



A Principle Compiler for Extensible Dependency Grammar

*Bachelor Thesis
Programming Systems Lab*

Jochen Setz, 24.05.2007

Betreuer: Ralph Debusmann

Extensible Dependency Grammar

- Grammatikformalismus für
Abhängigkeitsgrammatik

Extensible Dependency Grammar

- Grammatikformalismus für
Abhängigkeitsgrammatik
- Modell: Tupel von Abhängigkeitsgraphen

Extensible Dependency Grammar

- Grammatikformalismus für
Abhängigkeitsgrammatik
- Modell: Tupel von Abhängigkeitsgraphen
- Abhängigkeitsgraphen heißen Dimensionen

Extensible Dependency Grammar

- Grammatikformalismus für
Abhängigkeitsgrammatik
- Modell: Tupel von Abhängigkeitsgraphen
- Abhängigkeitsgraphen heißen Dimensionen
- alle Dimensionen haben die gleiche
Knotenmenge

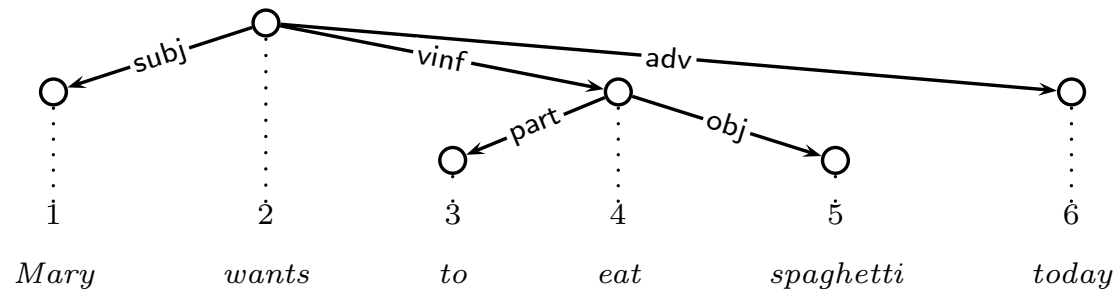
Extensible Dependency Grammar

- Grammatikformalismus für
Abhängigkeitsgrammatik
- Modell: Tupel von Abhängigkeitsgraphen
- Abhängigkeitsgraphen heißen Dimensionen
- alle Dimensionen haben die gleiche
Knotenmenge

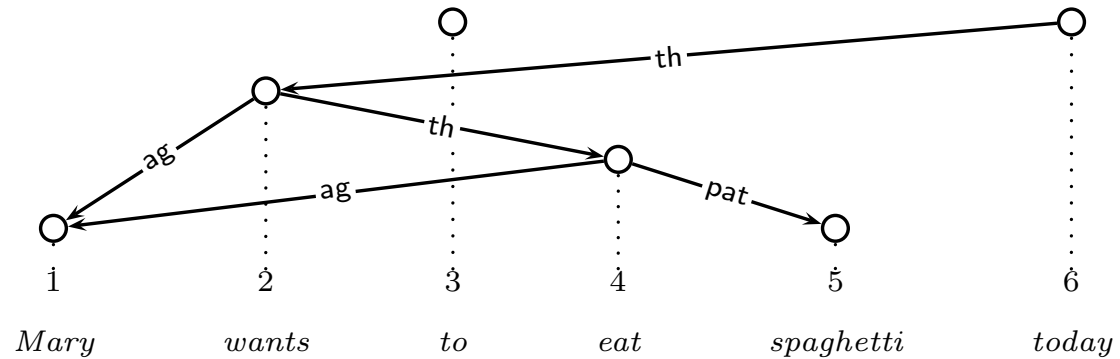
⇒ Modelle = Multigraphen

XDG Beispiel

SYN:



SEM:



Prinzipien

- Wohlgeformtheitbedingungen für Multigraphen

Prinzipien

- Wohlgeformtheitbedingungen für Multigraphen

Beispiele:

- Baumheit (eine Wurzel / keine Zyklen / keine oder eine Mutter pro Knoten)

Prinzipien

- Wohlgeformtheitbedingungen für Multigraphen

Beispiele:

- Baumheit (eine Wurzel / keine Zyklen / keine oder eine Mutter pro Knoten)
- Valenz (Anzahl und Art der ein-/ausgehenden Kanten)

Prinzipien

- Wohlgeformtheitbedingungen für Multigraphen

Beispiele:

- Baumheit (eine Wurzel / keine Zyklen / keine oder eine Mutter pro Knoten)
- Valenz (Anzahl und Art der ein-/ausgehenden Kanten)
- Projektivität (keine kreuzenden Kanten)

Prinzipien in FOL

- Prinzipien in First Order Logic (FOL) formalisiert

Prinzipien in FOL

- Prinzipien in First Order Logic (FOL) formalisiert

Z.B. Baumheit auf Dimension d :

- $NoCycles = \forall v : \neg(v \rightarrow_d^+ v)$
- $OneRoot = \exists^1 v : \neg \exists v' : v' \rightarrow_d v$
- $ZeroOrOneMother = \forall v : (\neg \exists v' : v' \rightarrow_d v) \vee (\exists^1 v' : v' \rightarrow_d v)$

XDK

The screenshot displays the XDK (XDK: Main window) interface, which is used for inspecting lexical entries and solving grammar rules. The main window shows the project 'Diplom.ul' and examples 'Diplom.txt'. The 'Inspect lexical entries' button is active, and the text 'eine huebsche kleine frau lacht' is entered in the 'Solve' field.

The interface includes several windows:

- Oz Explorer:** Shows a tree structure with nodes 1, 2, and 3.
- Oz Inspector:** Shows a list of lexical entries, including '5# unused lexical classes ...', 'prep obj id', 'piped lp', 'intra lp', 'vapp', 'prep ob', 'Collec', and 'Checki'.
- XDK: 1:** Shows a parse tree for the sentence 'eine huebsche kleine frau lacht' with nodes 1-5 and labels 'id', 'det', 'adj', 'adj', 'subj'.
- XDK: 2:** Shows a parse tree for the sentence 'eine huebsche kleine frau lacht' with nodes 1-5 and labels 'id', 'npf', 'nmf', 'nmf', 'n', 'v12'.

The main window also displays the following text:

```
Project Search Dimensions Principles Outputs Extras
Grammar: Diplom.ul
Examples: Diplom.txt
Inspect lexical entries
*****
* Thesis
* Chapter 1
maria liebt ihn
*****
* Chapter 2
maria hat einen reichen mann sehr geliebt
eine huebsche kleine frau lacht
*****
* Chapter 3
den mann hat maria geliebt
hat dem mann jemand erlaubt das buch zu lesen
Solve eine huebsche kleine frau lacht
```

Parser (Grundidee)

- Multigraphen kodiert durch endliche Mengen von Integern

Parser (Grundidee)

- Multigraphen kodiert durch endliche Mengen von Integern
- jeder Knoten v mit einem Index und verschiedenen Mengen assoziiert, z.B.:
 - mothers: Mutterknoten von v
 - daughters: Töchter von v
 - up: Vorfahren von v
 - daughtersL: Menge von Tupeln, jedes Tupel hat das gleiche Label

Parser (Grundidee)

- Multigraphen kodiert durch endliche Mengen von Integern
- jeder Knoten v mit einem Index und verschiedenen Mengen assoziiert, z.B.:
 - mothers: Mutterknoten von v
 - daughters: Töchter von v
 - up: Vorfahren von v
 - daughtersL: Menge von Tupeln, jedes Tupel hat das gleiche Label
- Prinzipien: Constraints auf diesen Mengen

Implementierung von Prinzipien

- Prinzipien müssen von Hand als Mozart/Oz-Constraints implementiert werden

Implementierung von Prinzipien

- Prinzipien müssen von Hand als Mozart/Oz-Constraints implementiert werden
- ⇒ nur Experten in Mozart/Oz und XDK können neue Prinzipien schreiben, nicht aber typische Anwender wie z.B. Linguisten

Implementierung von Prinzipien

- Prinzipien müssen von Hand als Mozart/Oz-Constraints implementiert werden
- ⇒ nur Experten in Mozart/Oz und XDK können neue Prinzipien schreiben, nicht aber typische Anwender wie z.B. Linguisten
- ⇒ große Lücke zwischen Formalisierung und Implementierung

Implementierung von Prinzipien

- Prinzipien müssen von Hand als Mozart/Oz-Constraints implementiert werden
- \Rightarrow nur Experten in Mozart/Oz und XDK können neue Prinzipien schreiben, nicht aber typische Anwender wie z.B. Linguisten
- \Rightarrow große Lücke zwischen Formalisierung und Implementierung
- provisorische Brücke: umfangreiche Bibliothek von vorimplementierten Prinzipien

Idee meiner BA-Thesis

- Lücke zwischen Formalisierung und Implementierung schließen

Idee meiner BA-Thesis

- Lücke zwischen Formalisierung und Implementierung schließen
- Ziel: Compiler (“PrincipleWriter”), der in FOL geschriebene Prinzipien in Mozart/Oz-Constraints übersetzt, und ins XDK integriert

Idee meiner BA-Thesis

- Lücke zwischen Formalisierung und Implementierung schließen
- Ziel: Compiler (“PrincipleWriter”), der in FOL geschriebene Prinzipien in Mozart/Oz-Constraints übersetzt, und ins XDK integriert
- dadurch XDK attraktiver für typische Anwender, die neue Prinzipien schreiben wollen: nur noch Kenntnisse in FOL benötigt

Idee meiner BA-Thesis

- Lücke zwischen Formalisierung und Implementierung schließen
- Ziel: Compiler (“PrincipleWriter”), der in FOL geschriebene Prinzipien in Mozart/Oz-Constraints übersetzt, und ins XDK integriert
- dadurch XDK attraktiver für typische Anwender, die neue Prinzipien schreiben wollen: nur noch Kenntnisse in FOL benötigt
- Nebeneffekt: Formalisierung und Implementierung rücken näher zusammen

Arbeitspakete

- konkrete Eingabesyntax definieren (fertig)

Arbeitspakete

- konkrete Eingabesyntax definieren (fertig)
- Mozart/Oz-Constraints für FOL finden (1. Version)

Arbeitspakete

- konkrete Eingabesyntax definieren (fertig)
- Mozart/Oz-Constraints für FOL finden (1. Version)
- Compiler bauen, der Eingabesyntax in Mozart/Oz-Constraints überführt, mit
 - Typ-Checker,
 - Typ-Inferenz

Arbeitspakete

- konkrete Eingabesyntax definieren (fertig)
- Mozart/Oz-Constraints für FOL finden (1. Version)
- Compiler bauen, der Eingabesyntax in Mozart/Oz-Constraints überführt, mit
 - Typ-Checker,
 - Typ-Inferenz

Wie gut kann der PrincipleWriter optimieren, verglichen mit einem Experten?

Literatur

- Ralph Debusmann (2006). Extensible Dependency Grammar: A Modular Grammar Formalism Based On Multigraph Description. PhD thesis (revised version)
- Ralph Debusmann (2007). Scrambling as the Intersection of Relaxed Context-Free Grammars in a Model-Theoretic Grammar Formalism. ESSLLI 2007 Workshop Model Theoretic Syntax at 10, Dublin

Optimierung - ZeroOrOneMother

FOL:

$$\forall v : (\neg \exists v' : v' \rightarrow_d v) \vee (\exists^1 v' : v' \rightarrow_d v)$$

UL:

forall V::node:

~exists V1::node: edge(V1 V D)

|

existsone V1::node: edge(V1 V D)

Optimierung - ZeroOrOneMother

Mozart/Oz:

```
proc {ZeroOrOneIncomingEdges NodeRecs}
  {ForAllNodes NodeRecs
    fun {$ NodeRec}
      {Disj
        {Nega
          {ExistsNodes NodeRecs
            fun {$ NodeRec1} {Edge NodeRec1 NodeRec d} end}}}}
      {ExistsOneNodes NodeRecs
        fun {$ NodeRec1} {Edge NodeRec1 NodeRec d} end}
      end 1}
    end
  end
```

Komplexität: $O(n^2)$

Optimierung - ZeroOrOneMother

$$\text{FOL: } \forall v : \underbrace{(\neg \exists v' : v' \rightarrow_d v)}_{|mothers(v)| = 0} \vee \underbrace{(\exists^1 v' : v' \rightarrow_d v)}_{|mothers(v)| = 1}$$

$$\text{Optimiert: } \forall v : |mothers(v)| \leq 1$$

Komplexität: $O(n)$