Fluid Construction Grammar

Joachim De Beule (VUB Artificial Intelligence Lab)

Technieken van de Artificiele Intelligentie 2
Linguistic Knowledge

- In FCG, general linguistic knowledge is stored in rules or templates. Knowledge about a specific phrase is kept in a semantic and syntactic structure.
- Each template has a left and a right pole.
- Typically, one pole is unified (matched) with one of the structures and, if successful, the other pole is merged with the other structure to obtain a new structure.
Example Semantic Structure

(((Top-unit
  (sem-subunits (Noun-unit)))

(Noun-unit
  (meaning ((filter-set-prototype R S p)))
  (context ((link R S)))
  (sem-subunits (ball-unit)))

(ball-unit
  (meaning ((prototype p [ball])))
  (context ((link p))))]}
Example Semantic Structure

((Top-unit
  (sem-subunits (Noun-unit)))

(Noun-unit
  (meaning ((filter-set-prototype R S p)))
  (context ((link R S)))
  (sem-subunits (ball-unit)))

(ball-unit
  (meaning ((prototype p [ball])))
  (context ((link p)))))
Example Semantic Structure

((Top-unit
  (sem-subunits (Noun-unit)))
(Noun-unit
  (meaning ((filter-set-prototype R S p))))
  (context ((link R S)))
  (sem-subunits (ball-unit)))
(ball-unit
  (meaning ((prototype p [ball])))
  (context ((link p))))

features of the Noun-unit
Example Semantic Structure

\[
\text{((Top-unit}
\text{(sem-subunits (Noun-unit)))}
\]

\[(\text{Noun-unit}
\text{(meaning ((filter-set-prototype R S p)))})
\]

\[(\text{context ((link R S))})
\text{(sem-subunits (ball-unit)))}
\]

\[(\text{ball-unit}
\text{(meaning ((prototype p [ball]))})
\]

\[(\text{context ((link p))}))
\]
((Top-unit
  (syn-subunits (Noun-unit)))
(Noun-unit
  (syn-cat ((constituent Noun)))
  (syn-subunits (ball-unit)))
(ball-unit
  (syn-cat ((lex-cat Noun) (number singular)))
  (form ((string “ball”))))
Example lexical rule application

Initial Semantic Structure:
((Top-unit
  (meaning ((filter-set-prototype R S p)
    (prototype p [ball]))))))

Initial Syntactic Structure:
((Top-unit))
Example lexical rule application

Initial Semantic Structure:
((Top-unit

(meaning ((filter-set-prototype R S p)

(prototype p [ball])))))))

Initial Syntactic Structure:
((Top-unit))

Part that should `trigger' the ball-rule's left pole

=> Unification
Example lexical rule application

=> Left pole of lex-stem-rule:

((?Top-unit

    (meaning (==

        (prototype ?p [ball])))))))
Example lexical rule application

=> Left pole of lex-stem-rule:
((?Top-unit
  (meaning (=
    (prototype ?p [ball]))))))

=> - Variable unit name
  - use of includes operator
  - variable arguments in meaning
Example lexical rule application

**Initial Semantic Structure:**

\[
((\text{Top-unit} \\
\quad (\text{meaning} ((\text{filter-set-prototype} R S p) \\
\quad \quad \text{(prototype} p \ [\text{ball}])))\)))
\]

**Expanded Syntactic Structure:**

\[
((\text{Top-unit} \ (\text{syn-subunits} ((\text{ball-unit})))) \\
\quad \text{(ball-unit} \ (\text{form} ((\text{string ball-unit "ball"}))))))
\]

Parts that should be added during the rule's application

\[=> \text{Merging of right pole}\]
Example lexical rule application

Right pole of lex-stem-rule:

((?Top-unit (syn-subunits (= ?ball-unit)))
 (?ball-unit (form (= (string ?ball-unit “ball”)))))
Example lexical rule application

=> - Variable unit name
   - (partial) specification of syn-subunits and form features

Right pole of lex-stem-rule:

((?Top-unit (syn-subunits (== ?ball-unit)))
 (?ball-unit (form (== (string ?ball-unit "ball"))))))
Example lexical rule application

Final Semantic Structure:

((Top-unit
  (sem-subunits ((ball-unit)))
  (meaning ((filter-set-prototype R S p))))
  (ball-unit
    (meaning ((prototype p [ball])))
    (context ((link p)))))

Manipulation of Hierarchy (J-operator)
(Also in syntactic structure.)
Complete lex-stem rule

((?Top-unit
  (meaning (== (prototype ?p [ball]))))
((J ?ball-unit ?top-unit)
  (context (== (link ?p))))))

<-->

((?Top-unit
  (syn-subunits (== ?ball-unit)))
(?ball-unit
  (form (== (string ?ball-unit “ball”))))))
• Extension to simple pattern matching

• ELIZA (Weizenbaum, 1960):

  Men are all alike
  IN WHAT WAY?
  They're always bugging us about something or another
  CAN YOU BE MORE SPECIFIC?
  Well, my boyfriend made me come here
  YOUR BOYFRIEND MADE YOU COME HERE?
  He says I'm depressed much of the time
  I AM SORRY TO HEAR YOU ARE DEPRESSED
... 

It's true, I am unhappy

DO YOU THINK COMING HERE WILL HELP YOU NOT BE UNHAPPY?

I need some help, that much seems certain

WHAT WOULD IT MEAN TO you IF YOU GOT SOME HELP?

Perhaps I could learn to get along with my mother

TELL ME MORE ABOUT YOUR FAMILY

...
• Pattern matching:

**Pattern**: (I need a ?x)

**Response**: (What would it mean to you if you got a ?x)

**Input**: (I need a vacation)

=> **Bindings**: ((?x . vacation))
• Unification: Both the pattern and the source (input) may contain variables:

\[
\text{match '}(\text{?x} + \text{?y}) ' (2 + 1)) \Rightarrow ((\text{?y} . 1) (\text{?x} . 2))
\]
\[
\text{unify '}(\text{?x} + 1) ' (2 + \text{?y})) \Rightarrow ((\text{?y} . 1) (\text{?x} . 2))
\]

• Substitution of a set of bindings gives a unifier:

\[
\text{unifier '}(\text{?a} + \text{?a} = 0) ' (\text{?x} + \text{?y} = \text{?y})) \Rightarrow (0 + 0 = 0)
\]
FCG Unification

- Returns multiple hypotheses if there are
- Includes special operators:
  - permutation operator
  - includes operator
  - includes uniquely operator
  - ...

An permutation list '(==_p e1 ... en) unifies with all source lists of which the elements are a permutation of the list (e1 ... en), with elements compared using unification:

(fcg-unify '(==_p ?x ?y) '(a b))

=> (((?x . a) (?y .b)) ((?x . b) (?y . a)))

(fcg-unify '(==_p ?x ?y) '(a b c)) => fail
An permutation list \( '((=_{p} e1 \ldots \ en)\) unifies with all source lists of which the elements are a permutation of the list \( (e1 \ldots \ en)\), with elements compared using unification:

\[
\begin{align*}
(fcg\text{-}unify \ '((=_{p} ?x \ ?y) \ '(a \ b)) & \Rightarrow (((?x . a) \ (?y . b)) \ ((?x . b) \ (?y . a))) \\
(fcg\text{-}unify \ '((=_{p} ?x \ ?y) \ '(a \ b \ c)) & \Rightarrow \text{fail}
\end{align*}
\]
An includes list 

`==(e1 ... en)` unifies with all source lists that at least contains the (different) elements `e1` to `en`, where an element `ei` is contained in a source list if one of its elements unifies with the `ei`:

```
(fcg-unify '==(b ?x) '(a b c))  => (((?x . a)) ((?x . c)))
(fcg-unify '==(b b ?x) '(a b c))  => fail
```
• Typically, in order to determine whether a template (rule) is triggered, one of its poles (a feature structure) is unified with the current semantic or syntactic feature structure.

• To use FCG unification for this, the pole should first be transformed
Unification of Feature structures

((?Top-unit
  (meaning (= (prototype ?p [ball]))))
  (other-feature ...))
  (other-unit ...))

Semantic structure

Pattern
(left pole of lex-stem rule)
Unification of Feature structures

((Top-unit
  (meaning ((filter-set-prototype R S p)
    (prototype p [ball]))))
  (other-feature ...))
(Other units may be present)

==> add includes operator

((J ?ball-unit ?top-unit)
  ((context (== (link ?p))))))
Unification of Feature structures

((Top-unit

(meaning ((filter-set-prototype R S p)

(prototype p [ball]))))

(other-feature ...))

(other-unit ...))

(== (?Top-unit

(meaning (== (prototype ?p [ball])))))

((J ?ball-unit ?top-unit)

(context (== (link ?p))))

J-units are ignored during unification
Unification of Feature structures

((Top-unit
  (meaning ((filter-set-prototype R S p)
    (prototype p [ball]))))
  (other-feature ...))

(prototype p [ball]))

(other-unit ...))

Other features
may be present

(== (?Top-unit
  == (meaning (== (prototype ?p [ball])))))

((J ?ball-unit ?top-unit)
  (context (== (link ?p))))

=> add includes operator
Merging

- Change a source (e.g. by adding missing parts) such that, after the change, it would unify with a given pattern.
- Example:

```lisp
(merge '(syn-cat (== (number singular)))
       '(syn-cat ((lex-cat Noun))))
=> (syn-cat ((lex-cat Noun) (number singular)))```
Typically, in order to apply a triggered template (rule), one of its poles (a feature structure) is merged with the current semantic or syntactic feature structure.

To use FCG merging for this, the pole should first be transformed.

But first, some additional special operators are required.
Merging of Feature structures

(merge '(?unit (sem-cat (== (agent ?e ?a) (human ?a))))
'((unit (sem-cat ((agent e a) (motion-event e))))).
Merging of Feature structures

(merge '((?unit (sem-cat (== (agent ?e ?a) (human ?a)))))
 '((unit (sem-cat ((agent e a) (motion-event e)))).

Result 1 with bindings ((?unit . unit) (?e . e) (?a . a)):

((unit (sem-cat ((agent e a) (motion-event e) (human a))))))
Merging of Feature structures

(merge '(((?unit (sem-cat (== (agent ?e ?a) (human ?a))))))
  '((unit (sem-cat ((agent e a) (motion-event e)))))).

Result 1 with bindings (((?unit . unit) (?e . e) (?a . a)):
  ((unit (sem-cat ((agent e a) (motion-event e) (human a)))))

Result 2 with bindings (((?unit . unit) (?e . motion-event) (?a . e)):
  ((unit (sem-cat ((agent e a) (agent motion-event e)
  (human e))))),
Merging of Feature structures

\[
\text{(merge '(((?unit (sem-cat (= (agent ?e ?a) (human ?a))))))}
\]
\[
'(((unit (sem-cat ((agent e a) (motion-event e)))))].
\]

Result 1 with bindings ((?unit . unit) (?e . e) (?a . a)):
\[
((unit (sem-cat ((agent e a) (motion-event e) (human a))))]
\]

Result 2 with bindings ((?unit . unit) (?e . motion-event) (?a . e)):
\[
((unit (sem-cat ((agent e a) (agent motion-event e) (human e))))].
\]
Includes uniquely operator `==₁`

All elements $e_i$ in `'(==₁ e₁ ... eₙ)` should (after substitution) be unique and, if a list, have a unique first element:

$$(\text{unify} \ '({\text{sem-cat}} \ (==₁ \ (\text{agent} \ e \ a)))$$

'({\text{sem-cat}} \ ((\text{agent} \ e \ a) \ (\text{agent motion-event} \ e)))$$

=> Fail
Merging of Feature structures

\[(\text{merge '}(==_1 \text{ (unit1 }==_1 \text{ (F1 V1))})\]
\[
\quad \text{(unit2 }==_1 \text{ (F2 V2))),}\]
\[
\'((\text{unit1})\]
\[
\quad \text{(unit2)))}\]

result 1: \((\text{unit1 (F1 V1))}\]
\[
\quad \text{(unit2 (F2 V2))).}\]

result 2: \((\text{unit1 unit2 (F1 V1))}\]
\[
\quad \text{(unit2 unit1 (F2 V2))).}\]
All elements $e_i$ in $'(==_{l} e_1 \ldots e_n)$ should (after substitution) be lists:

$$(\text{unify } ' (==_{l} (\text{unit1} ==_{l} (\text{F1 V1}))))$$

$'$((\text{unit1} \text{ unit2} (\text{F1 V1})))$

=> Fail
On the Emergence of Compositionality

21/04/06 | pag. 39

Merging of Feature structures

Syntactic structure

```
((Top-unit))
```

Pattern

```
((?Top-unit
  (syn-subunits (== ?ball-unit)))
(?ball-unit
  (form (== (string ?ball-unit "ball"))))))
```

(right pole of lex-stem rule)
On the Emergence of Compositionality

21/04/06 | pag. 40

Merging of Feature structures

Syntactic structure

Transformed Pattern

((Top-unit))

(== 1l (?Top-unit 1l)
  (syn-subunits (== ?ball-unit)))

(?ball-unit 1l
  (form (== (string ?ball-unit "ball"))))
On the Emergence of Compositionality

21/04/06 | pag. 41

Merging of Feature structures

((Top-unit
   (syn-subunits (ball-unit)))
(ball-unit
   (form ((string ball-unit “ball”))))))

(==_{1} (Top-unit ==_{1}
   (syn-subunits (== ?ball-unit)))
(??ball-unit ==_{1}
   (form (== (string ?ball-unit “ball”))))))

Merged Syntactic structure

given binding (?ball-unit . ball-unit)
J operator in merging

 Semantic structure

 Pattern
(left pole of lex-stem rule)

\[
((\text{Top-unit})
  \begin{align*}
    &\text{(meaning } (\text{filter-set-prototype R S p}) \\
    &\quad (\text{prototype p [ball]}))) \\
    &\text{(other-feature ...))} \\
    &\text{(other-unit ...))}
  \end{align*}
\]

\[
((\text{?Top-unit})
  \begin{align*}
    &\text{(meaning } (== (\text{prototype ?p [ball]}))) \\
    &((\text{J ?ball-unit ?top-unit}) \\
    &\quad (\text{context } (== (\text{link ?p}))))
  \end{align*}
\]
((Top-unit
  (meaning ((filter-set-prototype R S p)
    (prototype p [ball]))))
  (other-feature ...))
(other-unit ...))

create new binding
(?ball-unit . ball-unit)
((Top-unit
  (sem-subunits (ball-unit))
  (meaning ((filter-set-prototype R S p)
    (prototype p [ball])))) ...
  (ball-unit ...) ...)

add as subunit

((?Top-unit
  (meaning (== (prototype ?p [ball]))))
  ((J ?ball-unit ?top-unit)
    (context (== (link ?p))))))
J operator in merging

((Top-unit
  (sem-subunits (ball-unit))
  (meaning ((filter-set-prototype R S p)) ...)
(ball-unit
  (meaning ((prototype p [ball]))) ...)

((?Top-unit
  (meaning (== (prototype ?p [ball])))
  ((J ?ball-unit ?top-unit)
    (context (== (link ?p)))))

Pull in specified feature (values)
((Top-unit
    (sem-subunits (ball-unit))
    (meaning ((filter-set-prototype R S p)) ...) 
(ball-unit (context ((link p))))
    (meaning ((prototype p [ball])))...) ...)

((?Top-unit
    (meaning (== (prototype ?p [ball])))
(('J ?ball-unit ?top-unit)
    (context (== (link ?p))))

Add additional specified feature (values)
=> additional merge
• Rule application can differ in:
  - which pole is unified in production/parsing
  - with which structure (semantic or syntactic)
  - which pole is merged in production/parsing
  - with which structure (semantic or syntactic)

• Rule-set application can differ in:
  - whether competing rules can apply in parallel
  - whether application is recursive
Lex-stem and con-rules in production:

- left pole unified with semantic structure
- right pole merged with syntactic structure
- con-rules applied recursive
Lex-stem and con-rules in interpretation:
- right pole unified with syntactic structure
- left pole merged with semantic structure
- con-rules applied recursive
Sem-rules, both in production and parsing:
- left pole unified with semantic structure
- right pole merged with semantic structure
(re-conceptualization)
Single user interface

- Allows to set and show the current semantic and/or syntactic structure
- Allows to define and apply rules
- Requires ASDF package: :fcg-single-user-interface
- Good starting point:
  
  file “.../experiments/single-user-examples/IK2006-exercise.lisp”
  
  (Note: no IRL but first order meaning)