Fluid Construction Grammars. A Brief Tutorial

Luc Steels
Vrije Universiteit Brussel Artificial Intelligence Laboratory
SONY Computer Science Laboratory - Paris.

September 29, 2004

Abstract

In construction grammars, grammar rules (called constructions) combine, modulate, and enhance the meanings supplied by lexical items. The paper gives a short informal introduction to a particular computational implementation of construction grammars, called Fluid Construction Grammar (FCG). FCG emphasises bi-directional application and extreme fluidity in rule-application.

1 Introduction

Fluid Construction Grammar (FCG) is a formalisation and computational implementation of construction grammars. It has been under development since 2000, as a basis for experiments in the origins and evolution of language. This paper is a very brief informal introduction to Fluid Construction Grammar. All examples are drawn from a working implementation. We do not claim that FCG is the only possible way to formalise or implement construction grammars, nor that the specific view on construction grammar implied by the formalism is the 'right' one. Instead our design should be seen as a contribution to the ongoing linguistics dialog and to the developing body of implementation techniques.

1.1 What are Constructions?

Although there are multiple interpretations of the notion of a construction, most linguists agree that a construction is a pairing of a particular syntactic pattern with a particular semantic pattern (usually called a semantic frame). However, a construction does more than simply relate the two. It supplies additional information required for semantic interpretation, over and above

the information already provided by lexical items. Hence, the meaning of a sentence becomes more than the meaning of the individual parts. The lexicon supplies the meanings of individual words, and the grammar combines and modulates these meanings to compose the whole.

Figure 1 gives an example of a construction. A syntactic pattern Subject+Predicate+DirectObject+*to*+PrepObject is related to the semantic frame TRANSFER-TO-TARGET+Agent+Patient+ Target frame. The construction is applied to the specific case of a slide event, as in "Jill slides blocks to Jack", represented as a predicate with several arguments: slide1, slide2, etc. The same construction is more generally applicable to many other TRANSFER-TO-TARGET situations, as in: "Jill gives the block to Jack" or "Jane sent a letter to her mother".

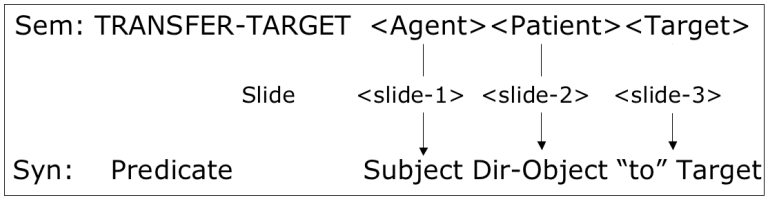

Figure 1: A construction relates a syntactic pattern such as Subject+Predicate+DirectObject+PrepObject with a semantic frame such as TRANSFER-TO-TARGET+Agent+Patient+Target.

The construction in figure 1 tightens the meanings supplied by the lexicon. For the sentence "Jill slides blocks to Jack", the lexicon provides the initial set of predicates to yield:
$Jill(?r), slide(?u), slide1(?u, ?x), slide2(?u, ?y), slide3(?u, ?z),$
$block(?v), Jack(?w)$
(Variables are denoted by letters preceded by a question mark.) Note that the variable ?r to be bound to the object named Jill is not equal to the variable ?x to be bound to the object causing the slide event. Similarly for ?v (the block) and ?y (the object implicated in the slide event) or ?w (Jack) and ?z (the target of the slide event). The role of the construction is to link these variables and thus clarify the argument structure, so that after applying the construction, the target meaning becomes:
$Jill(?x), slide(?u), slide1(?u, ?x), slide2(?u, ?v), slide3(?u, ?w),$
$block(?v), Jack(?w)$.

Here are some more examples of constructions (from French) which all involve the same verb 'rendre'. Because the verb is each time embedded in a different construction, the meanings become quite different as well, showing

that the meaning of the whole is a function of the meanings contributed by the lexicon *and* those contributed by the grammatical construction:

1a. Il me *rend* malade [Subject+DirectObject+*rendre*+Predicate]
1b. Lit: He me makes sick (He makes me sick)
2a. Je me *rends* a la maison. [Subject+DirectObject+*rendre*+*a*+PrepObject]
2b. Lit: I me go to the house (I go home)
3a. Il me *rend* mon livre [Subject+IndirectObject+*rendre*+DirectObject]
3b. He me give my book (He gives me my book)

Constructions clearly have different degrees of specifity (i.e. idiomaticity), ranging from very idiomatic constructions built around a particular noun or verb, to very general constructions with wide applicability, such as Subject+Predicate+DirectObject (as in "John gives a book"). Constructions thus form networks where more specific constructions inherit from more general ones and combine with each other to achieve higher expressive power. Moreover empirical observations of actual language use shows that the inventory of constructions used by an individual speaker (including as an adult) is constantly changing. Constructions capture conventionalised patterns of usage, but new patterns develop all the time and others may go out of fashion.

Just like there can be lexical homonymy (one word having multiple meanings) and synonymy (one meaning expressed with several words), there can be grammatical homonymy and synonymy in the sense that the same construction can have multiple effects on meaning or vice-versa the same effect can be expressed with more than one meaning. For example, an alternative construction for the one in figure 1 with a similar semantic function uses the Subject+Predicate+ IndirectObject+DirectObject pattern as in "Jill slides Jack the block" or "Jane sent her mother a letter".

1.2 Utility of Constructions

Grammatical constructions are useful for three reasons:

- They can express constraints on meaning which go beyond those of individual words, as the example of argument structure has illustrated.
- They help to reduce the size of the lexicon because the same word (like *rendre*) can be used productively in many different constructional contexts.

- They allow listeners to infer the meaning or usage of unknown words by using the semantic frames that the construction provides. For example, in the sentence "Jill frooples a box to Jack" we know that Jill somehow transfers the box to Jack, even though we do not know what "frooples" means. If a real world scene is available showing a physical transfer between Jill and Jack with specific characteristics (perhaps Jill throws the box to Jack), then semantic interpretation becomes possible and a good guess can be made of the meaning of "frooples". Such bootstrapping strategies have been demonstrated in children for the acquisition of new verb meanings.

- Constructions can be combined in a relatively open-ended way to build up more complex constructions. For example, the sentence "John slides a block off the table" can be viewed as a combination of two constructions: a CAUSE-MOVE+agent+patient frame expressed as a subject+predicate+patient pattern and a MOVE+object+trajectory frame expressed as a predicate+patient+prep+object pattern.

1.3 Syntactic and Semantic Categories

In order to apply a construction, it must be prepared both on the syntactic and the semantic side. On the syntactic side the pattern must be recognised in parsing (or introduced in producing) which amounts to recognising/introducing a set of syntactic constraints on word order, parts of speech (Noun, Verb, etc.), agreement, presence of function words, etc. For example, the construction in figure 1 uses a function word TO, imposes agreement in number and person between subject and verb, and introduces an SVO+pO ordering relation among the constituents.

On the semantic side, the events and objects must be conceptualised in terms of semantic frames. Thus the construction in figure 1 requires that the slide-event is seen as a specific instance of a transfer-to-target frame and the objects playing various roles in this event are conceptualised as the agent, patient and target of a transfer event.

To express all these syntactic and semantic constraints on the application of constructions requires a repertoire of syntactic and semantic categories. There are a very large number of categories (surely in the thousands) involved in the grammar of any specific natural language and there is known to be considerable variation among languages concerning the specific inventory of syntactic categories they employ (for example, Japanese has no syntactic gender, Chinese has no syntactic tense nor makes a tight distinction between

adjectives and nouns, etc.). Moreover syntactic categories exhibit prototype behavior with clear uses and boundary cases (e.g. many nouns can function as verbs in English, objects or events can often be coerced into new roles that violate their 'natural' selection restrictions, etc.)

A distinction can be made between natural and grammatical syntactic categories. The number of a noun (phrase) is typically telling us something about the status of the entity it refers to, namely whether it involves a single-object (singular) or a set of objects (plural). Number is in this case a natural syntactic category in the sense that it contributes to meaning. On the other hand, the number of a verb is not related to the event described by the verb. Number is in this case a grammatical syntactic category used for implementing specific syntactic phenomena (agreement between subject and predicate). Grammatical categories often start by being meaningful and then adopt a purely grammatical function. Thus the gender of nouns is meaningful in the case of animate beings (compare actor versus actress) but becomes arbitrary when it is generalised to other kinds of objects which have no natural gender. For example, moon is neuter in English, feminine in French or Italian, and masculine in German. This is not unlike some words (like "by" in "she is loved by him") which start out as being meaningful and then become grammaticalised as function words.

1.4 Types of Rules

To represent grammatical constructions we will need two types of rules. The first type prepares the application of constructions through syntactic or semantic categorisations. The second type implements the constructions themselves by relating aspects of semantic structure to aspects of syntactic structure. In addition, we need lexical rules which are similar to constructions in the sense that they also relate aspects of semantic structure to aspects of syntactic structure. Although rules thus fall into classes based on the role they play in the overall process, formally speaking the rules all have the same structure (defined shortly) and they could all be put into one big rule-set in which rules become active whenever they are applicable. The remainder of these sections lists the various rule-types as summarised in figure 2.

There are first of all two classes of rules which relate syntactic to semantic structure:

1. Lexical rules (lex-rules) map a lexical item into a set of clauses that specify the meaning of that item. They can be further subdivided into:

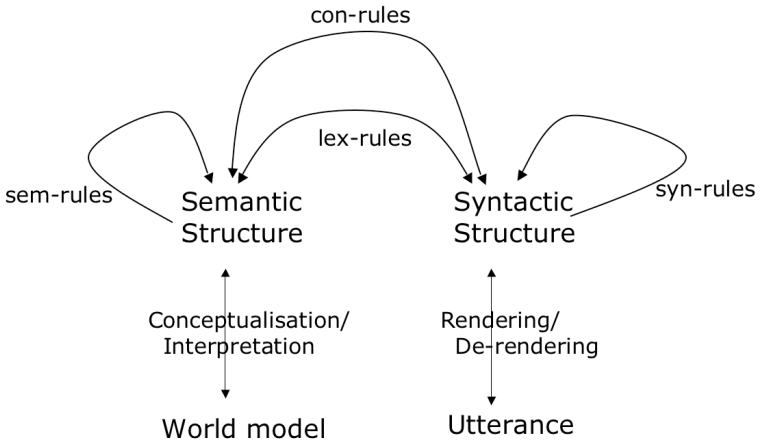

Figure 2: The different types of rules that are employed for constraining the semantic and syntactic aspects of a sentence structure.

   (a) Lexical stem rules (lex-stem-rules) which map the stem of words to a set of predicates.
   (b) Lexical category rules (lex-cat-rules) which map 'natural' syntactic categories (like the number of a noun, gender in the case of animate entities, tense in the case of the main verb, etc.) to additional meanings.

2. Construction rules (con-rules) map parts of syntactic structure (i.e. units with particular syntactic categories) into parts of semantic structure (i.e. semantic categories of some units or possibly additional clauses to be added to the meaning). These rules therefore implement constructions of the type shown in figure 1.

Next there are two classes of rules which prepare the application of constructions by adding syntactic resp. semantic categories:

1. Syntactic categorisation rules add syntactic categories to the syntactic structure. They come in two types:

   (a) Morphological rules (morph-rules) decompose words into a stem with affixes and add syntactic categories, such as gender or part of speech.
   (b) Syntactic rules (syn-rules) relate syntactic properties of a sentence (for example word order) to additional syntactic categories,

such as subject or direct-object.

2. Semantic categorisation rules (sem-rules) add semantic categories to the semantic structure. For example, they map the arguments of a specific predicate, like 'slide' into the more abstract TRANSFER-TO-TARGET frame. Sem-rules are the only rules that are uni-directional because both in parsing and in production they are applied in the same way: the left-pole matches and the right pole merges.

Examples follow.

1.5 When are Construction Grammars Fluid?

Any construction grammar formalism must obviously be capable to express the kinds of rules mentioned and to implement parsing and production processes. But there are two additional requirements that are characteristic for the Fluid Construction Grammar formalism discussed here: bi-directionality and fluidity.

1. In FCG all rules are applicable both in producing (i.e. constructing an utterance that expresses specific meanings derived through a conceptualisation process from a grounded world model) and in parsing (reconstructing the meaning of an utterance and mapping it back into reality by way of the grounded world model). This is a tough technical requirement which will be achieved by viewing constructions as constraints, and language processing as constraint propagation.

2. We assume that the inventory of an agent is not necessarily complete. Consequently agents must be capable to parse or produce a sentence 'as far as possible'. Even though a sentence may be ungrammatical, parsing should still try to come up with a reasonable parse, and even if the lexicon or grammar is incomplete, the agent should try to come up with an expression which covers as much as possible.

3. We also assume that the inventory of an agent is not necessarily conform to that of other agents in the population. As new words, new categories, and new constructions may appear at any time, each agent must keep track of competing hypotheses about what conventions are floating around in the population and choose the one that is likely to have the most communicative success. This will be handled by giving all rules a particular strength which reflects expected communicative success. Agents change the strength of their rules to become more

aligned to the verbal behavior of others in the population based on success or failure in the communication.

4. After every interaction, agents must be prepared to update their language inventory, either because they had to invent new forms to express meanings that they had not yet expressed before, or because they had to adopt new form-meaning mappings.

Because fluidity is a defining characteristic of our computational formalism, we call it 'Fluid Construction Grammar'. It imposes tough requirements on the datastructures and processing mechanisms that will be needed.

2 The FCG Formalism

2.1 Logic as Base Language

Logic is generally accepted in AI for representing both the facts in a world model and the meaning that is expressed by a sentence, and we follow that tradition. For example, we may have the following facts as a result of perceiving and categorising a scene:

```
slide(ev1), slide-1(ev1,obj1),
slide-2(ev1,obj2), slide-3(ev1,obj3),
name(obj1,Jill), status(obj1,single-object),
block(obj2), status(obj2,single-object),
name(obj3,Jack), status(obj3,single-object)
```

There are three 'single' objects involved and one slide-event. The different roles of objects in the event are represented as separate predicates instead of arguments of a single slide-predicate: slide introduces the event itself, slide-1 introduces the one who is doing the sliding, slide-2 the object involved, and slide-3 the target destination or goal towards which the object is moved. We use English words for the names of predicates, instead of predicate-1, predicate-2, etc. just to make it easier to understand what is going on, but these names should not be confused with 'real' English words and the meaning of these predicates comes entirely from how they are grounded in the world and in language.

Variables are denoted with a question mark in front and they are treated as logic variables that get bound by matching processes. For example, the following expression could be part of the parsed meaning of "Jill slides Jack the block":

```
slide(?ev1), slide-1(?ev1,?obj1),
slide-2(?ev1,?obj2), slide-3(?ev1,?obj3)
```

This matches with expressions in the world model above to give the following set of bindings:

```
((?ev1 . ev1) (?obj1 .obj1) (?obj2 . obj2) (?obj3 . obj3))
```

FCG uses a logic-based representation also for the syntactic and semantic categorisations and the constraints on semantic or syntactic structures that characterise a particular construction. Syntactic categories are predicated over units such as:

```
string(unit-1,"john"), precedes(unit-1,unit-2),
number(singular,natural), direct-object(unit-1,unit-2),
tense(unit2, past),  ...
```

The final form of a sentence is described in a declarative fashion. For example, "John walks home" is described as

```
string(unit-1, "john"), string(unit-2,"walks"), string(unit-3,"home")
preceeds(unit-1,unit-2), preceeds(unit-2,unit-3)
```

The explicit representation of ordering relations makes it much easier to deal with relatively free word order or parse sentences even if the word order constraints are partially violated. Instead of the preceeds predicate, there is an alternative sequence which can be used to order more than one unit as in:

```
sequence(unit-1,unit-2,unit-3)
```

Semantic categories are predicates over the entities in the domain of discourse, such as

```
time-period(?ev1, before-now), agent(ev1, obj1), etc.
```

These semantic categories are (re-)conceptualisations of the 'conceptual' predicates directly derived from the sensory datastreams.

Note that there is never any ordering intended, for example.

```
preceeds(unit-1,unit-2), string(unit-3,"home"), preceeds(unit-2,unit-3)
string(unit-2,"walks"), string(unit-1, "john")
```

is totally equivalent with the form description given earlier.

2.2 Representing Syntactic and Semantic Structure

Language processing consists in building up the semantic and syntactic aspects of a sentence structure. In FCG, one is not done before the other (as in strictly modular approaches to language processing) but both are built up at the same time. Syntactic and semantic structures will be represented as feature structures, compatible with the main trend in computation linguistics.

The sentence structure consists of units with slots containing information about the unit. There is no ordering among the slots and a slot is filled by a set of elements which are themselves also considered to be unordered. The elements are either atomic, or expressions in predicate calculus notation, or variables (which are atomic symbols preceded with a question mark as in *?event*). In contrast to other feature structure formalisms, feature structures are not used hierarchically as fillers of slots. All units are located at the same level but are labeled (unit-1, unit-2, etc.) and hierarchical structure is explicitly represented with the slots syn-subunits (for syntactic subunits) and sem-subunits (for semantic subunits). The labels allow reference to a unit, predication over units, for example to express ordering constraints, and the use of variables that become bound to specific units. This turns out to be much more efficient from a computational point of view.

2.3 Components of Syntactic and Semantic Structure

Basically there is a unit for every word in the sentence and for every group of words that forms a larger unit. But there may be units which have only a semantic role (i.e. they have no direct analog in the form) and units which have only a syntactic role (e.g. for grammatical function words like "by" in a passive construction which have no direct analog in meaning).

The main slots for the semantic structure of a unit are:

1. Referent: which is the entity in the world-model the unit is about (or a variable that will become bound to an entity)

2. Meaning: the clauses that will be used to identify the referent.

3. Sem-cat: the semantic categories in which the entity referred to by the unit participates.

4. Sem-subunits: the set of subunits of this unit from the viewpoint of semantic structure.

An example is shown in figure 8.

The main slots for the syntactic structure of a unit are:

1. Form: which is a declarative description of the form in terms of strings, stems, affixes, and precedence or sequencing relations between units
2. Syn-cat: the syntactic categories that are associated with the unit, such as number or gender.
3. Syn-subunits: the set of subunits of this unit from the viewpoint of syntax.

An example is shown in figure 6.

2.4 Representing Rules

In line with unification grammar formalisms, language processing will be viewed as a kind of inference process and mechanisms pioneered in logic programming and rule-based systems can therefore be employed to implement parsing and production. The sentence structure (syntactic and semantic) can be viewed as the 'fact base' over which rules operate. More specifically, each rule is a 4-tuple, written as:

```
def-<rule-type> <rule-name> <strength>
  <left pole> <--> <right pole>
```

The ⟨ strength ⟩ is the strength of the rule, a numerical value between 0.0 and 1.0 which is adjusted based on success of the rule in actual communication. The strength influences whether the rule will win against other competing rules. In this document the strength is not explored further and left out from the rule definitions.

The different definitions corresponding to the rule types introduced earlier are:

def-morph	morphological rules
def-lex-stem	lexical stem rule
def-lex-cat	lexical category rule
def-sem	semantic categorisation rule
def-syn	syntactic rule
def-con	construction

Both the left pole and the right pole are partial descriptions of sentence structure. The filler of a slot specifies which elements have to be part of the slot. It can be of three types:

1. Either the filler of a slot in a rule must be *exactly equal* to the filler in the sentence structure. In the PC-notation this will be written with curly brackets {, }.

2. The filler of a slot contains some of the elements that must be *included* in the filler of the sentence structure but there can be additional elements. This is written in the PC-notation by simply writing the list after the name of the slot.

3. The filler of a slot contains some of the elements included in the target but these elements can occur only once. In this case, the label ==1 is written before the element. [need more explanation here]

An example of a rule is shown in figure 3. It is a morphological rule which decomposes the word "slides" into a stem and some syntactic categories. Such a rule expands in both directions the syntactic structure with additional information. In production, the stem and syntactic categories are known and the word "slides" is added. In parsing, the word "slides" is known and the stem and syntactic categories are added. Both person and number are grammatical as opposed to natural syntactic categories because they are not meaningful for verbs but only play a role in agreement phenomena. Tense is a natural syntactic category for verbs.

```
def-morph "slides"
  ?unit
    syn-cat: verb,person(3d,grammatical),
             number(singular,grammatical)
    form: stem(?unit,"slide")
 <-->
  ?unit
    form: string(?unit,"slides")
```

Figure 3: Example of a morphological rule that decomposes the word "slides" into a stem and a number of syntactic categories.

In contrast to other unification-based formalisms, rules do not use numerical indices for referring back to other parts of a structure, but variables, which can be bound not only to specific items, such as the value for the syntactic feature person, but also to units themselves (as ?unit in figure 3).

2.5 Rule Application

The bi-directional application of a rule is based on the following algorithm:

1. PRODUCING (i.e. go from meaning to form)
   (a) Match the left pole of a rule against the structure already built. If the rule matches, this yields a set of bindings for the variables or a set of equalities between variables (because the sentence structure may also contain variables).
   (b) Merge the right pole of the rule with the sentence structure. Merging means that the instantiated right pole is first matched against the sentence structure, possibly yielding additional bindings, and then the union is taken of the further instantiated right pole and the sentence structure.
2. PARSING (i.e. go from form to meaning)
   (a) Match the right pole of a rule against the sentence structure. If the rule matches, this yields a set of bindings for the variables or a set of equalities between variables (because the sentence structure may also contain variables).
   (b) Merge the left pole of the rule with the sentence structure. Merging means that the instantiated left pole is first matched against the sentence structure, possibly yielding additional bindings, and then the union is taken of the further instantiated left pole and the sentence structure.

So producing and parsing are totally analogous, the only thing which changes is the direction of rule application. The type of the rule determines which part of the sentence structure (syntactic or semantic) is used in matching and merging. For example, in the case of a syn-rule (syntactic rule) both the left and the right pole are syntactic, whereas in the case of a con-rule (a construction) the left pole is semantic and the right pole syntactic.

3 An Example of Parsing

Let us now look at some examples of rules, all relevant for the parsing and production of the example sentence "Jill slides blocks to Jack". We first look at how the rules are used for parsing and then show later that exactly the same rules work for production. All rules are listed in the appendix. The ordering of rule-set application in parsing is as follows:

```
morph-rules =>
  lex-stem-rules =>
    lex-cat-rules =>
      sem-cat-rules =>
        syn-rules =>
          con-rules => final meaning
```

3.1 Morphological and Lexical Rules

A morphological rule that decomposes "slides" into a stem and a set of syntactic categories is shown in figure 3. Number and person are 'grammatical' as opposed to 'natural', because they do not contribute to meaning. The rule is applicable in both directions. If the stem and the syntactic categories are known (in production) the rule adds the right pole, namely the string "slides". If the string is seen during parsing, the left pole is unified with the syntactic structure providing information on the stem and the syntactic categorisation of this unit. The use of the affix "-s" is of course productive in English and therefore the rule in figure 3 should just split up "slides" into its two morphemes with another rule associating singular 3d person with "-s", but we simplify here because of space limitations.

Figure 4 is an example of a lex-stem rule that constrains the semantic structure based on the presence of the stem "slide" and figure 5 is an example of a lex-cat rule that relates a 'natural' syntactic feature namely singular with an aspect of meaning, namely that there is a single object. Another syntactic feature that will be used is 'person' which refers to the discourse role of the object (either participant as speaker (1st person) or hearer (2nd person) or external (3d person)).

```
def-lex-stem "slide"
 ?unit
  referent: ?ev
   meaning: slide(?ev),slide-1(?ev,?obj1)
       slide-2(?ev,obj2),slide-3(?ev,obj3)
 <-->
 ?unit
  form: stem(?unit, "slide")
```

Figure 4: Example of a lexical stem rule that associates the word stem "slide" with predicates to be added to the meaning of a semantic structure.

```
def-lex-cat singular-cat
 ?unit
   referent: ?x
   meaning: status(?x,single-object)
 <-->
 ?unit
   syn-cat: number(singular,natural)
```

Figure 5: Example of a lex-cat rule that associates a natural syntactic feature, in this case number, with a predicate added to the meaning in the semantic structure.

Suppose the agent is parsing the sentence "Jill slides blocks to Jack", then after application of morphological and lexical rules like the ones shown in these figures, the syntactic structure built so far looks as in figure 6. "to" remains unanalysed.

3.2 Preparing the Application of a Construction

An example of a semantic categorisation rule is shown in figure 7. It maps a specific predicate (in this case 'slide') into a semantic categorisation (in this case 'TRANSFER-TO-TARGET') that will be useful later as a selection restriction for the application of the TRANSFER-TO-TARGET construction. Notice that the arrow only goes in one direction. Semantic categorisation rules are indeed the only rules that are uni-directional, simply because a slide event is a TRANSFER-TO-TARGET event but not every TRANSFER-TO-TARGET is a slide event.

Figure 8 shows the semantic structure built up in parallel with the syntactic structure in figure 6 after application of the lex-stem and lex-cat rules and after application of the semantic categorisation rule in figure 7. Note that the variables introducing the different objects and those introducing the roles in the slide event are not yet equal. So the semantic structure does not yet contain the information that it is Jill who is the agent in the event, the block the patient, and Jack the target. Note also how unit3 contains a meaning slot as well as semantic categories.

An example of a syntactic rule is shown in figure 9. It extends the syntactic structure in figure 6 with additional syntactic categorisations. The rule recognises/specifies the Subject+Predicate+DirectObject+PrepObject pattern. The left pole of the rule contains this abstract pattern and the

```
unit1
 syn-cat: {sentence}
 form: {sequence(unit2,unit3,unit4,unit5,unit6)}
 syn-subunits: {unit2,unit3,unit4,unit5,unit6}
unit2
 syn-cat: {noun,person(3d,natural),
           number(singular,natural)}
 form: {string(unit2,"jill"), stem(unit2,"jill")}
unit3
 syn-cat: {verb,person(3d,grammatical),
           number(singular,grammatical)}
 form: {string(unit3,"slides"),stem(unit3,"slide")}
unit4
 syn-cat: {noun,person(3d,natural),
           number(plural,natural)}
 form: {string(unit5,”blocks"),stem(unit5,"block")}
unit5
 form: {string(unit5,"to")}
unit6
 syn-cat: {noun,person(3d,natural),
           number(singular,natural)}
 form: {string(unit4,"jack"), stem(unit4,"jack")}
```

Figure 6: Syntactic structure built during parsing after application of morphological and lexical rules

right pole the syntactic constraints for the realisation of this pattern. These include specific parts of speech, ordering constraints, and agreement between subject and predicate for number and person. Agreement is expressed by using the same variables.

The right pole of the syntactic rule in figure 9 matches with the syntactic structure built up so far (shown in figure 6). The different unit variables bind to the units in the sentence structure (e.g. ?subject gets bound to unit2, ?predicate to unit3, etc.) and after merging the instantiated left pole with the sentence structure in figure 6 we get the structure shown in figure 10.

3.3 Applying the Construction

Figure 11 shows an example of a construction. It implements the one shown schematically in figure 1. The main function of this construction is to ensure that the variables introducing the various objects and the roles in the event become equal, in other words that it is known what the roles are of the objects introduced by the various noun phrases in the event introduced by

```
def-sem slide-transfer-to-target
 ?unit
   meaning: slide(?event),slide(?event,?obj1),
            slide-2(?event,?obj2), slide-3(?event,?obj3)
 -->
 ?unit
  sem-cat: transfer-to-target(?event),agent(?event,?obj1),
           patient(?event,?obj2),target(?event,?obj3)
```

Figure 7: Example of semantic rule that reconceptualises the slide event in terms of a TRANSFER-TO-TARGET frame. This kind of rule is exceptional because it is always applied in the same direction (from left to right pole).

the verb. The work is done by the unification of the left pole of the rule with the semantic structure already built (see figure 8). The construction is shown in figure 11) which is a direct implementation of the one shown graphically in figure 1.

The semantic structure after application of the construction in figure 11 is shown in figure 12. The units in the right pole have been matched with units in the sentence structure resulting in a series of bindings (for example ?event-unit was bound to unit3). Then the left pole, after instantiating these variables, was matched against the semantic structure yielding additional variable bindings, for example, ?agent became bound to ?obj-20 and ?obj1-11, ?patient to ?obj-4 and ?obj3-13, etc. After instantiating the left pole with these new bindings (which includes the introduction of a single variable for all variables that are equal) it is merged with the semantic structure in figure 8 to yield the one shown in figure 12. Note that it was absolutely necessary to extract additional bindings by matching the (instantiated) left pole against the sentence structure before merging it.

3.4 Extracting the Final Meaning

The meaning of the phrase can be extracted from this semantic structure by taking the union of the meanings of all the units:

```
Jill(?obj-20), status(?obj-20, single-object),
discourse-role(?obj-20,external), slide(?ev-9),
slide-1(?ev-9,?obj-20), slide-2(?ev-9,?obj2-14),
slide-3(?ev-9,?obj-17), block(?obj-14),
status(?obj-14,object-set), discourse-role(?obj-14,external),
```

```
unit1
 sem-subunits: {unit2,unit3,unit4,unit6}
unit2
  referent: {?obj-20}
  meaning {jill(?obj-20), status(?obj-20,single-object),
          discourse-role(?obj-20,external)}
unit3
  referent: {?ev-9}
  meaning: {slide(?ev-9),slide-1(?ev-9,?obj-20),
            slide-2(?ev-9,?obj14),slide-3(?ev-9,?obj-17)}
  sem-cat: {transfer-to-target(?ev-9),agent(?ev-9,?obj-20),
            patient(?ev-9,?obj-14),target(?ev-9,?obj-17)}
unit4
  meaning: {block(?obj-14), status(?obj-14,object-set),
            discourse-role(?obj-14,external)}
unit6
  referent: {?obj-17}
  meaning: {jack(?obj-17), status(?obj-17,single-object),
            discourse-role(?obj-17,external)}
```

Figure 8: Semantic structure built after application of lexical rules and semantic categorisation rules

```
Jack(?obj-17), status(?obj-17, single-object),
discourse-role(?obj-17,external)
```

When this expression is matched against the world model, bindings for all variables are to be found. If there is a unique set, the listener has found a unique interpretation for the sentence.

4 An Example of Production

We now examine how the same set of rules can be used in production. The process starts from the meaning:

```
slide(ev1,true), slide-1(ev1,obj1), slide-2(ev1,obj2),
slide-3(ev1,obj3), block(obj2), status(obj2,object-set),
discourse-role(obj2,external), Jill(obj3),
status(obj3,single-object), discourse-role(obj3,external),
Jack(obj1), status(obj1,single-object), discourse-role(obj1,external)
```

The ordering of rule-set application for production is as follows:

```
lex-stem-rules =>
  lex-cat-rules =>
```

```
def-syn SVOtoO-sentence
 ?top-unit
   syn-cat: SVOtoO-sentence
   syn-subunits: ?predicate,?subject,?direct-object,
                 ?prep-object
 ?predicate:
   syn-cat: predicate(?top-unit,?predicate)
 ?subject:
   syn-cat: subject(?top-unit,?subject)
 ?direct-object:
   syn-cat: direct-object(?top-unit,?direct-object)
 ?prep-object
   syn-cat: prep-object(?top-unit,?prep-object)
<-->
?top-unit
    syn-cat: sentence
    form: sequence(?subject,?predicate,?direct-object
                 ?prep,?prep-object)
  ?predicate
    syn-cat: verb,number(?number,grammatical),
             person(?person,grammatical)
  ?subject
    syn-cat: noun,number(?number,?number-type),
             person(?person,?person-type)
  ?direct-object:
    syn-cat: noun
  ?prep
    form: string(?prep-unit,"to")
  ?prep-object
    syn-cat: noun
```

Figure 9: Example of syntactic rule specifying the syntactic constraints on the Subject+Predicate+DirectObject+PrepObject pattern

```
sem-cat-rules =>
  con-rules =>
    syn-rules =>
      morph-rules => final form
```

Let us start from the semantic structure in figure 13 which contains in the top-unit the meaning to be expressed.

4.1 Applying Lexical and Semantic Categorisation Rules

The first step now is the application of the rules of the lexicon in order to cover the meaning in unit1. Each time a lex-stem rule is applied, a new unit is created both in the syntactic and the semantic structure. The syntactic

```
unit1
 syn-cat: {sentence}
 form: {sequence(unit2,unit3,unit4,unit5,unit6)}
 syn-subunits: {unit2,unit3,unit4,unit5,unit6}
unit2
 syn-cat: {noun,person(3d,natural),
           number(singular,natural)}
 form: {string(unit2,"jill"), stem(unit2,"jill")}
unit3
 syn-cat: {verb,person(3d,grammatical),
           number(singular,grammatical)}
 form: {string(unit3,"slides"),stem(unit3,"slide")}
unit4
 syn-cat: {noun,person(3d,natural),
           number(plural,natural)}
 form: {string(unit5,"blocks"),stem(unit5,"block")}
unit5
 form: {string(unit5,"to")}
unit6
 syn-cat: {noun,person(3d,natural),
           number(singular,natural)}
 form: {string(unit4,"jack"), stem(unit4,"jack")}
```

Figure 10: Expansion of syntactic structure after application of the syntactic rule. The grammatical relations (subject, predicate, etc.) have been added
.

structure contains the stem for the unit and the semantic structure the part of the meaning that is covered by that stem. Once units for the words exist, the lex-cat rules can be applied, which add 'natural' syntactic features to the corresponding units. For example, the lex-cat rule in figure 5 adds singular if the object status is 'single-object'. At the same time, the sem-rules can add additional semantic categories thus preparing the application of a construction.

The resulting syntactic structure is shown in figure 14. There are units for each of the words that have covered part of the semantic structure. Note that there are no parts of speech assigned yet and that the verb has not yet received features for number or person. There are also no relational categories yet like subject, predicate, direct-object, etc. The semantic structure (after lexical rules and semantic categorisation rules have been applied) is shown in figure 15.

```
def-cons transfer-to-target-construction
  ?top-unit
    sem-subunits:
      ?event-unit,?agent-unit,?target-unit,?patient-unit
  ?event-unit
    referent: ?event
    sem-cat: transfer-to-target(?event),agent(?event,?agent)
       patient(?event,?patient),target(?event,?recipient)
   ?agent-unit
     referent: ?agent
   ?patient-unit
     referent: ?patient
   ?target-unit
     referent: ?recipient
  <-->
   ?top-unit
     syn-cat: SVOtoO-sentence
     syn-subunits:
      ?event-unit,?agent-unit,?patient-unit,?target-unit
    ?event-unit
      syn-cat: predicate(?top-unit,?event-unit)
    ?agent-unit
      syn-cat: subject(?top-unit,?agent-unit)
    ?patient-unit
      syn-cat: direct-object(?top-unit,?patient-unit)
    ?target-unit
      syn-cat: prep-object(?top-unit,?target-unit)
```

Figure 11: Example of a construction which relates a TRANSFER-TO-TARGET frame to a Subject+Verb+Direct-Object+to+Prep-Object pattern

4.2 Applying the Construction

The next step is the application of the TRANSFER-TO-TARGET construction (figures 1 and 11). The left-pole of the construction matches with the semantic structure in figure 15 so that the right pole can be merged with the syntactic structure, giving the expanded result shown in figure 16. The unit variables of the left pole were bound in the matching (e.g. ?event-unit is bound to unit3) and the right pole was instantiated with these variable bindings and then merged with the syntactic structure. The new syntactic structure contains now a specification of the grammatical relations (subject, predicate, etc.).

```
unit1
 sem-subunits: {unit2,unit3,unit4,unit6}
unit2
  referent: {?obj-20}
  meaning {jill(?obj-20),status(?obj-20,single-object),
           discourse-role(?obj-20,external)}
unit3
  referent: {?ev-9}
  meaning: {slide(?ev-9),slide-1(?ev-9,?obj-1-11),
            slide-2(?ev-9,?obj2-12),slide-3(?ev-9,?obj3-13)}
  sem-cat: {transfer-to-target(?ev-9),agent(?ev-9,?obj1-11),
            patient(?ev-9,?obj2-12),target(?ev-9,?obj3-13)}
unit4
  referent:{?obj-14}
  meaning: {block(?obj-14),status(?obj-14,object-set),
            discourse-role(?obj-14,external)}
unit6
  referent: {?obj-17}
  meaning: {jack(?obj-17),status(?obj-17,single-object),
            discourse-role(?obj-17,external)}
```

Figure 12: The final semantic structure after application of the construction. The variables are now equalised.

4.3 Shaping the Surface Form

Now the syn-rule shown in figure 9 can be applied, so that the Subject+Verb+ Direct-Object+to+Prep-Object pattern gets translated into surface forms. The left-pole of the syn-rule matches with the syntactic structure built up so far. ?predicate is bound to unit3, ?subject to unit2, etc. Then the instantiated right-pole is matched with the syntactic structure giving additional bindings (e.g. ?number is bound to singular and ?person to 3d) and the further instantiation is merged with the syntactic structure to yield the one in figure 17.

All that remains to be done is the application of the morph-rules, which turn the stem plus syntactic features into wordstrings. The final utterance is extracted directly from the syntactic structure by taking the union of all the form constraints. Ignoring the stem specifications, which have no influence on rendering, this yields:

```
sequence(unit2,unit3,unit4,unit5, unit6),
string(unit2,"Jill"), string(unit3,"slides"),
string(unit4,"blocks"), string(unit5,"to"), string(unit6,"Jack"),
sequence(unit2,unit3,unit4,unit5,unit6)
```

which is rendered as “Jill slides blocks to Jack”.

```
unit1
 meaning:
  {jill(obj1),status(obj1,single-object),
   discourse-role(obj1,external),
   slide(ev1),slide-1(ev1,obj1),slide-2(ev1,obj2),
   slide-3(ev1,obj3),block(obj2),
   status(obj2,object-set),
   discourse-role(obj2,external),
   jack(obj3),status(obj3,single-object),
   discourse-role(obj3,external)}
```

Figure 13: Initial semantic structure when production process commences. The meaning is a subset of the facts contained in the world model.

```
unit1
 syn-cat: {sentence}
 syn-subunits: {unit2,unit3,unit4,unit6}
unit2
 syn-cat: {person(3d,natural),number(singular,natural)}
 form: {stem(unit2,"jill")}
unit3
 form: {stem(unit3,"slide")}
unit4
 syn-cat: {person(3d,natural),number(plural,natural)}
 form: {stem(unit5,"block")}
unit6
 syn-cat: {person(3d,natural),number(singular,natural)}
 form: {stem(unit4,"jack")}
```

Figure 14: Syntactic structure after decomposition into different words and the application of lex-cat rules that turn aspects of meaning into syntactic features.

5 Notes on the Implementation

FCG can either run as a single-agent implementation in which the rule-sets as well as the syntactic and semantic structures are global variables.

5.1 Implementation of Structures

The current implementation of FCG uses LISP as substrate and follows the standard AI practice of prefix notation instead of infix notation for logic expressions. Parentheses appear before predicates and arguments are listed without commas, as illustrated in the following example:

```
unit1
 sem-subunits: {unit2,unit3,unit4,unit6}
unit2
  referent: {obj1}
  meaning {jill(obj1),status(obj1,single-object),
           discourse-role(obj1,external)}
unit3
  referent: {ev1}
  meaning: {slide(ev1),slide-1(ev1,obj1),
            slide-2(ev1,obj2),slide-3(ev1,obj3)}
  sem-cat: {transfer-to-target(ev1),agent(ev1,obj1),
            patient(ev1,obj2),target(ev1,obj3)}
unit4
  referent: {obj2}
  meaning: {block(obj2),status(obj2,object-set),
            discourse-role(obj2,external)}
unit6
  referent: {obj3}
  meaning: {jack(obj3),status(obj3,single-object),
            discourse-role(obj3,external)}
```

Figure 15: Semantic structure after application of the lexical rules which have decomposed the total semantic structure into separate units, one for each word, and after application of the sem rules which added semantic categories.

```
(slide ?ev1) (slide-1 ?ev1 ?obj1)
(slide-2 ?ev1 ?obj2) (slide-3 ?ev1 ?obj3)
```

which is equivalent to:

```
slide(?ev1), slide-1(?ev1,?obj1),
slide-2(?ev1,?obj2), slide-3(?ev1,?obj3)
```

There is no significance to the ordering of the elements.

Semantic and syntactic structures are implemented as lists:

```
((<unit-name-1>
  (<slot-1> <filler-1>)
  ... )
 ...)
```

The fillers are lists of elements. There is no ordering among the units nor among the elements making up a filler.

Here is an example of a syntactic structure:

```
unit1
 syn-cat: {sentence,SVOtoO-sentence}
 syn-subunits: {unit2,unit3,unit4,unit6}
unit2
 syn-cat: {person(3d,natural),number(singular,natural),
           subject(unit1,unit2)}
 form: {stem(unit2,"jill")}
unit3
 syn-cat: {predicate(unit1,unit3)}
 form: {stem(unit3,"slide")}
unit4
 syn-cat: {person(3d,natural),number(plural,natural),
           direct-object(unit1,unit3)}
 form: {stem(unit5,"block")}
unit6
 syn-cat: {person(3d,natural),number(singular,natural)
           prep-object(unit1,unit6)}
 form: {stem(unit4,"jack")}
```

Figure 16: Expanded syntactic structure after application of the TRANSFER-TO-TARGET SVOtoO construction. Grammatical relations have been added.

```
((unit1
    (form ((sequence unit2 unit3 unit4 unit5 unit6)))
    (syn-cat (sentence))
    (syn-subunits (unit2 unit3 unit4 unit5 unit6)))
 (unit2
    (form ((stem unit2 jack) (string unit2 jack)))
    (syn-cat (noun (person 3d natural)
                   (number singular natural))))
 (unit3
    (form ((stem unit3 slide) (string unit3 slides)))
    (syn-cat (verb (person 3d grammatical)
                   (number singular grammatical))))
 (unit4
    (form ((stem unit4 block) (string unit4 blocks)))
    (syn-cat (noun (person 3d natural)
                   (number plural natural))))
 (unit5
    (form ((string unit5 to))))
 (unit6
    (form ((stem unit6 jill) (string unit6 jill)))
```

```
unit1
 syn-cat: {sentence,SVOtoO-sentence}
 form: {sequence(unit2,unit3,unit4,unit5, unit6)}
 syn-subunits: {unit2,unit3,unit4,unit5,unit6}
unit2
 syn-cat: {noun,person(3d,natural),number(singular,natural)
           subject(unit1,unit2)}
 form: {stem(unit2,"jill")}
unit3
 syn-cat: {verb,person(3d,grammatical),
           number(singular,grammatical),predicate(unit1,unit3)}
 form: {stem(unit3,"slide")}
unit4
 syn-cat: {noun,person(3d,natural),
           number(plural,natural),direct-object(unit1,unit3)}
 form: {stem(unit5,"block")}
unit5
 form: {string(unit5,"to")}
unit6
 syn-cat: {noun,person(3d,natural),number(singular,natural)
           prep-object(unit1,unit6)}
 form: {stem(unit4,"jack")}
```

Figure 17: Syntactic structure after application of the syn-rule that translates the SVOtoO pattern into its surface form.

```
    (syn-cat (noun (person 3d natural)
                   (number singular natural)))))
```

Note how the unit-names can themselves be arguments of predicates constraining the filler of a slot.

Here is an example of a semantic structure:

```
((unit1
    (sem-subunits (unit2 unit3 unit4 unit5 unit6)))
 (unit2
    (meaning
     ((discourse-role ?x-126 external)
      (status ?x-126 single-object) (jack ?x-126)))
    (referent ?x-126))
 (unit3
    (meaning
     ((slide ?x-122 ?x-129) (slide-1 ?x-122 ?x-123)
      (slide-2 ?x-122 ?x-124) (slide-3 ?x-122 ?x-125)))
    (referent ?x-122)
    (sem-cat
     ((cause-move-target ?x-122)
```

```
     (agent ?x-122 ?x-123)
     (patient ?x-122 ?x-124) (target ?x-122 ?x-125))))
 (unit4
    (meaning
     ((discourse-role ?x-128 external)
      (status ?x-128 object-set) (block ?x-128)))
    (referent ?x-128))
 (unit5)
 (unit6
    (meaning
     ((discourse-role ?x-127 external)
      (status ?x-127 single-object)
      (jill ?x-127)))
    (referent ?x-127)))
```

This structure is at some stage in parsing (before application of the construction). The variables have been created by the parser. Note that there is a unit (unit-5) which does not have any associated slots. This unit has only a syntactic specification (it is the one for the function-word "to"). It is perfectly possible that more units appear in the syntactic structure than in the semantic structure or vice-versa.

5.2 Implementation of Rules

The FCG implementation uses a straightforward list-notation for the rules. The left pole as well as the right pole of a rule contain a list of units and each unit contains a list of features with a slot and a filler. For case 1 (all the elements must appear in the target structure but no other elements may), the filler contains the list of all the elements. For case 2 (all the elements must appear but the target structure may contain more of them), the filler contains a list starting with the includes sign ==. For case 3 (the elements must uniquely appear), the filler contains the uniquely-includes sign writen as ==1. An example of a rule written in list-notation (ignoring strength) is as follows:

```
(def-morph "slides"
   ((?unit
     (syn-cat (== verb (person 3d grammatical)
                  (number singular grammatical)))
     (form (==1 (stem ?unit "slide")))))
   <-->
```

```
((?unit
  (form (==1 (string ?unit "slides"))))))
```

In order for the left pole to match against the syntactic structure, the syn-cat slot must contain noun, (person 3d grammatical), and (number singular grammatical) and the form slot must contain the stem “slide”. In order for the right pole to match against the semantic structure, the form slot must contain the string ”slides”. The rule is bi-directional. When one pole matches, the other pole is added to the structure being built.

5.3 Rule-sets

The rule-sets are available as global variables:

```
*morph-rules*
*lex-stem-rules*
*lex-cat-rules*
*ps-rules*  ; should become *syn-rules*
*map-rules* ; should become *cons-rules*
```

Rule-sets are cleared using the following functions, each of them clearing one type of rule-set:

```
(clear-morph-rules)
(clear-lex-stem-rules)
(clear-lex-cat-rules)
(clear-ps-rules) ; should become clear-syn-rules
(clear-map-rules) ; should become clear-cons-rules
```

Rule sets are applied by the following function:

```
(apply-rule-set <rule-set> <rule-direction>)
```

where ¡rule-set¿ is one of the rule-sets and ¡rule-direction¿ is either -¿ if the rules are applied from left-pole to right-pole (as in production) or ¡- if they are to be applied from right-pole to left-pole (as in parsing).

5.4 Inspecting and Setting Structures

The following functions can be used to inspect the syntactic resp. semantic structures:

```
(show-syn)
(show-sem)
(show-current)
```

The last function shows both the current syntactic and semantic structure.

The following functions allow initialisation of the syntactic or semantic structure. This operation needs to be done with care!

```
(set-syn <syntactic-structure>)
(set-sem <semantic-structure>)
```

5.5 Rendering and derendering

The function render extracts from a syntactic structure the corresponding sequence of words:

```
(render (get-syn)) => ("Jack" "slides" "blocks" "to" "Jill")
```

The function de-render extracts from a set of words the initial syntactic structure:

```
(de-render '("Jack" "slides" "blocks" "to" "Jill"))
```

6 Conclusions

This brief tutorial introduced some of the major concepts of Fluid Construction Grammars and gave an example of parsing and production using the same set of rules. To remain didactic, many details have been left out and there are still many issues which need to be thought through to reach a formalism that is adequate for the full power of natural languages.

7 Acknowledgement

The Fluid Construction Grammar design progressively arose from experiments in the self-organisation of grammar, particularly grammars of case. The first implementation was achieved by Luc Steels in 2001 and used in case grammar experiments. A new implementation of the matcher was built by Nicolas Neubauer in 2002. In 2003, Joachim de Beule and Joris Van Looveren re-implemented the process engine and generalised to a larger population. Joachim De Beule then developed an application for evolving grammars of tense. In 2004 Joachim de Beule reimplemented the matcher and together with Luc Steels worked out several additional examples presented at the 3d Construction Grammar conference in Marseille.

The research is supported by the Sony Computer Science Laboratory (Paris), the CNRS OHLL project on simulating grammar evolution, and the ESF OHMM project.

. **APPENDIX I Rule sets**

(still need to change the names of the definitions)

```
; MORPH RULES decompose the string into a stem and grammatical categories
; it is the first step in parsing and the last step in production

(clear-morph-rules)

(def-morph-rule "Jack"
     ((?unit
       (syn-cat (== noun (person 3d natural)
   (number singular natural)))
       (form (==1 (stem ?unit "Jack")))))
   <-->
   ((?unit
     (form (==1 (string ?unit "Jack"))))))

(def-morph-rule "Jill"
   ((?unit
     (syn-cat (== noun (person 3d natural)
                  (number singular natural)))
     (form (==1 (stem ?unit "Jill")))))
   <-->
   ((?unit
     (form (==1 (string ?unit "Jill"))))))

(def-morph-rule "slides"
   ((?unit
     (syn-cat (== verb (person 3d grammatical)
                  (number singular grammatical)))
     (form (==1 (stem ?unit "slide")))))
   <-->
   ((?unit
     (form (==1 (string ?unit "slides"))))))

(def-morph-rule "blocks"
   ((?unit
     (syn-cat (== noun (person 3d natural)
                  (number plural natural)))
```

```
     (form (==1 (stem ?unit "block")))))
   <-->
   ((?unit
     (form (==1 (string ?unit "blocks"))))))

;; LEX-STEM RULES map parts of meaning to word stems

(clear-lex-stem-rules)

(def-lex-stem-rule Jack-entry
     ((?unit
       (referent ?obj)
       (meaning (== (Jack ?obj)
   (status ?obj ?status)
   (discourse-role ?obj ?role)))))
   <-->
   ((?unit
     (form (==1 (stem ?unit "Jack"))))))

(def-lex-stem-rule Jill-entry
     ((?unit
       (referent ?obj)
       (meaning (== (Jill ?obj)
   (status ?obj ?status)
   (discourse-role ?obj ?role)))))
   <-->
   ((?unit
     (form (==1 (stem ?unit "Jill"))))))

(def-lex-stem-rule block-entry
     ((?unit
       (referent ?obj)
       (meaning (== (block ?obj)
   (status ?obj ?status)
   (discourse-role ?obj ?role)))))
   <-->
   ((?unit
     (form (==1 (stem ?unit "block"))))))

(def-lex-stem-rule slide-entry
```

```
     ((?unit
       (referent ?ev)
       (meaning (== (slide ?ev ?truth)
   (slide-1 ?ev ?obj1)
   (slide-2 ?ev ?obj2)
   (slide-3 ?ev ?obj3)))))
   <-->
   ((?unit
     (form (==1 (stem ?unit "slide"))))))


;; LEX-CAT RULES map parts of meaning to grammatical categories
;; are to be applied in parsing after LEX-STEM rules

(clear-lex-cat-rules)

(def-lex-cat-rule singular-cat
     ((?unit
       (referent ?x)
    (meaning (== (status ?x single-object)))))
  <-->
  ((?unit
    (syn-cat (== (number singular natural))))))

(def-lex-cat-rule plural-cat
  ((?unit
    (referent ?x)
    (meaning (== (status ?x object-set)))))
  <-->
  ((?unit
    (syn-cat (== (number plural natural))))))

(def-lex-cat-rule 3d-person-cat
  ((?unit
    (referent ?x)
    (meaning (== (discourse-role ?x external)))))
  <-->
  ((?unit
    (syn-cat (== (person 3d natural))))))
```

```
;; SEM-CAT RULES categorise aspects of meaning into
;; language-internal semantic categories
;; they are applied after lex-rules in parsing

(clear-sem-rules)

(def-sem-rule slide-cause-move-target
    ((?event-unit
      (referent ?event)
      (meaning (== (slide ?event ?truth)
                   (slide-1 ?event ?obj1)
                   (slide-2 ?event ?obj2)
                   (slide-3 ?event ?obj3)))))
   -->
   ((?event-unit
     (sem-cat (== (cause-move-target ?event)
                  (agent ?event ?obj1)
                  (patient ?event ?obj2)
                  (target ?event ?obj3))))))

;; PS rules are phrase structure rules relating the
;; grammatical structure to grammatical functions used in constructions
;; They are applied in parsing after the morph rules

(clear-ps-rules)

(def-ps-rule SVOtoO-sentence
   ((?top-unit
     (syn-cat (== SVOtoO-sentence))
     (syn-subunits (== ?event-unit ?agent-unit ?target-unit ?patient-unit)))
    (?event-unit
     (syn-cat (== (predicate ?top-unit ?event-unit))))
    (?agent-unit
     (syn-cat (== (subject ?top-unit ?agent-unit))))
    (?patient-unit
     (syn-cat (== (direct-object ?top-unit ?patient-unit))))
    (?target-unit
     (syn-cat (== (prep-object ?top-unit ?target-unit)))))
   <-->
   ((?top-unit
```

```
     (syn-cat (== sentence))
     (syn-subunits
      (== ?event-unit ?agent-unit
          ?target-unit ?patient-unit ?prep-unit))
     (form
      (== (pattern ?agent-unit ?event-unit ?patient-unit
                   ?prep-unit ?target-unit))))
    (?event-unit
     (syn-cat (== verb (number ?number grammatical)
                  (person ?person grammatical))))
    (?agent-unit
     (syn-cat (== noun (number ?number natural)
                  (person ?person natural))))
    (?patient-unit
     (syn-cat (== noun)))
    (?prep-unit
     (form ((string ?prep-unit "to"))))
    (?target-unit
     (syn-cat (== noun)))))

;; MAP rules implement constructions
;; they map part of meaning into part of form

(clear-map-rules)

(def-map-rule cause-move-target-frame
   ((?top-unit
     (sem-subunits
      (== ?event-unit ?agent-unit ?target-unit ?patient-unit)))
    (?event-unit
     (referent ?event)
     (sem-cat (== (cause-move-target ?event)
  (agent ?event ?agent)
  (patient ?event ?patient)
  (target ?event ?recipient))))
    (?agent-unit
     (referent ?agent))
    (?patient-unit
     (referent ?patient))
    (?target-unit
```

```
   (referent ?recipient)))
<-->
((?top-unit
  (syn-cat (== SVOtoO-sentence))
  (syn-subunits
   (== ?event-unit ?agent-unit ?patient-unit ?target-unit)))
 (?event-unit
  (syn-cat (== (predicate ?top-unit ?event-unit))))
 (?agent-unit
  (syn-cat (== (subject ?top-unit ?agent-unit))))
 (?patient-unit
  (syn-cat (== (direct-object ?top-unit ?patient-unit))))
 (?target-unit
  (syn-cat (== (prep-object ?top-unit ?target-unit)))))
```

. **APPENDIX II Running the system**

```
;;;
;;;=====================================================
;;; EXAMPLE OF PARSING
;;;
;;;=====================================================

;; DE-RENDERING: "Jack slides blocks to Jill" produces:

(set-syn
  ((unit1
    (syn-cat (sentence))
    (form ((pattern unit2 unit3 unit4 unit5 unit6)))
    (syn-subunits (unit2 unit3 unit4 unit5 unit6)))
   (unit2
    (form ((string unit2 "Jack"))))
   (unit3
    (form ((string unit3 "slides"))))
   (unit4
    (form ((string unit4 "blocks"))))
   (unit5
    (form ((string unit5 "to"))))
   (unit6
    (form ((string unit6 "Jill"))))))

;; <- application of MORPH RULES
(show-morph-rules)
(apply-rule-set *morph-rules* '<-)
(show-syn)

;; <- application of LEX-STEM rules
(show-lex-stem-rules)
(apply-rule-set *lex-stem-rules* '<-)
(show-current)

;; <- application of LEX-CAT rules
(show-lex-cat-rules)
```

```
(apply-rule-set *lex-cat-rules* '<-)
(show-sem)

;; -> application of SEM rules
(show-sem-rules)
(apply-rule-set *sem-rules* '->)
(show-sem)

;; <- application of PS-rules
(show-ps-rules)
(apply-rule-set *ps-rules* '<-)
(show-current)

;; <- application of MAP rules
(show-map-rules)
(apply-rule-set *map-rules* '<-)
(extract-meanings (get-sem))
```

```
;;;
;================================================
;;; EXAMPLE OF PRODUCTION
;;;
;================================================

(set-sem
  ((unit1
    (referent (ev1))
    (meaning
     ((slide ev1 true)
      (slide-1 ev1 obj1)
      (slide-2 ev1 obj2) (slide-3 ev1 obj3)
      (block obj2) (status obj2 object-set)
      (discourse-role obj2 external)
      (Jill obj3) (status obj3 single-object)
      (discourse-role obj3 external)
      (Jack obj1) (status obj1 single-object)
      (discourse-role obj1 external))))))

;; -> application of LEX-STEM rules
;; decomposes both syn-struct and sem-struct into separate
;; units
(apply-rule-set *lex-stem-rules* '->)
(show-current)
;; -> application of LEX-CAT rules
;; add additional grammatical categories based on meaning
(show-lex-cat-rules)
(apply-rule-set *lex-cat-rules* '->)

;; -> application of SEM rules
;; add additional semantic categories to semantic structure
(apply-rule-set *sem-rules* '->)

;; -> application of map-rules
(apply-rule-set *map-rules* '->)

;; -> application of ps-rules
(apply-rule-set *ps-rules* '->)
;; -> application of morph-rules
```

```
(show-morph-rules)
(apply-rule-set *morph-rules* '->)
(show-current)

(render (get-syn))
; => ("Jack" "slides" "blocks" "to" "Jill")
```