Grammar Engineering

Weiwei Sun

Institute of Computer Science and Technology
Peking University

May 22, 2019

Outline

Scientific theory

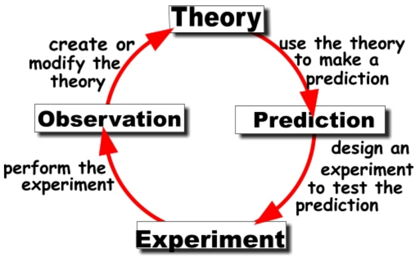
Wikipedia [https://en.wikipedia.org/wiki/Scientific_theory]

A scientific theory is a well-substantiated explanation of some aspect of the natural world that is acquired through the scientific method and repeatedly tested and confirmed through observation and experimentation.

A good scientific theory

- testable and make falsifiable predictions
- predictive power
- explanatory capability
- elegance and simplicity
- systematic

Scientific method


Theory
use the theory to make a prediction
Prediction
design an experiment to test the prediction
Experiment
perform the experiment
Observation
create or modify the theory

Linguistics is an empirical science

What we try to do is

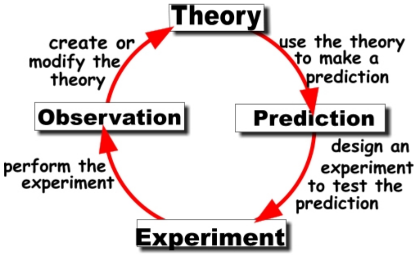
look at data and find the best theory that fits them,

until we find data that do not fit, at which point

we have to revise some of our conclusions, and so on.

As a theory about the natural world, our theory is, and should be, always under scrutiny and always developing.

Scientific method


Theory
use the theory to make a prediction
Prediction
design an experiment to test the prediction
Experiment
perform the experiment
Observation
create or modify the theory

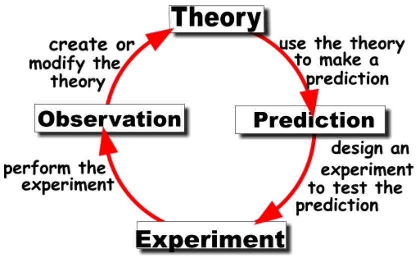
Linguistics is an empirical science

What we try to do is

> look at data and find the best theory that fits them,
>
> until we find data that do not fit, at which point
>
> we have to revise some of our conclusions, and so on.

As a theory about the natural world, our theory is, and should be, always under scrutiny and always developing.

Scientific method



Theory
use the theory to make a prediction
Prediction
design an experiment to test the prediction
Experiment
perform the experiment
Observation
create or modify the theory

Linguistics is an empirical science

What we try to do is

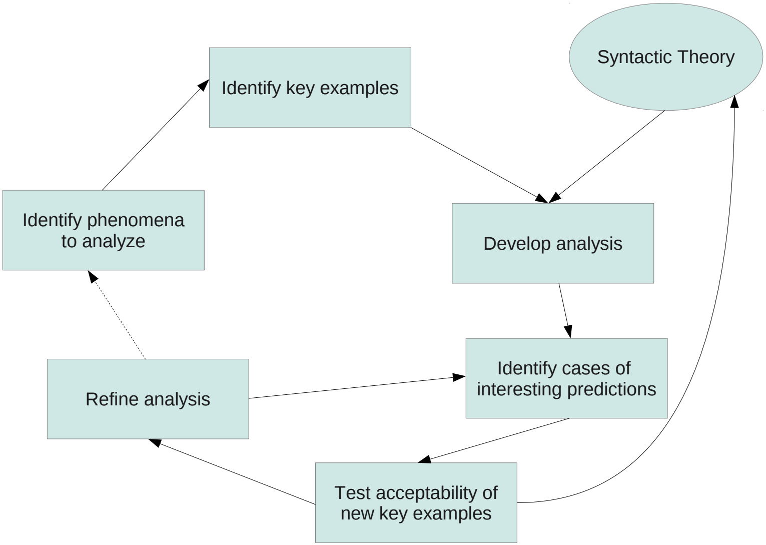
look at data and find the best theory that fits them,

until we find data that do not fit, at which point

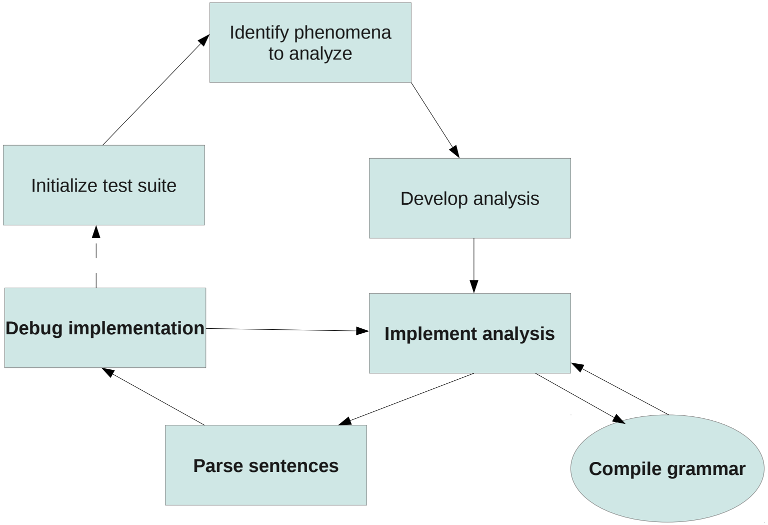
we have to revise some of our conclusions, and so on.

As a theory about the natural world, our theory is, and should be, always under scrutiny and always developing.

Pen and Paper Syntax Work-flow


Syntactic Theory
Identify key examples
Identify phenomena to analyze
Develop analysis
Identify cases of interesting predictions
Refine analysis
Test acceptability of new key examples

Grammar Engineering Work-flow



Grammar engineering

What we are going to do

- Develop understanding of (natural) language as a system of rules, i.e. a grammar fragment.
- Learn how to formalize grammars guided by HPSG and through typed feature structures.

Why computational grammars

Research	Formalize linguistic theories with complex interactions of language phenomena; advance theory building and implementation through synergy
Application	Embed grammar-based natural language analysis or generation in research prototypes and commercial applications.

The importance of developing a grammar fragment

otherwise it is extremely easy to think that you have a solution to a problem when in fact you don't.

B. Partee

Outline

Reference

The Linguistic Knowledge Builder

Copestake, Ann: *Implementing Typed Feature Structure Grammars*.

Deep Linguistic Processing with HPSG: DELPH-IN

`http://moin.delph-in.net/FrontPage`

- Loosely organized group of institutions and interested individuals;
- rooted in 'linguistic' NLP but geared towards practical applications;
- DELPH-IN resources are widely used: research, education, applications.

The LinGO English Resource Grammar

Development Background (1993–today)

- General-purpose, computational English grammar;
- mainly D. Flickinger, with R. Malouf, E. Bender, Jeff Smith;
- supported in multiple HPSG processing environments (LKB & PET);

Design

- HPSG: constraint-based, strongly lexicalized;
- MRS: flat, Davidsonian, underspecified;
- type hierarchies defining principles, lexical classes, constructions;
- strict grammaticality assumption: generator using same grammar.

The LinGO English Resource Grammar

- Demo
- On-line parser: `http://erg.delph-in.net`

- 7000 types in multiple-inheritance monotonic hierarchy
- 975 leaf lexical types
- 39,000 manually constructed lexemes
- 225 syntactic rules
- 70 morphological rules (inflection and derivation)
- Statistical parse selection model trained on 1.5 million word corpus

Practice

Add a new lexical entry

Open file *lexicon.tdl*. Try to add a lexical entry for verb *sleeps*. After saving your modifications, try to reload your grammar check for errors. When the grammar is loaded successfully, try to parse sentences like: *the cat sleeps*.

Outline

TDL

A Type Description Language for Constraint-Based Grammars

- [Krieger and Schäfer, 1994]
- Originally used in PAGE system
- Simplification and extensions in LKB

$\Rightarrow$ DELPH-IN reference formalism

- Fully compatible implementation in PET

TDL Syntax – Examples

Type inheritance:

```
feat-struc := *top*.
```

or

```
feat-struc :< *top*.
```

Type inheritance with attribute-value constraints:

```
agr-cat := gen-agr-cat &
[ PER per,
  NUM num,
  GEND gend ].
```

Multiple inheritance and coreference:

```
head-feat-principle := grule & head-dtr-type &
[ SYNSEM [ HEAD #head ],
  H-DTR [ SYNSEM [ HEAD #head ] ] ].
```

Lists

```
list :< *top*.
e-list :< list.
ne-list := list &
           [ FIRST *top*,
             REST list ].
```

Difference Lists

Allows more flexible list operation: concatenation, append, remove from end, ..., simply using unification.

```
*diff-list* := *top &
[ LIST *list*,
  LAST *list* ].
```

- LIST points to the beginning position
- LAST points to the end position

Abbreviations

```
<a,b,c>
[ FIRST a,
  REST [ FIRST b,
         REST [ FIRST c,
                REST e-list ] ] ]

<a,b,c,...>
[ FIRST a,
  REST [ FIRST b,
         REST [ FIRST c,
                REST list ] ] ]
<a.b>
[ FIRST a,
  REST b ]
```

Abbreviations

```
<!a,b,c!>
[ LIST [ FIRST a,
         REST [ FIRST b,
                REST [ FIRST c,
                       REST #last ] ] ],
  LAST #last ]
```

Difference lists allow concatenation by unification.

TFS Example (as AVM)

$$\left[\begin{array}{ll} \textit{phrase} & \\ \text{HEAD} & \textsf{verb} \\ \text{ARGS} & \left[\begin{array}{ll} \textit{*ne-list*} & \\ \text{FIRST} & \left[\begin{array}{ll} \textit{word} & \\ \text{ORTH} & \textsf{"chased"} \\ \text{HEAD} & \textsf{verb} \end{array}\right] \\ \text{REST} & \left[\begin{array}{ll} \textit{*ne-list*} & \\ \text{FIRST} & \left[\begin{array}{ll} \textit{expression} & \\ \text{HEAD} & \textsf{noun} \end{array}\right] \\ \text{REST} & \textsf{*null*} \end{array}\right] \end{array}\right] \end{array}\right]$$

TFS Example (in TDL)

```
vp := phrase &
[ HEAD verb,
  ARGS *ne-list* &
       [ FIRST word &
               [ ORTH "chased",
                 HEAD verb ],
         REST *ne-list* &
              [ FIRST expression &
                      [ HEAD noun ],
                REST *null* ]]] .
```

TFS Example (as AVM)

$$\begin{bmatrix} \textit{phrase} & \\ \text{HEAD} & \boxed{1}\,\textsf{verb} \\ \text{ARGS} & \begin{bmatrix} \textit{*ne-list*} & \\ \text{FIRST} & \begin{bmatrix} \textit{word} & \\ \text{ORTH} & \textsf{"chased"} \\ \text{HEAD} & \boxed{1} \end{bmatrix} \\ \text{REST} & \begin{bmatrix} \textit{*ne-list*} & \\ \text{FIRST} & \begin{bmatrix} \textit{expression} & \\ \text{HEAD} & \textsf{noun} \end{bmatrix} \\ \text{REST} & \textsf{*null*} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

TFS Example (in TDL)

```
vp := phrase &
[ HEAD #head \& verb,
  ARGS *ne-list* &
        [ FIRST word &
                [ ORTH "chased",
                  HEAD #head ],
          REST *ne-list* &
               [ FIRST expression &
                       [ HEAD noun ],
                 REST *null* ]]] .
```

Homework IV

Exercise I: Extending the grammar rules

You should have noticed that the grammar comes with only two rules that simulate the following CFG grammar:

- S $\rightarrow$ NP VP
- NP $\rightarrow$ DET N

Add a rule for verb phrases so that transitive verbs can be covered by the grammar. You are free to introduce extra features or make changes to types or lexicon if necessary.

Exercise II: Adding agreements

Introduce an AGR attribute to the type cat and modify the rules to garantee determiner-noun and subject-verb agreements.

Readings

- `http://moin.delph-in.net/HpsgTutorial`