Human Communication I
Lecture 8

A language-controlled calculator

- Natural language is very complex
- Thus, building a natural language model is very difficult
- One way of dealing with this is to restrict the language domain
- We will look at a *language-controlled calculator*
  - structure and grammar rules (syntax)
  - how to represent meaning (semantics)
  - how to answer questions

A language-controlled calculator

- Task: building a language-controlled calculator
- English-language input and output
- Two problems (sub-tasks)
  - syntax and semantics of numbers
  - syntax and semantics of commands and questions

User: How much is three times four?

Program: Twelve

User: Multiply three by three.

Program: OK

User: Add to that four times four.

Program: OK

User: What's the square root of that?

Program: Five

User: Add three hundred and eighty five to fifteen thousand nine hundred eighteen.

Program: OK

User: How much is that?

Program: Sixteen thousand three hundred and three

User: What's twenty five divided into one hundred and twenty thousand?

Program: Forty eight hundred

Syntax & Semantics of numbers

Num	→	zero	0
Num	→	To99	To99
Num	→	To999	To999
To99	→	Digit	Digit
To99	→	Teen	Teen
To99	→	Tens	Tens
To99	→	Tens Digit	Tens + Digit
Digit	→	one	1
		⋮	
Digit	→	nine	9
Teen	→	ten	10

Teen	→	eleven	11
		⋮	
Teen	→	nineteen	19
Tens	→	twenty	20
		⋮	
Tens	→	ninety	90
To999	→	Hun	Hun
To999	→	Hun To99	Hun + To99
To999	→	Hun and To99	Hun + To99
Hun	→	a hundred	100
Hun	→	Digit hundred	Digit * 100

What we did

- Define rewrite rules for numbers < 1000
- Add expressions to each to give meaning to structure built by rule
- Meaning of structure built from meaning of substructures
- Non-terminal in meaning stands for meaning of subtree

Parsing & determining meaning

"three hundred and forty two"

Parse bottom-up; add meaning [] as we go

Rules to use:

Digit → three [3]

Tens → forty [40]

Digit → two [2]

three hundred and forty two

- Digit → three [3]

Digit[3] hundred and forty two

- Tens → forty [40]

Digit[3] hundred and Tens[40] two

- Digit → two [2]

Digit[3] hundred and Tens[40] Digit[2]

Digit[3] hundred and Tens[40] Digit[2]

- Hun → Digit hundred [Digit * 100]

Hun[3 * 100] and Tens[40] Digit[2]

- To99 → Tens Digit [Tens + Digit]

Hun[3 * 100] and To99[40 + 2]

- To999 → Hun and To99 [Hun + To99]

To999[[3 * 100] + [40 + 2]]

- Num → To999 [To999]

Num[342]

So, "three hundred and forty two" is a Num with the meaning [342]

Syntax & semantics of the calculator

- The calculator dialogues consist of two kinds of input
  - questions
  - commands.
- S → Imp [!] save the value of Imp
- S → Q [?] print meaning of Q and save it
- This gives the basic semantics for the system
  - the value of every computation is saved
  - in the case of questions, all of the form "How much is …" or something equivalent we print out the result as well.

Syntax & semantics of questions

- Syntax of questions is simple
  - Q → how much is NP [NP]
  - Q → what is NP [NP]
- For example
  - "how much is seven added to six?"
  - "what is four divided by two?"

Syntax & semantics of commands

Commands like "Multiply three by two" all involve prepositions, separated by a NP from the verb, but determined by it:

Imp	→	multiply NP by NP	$[NP_1 * NP_2]$
Imp	→	multiply NP by NP	$[NP_1 * NP_2]$
Imp	→	divide NP by NP	$[NP_1 / NP_2]$
Imp	→	divide NP into NP	$[NP_2 / NP_1]$
Imp	→	add NP to NP	$[NP_1 + NP_2]$
Imp	→	add NP and NP	$[NP_1 + NP_2]$
Imp	→	subtract NP from NP	$[NP_2 – NP_1]$

Note: for dividing, meaning depends on preposition.

Simple noun phrases

NP → that	the saved meaning of previous computation
NP → the result {of that}	the saved meaning of previous computation
NP → Num	Num

Complex NPs – "three added to four"

This class of NPs requires rules of the general form

Imp → V NP Prep NP, e.g. "add seven to three"

NP → NP V-ed Prep NP, e.g. "seven added to three"

Both have the same meaning; the NP rules are:

NP → NP multiplied by NP	$[NP_1 * NP_2]$
NP → NP divided by NP	$[NP_1 / NP_2]$
NP → NP divided into NP	$[NP_2 / NP_1]$
NP → NP added to NP	$[NP_1 + NP_2]$
NP → NP subtracted from NP	$[NP_2 - NP_1]$

The Imp rules are analogous

Complex NPs – "the result of dividing three into four"

This class of NPs requires rules of the general form

NP	→	the result of PartP, e.g. "the result of dividing three into four"
PartP	→	V-ing NP Prep NP

where the PartP rules have the same meaning

PartP	→	multiplying NP by NP	$[NP_1 * NP_2]$
PartP	→	multiplying NP and NP	$[NP_1 * NP_2]$
PartP	→	dividing NP by NP	$[NP_1 / NP_2]$
PartP	→	dividing NP into NP	$[NP_2 / NP_1]$
PartP	→	adding NP to NP	$[NP_1 + NP_2]$
PartP	→	adding NP and NP	$[NP_1 + NP_2]$
PartP	→	subtracting NP from NP	$[NP_2 - NP_1]$

Are you hoping we're done yet?

Two more NP classes

NPs like "three plus four"

NP	→	NP Op NP	[NP_1 Op NP_2]
Op	→	plus	[+]
Op	→	minus	[–]
Op	→	times	[*]
Op	→	over	[/]

NPs like "the sum of three and four"

NP	→	the Nop NP and NP	[NP_1 NOp NP_2]
NOp	→	sum of	[+]
NOp	→	difference between	[–]
NOp	→	quotient of	[/]
NOp	→	product of	[*]

Grammar coverage

- This gives a glimpse into the problems and complexities involved in creating a realistic grammar that "can do stuff".
- Covers: "What is thirty five divided by the result of multiplying the sum of two and two and the product of four over five and five?"

Grammar coverage

- This grammar will not quite handle the example dialogue
- We could extend the grammar to cover
  - numbers greater than 999
  - the use of square root (extension of one rule; similar for sine, cosine, logarithm, etc.)
  - cases like "Add to that four times four."
    (We'd have to add a rule
    Imp → V Prep that NP
    for every Imp rule of the form
    Imp → V NP Prep NP)

Grammar Coverage

- For an application, the question of a grammar's coverage is very important
  - What and how much linguistic input can the grammar handle, i.e. is covered by the grammar?
  - This determines range of the application, i.e. the cases in which it can be used