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 Dependenzgrammatik

Extensible Dependency Grammar

- Grammatikformalismus für Dependenzgrammatik
- Modell: Tupel von Dependenzgraphen

Extensible Dependency Grammar

- Grammatikformalismus für Dependenzgrammatik
- Modell: Tupel von Dependenzgraphen
- Dependenzgraphen heißen Dimensionen

Extensible Dependency Grammar

- Grammatikformalismus für Dependenzgrammatik
- Modell: Tupel von Dependenzgraphen
- Dependenzgraphen heißen Dimensionen
- alle Dimensionen haben die gleiche Knotenmenge

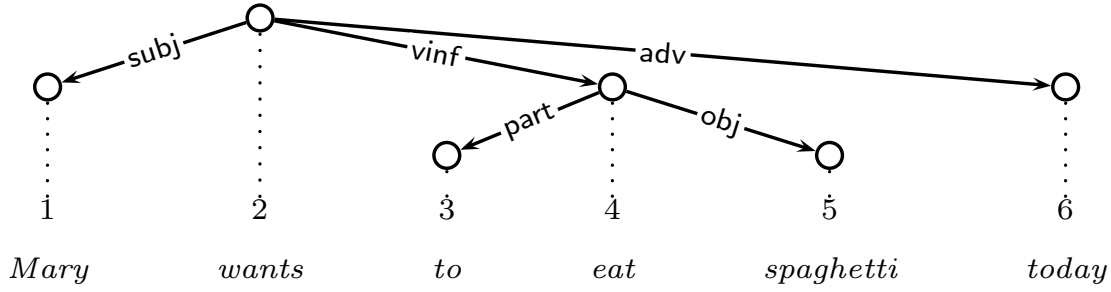
Extensible Dependency Grammar

- Grammatikformalismus für Dependenzgrammatik
- Modell: Tupel von Dependenzgraphen
- Dependenzgraphen heißen Dimensionen
- alle Dimensionen haben die gleiche Knotenmenge

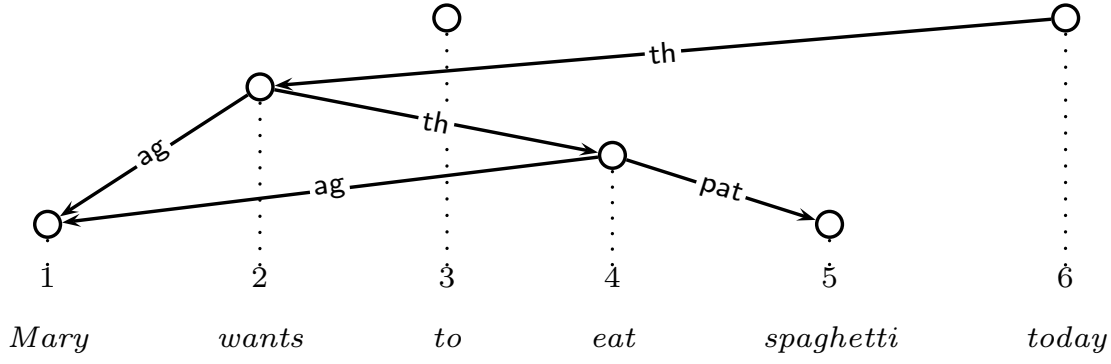
$\Rightarrow$ Modelle = Multigraphen

XDG Beispiel

SYN:



SEM:



Prinzipien

- Wohlgeformtheitbedingungen des Multigraphen

Prinzipien

- Wohlgeformtheitbedingungen des Multigraphen

Beispiele:

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

Prinzipien

- Wohlgeformtheitbedingungen des Multigraphen

Beispiele:

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

Prinzipien

- Wohlgeformtheitbedingungen des 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

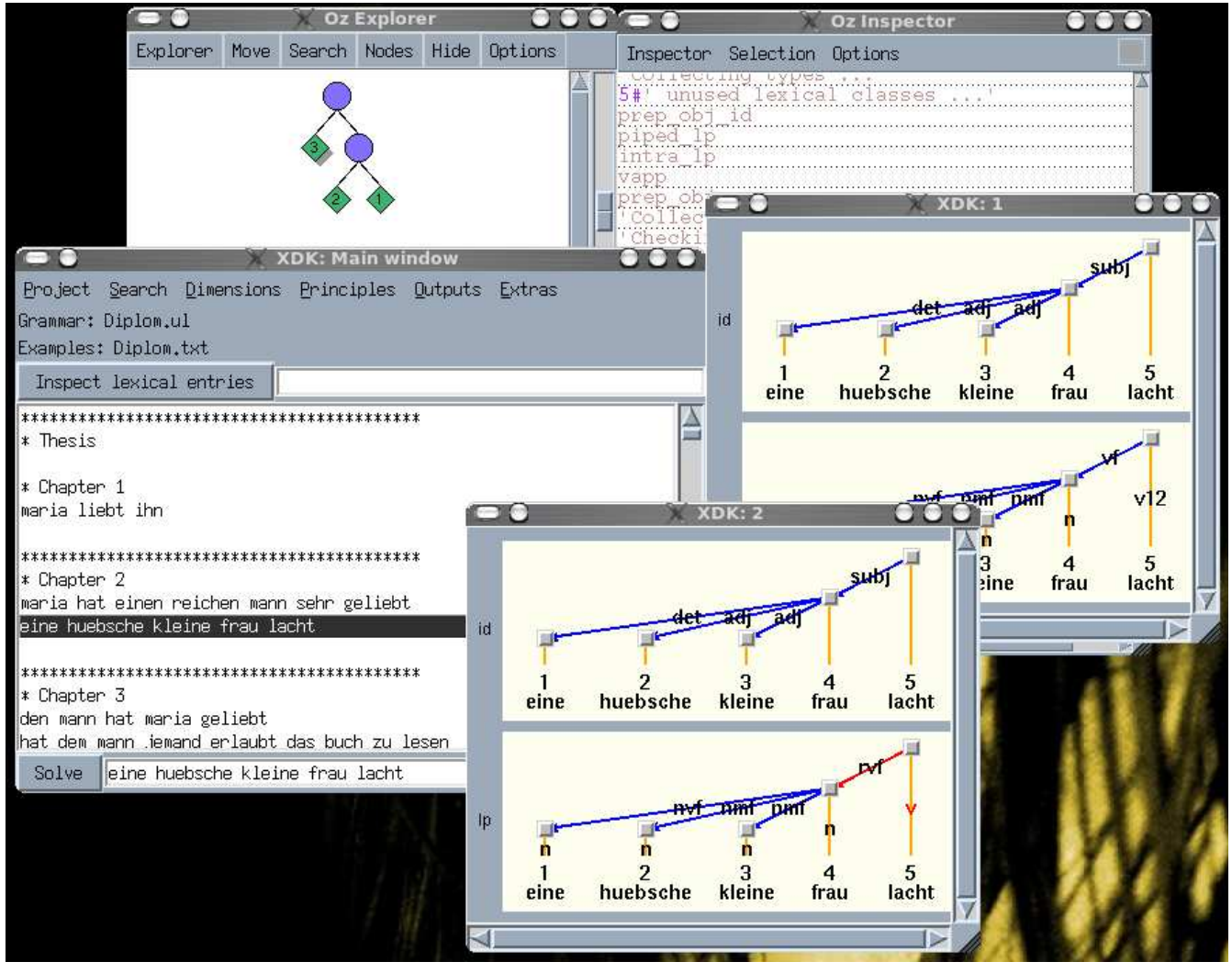
Prinzipien in FOL

- 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


Oz Explorer
Explorer Move Search Nodes Hide Options
Oz Inspector
Inspector Selection Options
XDK: Main window
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
XDK: 1
XDK: 2
id
lp
det adj adj subj
1 eine
2 huebsche
3 kleine
4 frau
5 lacht
nvf nmf pmf n v12 vf

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
- $\Rightarrow$ 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
- $\Rightarrow$ 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

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

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

Komplexität: $O(n^2)$

Optimierung - ZeroOrOneMother

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

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

Komplexität: O(n)