The Free Linguistic Environment

Theoretical Concepts and Basic Architecture

Damir Cavar

Department of Linguistics
Indiana University

IPI-PAN, May 23rd 2016

Outline

Team
IU Crew

- Hai Hu
- Lwin Moe
- Kenneth Steimel
- Tim Gilmanov
- Joshua Herring
- Michael Czerniakowski

Team
Supporters

Provided advice and ideas:

- Kenneth Beesley
- Lionel Clement
- Thomas Hanneforth
- Ronald Kaplan
- Gerald Penn
- Richard Sproat
- Annie Zaenen

Team
Supporters

- Provided morphologies and grammars to test:
  - Mary Dalrymple (Malagasi)
  - Helge Dyvik and Paul Meurer (Norwegian)
  - Agnieszka Patujek and Adam Przepiórkowski (Polish)
- The BNFC-team fixed several compiler issues for our code generation
- Morally supported and brought up the idea of various kinds of semantic processing, e.g. the Monotonicity Calculus:
  - Larry Moss

LFG Formalism

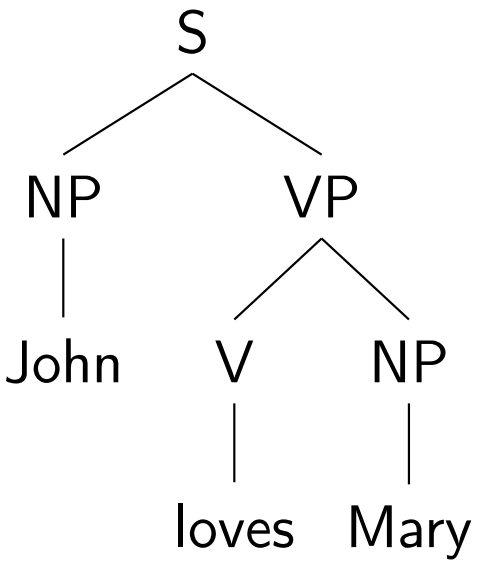
Rules:

$$\begin{array}{lcccc} S & \rightarrow & NP & & VP \\ & & (\uparrow \text{SUBJ})=\downarrow & & \uparrow=\downarrow \\ & & (\downarrow \text{CASE})=\text{NOM} & & \end{array}$$

$$NP \rightarrow \begin{pmatrix} D \\ \uparrow=\downarrow \end{pmatrix} \quad \begin{array}{c} N \\ \uparrow=\downarrow \end{array}$$

$$VP \rightarrow \begin{array}{c} V \\ \uparrow=\downarrow \end{array} \quad \begin{pmatrix} NP \\ (\uparrow \text{OBJ})=\downarrow \end{pmatrix}$$

LFG Formalism



$$\begin{bmatrix} \text{PRED} & \text{`}voli\langle \text{SUBJ,OBJ} \rangle\text{'} \\ \text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{`}Ivan\text{'} \\ \text{NUM} & \text{SG} \end{bmatrix} \\ \text{OBJ} & \begin{bmatrix} \text{PRED} & \text{`}Marija\text{'} \\ \text{NUM} & \text{SG} \end{bmatrix} \\ \text{TENSE} & \text{PRES} \\ \text{NUM} & \text{SG} \\ \text{PERS} & 3 \end{bmatrix}$$

LFG Formalism

John, N
$(\uparrow \text{PRED}) = \text{`}John\text{'}$
$(\uparrow \text{NUM}) = \text{SG}$
$(\uparrow \text{PERS}) = 3$

loves, V
$(\uparrow \text{PRED}) = \text{`}love\langle\text{SUBJ,OBJ}\rangle\text{'}$
$(\uparrow \text{NUM}) = \text{SG}$
$(\uparrow \text{PERS}) = 3$
$(\uparrow \text{TENSE}) = \text{PRES}$

Mary, V
$(\uparrow \text{PRED}) = \text{`}Mary\text{'}$
$(\uparrow \text{NUM}) = \text{SG}$
$(\uparrow \text{PERS}) = 3$

Constraints

C- and F-structure well-formedness

Lexical Integrity
The terminal nodes of c-structures are morphologically complete words.

Uniqueness/Consistency Condition
Every f-structure is such that every attribute has exactly one value.

Governable grammatical functions: SUBJ, OBJ, OBJ_Θ, OBL_Θ, COMP, XCOMP

Discourse functions: TOPIC, FOCUS, SUBJ

Completeness
An f-structure is complete if and only if it contains all the governable grammatical functions that its predicate governs.

Constraints

Coherence
An f-structure is coherent if and only if all the governable grammatical functions it contains are governed by a predicate.

Constraining equation
$(f$ PARTICIPLE$) =_c$ PRESENT

Negative equation
$(f$ CASE$) \neq$ NOMINATIVE
$\neg[(f$ CASE$) =$ NOMINATIVE$]$

Existential constraint
$(f$ CASE$)$

Negative existential constraint
$\neg(f$ CASE$)$

Constraints

Conjunction
Typically implicit or using & or $\wedge$

Disjunction
$\vee$ or $\{X|Y\}$

Grouping
$[\ldots]$

Optionality
$(\ldots)$

Regular Expression Operators
$*$ and $+$

Constraints

Empty categories: *pro*

$$f\begin{bmatrix}\text{PRED} & \text{`}\textit{continue}\langle\text{SUBJ}\rangle\text{'} \\ \text{TENSE} & \text{PRESENT} \\ \text{SUBJ} & \begin{bmatrix}\text{PRED} & \text{`pro'} \\ \text{NUMBER} & \text{SG}\end{bmatrix}\end{bmatrix}$$

Functional Uncertainty

wh-phrases have function FOCUS:

$(f\ \text{FOC})=(f\ \{\text{SUBJ}|\text{OBJ}\})$

Unbounded Uncertainty over Grammatical Functions

$(f\ \text{FOC})=(f\ \{\text{XCOMP}|\text{COMP}\}*\{\text{SUBJ}|\text{OBJ}\})$

Lexical Functional Grammar

Language Documentation and Engineering

- LFG is an attractive language documentation formalism (accessible to documentary, theoretical, and computational linguists).
- It appears to be suitable for computational applications: grammar engineering, real world applications for syntactic and semantic parsing (industry-level grammars).
- Ideal platform for white-box modeling.
- Missing: Induction, Probabilistic Modeling, *RAD*-type of Engineering Platform for Linguists, ...

Lexical Functional Grammar

Parser and Grammar Engineering Workbench

- Xerox Linguistic Environment (XLE) and Xerox Finite State Toolkit (XFST)
  - XLE-Web
  - XFST Homepage
- XLFG by Lionel Clément (University Bordeaux), LaBRI
- HPSG-framework
  - LKB

Probabilistic LFG
Hybrid LFGs

- Probability of c- and f-structure fragment tuples combined to larger structures (Kaplan, 1996), see Rens Bod's *Data Oriented Parsing* (DOP)
- Stochastic LFG (Johnson, 2000): Non-stochastic component: LFG Grammar
  Stochastic component: relation between input and candidate pairs of c- and f-structure
- Learning LFGs from corpora (Cahill, 2004)

Probabilistic LFG
Hybrid LFGs

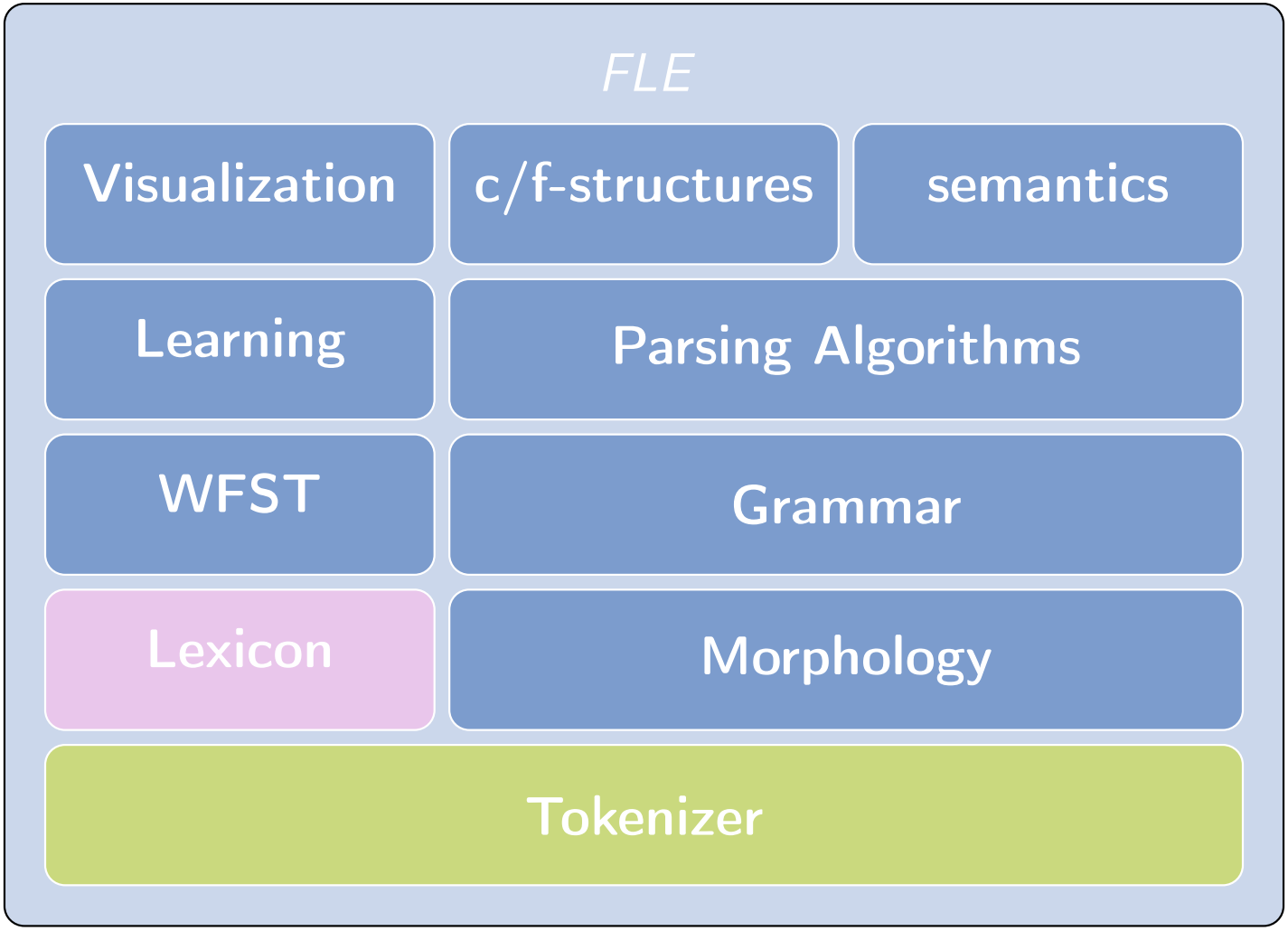
Goals:

- Combination of white-box and black-box modeling with white-box transparency
- Cognitive or Psycholinguistic modeling
- Disambiguation and most probable parse tree
- Large scale data (treebanks) for evaluation and extension of grammars
  - Lexical acquisition
  - Rule induction

Problem:

- Missing software and algorithmic base.

Architecture
General Components



General Environment
Code, Compilers, and Tools

Basic development environment:

- Coding in C++11 and newer using GCC/G++, Clang/LLVM, MS VisualStudio.
- CMake-based compiler configuration.
- BNFC-based grammar to code conversion (using flex and bison).
- Doxygen-based code documentation.
- Git-based code and version management (using Bitbucket).
- Clion IDE.
- On OS: Linux, Mac, Windows

General Environment
Code and Libraries

The following libraries are required:

- C++ Standard Library
- Boost Libraries
- OpenFST
- Foma

The following libraries are optional:

- Ucto for rule-based tokenization (Jain et al., 2012)

All libraries and tools are available for all common OS.

Code-base

Code and Language Data

Some Statistics

Using: 2nd generation i7 CPU and Ubuntu 16.04

- Lines of code (mainly C++, some shell-scripts): > 65,000
- Runtime-behavior:
  - Processing word-list using Foma-based morphology: **112,656 tokens** with some lexical ambiguities (results: 127,713): **300 milliseconds total**
  - Parsing **3,000 structurally ambiguous sentences** (avg. length 4.5) with 7,003 resulting c-structures: **917 milliseconds**. (not optimized, with tokenization, morphological analysis)

Tokenizer

Finite State and rule-based

Components:

- Basic tokenizers as part of the FLE-code (simple C++-functions).
- Foma-based tokenization (Hulden, 2009) for various languages, including Burmese.
- Ucto for rule-based tokenization (Jain et al., 2012). Ucto is coded in C++ and open and free license.

Language Resources:

- Own tokenizer framework for languages like Burmese, Thai, Lao and Khmer.
- Ucto covers English, Dutch, French, Italian, and Swedish.

Morphology
Finite State based Two-level Morphology

Components:

- Foma-based compilation of XFST-compatible morphologies to Finite State Transducer networks (Hulden, 2009)
- OpenFST-based morphologies and parsers (Allauzen et al., 2007)

Language resources:

- Our own open morphologies (English, Croatian, Burmese, . . .)
- Freely available large scale morphologies (e.g. Arabic, Malagasy, Polish, Norwegian, . . .)

Morphology
Mapping Morphology

Mapping of morphological features:

- Morphology output translation:
  - Mapping to category symbols as used in grammar.
  - Mapping of feature-labels to feature structures in syntax and unification algorithm.

Grammars
Processing Grammars

Grammar formalisms defined using Labeled Backus–Naur Form (LBNF):

- Context Free Frammars (CFG)
- Probabilistic Context Free Grammars (PCFG)
- XLE-grammars

Code Generation using BNFC:

- BNFC generates C, C++, Haskell, Java, etc. code from LBNF-grammars.
- Implementation of a Visitor-based parser for all three grammar formats mapping them on one uniform Weighted Finite State Transducer (WFST) model.

Grammars
WFST Mapping

Grammars as WFSTs:
T as a 7-tuple $(Q, \Sigma, \Gamma, I, F, \lambda, \rho)$ with

- Q a finite set of states
- Σ a finite set over the input alphabet
- Γ a finite set over the output alphabet
- I a subset of Q of initial states (we use only one initial state)
- F a subset of Q of final states
- $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times Q \times K$, a mapping of a state $\in Q$ and an input symbol $\in \Sigma \cup \{\epsilon\})$ to an output symbol $\in \Gamma \cup \{\epsilon\}$ and a new state $\in Q$; and $\lambda : I \rightarrow K$ mapping initial states and $\rho : F \rightarrow K$ final states to weights.

Grammars
WFST Mapping

Grammars as WFSTs:

- Transitions from one state to another consume a right-hand-side (RHS) symbol and emit ϵ or the corresponding left-hand-side (LHS) symbol of a CFG-rule, and an associated weight w.
- The weight w can be associated with:
  - a transitional probability over the RHS-symbols,
  - the fraction of a rule probability (e.g. PCFG),
  - an abstract function (e.g. unification)

Grammars
XLE-Grammar Part 1

```
S -> e: (^ TENSE);
        (NP: (^ XCOMP* { OBJ | OBJ2 })=!
        (^ TOPIC)=!)
    NP: (^ SUBJ)=!
        (! CASE)=NOM;
    { VP | VPaux }.

VP -> V
     (NP: (^ OBJ)=!
          (! CASE)=ACC)
     PP*: ! $ (^ ADJUNCT).

VPaux -> AUX
         VP.
```

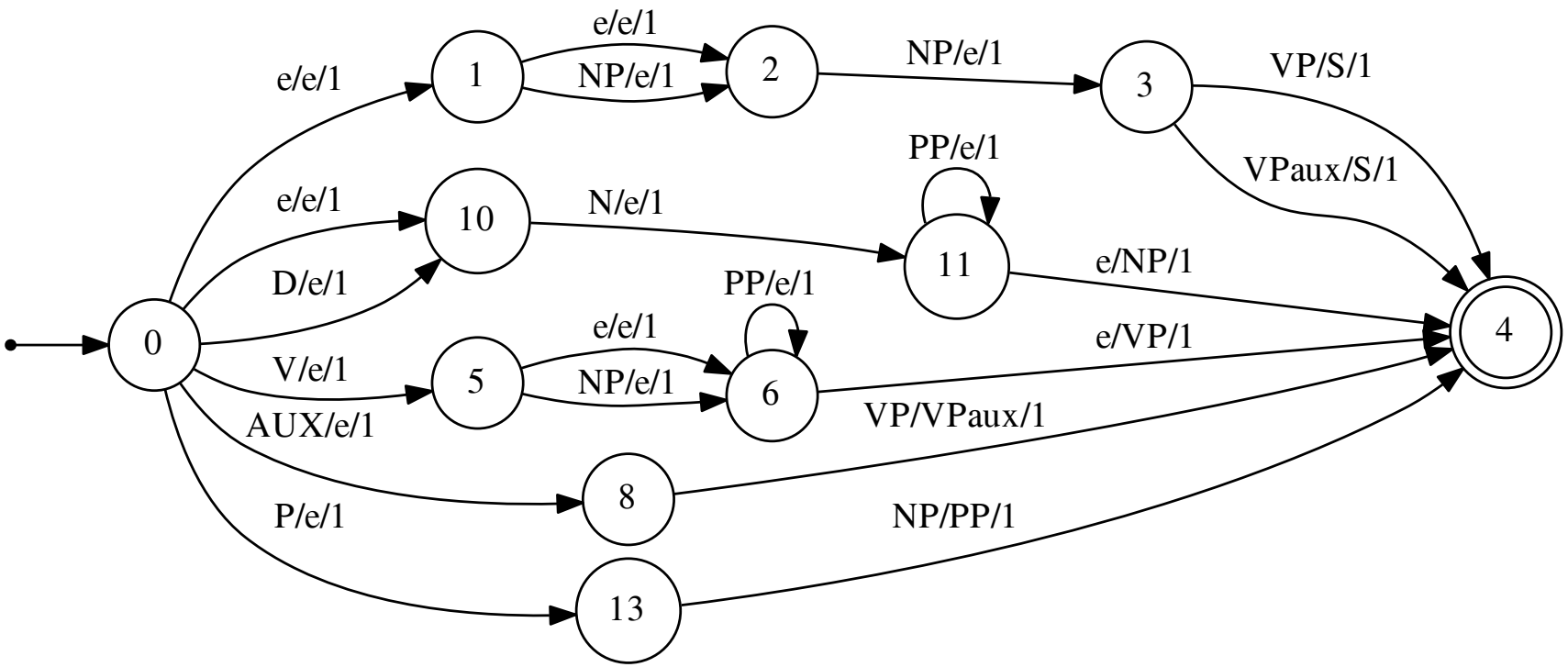
Grammars
XLE-Grammar Part 2

```
NP -> (D)
       N
      PP*: ! $ (^ ADJUNCT).

PP -> P
      NP: (^ OBJ)=!
          (! CASE)=ACC.
```

Grammars
WFST Mapping

Grammars as WFSTs:
(hidden feature structures, showing weight 1)



Parser

WFST-based Early-type Chart-parser

Parsing: Early-like Unification Parser (Earley, 1968, 1970; Aycock and Horspool, 2002; Sikkel, 1997)

- Recursive application of WFST and edge generation on chart.
  - Unification is a filter on the transitions in the WFST, or
  - Unification applies to the complete parse trees after the (P)CFG-backbone terminates (Maxwell III and Kaplan, 1996).
- Chart expansion based on classical Earley-algorithm.
- Unification algorithm: (Aho et al., 1974; Sikkel, 1997)
  - Feature structures are Directed Acyclic Graphs (DAG)
  - Non-destructive Graph-based Unification Algorithm

Solution Algorithm

Currently:

- Nothing yet, except first concept of the graph-based unification algorithm, but necessary:
  - handling of constraints and negation
  - complex processing of feature disjunctions
  - handling of semantic features (beyond the LFG-framework, e.g. lexical semantics, Monotonicity)

Solution Algorithm

Currently:

- Nothing yet, except first concept of the graph-based unification algorithm, but necessary:
  - handling of constraints and negation
  - complex processing of feature disjunctions
  - handling of semantic features (beyond the LFG-framework, e.g. lexical semantics, Monotonicity)

Solution Algorithm

Currently:

- Nothing yet, except first concept of the graph-based unification algorithm, but necessary:
  - handling of constraints and negation
  - complex processing of feature disjunctions
  - handling of semantic features (beyond the LFG-framework, e.g. lexical semantics, Monotonicity)

Solution Algorithm

Currently:

- Nothing yet, except first concept of the graph-based unification algorithm, but necessary:
  - handling of constraints and negation
  - complex processing of feature disjunctions
  - handling of semantic features (beyond the LFG-framework, e.g. lexical semantics, Monotonicity)

Solution Algorithm

Currently:

- Nothing yet, except first concept of the graph-based unification algorithm, but necessary:
  - handling of constraints and negation
  - complex processing of feature disjunctions
  - handling of semantic features (beyond the LFG-framework, e.g. lexical semantics, Monotonicity)

Solution Algorithm

Currently:

- Nothing yet, except first concept of the graph-based unification algorithm, but necessary:
  - handling of constraints and negation
  - complex processing of feature disjunctions
  - handling of semantic features (beyond the LFG-framework, e.g. lexical semantics, Monotonicity)

Solution Algorithm

Currently:

- Nothing yet, except first concept of the graph-based unification algorithm, but necessary:
  - handling of constraints and negation
  - complex processing of feature disjunctions
  - handling of semantic features (beyond the LFG-framework, e.g. lexical semantics, Monotonicity)

Solution Algorithm

Currently:

- Nothing yet, except first concept of the graph-based unification algorithm, but necessary:
  - handling of constraints and negation
  - complex processing of feature disjunctions
  - handling of semantic features (beyond the LFG-framework, e.g. lexical semantics, Monotonicity)

Solution Algorithm

Currently:

- Nothing yet, except first concept of the graph-based unification algorithm, but necessary:
  - handling of constraints and negation
  - complex processing of feature disjunctions
  - handling of semantic features (beyond the LFG-framework, e.g. lexical semantics, Monotonicity)

Solution Algorithm

Currently:

- Nothing yet, except first concept of the graph-based unification algorithm, but necessary:
  - handling of constraints and negation
  - complex processing of feature disjunctions
  - handling of semantic features (beyond the LFG-framework, e.g. lexical semantics, Monotonicity)

Solution Algorithm

Unification and other feature constraint resolution

Currently:

- Nothing yet, except first concept of the graph-based unification algorithm, but necessary:
  - handling of constraints and negation
  - complex processing of feature disjunctions
  - handling of semantic features (beyond the LFG-framework, e.g. lexical semantics, Monotonicity)

Parser

WFST-based Parsing using OpenFST and Thrax

WFST and Probabilistic Push-down Automata

- WFST using OpenFST with all its rich infrastructure.
- OpenGrm Thrax based on OpenFST (Tai et al., 2011; Roark et al., 2012)
  - Single-stack PDA for CFG
  - Multi-stack (or rather dual-stack PDA) for higher complexity

Parser

WFST and Probabilistic Push-down Automata

- Using WFST in advanced ways:
  - Restriction of certain recursion types to depth of N, e.g. center embeddings
  - Pre-compilation of feature structures to symbols on transducer arches
  - Learning while parsing, generating pre-compiled paths in the WFST on the fly, caching feature structures, etc.

FLE Development
Graphical User Interface

- Classic approach:
  - Qt 5 for Desktops
- New strategy
  - Universal GUI using:
    - JavaScript (and libs) and HTML5
    - Node.js (system access and client-server connectivity)
    - Node-Webkit, now NW.js (using Chromium) embeds Node.js in DOM, Web-technologies and Node.js in one
  - One GUI for web, mobile, platform independent desktops
  - Dynamic applications on- and offline (see JavaScript parser implementation)
  - Rich libraries for graphs (e.g. SVG, DS.js), widgets

FLE Development

Timeline

- Finalization of basic algorithms and pipeline.
- Compilation of executables and libraries for all appropriate platforms, and documentation, grammar engineering binary and runtime binary for efficient application.
- Visualization and grammar engineering front-end: JavaScript and HTML5-based widgets/application, SVG graphs.
- Grammar compilation components: reduction of unification steps, modularization of grammars and morphologies.
- Grammar engineering functionality: chart inspection, tracking of unification errors, parse stepper, feature guessing and induction, rule induction (CFG- or MCS-backbone), etc.
- Semantic components.
- Parallelization and performance tuning.

FLE Development
Reality Timeline

- Debugging
- Debugging
- Debugging
- ...

Git Repositories
on Bitbucket

- Hidden repository for developers and interested observers:
  - `http://btbucket.org/dcavar/fle/`
- Open repository:
  - `http://btbucket.org/dcavar/fle/`
- Mailing list:
  - `http://listserv.linguistlist.org/mailman/listinfo/lfgparser/`

References I

Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFst: A General and Efficient Weighted Finite-State Transducer Library. In *Proceedings of the Twelfth International Conference on Implementation and Application of Automata, (CIAA 2007)*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23, Prague, Czech Republic, 2007. Springer.

References II

John Aycock and R. Nigel Horspool. Practical Earley parsing. *The Computer Journal*, 45:620—630, 2002. doi: doi:10.1093/comjnl/45.6.620.

Aoife Cahill. *Parsing with Automatically Acquired, Wide-Coverage, Robust, Probabilistic LFG Approximations*. Doctoral dissertation, Dublin City University, 2004.

Jay Earley. *An Efficient Context-Free Parsing Algorithm*. PhD thesis, Carnegie-Mellon University, 1968.

Jay Earley. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102, February 1970. ISSN 0001-0782. doi: 10.1145/362007.362035. URL `http://doi.acm.org/10.1145/362007.362035`.

References III

Mans Hulden. Foma: a finite-state compiler and library. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 29–32. Association for Computational Linguistics, 2009.

Anil K. Jain, Lin Hong, and Sharath Pankanti. Ucto: Unicode Tokenizer version 0.5.3 Reference Guide. Technical Report ILK 12-05, Induction of Linguistic Knowledge Research Group, Tilburg Centre for Cognition and Communication, Tilburg University, Tilburg, The Netherlands, November 2012. URL `https://ilk.uvt.nl/ucto/ucto_manual.pdf`.

References IV

Mark Johnson. Stochastic lexical-functional grammars. Paper presented at the Lexical Functional Grammar Conference 2000, July 2000.

Ronald Kaplan. A probabilistic approach to lexical-functional grammar, August 1996. Presentation at the LFG Colloquium and Workshops, Rank Xerox Research Centre.

John T. Maxwell III and Ronald M. Kaplan. Unification parser that automatically take advantage of context freeness. In *Proceedings of the first LFG Conference (Grenoble)*, Stanford, 1996. CSLI Publications.

References V

Brian Roark, Richard Sproat, Cyril Allauzen, Michael Riley, Jeffrey Sorensen, and Terry Tai. The opengrm open-source finite-state grammar software libraries. In *Proceedings of the ACL 2012 System Demonstrations*, pages 61–66, 2012. URL `http://www.opengrm.org/`.

Klaas Sikkel. *Parsing Schemata*. Texts in Theoretical Computer Science. Springer-Verlag, Berlin, 1997.

Terry Tai, Wojciech Skut, and Richard Sproat. Thrax: An open source grammar compiler built on openfst, December 2011.