Letzte Wochen / Diese Woche / Nächste Woche

Parsen Kontextfreier Grammatiken	Letzte Wochen
Top-Down, Bottom-Up	
Chart, Agenda	
Unifikation	Diese Woche
Parser mit Unifikation	Nächste Woche

Unifikation

Wozu Unifikation?

Viele Knoten unterscheiden sich nur durch wenige **Merkmale**

Z.B.:

- *sleeps* V, Präsens, Singular, 3. Person
- *sleep* V, Präsens, Plural, 2. Person (...und andere Bedeutungen)

Es gibt wenigstens zwei Möglichkeiten:

- Komplexe Knotennamen: `V_PRAES_SG_3, V_PRAES_PL_2`
- Merkmalsstrukturen (auch Attribute oder engl. *features*:

```
V [ temp: praes, num: sg, pers: 3]
V [ temp: praes, num: pl, pers: 2]
```

Merkmalsstrukturen

Merkmalsstrukturen sind meist nicht völlständig

- I.A. sind die Informationen in Merkmalsstrukturen nicht vollständig:

  Der Artikel *the* steht für Singular und Plural, also wird in seiner Merkmalsstruktur kein Wert angegeben: entweder fehlt der Eintrag für `num`, oder der Wert ist leer.
- Merkmalsstrukturen werden auch Featurestrukturen oder Attribut-Wert-Matrizen genannt. Ein Merkmal mit seinem Wert heisst auch Feature-Value-Pair

Was ist Unifikation

Unifikation vereinigt die Information zweier Merkmalsstrukturen – falls möglich

- Unifikation scheitert, wenn widersprüchliche Werte zu einem Merkmal vorliegen. Das Ergebnis der Unifikation wird dann als `FAIL` angegeben.
- Wenn ein Merkmal (oder sein Wert) in der Struktur A fehlt, aber in B vorhanden ist, dann wird es in das Unifikationsergebnis aufgenommen. Sonst müssen die Werte des Merkmals *gleich* sein.
- Beispiel:

```
A:   [ temp: praes ]    B:   [ temp:  NIL ]              A U B:  [ pers: 1     ]
     [ num:  pl    ]         [ pers:  pl  ]     --->             [ num:  pl    ]
     [ pers: 1     ]                                             [ temp: praes ]
```

Verschachtelte Merkmalsstrukturen

Der Wert eines Merkmals kann wieder eine Merkmalsstruktur sein.
Rekursion :-)

- So können verschachtelte Merkmalsstrukturen gebildet werden.
- Beispiel:

```
[ tempus:    pras                 ]
[ kongruenz: [ numerus: plural ]  ]
[            [ person:  1      ]  ]
```

Wir brauchen eine erweiterte Definition von Unifikation

Unifikation von Verschachtelten Merkmalsstrukturen

Rekursive Definition: für jedes (Toplevel-) Merkmal

- Wenn ein Merkmal in A nicht vorhanden ist oder keinen Wert hat, nehme den Wert von B
- Wenn beide Werte *Atome* sind, müssen sie *gleich* sein, sonst `FAIL`.
- Wenn beide Werte *Merkmalsstrukturen* sind, müssen sie *unifizierbar sein*.
- Sonst `FAIL`.

```
A: [ temp: praes      ]  B: [ temp:  NIL       ]     A U B: [ temp: prase       ]
   [ agr: [ num:  pl ]]     [ agr: [ pers: pl ]] -->        [ agr:  [ num:  pl ]]
   [      [ pers: 1  ]]                                     [       [ pers: 1  ]]
```

Eine Unifikationsfunktion

```
(defun unify (fs-1 fs-2)
  (cond ((null fs-1) fs-2)
        ((null fs-2) fs-1)
        ((eq fs-1 fs-2) fs-1)
        ((or (atom fs-1) (atom fs-2)) *bottom*)
        (t (loop for (key . value-1) in fs-1
                 for fvp-2 = (assoc key fs-2)
                 for value-2 = (cdr fvp-2)
                 for unify = (if fvp-2
                                 (unify value-1 value-2)
                               value-1)
                 if (bottomp unify)
                 return *bottom*
                 else collect (cons key unify) into result
                 finally (return (append result (set-difference fs-2 result :key #'firs
```


Koreferenzen

Manchmal will man, daß Werte an verschiedenen Stellen innerhalb einer Merkmalsstruktur *gleich* sind.

- *Gleich* bedeutet hier `eq`.
- Dies wird durch *Koreferenzen* ausgedrückt; das sind Zahlen, hier noch in rechteckigen Klammern: `[3]`
- Und (abhängig von der Implementierung) auch *structure sharing* genannt.

Als Beispiel eine Merkmalsstruktur für ein Verb, wie sie *im Prinzip* in einer HPSG Grammatik vorkommt:

```
[ cat: [ typ:  verb                              ] ]
[      [ agr:  [1]                               ] ]
[      [ valence: [ subj: [ cat: [ typ: noun ]]] ] ]
[      [          [       [      [ agr: [1]  ]]] ] ]
[      [          [ compl: ...                   ] ]
```

Koreferenzen

Unsere erste Implementierung trennt die Merkmalsstruktur von den Koreferenzen (Bindungen)

```
(defstruct fs
  structure   ;; Merkmalsstruktur
  bindings)   ;; Bindungen
```

Das Beispiel als LISP-Struktur:

```
#S(FS :STRUCTURE ((CAT . ((TYP . VERB)
                          (AGR . 1)))
                  (VAL . ((SUBJ . ((CAT . ((TYP . NOUN))
                                          (AGR . 1)))))))
      :BINDINGS (1 ((NUM . PL)(PERS . 3))))
```


Unifikation mit Koreferenzen

Welche Probleme können auftreten?

- Die Koreferenzen können gleiche Zahlen sein
  → umbenennen
- Mehrere Stellen (Koreferenzen) können auf die gleiche Struktur zeigen
  → Gleichheiten aufsammeln und hinterher einheitlich benennen

Eine Unifikationsfunktion mit Koreferenzen

```
(defun unify4 (fvps-1 bind-1 fvps-2 bind-2)
  (let* ((offset (if bind-1
                   (apply #'max (remove-if-not #'integerp bind-1))
                   0)))
    (setq *bindings* (append bind-1 (add! offset bind-2))
          *equalities* nil)
    (let ((unification (unify! fvps-1
                               (add! offset fvps-2))))
      (when (bottomp unification)
        (return-from unify4 *bottom*))
      (loop for (variable equality-list) on *equalities* by #'cddr
            do (loop for equality in equality-list
                     do (nsubstitute! variable equality unification :test #'eql)))
      (make-fs :structure unification :bindings *bindings*))))
```


Die eigentliche Unifikationsfunktion

```
(defun unify! (fvps-1 fvps-2)
  (cond ((null fvps-1) fvps-2)
        ((null fvps-2) fvps-1)
        ((or (integerp fvps-1) (integerp fvps-2))
         (let* ((value-1 (if (integerp fvps-1) (getf *bindings* fvps-1) fvps-1))
                (value-2 (if (integerp fvps-2) (getf *bindings* fvps-2) fvps-2))
                (unification (unify! value-1 value-2)))
           (cond ((bottomp unification) *bottom*)
                 ((integerp fvps-1)
                  (setf (getf *bindings* fvps-1) unification)
                  (when (integerp fvps-2)
                    (push fvps-2 (getf *equalities* fvps-1)))
                  fvps-1)
                 (t;; fvps-2 is integerp
                  (setf (getf *bindings* fvps-2) unification)
                  fvps-2))))
   [...]
```

Die eigentliche Unifikationsfunktion (2)

```
((eq fvps-1 fvps-2) fvps-1)
((or (atom fvps-1) (atom fvps-2)) *bottom*)
(t
 (loop for (key . value-1) in fvps-1
       for value-2 = (rest (assoc key fvps-2))
       for unification = (unify! value-1 value-2)
       when (bottomp unification) do
       (return-from unify! *bottom*)
       when (integerp value-2)
       collect (cons key value-2) into result
       else
       collect (cons key unification) into result
       finally (return (append result
                               (set-difference fvps-2 result :key #'first))))))
```