Computational Linguistics II: Parsing

Unger's Parsing Method

Frank Richter & Jan-Philipp Söhn

fr@sfs.uni-tuebingen.de, jp.soehn@uni-tuebingen.de

November 29th, 2006

Unger's Parser

- top-down processing
- guesses how to split the input string into partitions that can be derived from a particular daughter
- all possible splits are tried
- assume: ϵ-free grammar
- example: rule: S $\rightarrow$ PP NP VP | NP VP | VP
  sentence: `In the Olympic Games, Greeks ran races, jumped, hurled the biscuits, and threw the java.`

Unger's Parser – Example

- S → VP: easy
  ⇒ VP → `In the Olympic Games, Greeks ran races, jumped, hurled the biscuits, and threw the java.`
- S → NP VP:

NP	VP
In	the Olympic Games, Greeks...
In the	Olympic Games, Greeks ran...
In the Olympic	Games, Greeks ran races...
In the Olympic Games,	Greeks ran races, jumped...
	...
In the Olympic...	java.

Unger's Parser – Example II

- S → PP NP VP:

PP	NP	VP
`In`	`the`	`Olympic Games,...`
`In`	`the Olympic`	`Games, Greeks...`
	...	
`In the`	`Olympic`	`Games, Greeks ran...`
`In the`	`Olympic Games,`	`Greeks ran...`
	...	
`In the Olympic...`	`the`	`java.`

- then try all rules and all partitions for *PP, NP, VP*
- each symbol needs to cover at least one word ⇒ the strings will always become shorter

Unger's Parser – Details

- can be executed depth-first or breadth-first
- immense number of comparisons: exponential time complexity
- possible optimization: discard splits for which terminals do not match:
  rule: NPK $\rightarrow$ NP `and` NP
  impossible split:
  {NP `many poems and`}{`and verse`}{NP `and also literature`}
- more optimizations: e.g. compute minimum number of terminals that derive from a non-terminal
  i.e. non-terminal: *VP*, minimal length for $VP = 3$, then discard all partitions of less than 3 words

Unger Algorithm – parallel

1. if $Z \in T$ and $Z = w_k$, finish
2. select rule $Z \rightarrow X_1 \ldots X_n$
3. split up sentence in n parts $w_1 \ldots w_n$ in all different ways
4. for all $k = 1$ to n: if $X_k \in T$ and $X_k \neq w_k$, discard split otherwise store split
5. select one split, for all parts Z repeat steps 1 – 4

Towards a Real Algorithm

- What knowledge needs to be preserved during the parse?
- What data structures do we need?
- What happens if a possibility turns out to be wrong?

Unger's Parser with ϵ Rules

- allow empty string as partition:
  rule: S $\rightarrow$ NP VP:

NP	VP
	`In the Olympic Games,...`
`In`	`the Olympic Games, Greeks...`
`In the`	`Olympic Games, Greeks ran...`
`In the Olympic`	`Games, Greeks ran races...`
`In the Olympic Games,`	`Greeks ran races, jumped...`
...	...
`In the Olympic...`	`java.`
`In the Olympic...`	

Unger's Parser with ϵ Rules II

- problem: loops
  rules: S $\rightarrow$ NP VP, and VP $\rightarrow$ V S
  sentence: `The Magna Carta provided that no free man should be hanged twice for the same offense.`
- problematic partition:

NP	VP
	`The Magna Carta provided that...`

	V	S
		`The Magna Carta provided...`

Unger's Parser with ϵ Rules III

Solution: check in decision history whether the same situation has occurred before

S $\Rightarrow$ `The Magna ... same offense.`

  NP $\Rightarrow$ ϵ; VP $\Rightarrow$ `The Magna ... same offense.`

    V $\Rightarrow$ ϵ; S $\Rightarrow$ `The Magna ... same offense.`

    cut off!

...

  NP $\Rightarrow$ `The`; VP $\Rightarrow$ `Magna ... same offense`

Example

Sentence:

`shit happens on the other side of the wormhole` (Trekkism, DS9)

Grammar:

S	$\rightarrow$	NP VP
NP	$\rightarrow$	N | DET N | DET ADJ N | NP PP
VP	$\rightarrow$	V PP
PP	$\rightarrow$	P NP
ADJ	$\rightarrow$	other
DET	$\rightarrow$	the
N	$\rightarrow$	shit | side | wormhole
P	$\rightarrow$	on | of
V	$\rightarrow$	happens