is the abstract syntax tree for the segment of the token list parsed as the corresponding category. The `ast` value is a structure whose root indicates the category and whose arguments are the abstract syntax trees for the children of the category in question. The feature structures for the categories that comprehend the main verb include a `force` feature to indicate the illocutionary force of the speech act. Other features have semantic values constraining how lexical items are combined.

The top-level category is `tagged_sentence`, which consists of a proper noun (denoting the utterer), then a `:`, and finally the speech act itself (as a sentence):

```
tagged_sentence(t_sent(ast => tagged(NT,ST), force => F)) -->
  noun(n(ast => NT, proper => yes)), [':'],
  sentence(sent(ast => ST, force => F)).
```

The sentence begins with [ `i` ], the person who is speaking, and contains a verb phrase:

```
sentence(sent(ast => s(pn(i),VPT), force => F)) --> [i],
  verb_phrase(verb_p(ast => VPT, force => F)).
```

There are two `verb_phrase` clauses relevant to this context. Each one contains a verb (which identifies the illocutionary force), a noun (which identifies the hearer), a locative (which denotes the place), and a `time_ref` (which gives a time of the transaction). One clause has an infinitive clause (`inf_clause`) stating the propositional content of the speech act.

```
verb_phrase(verb_p(ast => vp(VT,NT,ICT,TT), force => F)) -->
  verb(v(ast => VT, agree => ag(num => sing, pers => first),
    force => F:{commissive;directive;permissive;prohibitive})),
  noun(n(ast => NT, proper => yes)),
  inf_clause(inf_cl(ast => ICT)),
  time_ref(tr(ast => TT)).
```

An `inf_clause` involves an infinitive, a noun, a locative and a `time_ref`. The other `verb_phrase` clause uses a relative clause to state the propositional content.

Verbs are represented as single lexical items (enclosed in [...]). Besides the `ast` feature, a verb has an `agree` feature (for agreement information, needed to constrain the noun phrases with which it may be coupled) and a `force` feature (indicating its illocutionary force, which is `non_performative` if no illocutionary force is associated with the verb). A verb in infinitive form is given simply `infinite` for the value of its `agree` feature; otherwise, this feature indicates the number and person of the verb form. Some examples are

```
verb(v(ast => ver(meet), agree => infinite,
  force => non_performative)) --> [meet].
verb(v(ast => ver(meets), agree => ag(num => sing, pers => third),
  force => non_performative)) --> [meets].
verb(v(ast => ver(promise), agree => ag(num => sing, pers => first),
  force => commissive)) --> [promise].
```

We consider only relative clauses that begin with *that*.

```
rel_clause(rel_cl(ast => rc(DST))) -->
  [that], decl_sentence(decl_s(ast => DST)).
```

The `decl_sentence` contains a noun phrase and a verb phrase

```
decl_sentence(decl_s(ast => ds(NPT,VPT))) -->
  noun_phrase(noun_p(ast => NPT, agree => A)),
  verb_phrase(verb_p(ast => VPT, agree => A)).
noun_phrase(noun_p(ast => np(NT), agree =>
  ag(num => N, pers => third))) -->
  noun(n(ast => NT, num => N, proper => yes)).
```

The `verb_phrase` was discussed earlier. The `noun_phrase` contains a noun or a determiner and a noun. Nouns are identified as singular or plural, proper or common.

## 5. Conclusion

We have described a prototype performative scheduler that tracks the effect of speech acts. This scheduler has a limited vocabulary and addresses a restricted microworld. The prototype is implemented in the language LIFE and uses definite clause grammar rules and feature structures. This work has clear significance for natural language understanding [A195], where speech acts have been largely ignored. It also has potential for human-computer interaction, where deontic notions (as arise with the effects of speech acts) have not yet been applied despite their latent importance. Finally, although agent communication languages are generally discussed in terms of speech acts, the grammars of the performative aspects of these languages are much less developed than they are here. Something akin to our performative scheduler could be used for coordination in a system of artificial agents.

Many enhancements to our performative scheduler could be pursued. A larger repertoire of performative verbs could be used, and the microworld could be expanded. We could allow effects of speech acts that are more general. For example, we could allow someone to prohibit another from ever giving a book to another person, as opposed to prohibiting him from giving a book to the other person at a particular time and place. Finally, more convenient input and output modalities can be envisioned. For example, voice recognition could be adapted for our lexicon and grammar, voice output could be synthesized, sensor input could be used to supply facts, and a clock could be used for recording the passage of time.

## References

- [Ai94] Ait-Kaci, H. *The Wild LIFE Handbook* (prepublication edition). Paris: DEC Paris Research Laboratory, 1994.
- [A195] Allen, James. *Natural Language Understanding. 2<sup>nd</sup> Edition* Redwood City, Ca: Benjamin/Cummings. 1995.
- [Ca92] Bob Carpenter, *The Logic of Typed Feature Structures*. Cambridge, UK: Cambridge University Press, 1992.
- [Se69] Searle, John R. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge, UK: Cambridge University Press, 1969.
- [Si91] Singh, M. P. "A Semantics for Speech Acts", in M. N. Huhns and M. P. Singh (eds.), *Readings in Agents*. San Francisco, CA: Morgan Kaufmann, 1998, pp. 458-470.