Parsing of Open Domain Text with GF

Krasimir Angelov

Chalmers University of Technology

August 24, 2011

Can we apply GF to open-domain text?

Current state *(application grammars)*

- Parsing for small controlled languages
- Language Generation from formal representation

Long-term goal

- Parsing with the **resource grammars**
- **Robustness** for out of coverage content
- Statistical **disambiguation**

Application Grammars

- domain with **constrained** language
- requirement for clear **semantics**
- mission critical **quality**
- **lightweight**

Resource Grammars

- free **unrestricted** language
- **no** need for **semantic** understanding
- small percentage of **errors** is permissible
- computationally **hard**

The Concrete Experiment

We evaluated the combination:

- English Resource Grammar
- Oxford Advanced Learners Dictionary *(adapted)*
- Simple Named Entities Recognizer

with sections 2–21 from PennTreebank.

Robustness

Try to read this:

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Robustness

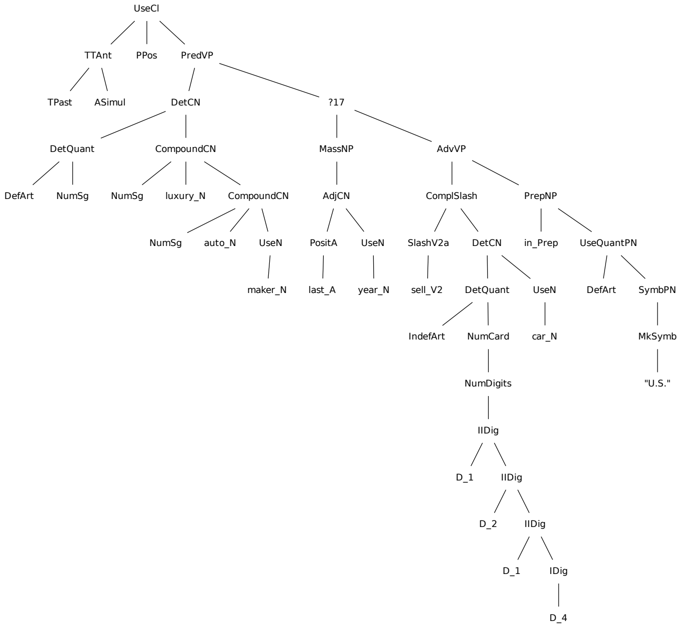
Try to read this:

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

The Standford Parser reads it as:

```
(ROOT
  (S
    (NP (JJ Lorem) (NN ipsum) (NN dolor))
    (VP (VBP sit)
      (NP (JJ amet) (, ,) (JJ consectetur)
          (NN adipiscing) (NN elit)))
    (. .)))
```

Robustness: Incomplete Abstract Trees in GF



1. Overview
2. Robustness
   - Low-Level API
   - Named Entity Recognizer
   - Chunking
3. Disambiguation
4. Penn Treebank for GF
5. Open-Domain Parsing by Cheating
6. Conclusion

Parse State Threading

$$initState :: PGF \to Language \to Type \to ParseState$$

$$nextState :: ParseState \to ParseInput \to Either\ ErrorState\ ParseState$$

$$\begin{aligned} getParseOutput :: \ & ParseState \to Type \to Maybe\ Int \\ & \to (ParseOutput, BracketedString) \end{aligned}$$

Reading the Input

$$\begin{aligned}
mkParseInput :: PGF &\rightarrow Language \\
&\rightarrow (\textbf{forall}\ \alpha\ .\ \ \beta \rightarrow Map\ \ Token\ \alpha \rightarrow Maybe\ \alpha) \\
&\rightarrow [(CId, \beta \rightarrow Maybe\ (Tree, [Token]))] \\
&\rightarrow (\beta \rightarrow ParseInput)
\end{aligned}$$

Design

- we cannot build Named Entity Recognizer directly in GF
- ... but we can complement GF grammars with custom code
  - The user defines some category as "literal"

    $ gf -make -literal=Symb PennTreebankEng.gf

  - and also provides a callback:

    [Token] −> Maybe (Tree,[Token])

  - The GF parser doesn't parse literal categories but delegates this to the callback

Rules

Currently very naïve rules:

> *Every sequence of tokens starting with captial letter is a candidate for name*

Full Source Code

```
parse :: PGF -> Language -> Type -> Maybe Int -> [Token] -> ParseOutput
parse pgf lang typ dp toks = loop (initState pgf lang typ) toks
  where
    loop ps []     = getParseOutput ps typ dp
    loop ps (t:ts) = case nextState ps (inputWithNames (t:ts)) of
                       Left  es -> []
                       Right ps -> loop ps ts

    inputWithNames = mkParseInput pgf lang
                                  tok
                                  [mkCId "String", name]
      where
        tok (t:ts) = Map.lookup (map toLower t)
        tok _      = Nothing

        name ts = let nts = takeWhile isNameTok ts
                  in if null nts
                       then Nothing
                       else Just (mkStr (unwords nts),nts)

        isNameTok (c:cs) | isUpper c = True
        isNameTok _                  = False
```

Idea

Simple:

- When we analize a sentence piece by piece and not as a whole, then the whole process is more robust.
- If a single piece (chunk) is not parseable the rest are still recognized.

For example: We can parse up to 75% of the basic noun phrases in PennTreebank but we can parse only few complete sentences.

Example

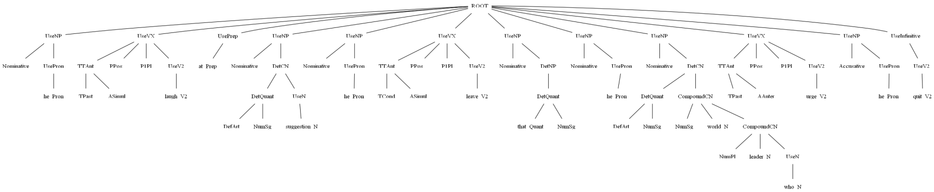
- We import parts of the English RGL + OALD
- ...but we introduce a new category *Chunk* instead of the category for sentences S.
- We add the rules:

$$\begin{aligned} \textbf{fun}\ & \mathit{UseNP} : \mathit{NP} \rightarrow \mathit{Chunk} \\ & \mathit{UseAP} : \mathit{AP} \rightarrow \mathit{Chunk} \\ & \mathit{UseVX} : \mathit{VX} \rightarrow \mathit{Chunk} \\ & \mathit{UsePrep} : \mathit{Prep} \rightarrow \mathit{Chunk} \end{aligned}$$

Parsing Strategy

- Call *initState* and start parsing from the beginning of the sentence.
- Consume as many tokens as possible by using *nextState*
- At the first failure, read the abstract syntax trees for the chunk that is recognized so far (*getParseOutput*).
- Continue again from the next token.

Example Output



Disadvantages

- Due to ambiguities we might miss the right chunking

  (works better when it is guided by some statistical chunker)

- Does not recover the structures that are far from the surface

- For disambiguation we might still need the global context

Disambiguation

The resource grammars can assign thousands of different analizes for a nontrivial sentence.

- Most alternative are missleading and useless
- Disambiguation is usually done statistically

Simple Probabilistic Model

- A configuration file assigns probability to every function:

AAnter	0.0074483421432004
ASimul	0.9255165785679962
AdVVP	0.0012515808814756
AdjCN	0.2070667478106594
AdvNP	0.1648705815470020
AdvS	0.0043645480559712
AdvVP	0.2808957930655976
AdvVPSlash	0.0008114836248423
BaseNP	0.6195076570774295
CompAP	0.4186262017838527
CompAdv	0.0099386076682497

- Compile the probabilities into the grammar:

```
> gf -make -probs=PennTreebank.probs PennTreebankCnc.gf
```

Dependency Model

- The simple model tends to get the PP attachement wrong
- State of the art parsers use head-driven dependency models

  (Not yet available in GF)
- Ranking should be integrated with the three extraction

  (Not yet available in GF)

Training Data?

- All statistical parsers rely on training data (treebank)
- Nothing is available for GF
- Building treebanks from scratch is costly

Grammar Evaluation

Success

- 85% single words
- 75% basic noun phrases
- 0% full sentences

Failure

- Incomplete patterns for Named Entities (ex: the United States)
- Syntax for dates?
- Missing constructions
- Missing words

Early Lessons

- Trying to directly parse a treebank is pointless (**failure**)
- Extending the grammar should not be the ultimate goal (**all grammars leak**)
- Even if you can parse a sentence this doesn't mean that you do it right (**false positives**)

Parsing vs Transformation

- Transformation Patterns

$$\frac{\left(\mathit{ADJP}\ \mathit{ad1}@\left(\mathit{RB}\ldots\right)\ \mathit{ad2}@\left(\mathit{RB}\ldots\right)\ \ldots\ \mathit{adj}@\left(\mathit{JJ}\ldots\right)\right)}{\mathit{AdAP}\ \mathit{ad1}\ \left(\mathit{AdAP}\ \mathit{ad2}\ldots\left(\mathit{PositA}\ \mathit{adj}\right)\ldots\right)}$$

- ... or as Haskell code:

```
grammar (mkApp meta)
   [ "ADJP":-> do adas <- many pAdA
                  adj  <- cat "JJ"
                  return (foldr (\ada ap -> mkApp cidAdAP [ada,ap])
                                (mkApp cidPositA [adj])
                                adas)
   , "JJ"  :-> ...
   ]

pAdA = do ada <- inside "RB" (lemma "AdA" "s")
          return (mkApp ada [])
```

Current status

- There is a script which can transform the whole treebank to GF for **few minutes**.
- We have recovered **92%** of tree nodes in the treebank
- Manual transformation can be as good as **97%**

Number of Guesses per Sentence

- Distribution:
  1416,4467,4623,4871,4561,4303,3763,3137,2501,
  1857,1353,952,646,483,332,200,116,89,54,41,20,
  22,7,2,4,5,0,3,2,1,0,0,0,0,0,1
- Average: 5

Summary of Grammar Extensions

- Very few syntax extensions
  (simple and easy things)
- A lot of changes in the lexicon
  - Structural words
  - Irregular verbs
  - Valency frames

Robust GF parser by cheating

- Take any statistical parser which produces Penn Treebank trees
- Use the parser to produce the Penn Treebank tree
- Convert the output to GF abstract syntax tree

Pros and Cons

Pros

- Simple and easy
- Already possible

Cons

- Doesn't utilize the GF infrastructure
- It is more interesting to have native GF parser
- The grammar is duplicated in the transformation rules.
- The Stanford Parser skip some annotations

Conclusion

This is only the beginning:

- The first draft import of Penn Treebank to GF
- Named entity recognizer in GF
- Preliminary experiments in robust parsing