Grammar Engineering for Deep Linguistic Processing SS2009

Lecture 3: TDL

Yi Zhang

Department of Computational Linguistics & Phonetics
Saarland University

Language Technology Lab
German Research Center for Artificial Intelligence

12th May, 2008
Flashbacks

Previously on Grammar Engineering . . .
- Formalism based on Typed Feature Structures (TFS)
- Multiple inheritance type hierarchy
- Linguistic Knowledge Builder (LKB) — a first impression
Outline

1. Typed Description Language

2. LKB Grammar Files
Outline

1. Typed Description Language

2. LKB Grammar Files
TDL— A Type Description Language for Constraint-Based Grammars

- [Krieger and Schäfer, 1994]
- Originally used in PAGE system
- Simplification and extensions in LKB
  ⇒ DELPH-IN reference formalism
- Fully compatible implementation in PET
TDL Syntax — Examples

- **Type inheritance:**
  
  \[
  \text{feat-struc} := *\text{top}* . \\
  \text{or} \\
  \text{feat-struc} :< *\text{top}* .
  \]

- **Type inheritance with attribute-value constraints:**
  
  \[
  \text{agr-cat} := \text{gen-agr-cat} \ & \ [ \ \text{PER per,} \ \text{NUM num,} \ \text{GEND gend} \ ] .
  \]

- **Multiple inheritance and coreference:**
  
  \[
  \text{head-feat-principle} := \text{grule} \ & \ \text{head-dtr-type} \ & \ [ \ \text{SYNSEM} \ [ \ \text{HEAD} \ #\text{head} \ ] , \ \text{H-DTR} \ [ \ \text{SYNSEM} \ [ \ \text{HEAD} \ #\text{head} \ ] \ ] \ ] .
  \]
Typed Description Language

TDL Syntax

Formal Description

- Type-def → Type Avm-def . | Type Subtype-def .
- Type → identifier
- Subtype-def → <: Type
- Avm-def → := Conjunction
- Conjunction → Term | Term & Conjunction
- Term → Type | string | Feature-term | Coreference
- Feature-term → [ ] | [ Attr-val-list ]
- Attr-val-list → Attr-val | Attr-val, Attr-val-list
- Attr-val → Attr-list Conjunction
- Attr-list → Attribute | Attribute.Attr-list
- Attribute → identifier
- Coreference → #identifier
**TDL Syntax — Continued**

- *top* is a built-in type in LKB
- Identifiers are composed of \{a-z, A-Z, 0-9, _, -, +, *, ?\}
  - Identifiers are case-insensitive
  - Conventionally, attributes in upper cases, types in lower cases
- Lisp-style comments:
  - Single line comments are started with `;`
  - Multi-line comments are bracketed by `#| |#`
Lists

list ::= *top*.
e-list ::= list.
ne-list ::= list &
           [ FIRST *top*,
            REST list ].
List Abbreviations

- `<a, b, c>`
  
  
  [ FIRST a,
  
  REST [ FIRST b,
  
  REST [ FIRST c,
  
  REST e-list ] ] ] ]

- `<a, b, c,...>`
  
  [ FIRST a,
  
  REST [ FIRST b,
  
  REST [ FIRST c,
  
  REST list ] ] ] ]

- `<a.b>`
  
  [ FIRST a,
  
  REST b ]
Difference Lists

- Allows more flexible list operation: concatenation, append, remove from end, . . . , simply using unification.

- Definition:

  \[ \text{diff-list} := \text{top} \& \left[ \text{LIST} \text{ list}, \text{LAST} \text{ list} \right]. \]

  - \text{LIST} points to the beginning position
  - \text{LAST} points to the end position
Abbreviation: \(<!a,b,c!>\)

\[
[ \text{LIST} [ \text{FIRST} \ a, \\
\text{REST} [ \text{FIRST} \ b, \\
\text{REST} [ \text{FIRST} \ c, \\
\text{REST} \text{#last} ] ] ], \\
\text{LAST} \text{#last} ]
\]
List & Diff-list

Formal Description

- **Term** → Type | string | Feature-term | Coreference | List | Diff-list
- **Diff-list** → < ! ! > | < ! Conjunction-list ! >
- **Conjunction-list** → Conjunction | Conjunction, Conjunction-list
- **List** → <> | < Conjunction-list > | < Conjunction-list, ... > | < Conjunction-list . Conjunction >
Lexical Entries

Formal Description

- **Lexentry → LexID Avm-def .**
- **LexID → identifier**

\[
\text{me}_1 := \text{pron-lxm} \& \\
[ \text{ORTH } "me", \\
\text{SYNSEM } [ \text{HEAD noun} \& \\
[ \text{AGR non-3sing} \& \\
[ \text{PER 1st } ], \\
\text{CASE acc } ] ] ].
\]

- Should not be confused with *types*
Grammar Rules

Formal Description

- Ruleentry → RuleID Avm-def .
- RuleID → identifier

binary-rule := rule &

[ ARGΣ < sign, sign > ].

- A means of constructing new signs
- Have a definite number of daughters
- Lexical rules are treated as unary grammar rules that may apply before affixation
Morphological Rules

Formal Description

- $\text{Mruleentry} \rightarrow \text{RuleID} \ \text{Mgraph-spec-list} \ \text{Avm-def} \ .$
- $\text{Mgraph-spec-list} \rightarrow \text{Mgraph-spec} \ | \ \text{Mgraph-spec} \ \text{Mgraph-spec-list}$
- $\text{Mgraph-spec} \rightarrow \%\text{prefix} \ \text{SPair-list} \ | \ %\text{suffix} \ \text{SPair-list}$
- $\text{SPair-list} \rightarrow \text{SPair} \ | \ \text{SPair} \ \text{SPair-list}$
- $\text{SPair} \rightarrow ( * \ \text{Char-list} ) \ | \ ( \ \text{Char-list} \ \text{Char-list})$
- $\text{Char-list} \rightarrow \text{letter} \ | \ \text{Macro} \ | \ \text{letter} \ \text{Char-list} \ | \ \text{Macro} \ \text{Char-list}$
- $\text{Letterset} \rightarrow \% \ ( \text{letter-set} \ (\text{Macro letters}) \ )$
- $\text{Macro} \rightarrow \!\text{letter}$
- $\text{Irregentry} \rightarrow \text{base Rulespec inflected}$
Morphological Rules
Letter sets

Formal Description

- Letterset → % (letter-set (Macro letters))
- Macro → !letter

%(letter-set (!c bdfglmnprstz))
%(letter-set (!s abcdefghijklmnopqrstuvwxyz))
%(letter-set (!t bcdfghjklmnpqrstvwxz))
%(letter-set (!v aeiou))
Morphological Rules
Inflectional rules

Formal Description

- Mruleentry → RuleID Mgraph-spec-list Avm-def .
- Mgraph-spec-list → Mgraph-spec | Mgraph-spec Mgraph-spec-list
- Mgraph-spec → %prefix SPair-list | %suffix SPair-list
- SPair-list → SPair | SPair SPair-list
- SPair → ( * Char-list ) | ( Char-list Char-list)
- Char-list → letter | Macro | letter Char-list | Macro Char-list

\[
past\_verb\_infl\_rule :=
%suffix ( * ed ) (!ty !tied) (e ed)
lex\_rule\_infl\_affixed &
[ NEEDS-AFFIX +,
  ARGs [ FIRST [ AFFIX past\_verb ] ] ] ]
\]
Morphological Rules
Irregular forms

Formal Description
Irregentry $\rightarrow$ inflected Rulespec base

"saw PAST-VERB see
seen PSP-VERB see"

"
Outline

1. Typed Description Language

2. LKB Grammar Files
What Makes A **LKB** Grammar?

- Types and constraints
- Lexical entries
- Grammar rules
- Lexical and morphological rules
  * Start symbol descriptions
  * Parse node descriptions
The Script File

(lkb-load-lisp (this-directory) "globals.lsp")
(lkb-load-lisp (this-directory) "user-fns.lsp")
(load-lkb-preferences (this-directory) "user-prefs.lsp")
(read-tdl-type-files-aux
   (list (lkb-pathname (this-directory) "types.tdl")))
(read-tdl-lex-file-aux
   (lkb-pathname (this-directory) "lexicon.tdl"))

(read-tdl-grammar-file-aux
   (lkb-pathname (this-directory) "rules.tdl"))
(read-tdl-start-file-aux
   (lkb-pathname (this-directory) "start.tdl"))
(read-tdl-parse-node-file-aux
   (lkb-pathname (this-directory) "parse-nodes.tdl"))

... ...
References I

TDL - a Type Description Language for HPSG.
Technical Report RR-94-37, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH.