An Adaptation Technique for GF-Based Dialogue Systems

Faegheh Hasibi

August 31, 2012

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Outline

- Introduction
- The Baseline System
- Adaptation Technique
- Example: An Adaptable Transport Query System
- Evaluation
- Directions

Introduction

Examples in a transport dialogue system:

- Normal dialogue:

  User: I want to go from Chalmers to Valand today at 7:30

  System: Take tram number 7 from Chalmers to Valand at 7:33

- User adaptable dialogue:

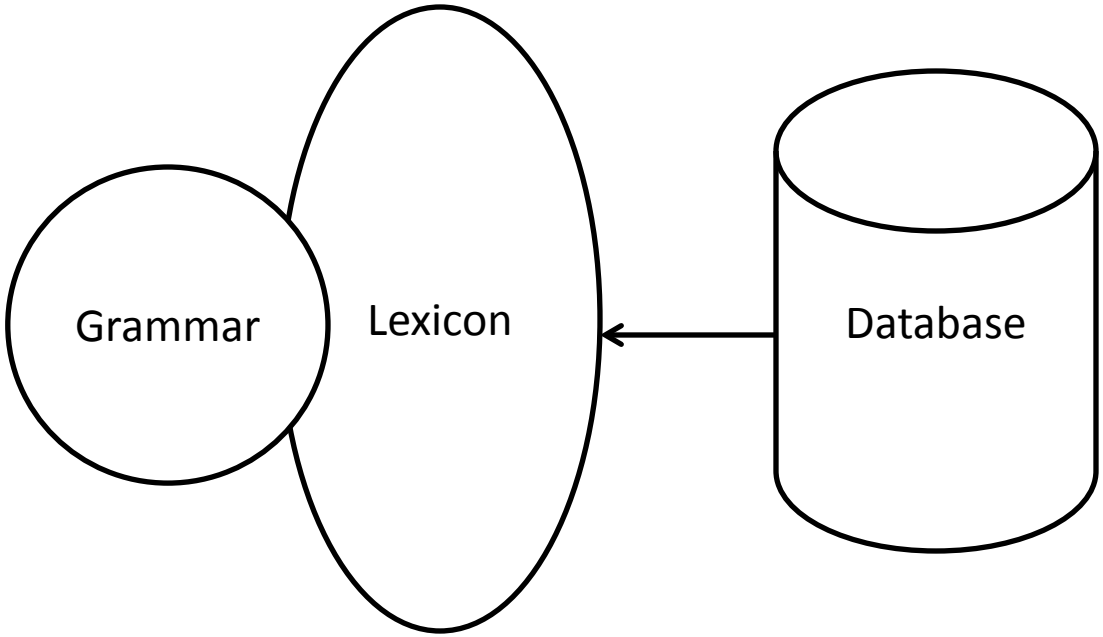
    User: work means Chalmers on Monday at 7:30

    User: home means Valand

  User: I want to go from work to home

  System: Take tram number 7 from Chalmers to Valand at 7:33

Objective



- To adapt the GF grammar of information-seeking dialogue systems

Grammatical Framework (GF)

- GF is a multilingual grammar formalism.
- Translate phrases between several languages.
- GF grammars are divided to two module types:
  - **Abstract module** – The ontology of a domain
  - **Concrete module** – Linearization of a abstract syntax in a particular language

Portable Grammar Format (PGF)

- PGFs are generated by compiling a set of concrete grammars with the same abstract syntax.
- PGF is a low-level binary format of GF grammars
- A PGF interpreter is needed to work with PGF files
  - performs parsing, linearization, random generation and type checking

The transport query system

Features:

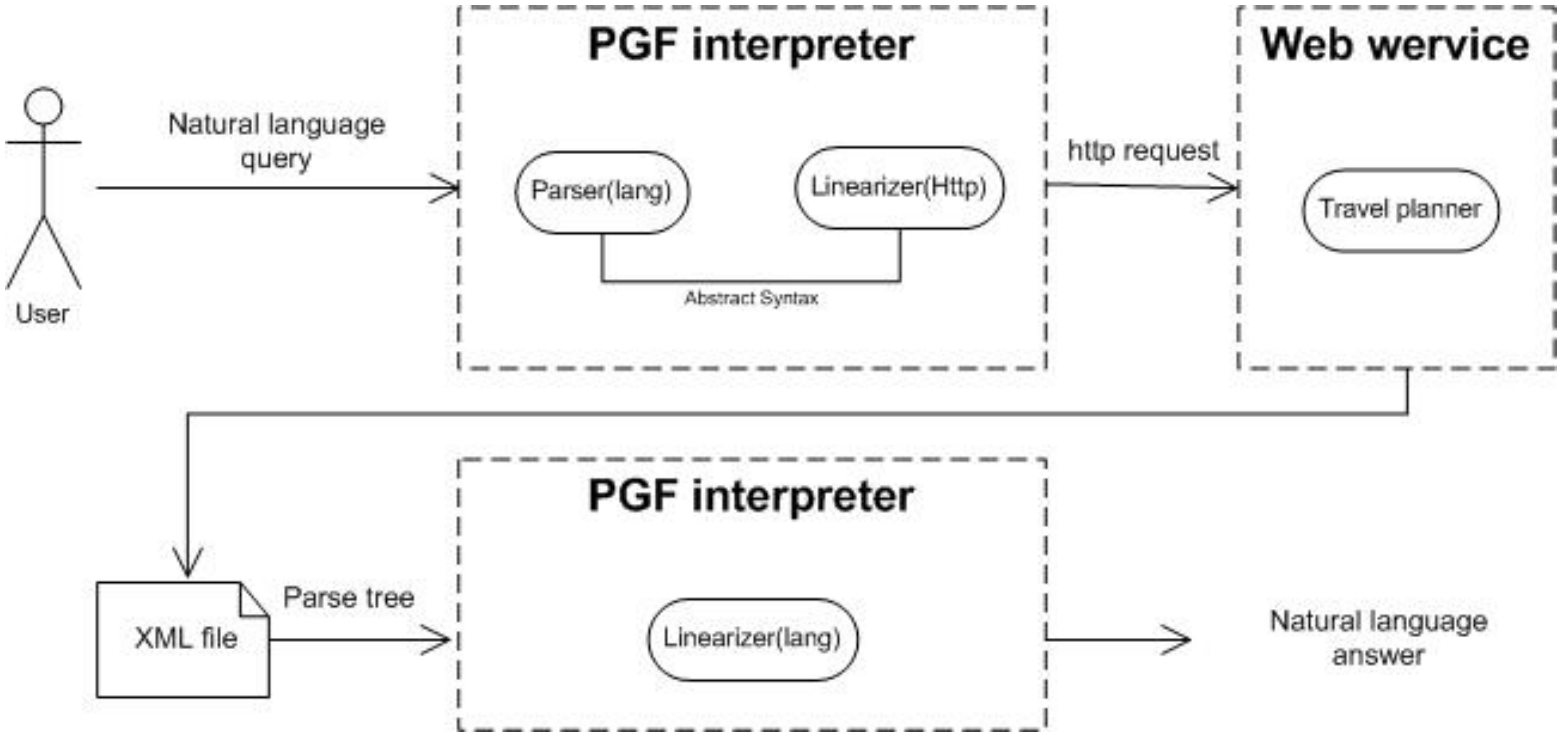
- Multilingual GF-based query system
  - English and Swedish languages
- Presents up-to-date travel plan
  - Communicates with a transport web service
- Can be used for other transport networks
  - Stop names are automatically added to the grammar by the *GF Writer* application

GF Writer Application

The embedded GF writer:

- Writes GF grammar rules during the execution of a program
- Generates or modifies abstract and concrete GF modules
- Compiles GF grammars and generates PGF files
  - Run GF software commands

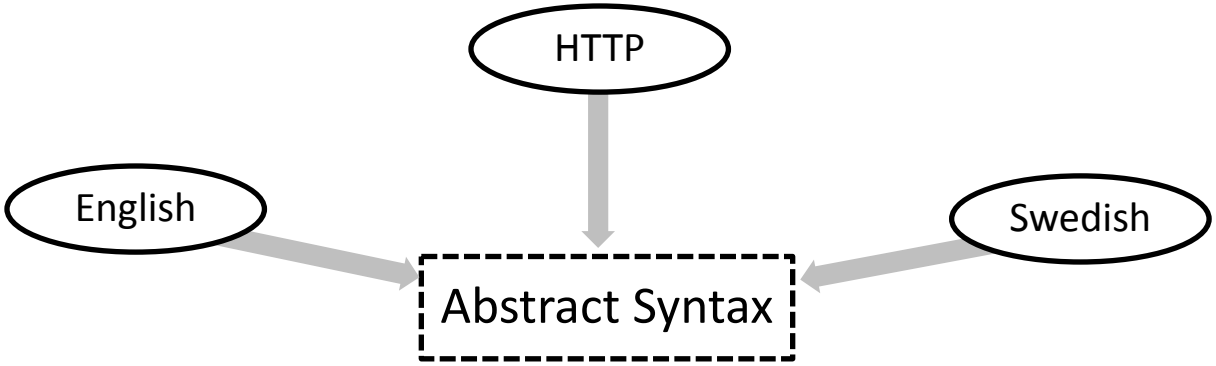
System Overview



Grammar overview

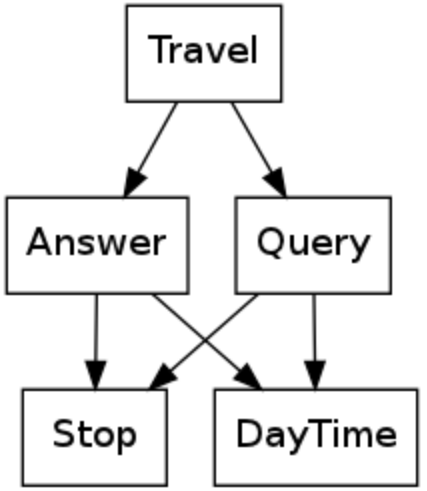
- Natural language grammars and HTTP grammar have the same abstract syntax

- Mapping between natural language queries and HTTP requests



Grammar Structure

- Separate modules for query and answer utterances
- Stop grammar holds stop names and identifiers
- Stop names can be offered to users



Stop Grammar

- Abstract Syntax

```
fun
  St_1 : Stop;
```

- English linearization

```
lin
  St_1 = mkStop " Valand "   "Göteborg"   "track A";
oper
   mkStop : Str -> Str -> Str -> TStop =
      \stop, region, track -> { s = stop; r = region; t = track; alt = stop ++ region};
```

- HTTP linearization

```
lin
  St_1 = {s = "9022014004420003"};
```

Query Grammar

- Abstract Syntax

```
fun
  GoFromTo : Stop -> Stop -> Day -> Time -> Query ;
```

- HTTP linearization

```
lin
  GoFromTo  from  to  day  time =
    {s = "date=“ ++ day.s ++
         "&time=" ++ time.s ++
         "&originId=" ++ from.s ++
         "&destId=" ++ to.s};
```

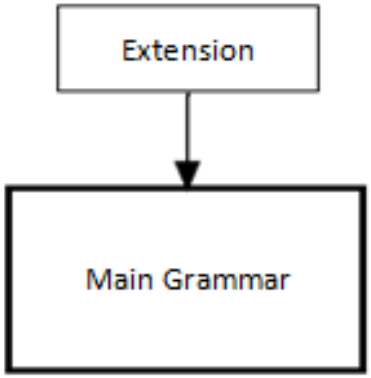
Adaptation Challenges in GF

- GF grammar adaptation can be costly in 2 aspects:
  1. Modifying GF modules
     - opening a GF module
     - Searching through rule
  2. Reproducing the PGF file

- Adaptation can be even worse when:
  - Changes need to be applied for several modules
  - GF modules are huge

Adaptation Technique

- Changes are applied to an extension grammar
- The Extenxion grammar extends all other grammars
- New grammar rules will be added gradually



Adaptation Technique

Grammar Compilation:

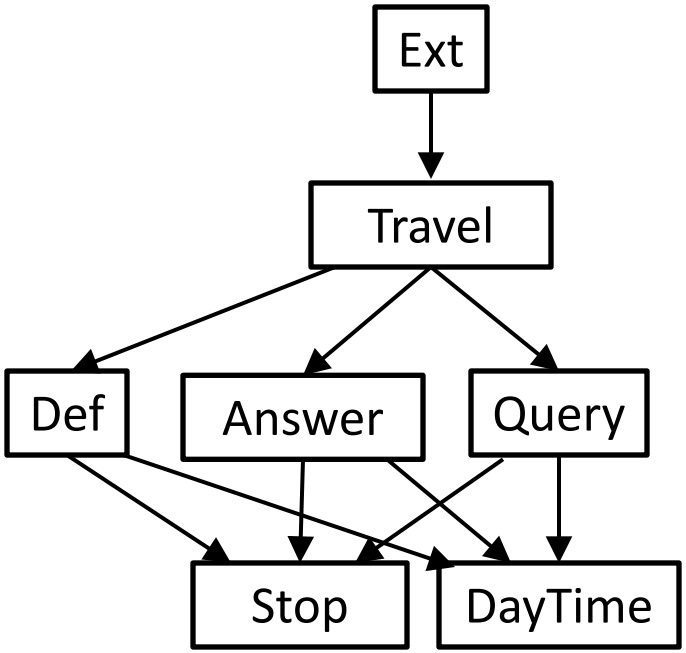
- GF compiles grammars separately

- The only compiled grammar is the Extension grammar
  - The huge main grammar is not compiled

- The compilation time is short

Adaptable Travel Grammar

- Convert the query system to an adaptive one



- The adaptation technique is used for:
  1. User adaptation
  2. Self-adaptation

1. User Adaptation

Stop, Day and Time Customization:

- User definition*: Work means Chalmers on Monday at 7:30*

- **Abstract syntax tree:**

```
(Customize
        ((((DefPlaceDayTime  Work)
                             St_2)
                             Monday)
                             ((HourMin (Num N7)) ((Nums N3) (Num N0)))))
```

1. User Adaptation

Stop, Day and Time Customization:

Abstract syntax:

```
abstract Ext = Travel ** {
  fun
    WorkStopDayTime : StopDayTime;
}
```

1. User Adaptation

Stop, Day and Time Customization:

English concrete syntax:

```
lin
  WorkStopDayTime = toStopDayTime   TravelEng.Work   TravelEng.St_2
                                    TravelEng.Monday    "7:30";

oper
  toStopDayTime:  {s : Str} ->  Stop ->  Day ->  Str ->  StopDayTime =
            \new, s, d, t -> {stop = s;   day= d;   time= t;   alt = new.s};
```

1. User Adaptation

Stop, Day and Time Customization:

HTTP concrete syntax:

```
lincat
  StopDayTime = {stop : Stop;  day : Day;  time : Str};

lin
  WorkStopDayTime = toStopDayTime   TravelEng.St_2
                                    TravelEng.Monday    "7:30";

oper
  toStopDayTime : Stop  ->  Day  ->  Str -> StopDayTime =
    \st, d, t -> { stop = st; day = d; time = t };
```

1. User Adaptation

Stop customization:

- User definition: *Home means Valand.*

Abstract syntax: No changes

Concrete syntax:

```
concrete ExtEng of Ext = TravelEng - [ St_1 ]  **  {
  lin
    St_1 = toStop   TravelEng.Home   TravelEng.St_1;
}

toStop :   {s : Str}  ->  Stop  ->  Stop = \new,  stop ->
  {s = stop.s;    r = stop.r;    t = stop.t;   alt  = stop.alt | new.s};
```

2. Self-adaptation

- Keep the system always updated
  - Adding new vehicle labels

Steps:

- Check the label is exists in the grammar
  - Try to parse the label

- If the parser does not succeeds
  - A new rule will be added

2. Self-adaptation

Abstract syntax:

```
abstract Ext = Travel ** {
  fun
    Lbl_new : Label;
. . . }
```

Concrete syntax:

```
concrete ExtEng of Ext = TravelEng-[ . . . ]** {
  lin
    Lbl_new = { s="Grön Express"};
. . . }
```

Adaptable Transport Query System

- Supported format for user's definitions:
  - *value* **means** *stop-name*
  - *value* **means** *stop-name day*
  - *value* **means** *stop-name day time*
  - *value* **means** *day*

- The changes are applied to all supported languages while adapting the system in one language.

Evaluation

The effect of user adaptation on speech recognition was assessed:

- 120 random generated queries were fed to an ordinary speech recognizer (Google)
  - All non-adapted queries were failed.
  - Most of the adapted queries were passed.

Evaluation (Non-adapted queries)

A failed non-adapted query:

- I want to go from **Skra Bro to Vrångö** on Tuesday at 13:12
- I want to go from **scratch the tools call** on Tuesday at 13:12

Speech recognizer behavior :

- Can find known words
- Cannot recognize foreign words by guessing alphabets
- Translate foreign words to known words

Evaluation (Adapted queries)

A failed adapted query:

- I want to go from **pub** to park on Friday at 15:35
- I want to go from **park** to park on Friday at 15:35

Speech recognition is improved by adaptation:

- Elimination of foreign words
- Shorter queries

Evaluation (Error rate)

- Word error rate:

  Rate of un-recognized words in a sentence

  – each query and the recognized one was checked word by word.

- Sentence error rate:

  Rate of failure in recognizing the whole sentence correctly

	Word error rate	**Sentence error rate**
Non-adapted queris	58%	100%
Adapted queries	26%	53%

Directions

- Extend the transport query system to a multimodal system (to support speech, text and map clicks).
- Use more efficient algorithm for searching through a GF module
- Apply the adaptation to various domains
  - Media players
  - Price comparison services