Properties of Derivations SHDG Can Create

- **Semantic monotonicity, a constraint on earlier systems**
  - In every local tree in a derivation, the daughter semantics never contains more information than the mother semantics
  - Formally: the semantics of each daughter subsumes the mother semantics
  - This is a tough constraint since even simple semantic languages may be non-monotonous, as in the example derivation
- **SHDG can generate with semantically non-monotonous grammars**
  - In contrast to [Shieber 1988]
  - See example derivation (p/up)
- **What grammars can SHDG generate with?**
  - Unknown ☹
  - Dependencies between grammars and interpreter exist
  - SHDG cannot generate from ANY semantically monotonous grammar (see case study)

Discussion of the SHDG Algorithm: Failures

- **Left recursion, non-termination**
  - NCR expansion does not terminate [Strzalkowski 1994]
  - Pivot semantics may grow
    - **`/RESTSEM` → `/[DTRSEM | RESTSEM], /DTRSEM`**
  - Remedy for semantically monotonous grammars: test for empty semantics
- **Failure of top-down expansion**
  - Left-to-right NCR expansion does not consider dependencies between daughters [Martinovic/Strzalkowski 1992]
  - E.g. the semantics of the first daughter is underspecified before the second one is expanded [Russell et al. 1990]
  - Remedy: expand left-most daughter with instantiated semantics first
- **Semantic non-determinism (see case study)**

Discussion of the SHDG Algorithm: Inefficiencies

- **Backtracking**
  - In lexicalized frameworks, rules are very general (little information)
  - Failure of a rule may be detected only late
  - Use context in derivation to guess the best candidate rule (a machine learning problem!)
    - **e.g. if current rule is R1, then try R2. Never try R3 (as it will always fail)**
  - It is up to the grammar writer to minimize “late” backtracking
- **Lexicon access**
  - Insertion of thousands of lexical NCR is not feasible
  - Use a stem lexicon (and a morphological inflection system)
  - Index the lexicon according to semantic information
  - Insert only those entries indexed according to pivot semantics

Discussion of the SHDG Algorithm: Interfaces

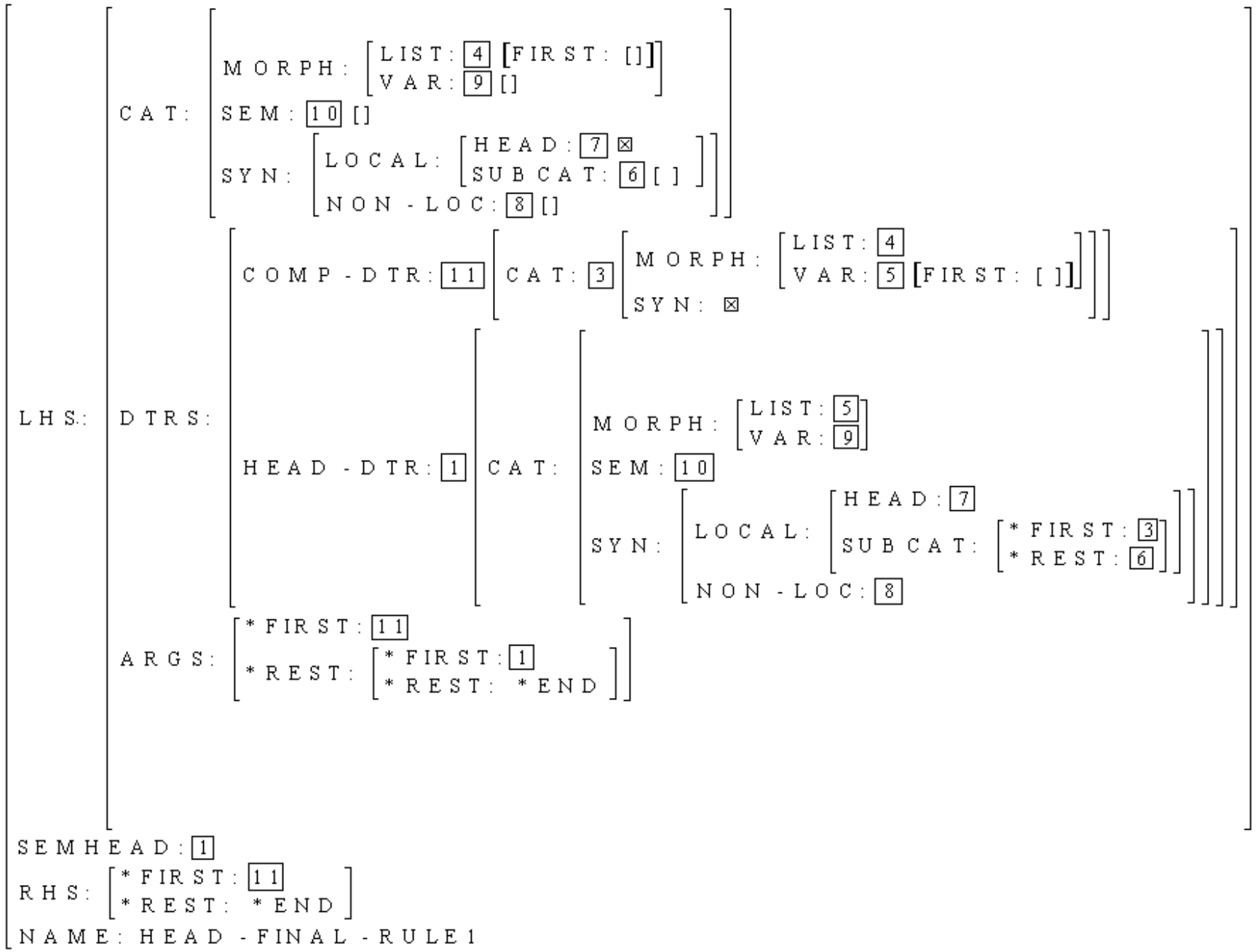
- **Within the Grammar**
  - Strata: morphology, syntax, semantics
  - Identity of semantics in different categories is based on theoretical grounds (e.g. HPSG), not to support SHDG
  - Common misunderstanding: a declarative representation of grammars does NOT guarantee interpretability (for instance: GPSG in its 1985 version)
- **Grammar and Generation**
  - What counts as a CR is determined by the grammar writer quite accidentally
  - Thus the actual SHDG control is implicitly left to the grammar writer (who should not have to take care about detailed processing issues)

The interface problem between declaratively represented knowledge and its interpretation procedures is not peculiar to SHDG - it is general

The control strategy must encode a *concept of derivation*

- **Concept of derivation, e.g.:**
  - Grammar and interpreter "know" left-hand and right-hand sides of rules
  - Definition of corresponding categories or features within the grammar and the interpreter
  - Often a concept of derivation is implicitly modeled within the grammar
- **Control strategies are based on "essential features"**
  - Parsing: Morphology
  - Generation: Semantics
  - The timely specification of essential features during processing usually depends on the way the grammar is modeled

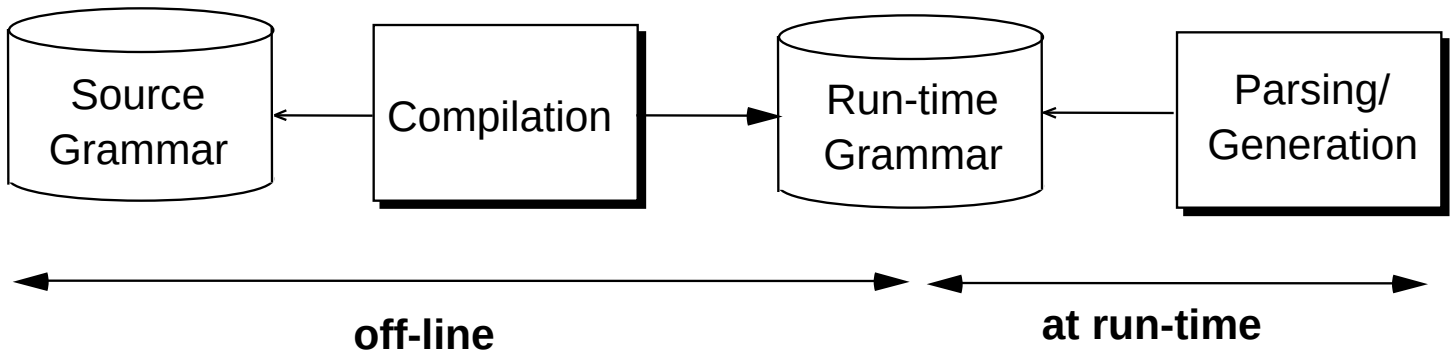
What makes processing of grammars possible after all?


LHS:
CAT:
MORPH:
LIST: 4 [FIRST: []]
VAR: 9 []
SEM: 10 []
SYN:
LOCAL:
HEAD: 7 ⊠
SUBCAT: 6 [ ]
NON-LOC: 8 []
DTRS:
COMP-DTR: 11
CAT: 3
MORPH:
LIST: 4
VAR: 5 [FIRST: [ ]]
SYN: ⊠
HEAD-DTR: 1
CAT:
MORPH:
LIST: 5
VAR: 9
SEM: 10
SYN:
LOCAL:
HEAD: 7
SUBCAT:
*FIRST: 3
*REST: 6
NON-LOC: 8
ARGS:
*FIRST: 11
*REST:
*FIRST: 1
*REST: *END
SEMHEAD: 1
RHS:
*FIRST: 11
*REST: *END
NAME: HEAD-FINAL-RULE1

The control strategy must encode a solution to the following, basic processing problem:

The essential feature must specify exactly the constituent to be generated at the time the generation procedure is executed on it

An implementation of this requirement is not part of the interface between grammar and control strategy



- **Control structure unchanged, grammar is adapted**
- **Source grammar is compiled into run-time grammar with the desired properties**
- **Disadvantage: the processing problem is shifted into the compiler**

Solution 1: Compilation

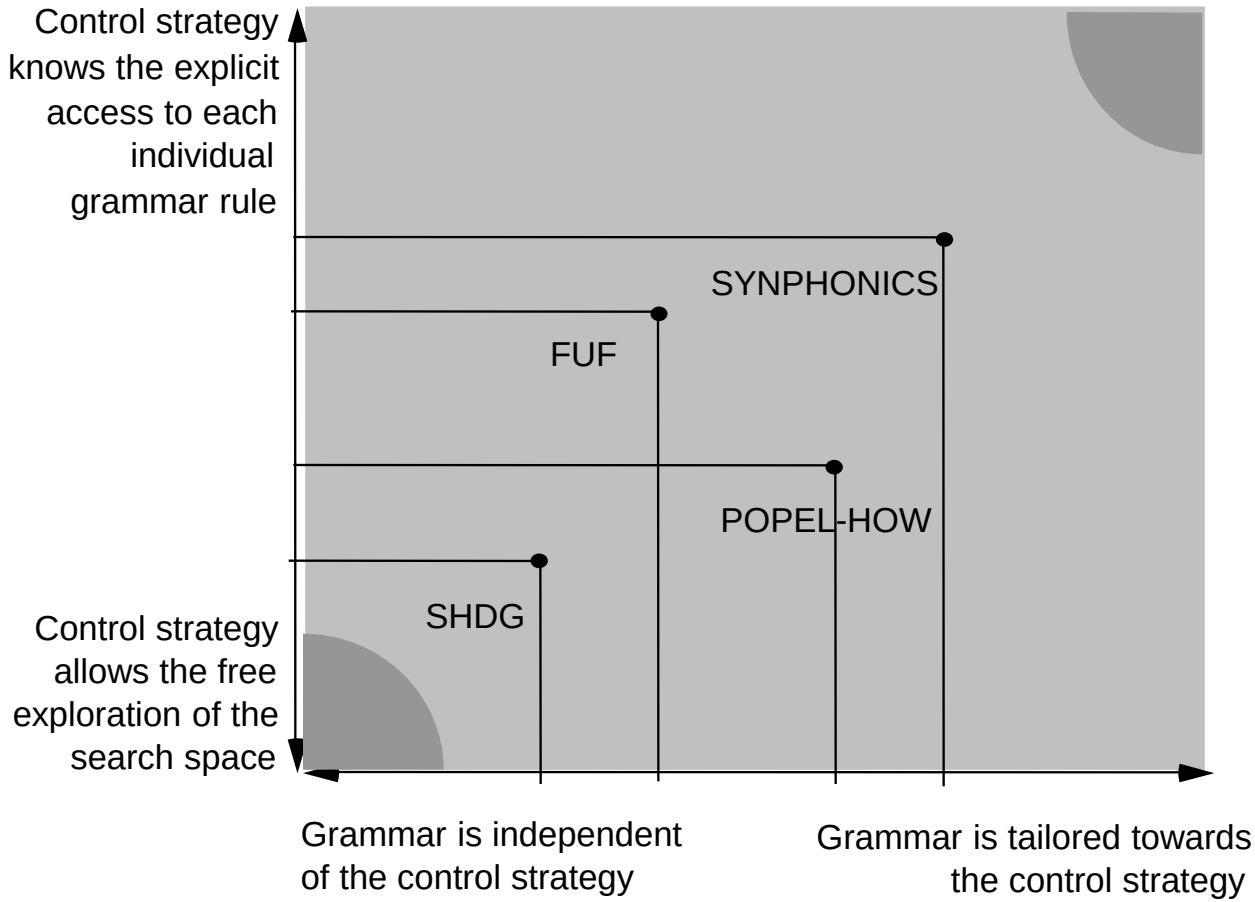
- **Linguistic knowledge is integrated into the control structure**
  - parameterizable (and thus sufficiently general), e.g.
    - If you have feature structure F, carry out a breadth-first step
  - grammar-specific (and thus problematic), e.g.
    - Apply rule R only if feature f has value v
- **Disadvantage: often theoretically obscure, hardly transferable**

Solution 2: augment the control strategy

- **Control knowledge is added to the linguistic knowledge**
  - Representation within the same formalism as the grammar
  - Separation of linguistic knowledge is a modeling problem; solvable
  - "Control" features can
    - influence the applicability of rules
    - signal the fulfillment of conditions
    - influence the control structure
- **Disadvantage: obscure and not transferable if intermingled with linguistic knowledge**

Solution 3: augment the grammar

Relative position of some systems



The Problem Space as a Continuum

- Implicit relationships between grammar and control
  - Concept of derivation
  - Sufficient specification of essential features
- Solutions to the basic processing problem are different
  - Control-oriented approaches base the control strategy on integrated linguistic knowledge
  - Grammar-oriented approaches use compilation or use declaratively defined control knowledge
- Missing theoretical foundation of the relations within the system

Summary

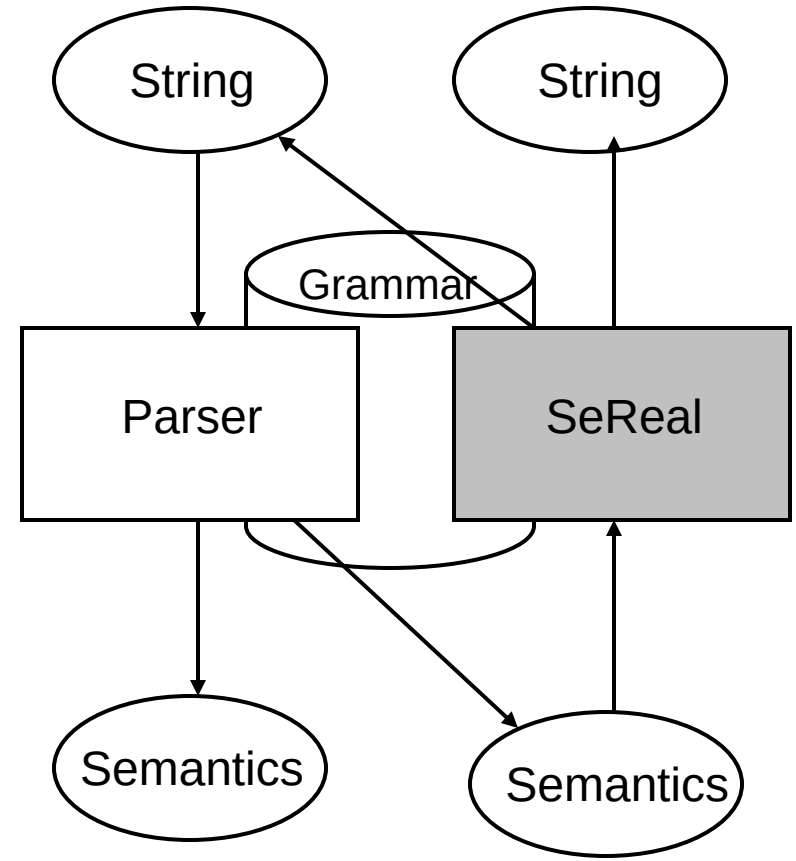
The DISCO Natural Language System

OVERVIEW

- **Linguistic core engine for parsing sentences into quasi-logical formulae** [Uszkoreit et al. 1994]
  - Large constraint-based grammar of German (several 100 types, 20 rules) [Netter 1994, Kasper 1994]
  - TDL: type definition and inheritance mechanism [Krieger/Schäfer 1994]
  - UDiNe: constraint solver for (destructive) unification of feature structures
  - Morphix: morphological analysis, “stemming” [Finkler/Neumann 1988]
- **DISCO has grown into PAGE, a Platform for Advanced Grammar Engineering**
  - Used by several research institutes (DFKI, SRI, ...)
  - **`http://www.dfki.de/pas/f2w.cgi?lts/page-e`**

The SeReal Sentence Realizer

- **Common Lisp Implementation of SHDG** [Busemann 1997]
- **Extensions and Improvements**
  - Efficient lexicon access and lexicon organization
  - Tuning of rule applications
  - Recycling of used structures
  - Avoiding left recursion with NCRs (termination)
  - Intelligent determination of the order in which non-head daughters are expanded
- **Integration with the DISCO system**



Semantics in the DISCO Grammar: QUANT

- **The semantics of the sentence constituents is represented as a list**
- **List reflects the surface order of the constituents**

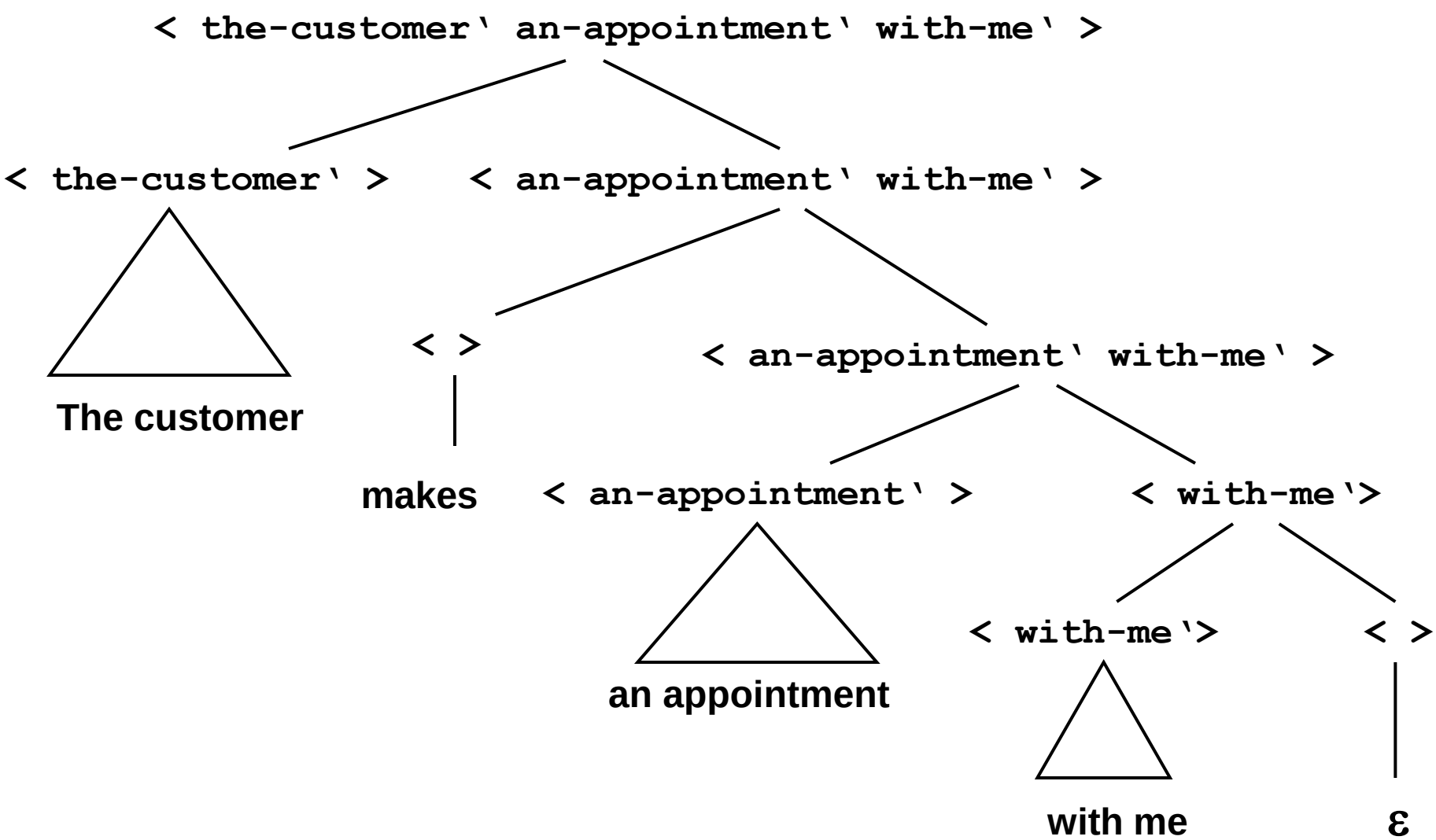
Example

Constituents: (The customer) makes (an appointment) (with me)

```
[ QUANT < the-customer‘ an-appointment‘ with-me‘> ]
```

- **Linguistic motivation**
  - semantics interfaces to subsequent non-compositional, extra-grammatical semantic interpretation (anaphora resolution, scope disambiguation)
- **Use of QUANT**
  - QUANT values are “collected” during bottom-up parsing in a list
  - The root node of the parse tree shows the complete list
  - List concatenation is achieved using difference lists

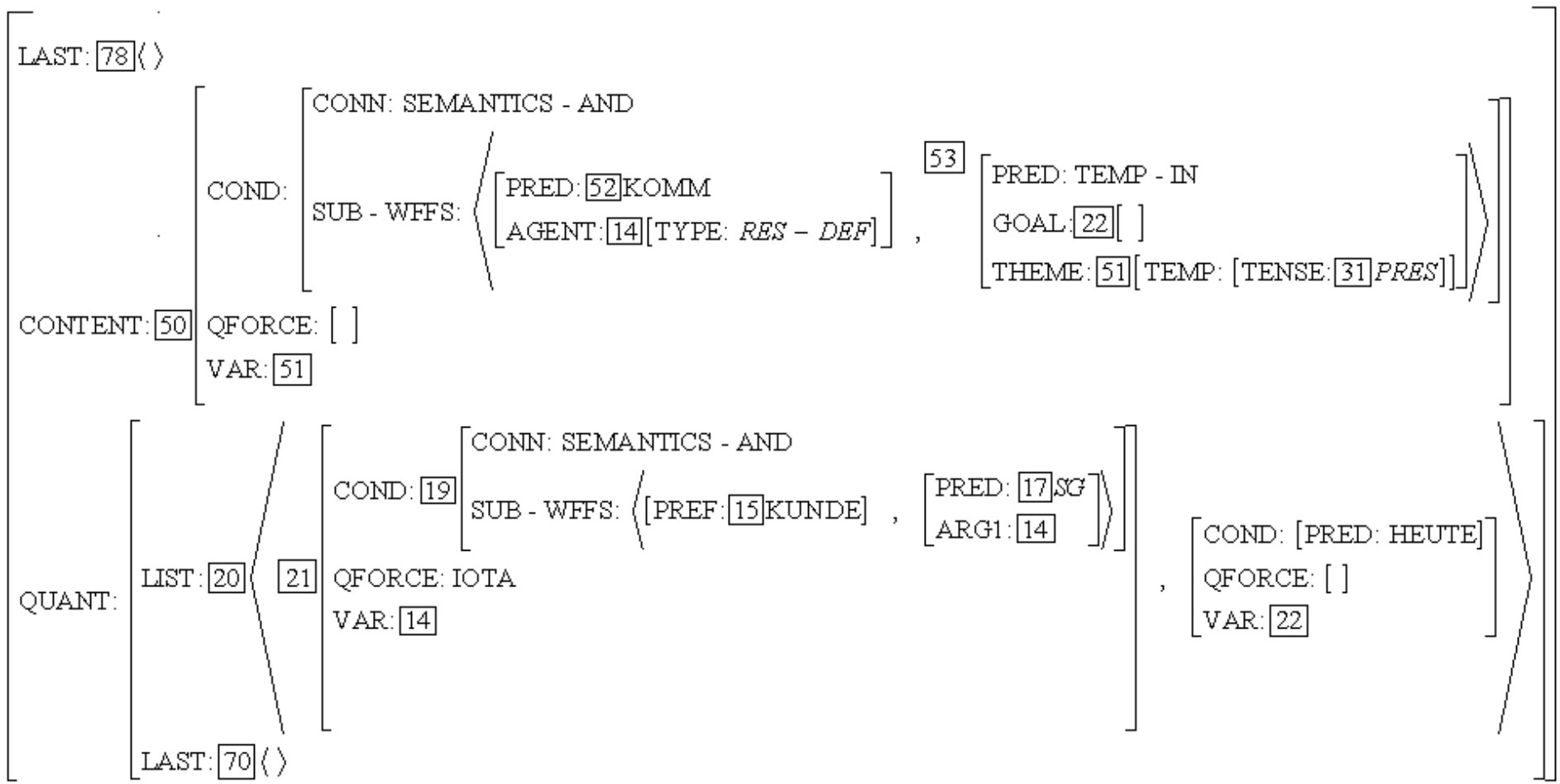
A Derivation Tree with QUANT



Semantics in the DISCO Grammar: CONTENT

- **Predicate-Argument structure**
- **Relations between sentence constituents in QUANT (cf. example)**
  - Conjunction of sub-formulae
  - The agent role of “come” is the first element in QUANT
  - The goal of the temporal relation is the second element in QUANT
  - The theme of the temporal relation relates to CONTENT
- **QFORCE**
  - Quantification, values include *iota* (definiteness), *exists* (indefiniteness), *forall, some, no* etc.

Semantic Representation for ‘Der Kunde kommt heute’ (The customer arrives today)



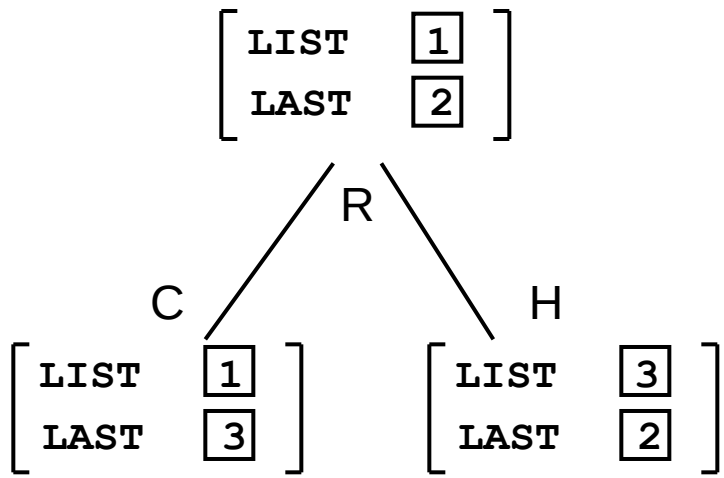
List Operations Can Be Dangerous

DIFFERENCE LISTS

- **Composition and decomposition of lists (cf. `append` in Prolog)**
  - **`append([a, b], [c, d], _)`** deterministically yields **`[a, b, c, d]`**
  - **`append(_, _, [a, b, c, d])`** has no unique solution, but rather several (in Prolog offered via backtracking)
- **List decomposition is not a function (unless the value is a disjunction of all solutions)**
- **The functionality of `append` can be described by *difference lists***
- **Feature unification is strictly functional**
- **Using difference lists in a constraint-based grammar can impose restrictions on processing**

Difference Lists in Constraint-Based Grammars

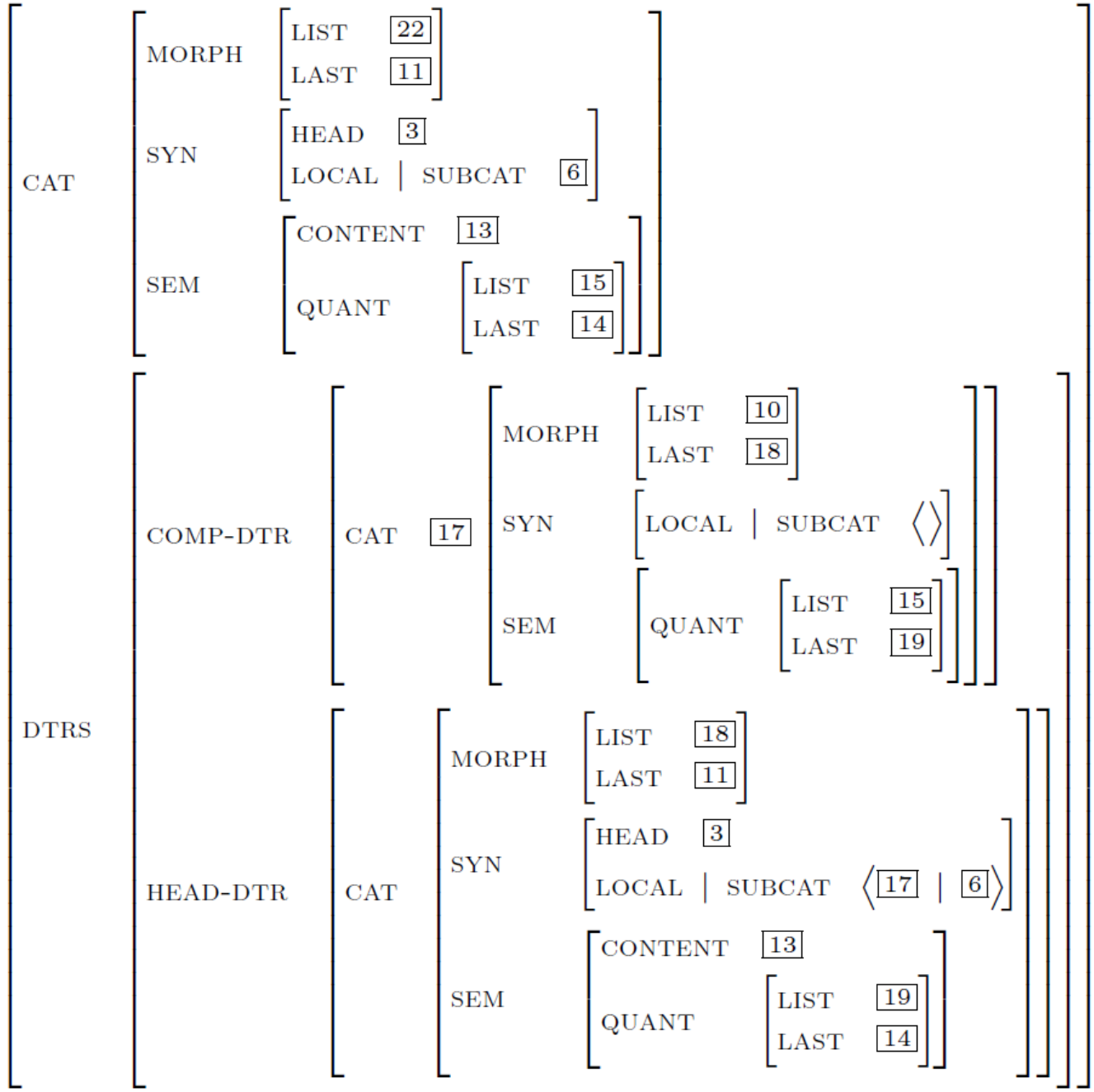
Example: $\begin{bmatrix} \texttt{LIST} & \texttt{<a b c d>} \\ \texttt{LAST} & \texttt{<c d>} \end{bmatrix}$ denotes <a b>



append (C, H, R)

Sufficient condition for deterministic behavior, discarding directionality

- All three variables are known
- Any two of the three feature structures are fully specified



SHDG is difficult with the DISCO Grammar

- **Pivot semantics: SEM**
  - There are hardly any CR because QUANT differs at all categories
  - SHDG thus instantiates a mere top-down generation strategy
  - The daughter QUANT values cannot be determined deterministically
  - Derivations are not found

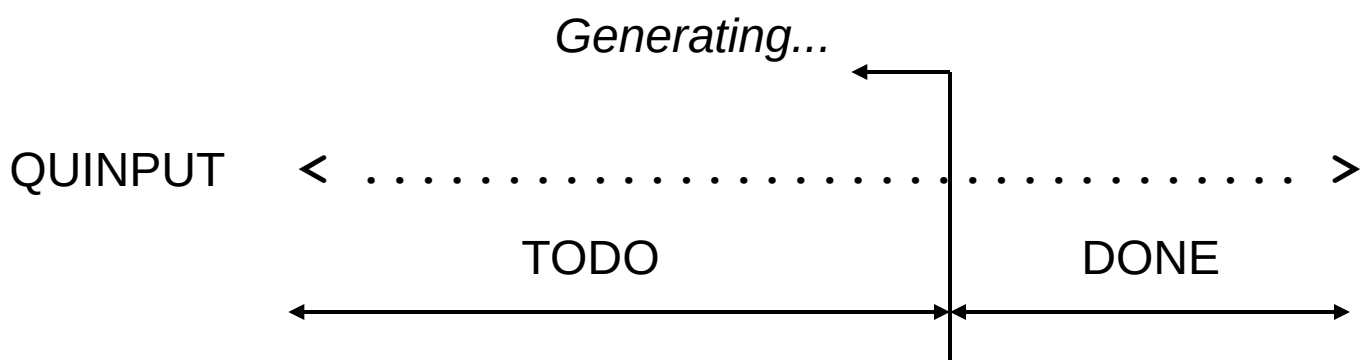
The essential feature is not specified when SHDG needs it

- **Notes**
  - Not a problem of SHDG/DISCO alone, also SHDG/MRS [Copestake et al. 96] and other Generator/Grammar pairs
  - However, SHDG ignores information that is available (in the input)
  - To solve it, neither change semantics nor adopt a different generator...

Defining a New Pivot Semantics: PIVOT

[Busemann 1997]

- **Idea: make input QUANT information available to the pivot semantics (QUINPUT) by defining an additional layer to the grammar, GEN**
- **Receive chain rules and implement a mere bottom-up strategy**
- **Book keeping**
  - DONE: what parts of QUINPUT have been generated
  - TODO: what part of QUINPUT remains to be generated
- **Using difference lists, we implement `append(TODO, DONE, QUINPUT)`**



The DISCO Grammar is Encoded in TDL

- **Type Description Language implemented in Common Lisp** [Krieger/Schäfer 1994]
- **Type definition (off-line) using multiple inheritance**
- **Type expansion within the Lisp system**
- **Inheritance is achieved through unification (TDL - UDiNe)**

Some TDL constructs

```
S1 :< S2.                    ;;S1 is a subsort of S2
T2 := T3 & T4.               ;;T2 inherits from both T3 and T4
T5 := 'a | 'b.               ;;T5 is a disjunction (of constants)
T7 := <...> : T5.            ;;T7 is a list with elements of type T5
T3 := T6 & [ f : T5 ].       ;;the value of f is restricted to type T5
T6 := [ f  <! #1 !>, g #1 ]. ;;coreferences; restricting f to a difflist of length 1.
```

```
sem-expr-type       :< *avm*.

semantics-type      := *avm* &
                       [ CONTENT : sem-expr-type,
                         QUANT : *diff-list* ].

semantics           := *avm* &
                       [ SEM : semantics-type ].

term-type           :< sem-expr-type.
wff-type            :< sem-expr-type.
pred-type           :< sem-expr-type.
var-type            :< term-type.

atomic-wff-type     := wff-type &
                       [ PRED : pred-type ].

identity-wff-type   := atomic-wff-type &
                       [ PRED 'IDENTITY ].

wff-list-type       := <...> : wff-type.

connective-type     := 'SEMANTICS-AND \|
                       'SEMANTICS-OR \|
                       'SEMANTICS-NOT .

conn-wff-type       := wff-type &
                       [ CONN : connective-type,
                       SUB-WFFS : wff-list-type ].
```

```
conj-wff-type      := conn-wff-type &
                         [ CONN 'SEMANTICS-AND ].

var-bndr-type       := 'FOR-ALL\|
                       'EXISTS \|
                       'IOTA\|
                       'NO .

indexed-wff         := term-type &
                       [ VAR var-type,
                         COND wff-type ].

rp-type             := indexed-wff &
                       [ QFORCE : var-bndr-type ].

quant-1-ele-type    := semantics &
                       [ SEM semantics-type &
                           [ QUANT <! *top* !> ] ].

cont2quant          := semantics-type &
                       [ QUANT <! #sem !>,
                         CONTENT #sem,
                         LAST < > ].

cont2quant-type     := quant-1-ele-type &
                       [ SEM cont2quant ].
```

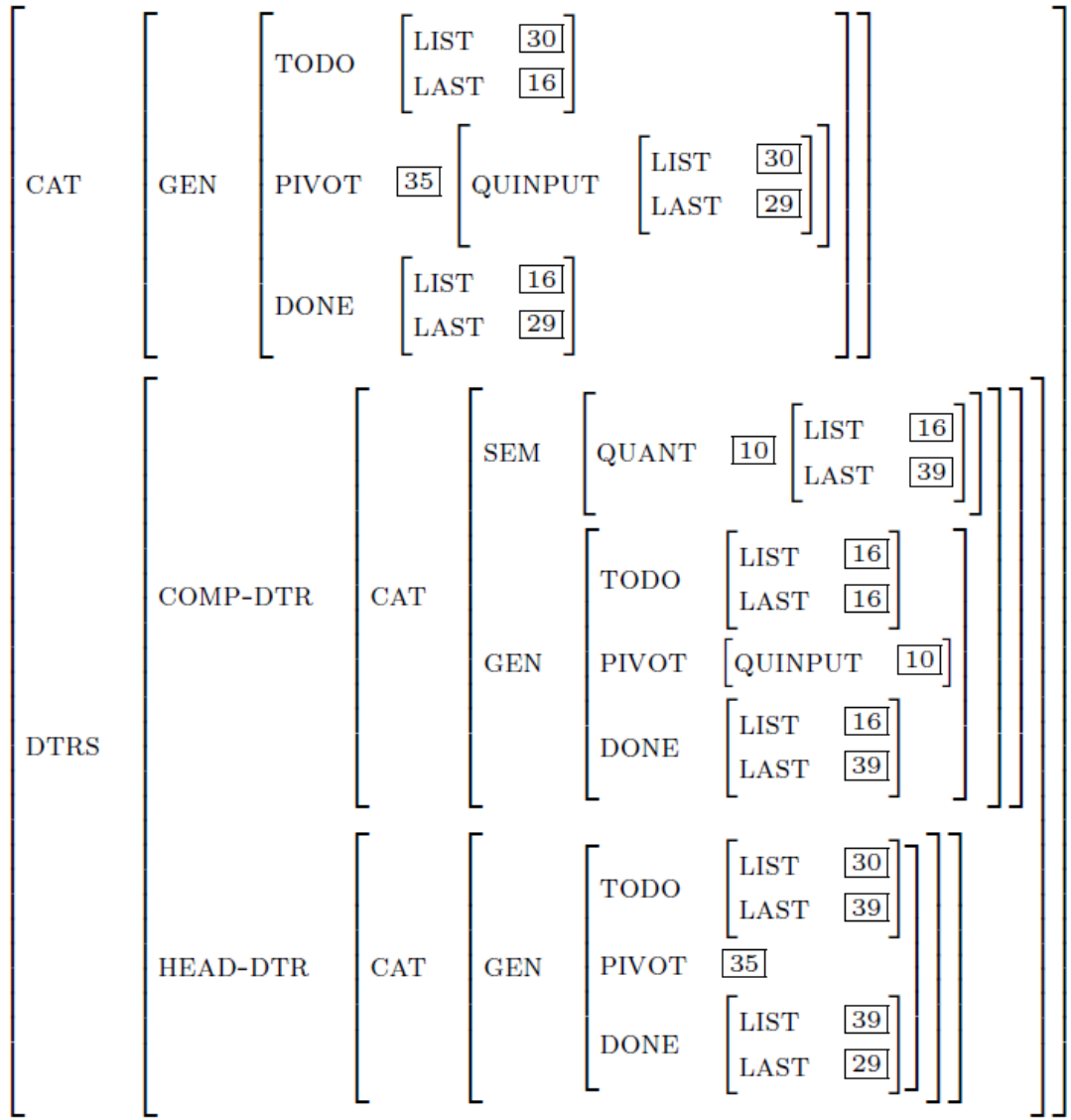
```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;   GEN Features                                 ;;
;;   pivot, todo, done, quinput                   ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

todo-type    :<  *diff-list*.

done-type    :<  *diff-list*.

quinput-type :<  *diff-list*.


pivot-type   := [ QUINPUT quinput-type,
                  CONTENT-GEN *avm* ].

gen-type     := [ PIVOT pivot-type &
                  [ QUINPUT quinput-type &
                          [ LIST #list1,
                            LAST #list2 ] ],
                    TODO  todo-type &
                          [ LIST #list1,
                            LAST #list3 ],
                    DONE  done-type &
                          [ LIST #list3,
                            LAST #list2 ] ].
```

```
gen-cat-type := [ GEN  gen-type ].

cat-gen-type := cat-type & gen-cat-type &
                [ GEN.PIVOT.CONTENT-GEN #con,
                  SEM.CONTENT #con ].


gen-sign-type := sign-type &
                   [ CAT cat-gen-type ].
```

The GEN Feature Structure

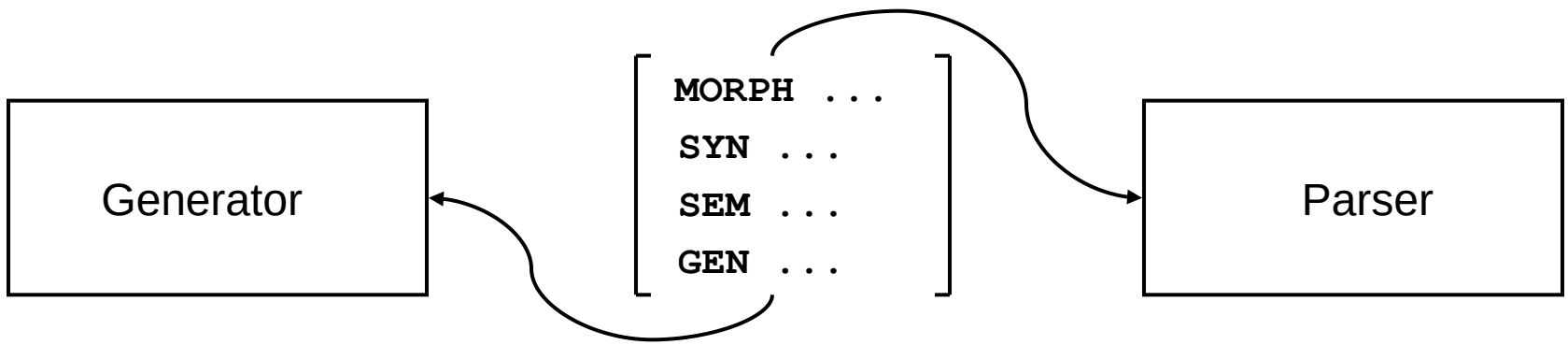
$$\begin{bmatrix} \text{SEM} & \begin{bmatrix} \text{CONTENT} & \boxed{1} \end{bmatrix} \\ \text{GEN} & \begin{bmatrix} \text{TODO} & \begin{bmatrix} \text{LIST} & \boxed{2} \\ \text{LAST} & \boxed{3} \end{bmatrix} \\ \text{PIVOT} & \begin{bmatrix} \text{CONTENT-GEN} & \boxed{1} \\ \text{QUINPUT} & \begin{bmatrix} \text{LIST} & \boxed{2} \\ \text{LAST} & \boxed{4} \end{bmatrix} \end{bmatrix} \\ \text{DONE} & \begin{bmatrix} \text{LIST} & \boxed{3} \\ \text{LAST} & \boxed{4} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Head-Complement Rule with GEN



Consequences

- **Reorganizing SEM within GEN, without replacing it**
- **Adding SHDG-specific control knowledge to the grammar**
- **Preserving reversability properties of the grammar**
- **Preserving modularity within the grammar and between the grammar and their interpreters (parser, generator)**



The essential feature for generation is GEN rather than SEM

Using GEN as the Essential Feature

- **Consequences for SHDG/DISCO**
  - Nonterminal rules are chain rules
  - Only lexicon rules are NCRs
  - Induces a bottom-up control strategy

- **General insights**
  - General: For modular constraint-based grammars, any interpreter/grammar pair may be „made to work“ by adding a control layer to the grammar
  - Skill: Adding a control layer requires deep understanding of both grammar architecture and interpreters
  - Effort: Choice of solution (compilation, (ex)changing grammar, (ex)changing interpreter, adding control layer) is also a question of effort to be invested