ConTroll in context

- Linguistic data structures in computational linguistics
  - Development
  - Typed feature logic
- Classical typed feature logic systems
  - Definite clause based: CUF
  - Phrase structure based: ALE
- Using the data structure for program organization: Universal Principles
- A combined architecture: ConTroll
  - Modes of combination: relations + principles
  - Example grammar
- Implementation
  - Compilation
  - Interpretation
  - Debugging
- Comparison to other systems
- Summary

Data structures of grammar formalisms

One focus of research in the area of grammar formalisms has been the development of better data structures for modeling linguistic objects.

- (first order) terms in DCGs
- feature structures in PATR
- typed feature structures in ALE, CUF, . . .

Typed feature logic

King (1989, 94), Carpenter (1992b)

The linguistic ontology is defined in the *signature*.

- Type hierarchy: which type of objects exist.
- Appropriateness Conditions: which objects have which properties.

Using a formal description language, one can make statements about

- objects and
- the value of attributes of objects

with respect to the

- type of an object, and the
- token identity of two objects (path equality)

These atomic formulae are combined to more complex ones using

- conjunction,
- disjunction, and
- negation.

Examples for the description language

- `phrase`

  objects of type *phrase*

- `bar_level:two`

  objects, which have an appropriate attribute BAR_LEVEL and its value is *two*

- `~ bar_level: (zero;one)`

  objects for which `bar_level` is not appropriate or whose BAR_LEVEL is *zero* or *one*. $\neq$ previous expression

- `head:X, head_dtr:head:X`

  objects whose HEAD value is token-identical to the HEAD value of their HEAD_DTR.

- `~ word, head:_`

  objects of a type different from *word* which have an appropriate attribute HEAD.

Classical TFL systems
1. definite clause based

CUF (Dörre and Eisele, 1991; Dörre and Dorna, 1993)

- A program is expressed by a set of definite clauses, i.e., a relational extension of the description language.
  - The recursive constraints are encoded on a different level than the descriptions of the linguistic objects (relations vs. arguments of relations), e.g.,

    ```
    relate_to_its_head(head:X, X).
    ```

  - CUF is like PROLOG, except that it uses feature terms as arguments instead of first order terms.
  - A grammar is implemented by encoding it as a logic program.

- A query to a system is a call to a relation. The system returns the possible argument instantiations.

Classical TFL systems
2. phrase structure based

ALE (Carpenter, 1992a; Carpenter and Penn, 1994), TROLL (Gerdemann & Götz)

- A program is organized around a special kind of relation, phrase structure.
  - ALE grammars (in this respect) are like DCGs, except that they use typed feature terms as arguments instead of first order terms.
  - The most important recursive structure is treated separately in the phrase structure.

- Phrase structure is a relation
  - demanding a fixed number of daughters (at runtime)
  - having a designated phonology argument which satisfies the following conditions: $\text{phon}(\text{Mother}) = \text{phon}(\text{Dtr}_1) \oplus \ldots \oplus \text{phon}(\text{Dtr}_n)$

  $\Rightarrow$ very efficient algorithms for parsing due to possible indexing on the input string

Data structure and program structure

A disadvantages of a relationally encoded grammar is that it does not use the elaborate structure imposed on the domain by typed feature structures to structure the program.

Alternative: Universal principles

- A grammar is a set of (usually implicational) descriptions, all of which must be satisfied by every grammatical object.

- Example: Head-Feature Principle

  ```
  phrase, dtrs:headed_struc →
        synsem:loc:cat:head:X,
        dtrs:head_dtr:synsem:loc:cat:head:X
  ```

- Grammar constraints and description of linguistic objects are at the same level.

Advantages of universal principles

The use of universal principles allows for a more modular, underspecified encoding of grammar constraints.

HPSG example. A CUF encoding of the HFP as relation:

```
hfp  :=  synsem:loc:cat:head:X ,
         dtrs:head_dtr:synsem:loc:cat:head:X
```

- has to be called in the relevant places
- requires the encoding of parts of the ontology in the program structure

Assume the following relation `wf_phrase` to constraint grammatical phrases

`wf_phrase := phrase,` ϕ

- where ϕ is some feature term constraining all grammatical phrases.

Advantages of universal principles
(cont.)

To call the relation `hfp` at the appropriate places, the relational encoding has to account for the ontological difference between phrases (those with DTRS of type *headed_phrase* vs. others:

```
wf_phrase  :=  dtrs:headed_struc , hfp  , φ
wf_phrase  :=  dtrs:~headed_struc       , φ
```

Such an encoding is neither compact nor efficient, since (without special indexing) its execution always leaves a choice-point behind.

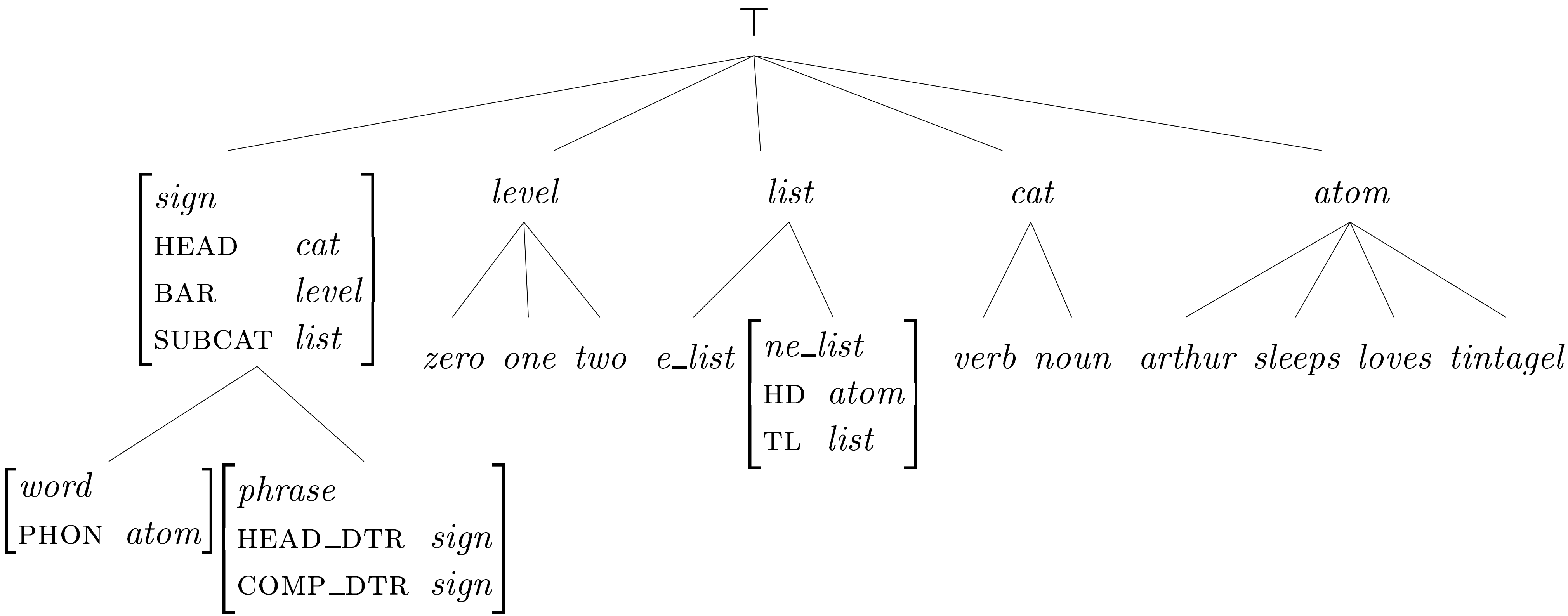
In the relational encoding it is not possible to only refer to those phrases having DTRS of type *headed_phrase*, since very subcase has to be licensed.

Universal principles on the other hand are constraint based in the intuitive sense: Every structure that is not explicitly excludes is well-formed.

A combined architecture: ConTroll

- Relations are a traditional encoding which can be processed efficiently.
- Universal principles use the structure of the domain for a more modular, underspecified grammar encoding.
- Both constraints can be combined in one architecture
  - principles with relational calls (HPSG)
  - relational encoding plus some universal principle

An example grammar: signature



An example grammar: relations

The relation `wfs` encodes the lexicon and the constituent structure:

```
wfs(P0,P)    :=  phrase, head:verb, bar:two,
                 subcat:[],
                 comp_dtr:wfs(P0,P1),
                 head_dtr:wfs(P1,P)
wfs(P0,P)    :=  phrase, head:verb,
                 subcat:ne_list,
                 head_dtr:wfs(P0,P1),
                 comp_dtr:wfs(P1,P)
wfs([X|Y],Y) :=  word, head:verb, bar:zero,
                 subcat:[head:noun, head:noun],
                 phon:(loves,X)
wfs([X|Y],Y) :=  word, head:verb, bar:zero,
                 subcat:[head:noun],
                 phon:(sleeps, X)
wfs([X|Y],Y) :=  word, head:noun, bar:two,
                 subcat:[],
                 phon:((arthur;tintagel), X)
```

An example grammar: principles

- A simple X-bar theory

```
1. bar:zero   →   word
2. bar:one    →   head_dtr:bar:~two
3. bar:two    →   subcat:[]
4. phrase     →   comp_dtr:bar:two
```

- Valence percolation

```
5. phrase   →   comp_dtr:X, subcat:Y,
                head_dtr:subcat:[X|Y]
```

- Head feature percolation

```
6. phrase   →   head:X, head_dtr:head:X
```

Querying a relation

```
?- query(wfs([arthur,sleeps],[])).
```

$$
\Rightarrow \left[\begin{array}{ll}
\textit{phrase} & \\
\text{BAR} & \textit{two} \\
\text{HEAD} & \boxed{2}\ \textit{verb} \\
\text{SUBCAT} & \boxed{3} \\
\text{COMP\_DTR} & \boxed{1} \left[\begin{array}{ll}
\textit{word} & \\
\text{PHON} & \textit{arthur} \\
\text{BAR} & \textit{two} \\
\text{HEAD} & \textit{noun} \\
\text{SUBCAT} & \textit{e\_list}
\end{array}\right] \\
\text{HEAD\_DTR} & \left[\begin{array}{ll}
\textit{word} & \\
\text{PHON} & \textit{sleeps} \\
\text{BAR} & \textit{zero} \\
\text{HEAD} & \boxed{2} \\
\text{SUBCAT} & \left[\begin{array}{ll}
\textit{ne\_list} & \\
\text{HD} & \boxed{1} \\
\text{TL} & \boxed{3}\ \textit{e\_list}
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Querying for objects

```
?- query(word , subcat:ne_list).
```

$$\Rightarrow \begin{bmatrix} \textit{word} & \\ \textsc{bar} & \textit{zero} \\ \textsc{subcat} & \textit{ne\_list} \end{bmatrix}$$

ConTroll: compilation

Compilation is based on Götz and Meurers (1995)

- eliminates complex antecedents by equivalence transformation:

$$
\begin{array}{lll}
a \wedge b \wedge c \;\rightarrow\; d & \Rightarrow & t \;\rightarrow\; d \wedge \neg(a \wedge b \wedge c) \\
 & \Rightarrow & \text{standard} \\
 & & t \;\rightarrow\; d \wedge \neg a \vee \neg b \vee \neg c \\
 & \Rightarrow & \text{disjunkte Disjunktionen} \\
 & & t \;\rightarrow d \wedge \neg(a \wedge b \wedge c) \vee \\
 & & \quad a \wedge \neg b \wedge \neg c \vee \\
 & & \quad b \wedge \neg c \vee \\
 & & \quad \neg c
\end{array}
$$

  where t is the type inferred for $(a \wedge b \wedge c)$.

- collects all principles defined for a type.

- defines a relational encoding for each type (incl. inheritance of constraints)

- specifies calls to the relational encoding at all nodes in the data structures where a restricted type can occur.

ConTroll: more on compilation

Laziness by compilation (Götz and Meurers, 1996)

- Three possible modes of answering a query:
  - enumeration of complete models
  - resolution of all constraints
  - resolution of those constraints which are defined on nodes that are specific enough to cause a failure.
    ⇒ lazy interpretation (Aït-Kaci et. al., 1993)

- Laziness is not sound for all grammars.
  Each type in the grammar must have a nonempty denotation. (type consistency)

- Not every object described by a solution is grammatical, but there is at least one.

Inheritance of constraints at compile time

- can significantly slow down compilation

- user can name principles for which the system then defines auxiliary relations encapsulating the disjuncts

ConTroll: interpretation

Following the Andorra model (Haridi and Janson, 1990)

- execute deterministic goals first
- deterministic: only one head unifies with the call
- Advantage:
  - minimizes number of choice points
  - early recognition of failure branches in search tree
- Disadvantage:
  - Computation required to detect determinacy

⇒ In practice the advantages by far outweigh the disadvantage.

- Further improvements possible by replacing test unifications with deep indexing through automatically generated *determinacy code* (Santos Costa, Warren, and Yang, 1991).

ConTroll: more on interpretation

On-line goal reordering through delays: Only execute a goal if the call is more specific than a user specified statement (wait declarations, Naish (1985))

- Advantage: reduces speculative processing
- Two kinds of delay statements in ConTroll:
  - `delay(Goal-or-Type, Desc)`
    e.g. `delay(wfs,arg1:list)`,
    `delay(phrase,subcat:list)`
  - `delay_deterministic(Goal-or-Type)`
    e.g. `delay_deterministic(sign)`
- Based on CUF's mechanism for definite clauses, but extended
  - to types (in general), and
  - to principles (a single constraint on a type) via auxiliary relations resulting from naming of principles.

ConTroll: visualization

The complex data structures contain a large number of specifications which are difficult to visualize.

$\Rightarrow$ graphical user interface Xtroll, which offers

- (interactive) viewing of AVMs and program states
- comparison of AVMs, search in AVMs, . . .
- user definable visualization possibilities for trees

ConTroll: debugging

The interaction of principles and relations in a complex theory is highly complicated.

$\Rightarrow$ ConTroll has a graphical debugger to trace queries

- displays the structure to be tested for grammaticality and the nodes to be checked (marked as ordinary goal failed, deterministic, delayed)
- Basic functionality: unfold, fail
- Extensions:
  - choosing all deterministic processing
  - viewing and choosing any matching clause (instead of the first)

Comparison with other systems

- TFS (Emele and Zajac, 1990; Emele, 1994)
  - first working TFL system
  - offers relations and type constraints
  - no principles with complex antecedents
  - no delay mechanism: often impossible to ensure termination or efficient processing
- ALE (Carpenter, 1992a; Carpenter and Penn, 1994)
  - offers phrase structure, relations, type constraints
  - no principles with complex antecedents
  - use of type constraints very restricted
    * no lazy processing, instead type constraints eagerly applied during unification
    * no delay mechanism

Comparison with other systems (cont.)

- CUF (Dörre and Eisele, 1991; Dörre and Dorna, 1993)
  - offers relations
  - good delay mechanism
  - preferred execution of deterministic goals
  - no principles
- LIFE (Aït-Kaci et al., 1994)
  - “real” programming language (foreign language interface, . . .)
  - combines functional, relational and object oriented imperative programming
  - feature structures not strongly typed (no appropriateness conditions)
  - no preferred execution of deterministic goals
  - no user definable delay statements

Summary

- Type feature structures as data structure offer new possibilities for the organization of grammatical constraints.

- An architecture combining universal principles with relational constraints supports
  - a more transparent encoding grammars with a relational backbone
  - the implementation of HPSG theories without requiring their reimplementation as a logic program

- In ConTroll
  - universal principles and relational constraints may be interleaved without considering one or the other as primary.
  - lazy interpretation is used
  - deterministic computation is preferred
  - delays can be used to influence execution order of goals

References

Aït-Kaci, Hassan, Bruno Dumant, Richard Meyer, Andreas Podelski, and Peter van Roy. 1994. The Wild LIFE handbook (prepublication edition). Technical report, Digital Equipment Corporation.

Aït-Kaci, Hassan, Andreas Podelski, and Seth Copen Goldstein. 1993. Order-sorted theory unification. Technical Report 32, Digital Equipment Corporation.

Bolc, Leonard, Krzysztof Czuba, Anna Kupść, Małgorzata Marciniak, Agnieszka Mykowiecka, and Adam Przepiórkowski. 1996. A survey of systems for implementing HPSG grammars. Technical Report 814, Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland, October.

Carpenter, Bob. 1992a. ALE – the attribute logic engine, user's guide. Laboratory for Computational Linguistics Report CMU-LCL-92-1, Laboratory for Computational Linguistics, Carnegie Mellon University.

Carpenter, Bob. 1992b. *The Logic of Typed Feature Structures - With Applications to Unification Grammars, Logic Programs and Constraint Resolution*, volume 32 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, New York.

Carpenter, Bob and Gerald Penn. 1994. ALE – the attribute logic engine, user's guide, version 2.0.1, December 1994. Technical report, Computational Linguistics Program, Philosopy Department, Carnegie Mellon University.

Dörre, Jochen and Michael Dorna. 1993. CUF - a formalism for linguistic knowledge representation. In Jochen Dörre, editor, *Computational aspects of constraint based linguistic descriptions I*. DYANA-2 Deliverable R1.2.A, Universität Stuttgart, August, pages 1–22.

Dörre, Jochen and Andreas Eisele. 1991. A comprehensive unification based formalism. DYANA Deliverable R3.1.B, Universität Stuttgart, January.

Emele, Martin. 1994. The typed feature structure representation formalism. In *Proceedings of the International Workshop on Sharable Natural Language Resources*, Ikoma, Nara, Japan.

Emele, Martin C. and Rémi Zajac. 1990. Typed unification grammars. In *Proceedings of the* 13^{th} *International Conference on Computational Linguistics*.

Götz, Thilo and Walt Detmar Meurers. 1995. Compiling HPSG type constraints into definite clause programs. In *Proceedings of the Thirty-Third Annual Meeting of the ACL*, Boston. Association for Computational Linguistics.

Götz, Thilo and Walt Detmar Meurers. 1996. The importance of being lazy - using lazy evaluation to process queries to HPSG grammars. In *Proceedings of TALN 96 (joint session with the Third International Conference on HPSG)*, Marseille, France.

Haridi, Seif and Sverker Janson. 1990. Kernel Andorra Prolog and its computation model. In David H. D. Warren and Peter Szeredi, editors, *Proceedings of the seventh international conference on logic programming*, pages 31–46. MIT Press.

King, Paul. 1989. *A Logical Formalism for Head-Driven Phrase Structure Grammar*. Ph.D. thesis, University of Manchester.

King, Paul. 1994. An expanded logical formalism for Head-Driven Phrase Structure Grammar. Arbeitspapiere des SFB 340 Nr. 59, Universität Tübingen.

Meurers, W. Detmar. 1994. On implementing an HPSG theory – Aspects of the logical architecture, the formalization, and the implementation of head-driven phrase structure grammars. In: Erhard W. Hinrichs, W. Detmar Meurers, and Tsuneko Nakazawa: *Partial-VP and Split-NP Topicalization in German – An HPSG Analysis and its Implementation*. Arbeitspapiere des SFB 340 Nr. 58, Universität Tübingen.

Naish, Lee. 1985. *Negation and Control in Prolog*. Springer-Verlag.

Santos Costa, Vítor, David H. D. Warren, and Rong Yang. 1991. The Andorra-I preprocessor: Supporting full Prolog on the Basic Andorra model. In *Proceedings of the Eighth International Conference on Logic Programming*, pages 443–456.