Codeco: A Grammar Notation for Controlled Natural Language in Predictive Editors

Tobias Kuhn

Department of Informatics
University of Zurich

Second Workshop on Controlled Natural Language
15 September 2010
Marettimo (Italy)

Introduction

- Problem: Existing grammar frameworks do not work out particularly well for CNLs.
- Reason:
  - CNLs have essential differences to other languages (natural and formal ones)
  - To solve the writability problem, CNLs have to be embedded in special tools with very specific requirements.
    - Error messages and suggestions
    - Predictive editors
    - Language generation
- Solution: A new grammar notation that is dedicated to CNLs and predictive editors.

CNL Grammar Requirements

Concreteness: CNL grammars should be fully formalized and interpretable by computers.

Declarativeness: CNL grammars should not depend on a concrete algorithm or implementation.

Lookahead Features: CNL grammars should allow for the retrieval of possible next tokens for a partial text.

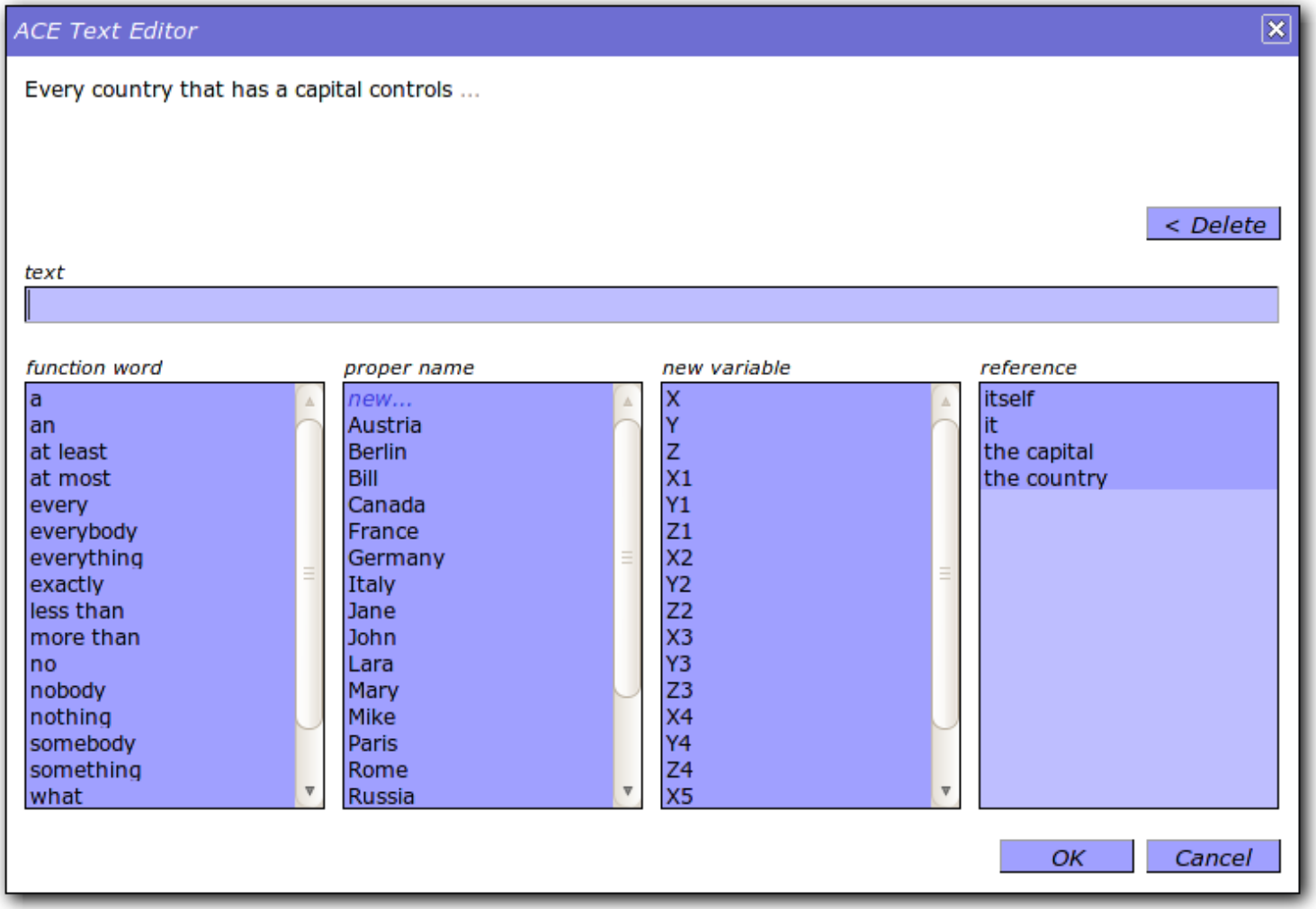
Anaphoric References: CNL grammars should allow for the definition of nonlocal structures like anaphoric references.

Implementability: CNL grammars should be easy to implement in different programming languages.

Expressivity: CNL grammars should be sufficiently expressive to express CNLs.

Lookahead Features

Predictive editors need to know which words can follow a partial text:



Anaphoric References

- Anaphoric references in CNLs are resolvable in a deterministic way:

  A country contains an area that is not controlled by the country.
  If a person X is a relative of a person Y then the person Y is a relative of the person X.
  John protects himself and Mary helps him.

- Anaphoric references that cannot be resolved should be disallowed:
  * *Every area is controlled by it.*
  * *The person X is a relative of the person Y.*

- Scopes have to be considered too:

  Every man protects a house from every enemy and does not destroy ...
  ... *himself.*
  ... *the house.*
  * ... *the enemy.*

Existing Grammar Frameworks

- Grammar Frameworks for Natural Languages
  - Head-Driven Phrase Structure Grammars
  - Lexical-Functional Grammars
  - Tree-Adjoining Grammars
  - Combinatory Categorial Grammars
  - Dependency Grammars
  - *... and many more*
- Backus-Naur Form (BNF)
- Parser Generators
  - Yacc
  - GNU Bison
  - ANTLR
- Definite Clause Grammars (DCG)
- Grammatical Framework (GF)

How Existing Grammar Frameworks Fulfill our Requirements for CNL Grammars

	NL	BNF	PG	DCG	GF
Concreteness	+	+	+	+	+
Declarativeness	+/−	+	−	(+)	+
Lookahead Features	−	+	(+)	(+)	+
Anaphoric References	(+)	−	−	(+)	−
Implementability	−	+	−	−	**?**
Expressivity	+	−	+	+	+

The Codeco Notation

Codeco = "Concrete and Declarative Grammar Notation for Controlled Natural Languages"

- Formal and Declarative
- Easy to implement in different programming languages.
- Expressive enough for common CNLs.
- Lookahead features can be implemented in a practical and efficient way.
- Deterministic anaphoric references can be defined in an adequate and simple way.

Grammar Rules in Codeco

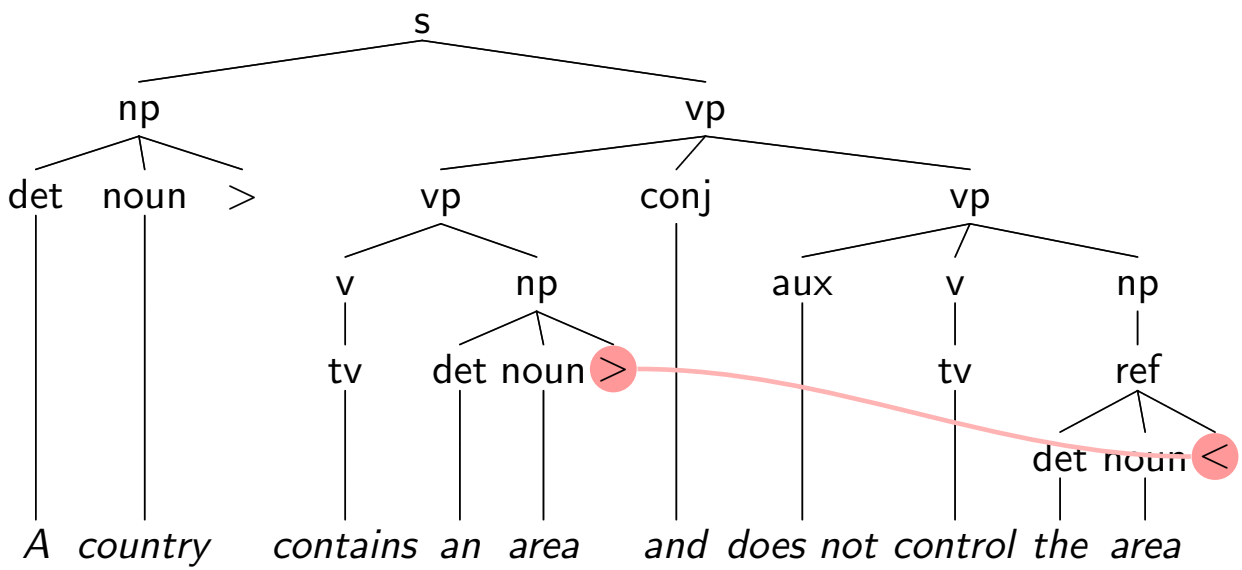
- Grammatical categories with flat feature structures
- Category Types:
  - non-terminal (e.g. *vp*)
  - pre-terminal (e.g. *noun*)
  - terminal (e.g. [does not])
- Grammar Rule Examples:
  - $vp\begin{pmatrix}\text{num: }\boxed{\text{Num}}\\ \text{neg: }\boxed{\text{Neg}}\end{pmatrix} \overset{:}{\longrightarrow} v\begin{pmatrix}\text{num: }\boxed{\text{Num}}\\ \text{neg: }\boxed{\text{Neg}}\\ \text{type: tr}\end{pmatrix} \; np\begin{pmatrix}\text{case: acc}\end{pmatrix}$
  - $v\begin{pmatrix}\text{neg: }+\\ \text{type: }\boxed{\text{Type}}\end{pmatrix} \overset{:}{\longrightarrow} [\text{does not}] \; verb\begin{pmatrix}\text{type: }\boxed{\text{Type}}\end{pmatrix}$
  - $np\begin{pmatrix}\text{noun: }\boxed{\text{Noun}}\end{pmatrix} \overset{:}{\longrightarrow} [\text{a}] \; \underline{noun}\begin{pmatrix}\text{text: }\boxed{\text{Noun}}\end{pmatrix}$
  - $\underline{noun}\begin{pmatrix}\text{text: woman}\\ \text{gender: fem}\end{pmatrix} \rightarrow [\text{woman}]$

Forward and Backward References

The special categories "$>$" and "$<$" can be used to establish nonlocal dependencies, e.g. for anaphoric references:

$$np \xrightarrow{:} \underline{det}\left(\text{def: } -\right)\ \underline{noun}\left(\text{text: } \boxed{\text{Noun}}\right)\ >\left(\begin{array}{l}\text{type: noun}\\ \text{noun: } \boxed{\text{Noun}}\end{array}\right)$$

$$ref \xrightarrow{:} \underline{det}\left(\text{def: } +\right)\ \underline{noun}\left(\text{text: } \boxed{\text{Noun}}\right)\ <\left(\begin{array}{l}\text{type: noun}\\ \text{noun: } \boxed{\text{Noun}}\end{array}\right)$$



Scopes

- Opening of scopes:
  - Scopes in (controlled) English usually open at the position of the scope triggering structure, or nearby.
  - Scope opener category "$/\!/$" in Codeco:

$$quant\left(\text{exist: } -\right) \xrightarrow{\vdots} /\!/ \quad [\,\text{every}\,]$$

- Closing of scopes:
  - Scopes in (controlled) English usually close at the end of certain structures like verb phrases, relative clauses, and sentences.
  - Scope-closing rules "$\xrightarrow{\sim}$" in Codeco:

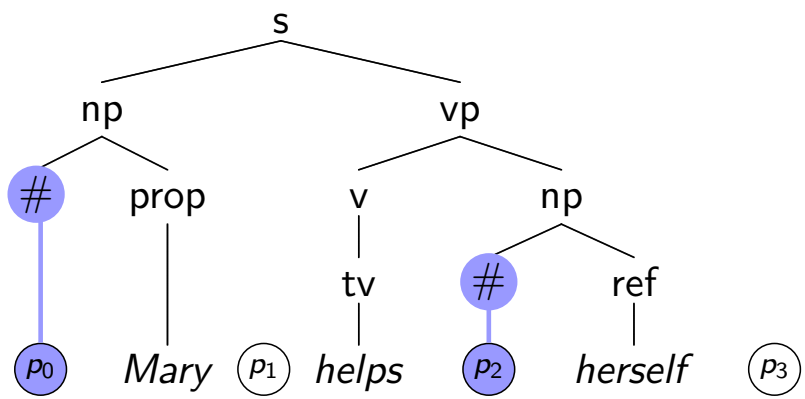
$$vp\left(\text{num: } \boxed{\text{Num}}\right) \xrightarrow{\sim} v\left(\begin{array}{l}\text{num: } \boxed{\text{Num}}\\ \text{type: tr}\end{array}\right) \; np\left(\text{case: acc}\right)$$

Position Operators

- How to define reflexive pronouns like "*herself*" that can only attach to the subject?
- With the position operator "#", position identifiers can be assigned:

$$np\left(\text{id: } \boxed{\text{Id}}\right) \overset{:}{\longrightarrow} \#\boxed{\text{Id}} \quad prop\left(\text{gender: } \boxed{\text{G}}\right) > \begin{pmatrix} \text{id: } \boxed{\text{Id}} \\ \text{gender: } \boxed{\text{G}} \\ \text{type: prop} \end{pmatrix}$$

$$ref\left(\text{subj: } \boxed{\text{Subj}}\right) \overset{:}{\longrightarrow} [\,\text{herself}\,] < \begin{pmatrix} \text{id: } \boxed{\text{Subj}} \\ \text{gender: fem} \end{pmatrix}$$

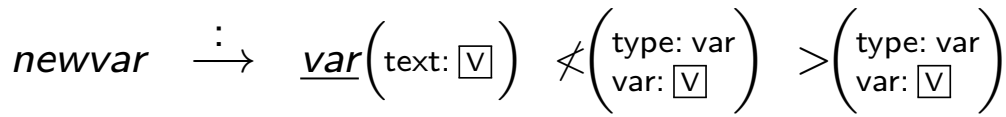


Negative Backward References

- How to define that the same variable can be introduced only once?

  **A person X knows a person X.*

- Negative backward references "$\not<$" succeed only if no matching antecedent is accessible:



Complex Backward References

- How to define pronouns like "*him*" that cannot attach to the subject?

  **John helps him.*

- Complex backward references "$<^{+}\ldots{}^{-}\ldots$" have one or more positive feature structure "$^{+}$" and zero or more negative ones "$^{-}$".
- They succeed if there is an antecedent that matches one of the positive feature structures but none of the negative ones:

$$\mathit{ref}\left(\begin{array}{l}\text{subj: } \boxed{\text{Subj}} \\ \text{case: acc}\end{array}\right) \overset{\vdots}{\longrightarrow} \quad [\,\text{him}\,] \quad <^{+}\left(\begin{array}{l}\text{human: } + \\ \text{gender: masc}\end{array}\right)^{-}\left(\text{id: } \boxed{\text{Subj}}\right)$$

- A more complicated (but probably less useful) example:

$$\mathit{ref}\left(\text{subj: } \boxed{\text{Subj}}\right) \overset{\vdots}{\longrightarrow} \quad [\,\text{this}\,] \quad <^{+}\left(\begin{array}{l}\text{hasvar: } - \\ \text{human: } -\end{array}\right)\left(\text{type: relation}\right)^{-}\left(\text{id: } \boxed{\text{Subj}}\right)\left(\text{type: prop}\right)$$

Strong Forward References

- How to define that propernames like "*Bill*" are always accessible?

  **Mary does not love a man. Mary hates him.*

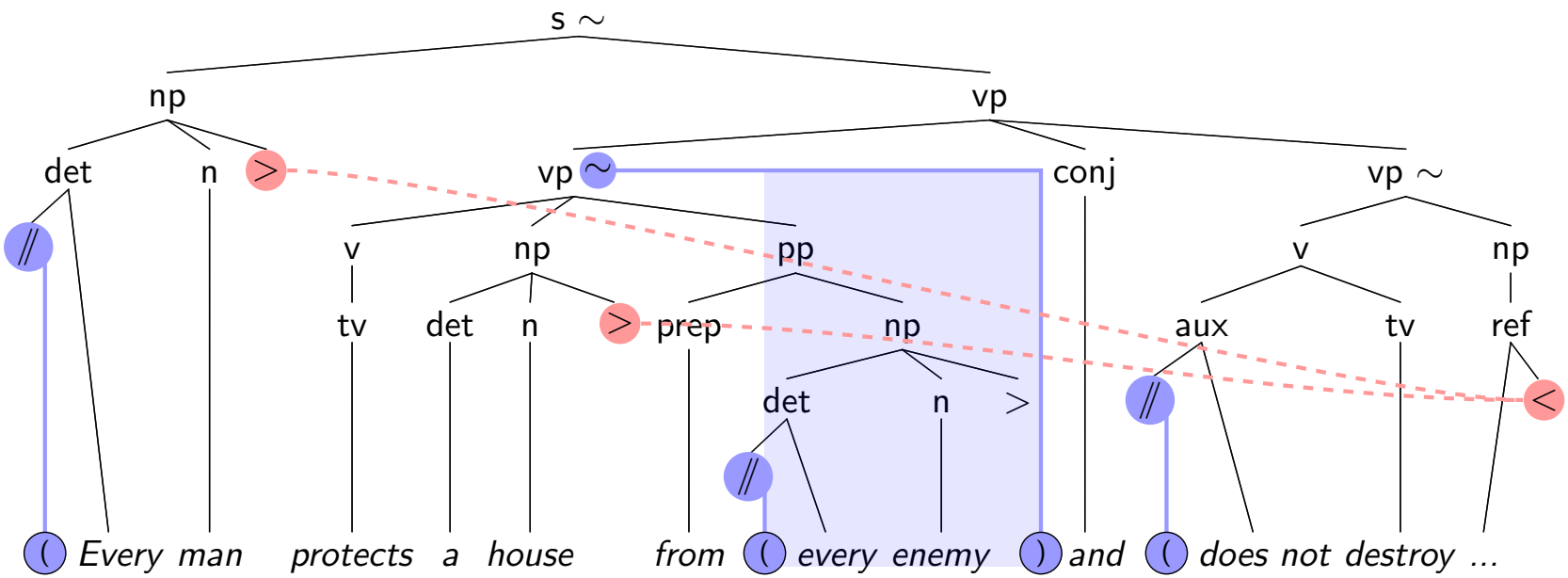
  Mary does not love Bill. Mary hates him.

- Strong forward references "$\gg$" are always accessible:

$$np\left(\text{id: } \boxed{\text{Id}}\right) \xrightarrow{:} prop\left(\begin{matrix}\text{human: } \boxed{\text{H}} \\ \text{gender: } \boxed{\text{G}}\end{matrix}\right) \gg \left(\begin{matrix}\text{id: } \boxed{\text{Id}} \\ \text{human: } \boxed{\text{H}} \\ \text{gender: } \boxed{\text{G}} \\ \text{type: prop}\end{matrix}\right)$$

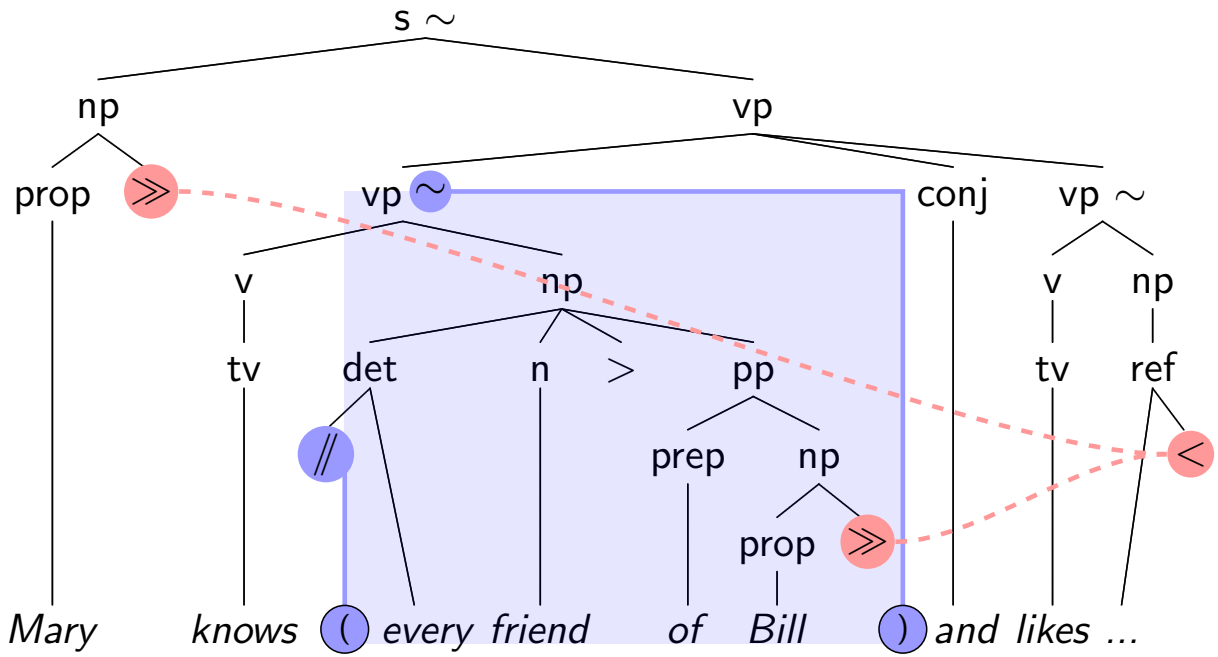
Reference Resolution: Accessibility

Forward references are only accessible if they are not within a scope that has already been closed before the position of the backward reference:



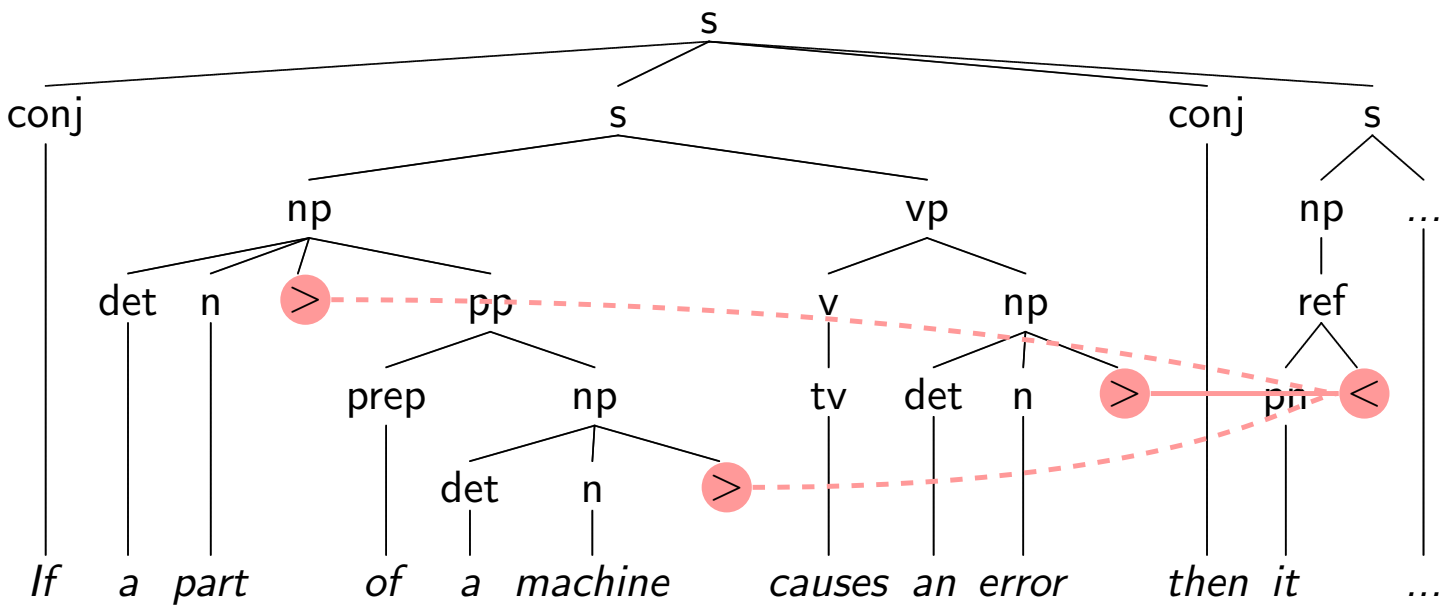
Reference Resolution: Accessibility

Strong forward references are always accessible:



Reference Resolution: Proximity

If a backward reference matches more than one forward reference then the closest one is taken:



Possible Extensions

- Semantics (e.g. with λ-DRSs)
- General feature structures (instead of flat ones)
- ...

Parsers for Codeco

Two parsers with different parsing approaches exist:

- Transformation into Prolog DCG
  - fast (1.5 ms per sentence)
  - no lookahead features
  - ideal for regression tests and parsing of large texts in batch mode
- Execution in a chart parser (Earley parser) under Java
  - slower, but still reasonably fast (130 ms per sentence)
  - lookahead features
  - ideal for predictive editors in Java

ACE in Codeco

- Large subset of ACE in Codeco
  - Includes: countable nouns, proper names, intransitive and transitive verbs, adjectives, adverbs, prepositions, plurals, negation, comparative and superlative adjectives and adverbs, *of*-phrases, relative clauses, modality, numerical quantifiers, coordination of sentences / verb phrases / relative clauses, conditional sentences, questions, and anaphoric references (simple definite noun phrases, variables, and reflexive and irreflexive pronouns)
  - Excludes: Mass nouns, measurement nouns, ditransitive verbs, numbers and strings as noun phrases, sentences as verb phrase complements, Saxon genitive, possessive pronouns, noun phrase coordination, and commands
- 164 grammar rules
- Used in the ACE Editor:
  `http://attempto.ifi.uzh.ch/webapps/aceeditor/`

Evaluation of ACE Codeco

Exhaustive Language Generation:

- Evaluation subgrammar with 97 grammar rules
- Minimal lexicon
- 2'250'869 sentences with 3–10 tokens:

sentence length	number of sentences	growth factor
3	6	
4	87	14.50
5	385	4.43
6	1'959	5.09
7	11'803	6.03
8	64'691	5.48
9	342'863	5.30
10	1'829'075	5.33
3–10	2'250'869	

- All are accepted by the ACE parser
  → ACE Codeco is a subset of ACE
- None is generated more than once
  → ACE Codeco is unambiguous

Evaluation of the Codeco notation and its implementations

Prolog DCG representation versus Java Earley parser:

- Equivalence of the Implementations:

  Generate the same set of sentences up to 8 tokens
  → The two implementations process Codeco in the same way

- Performance Tests:

task	grammar	implementation	seconds/sentence
generation	ACE Codeco eval. subset	Prolog DCG	0.00286
generation	ACE Codeco eval. subset	Java Earley parser	0.0730
parsing	ACE Codeco eval. subset	Prolog DCG	0.000360
parsing	ACE Codeco eval. subset	Java Earley parser	0.0276
parsing	full ACE Codeco	Prolog DCG	0.00146
parsing	full ACE Codeco	Java Earley parser	0.134
parsing	full ACE	APE	0.0161

Conclusions

Codeco ...

- ... fulfills our requirements for CNLs in predictive editors.
- ... is suitable to describe a large subset of ACE.
- ... allows for automatic tests.

- ... stands for a principled and engineering focused approach to CNL.

Thank you for your attention!

Questions & Discussion