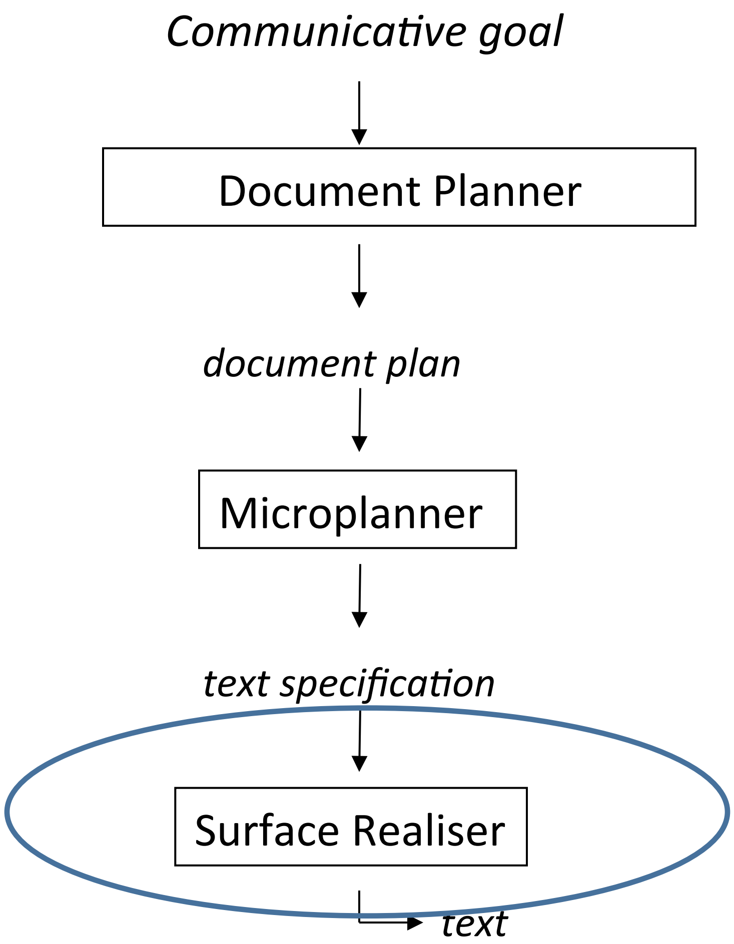
Realisation

Albert Gatt
Institute of Linguistics, University of Malta
http://staff.um.edu.mt/albert.gatt/
albert.gatt@um.edu.mt

The "consensus" architecture



By way of a characterisation

- Input:
  - A “sentence plan”
  - Unordered.
  - Uninflected.
- Task:
  - Map this to a syntactic structure
  - Apply morphological rules
  - Render as a string

Realisation = parsing in reverse?

- Parser:
  - A transducer from strings to structures.
  - Parsers model hypotheses.

- Realiser
  - A transducer from deep structures (semantic-syntactic) to strings.
  - Realisers model choices.

(e.g. Rajkumar & White 2014)

Preview

1. Overview of the realisation process
   – Choices involved

2. Types of realisers.

3. Statistical realisation in more detail
   – Overgeneration and ranking
   – The basic chart generation algorithm
   – Ranking via corpus data

4. Realisation engines: the case of SimpleNLG

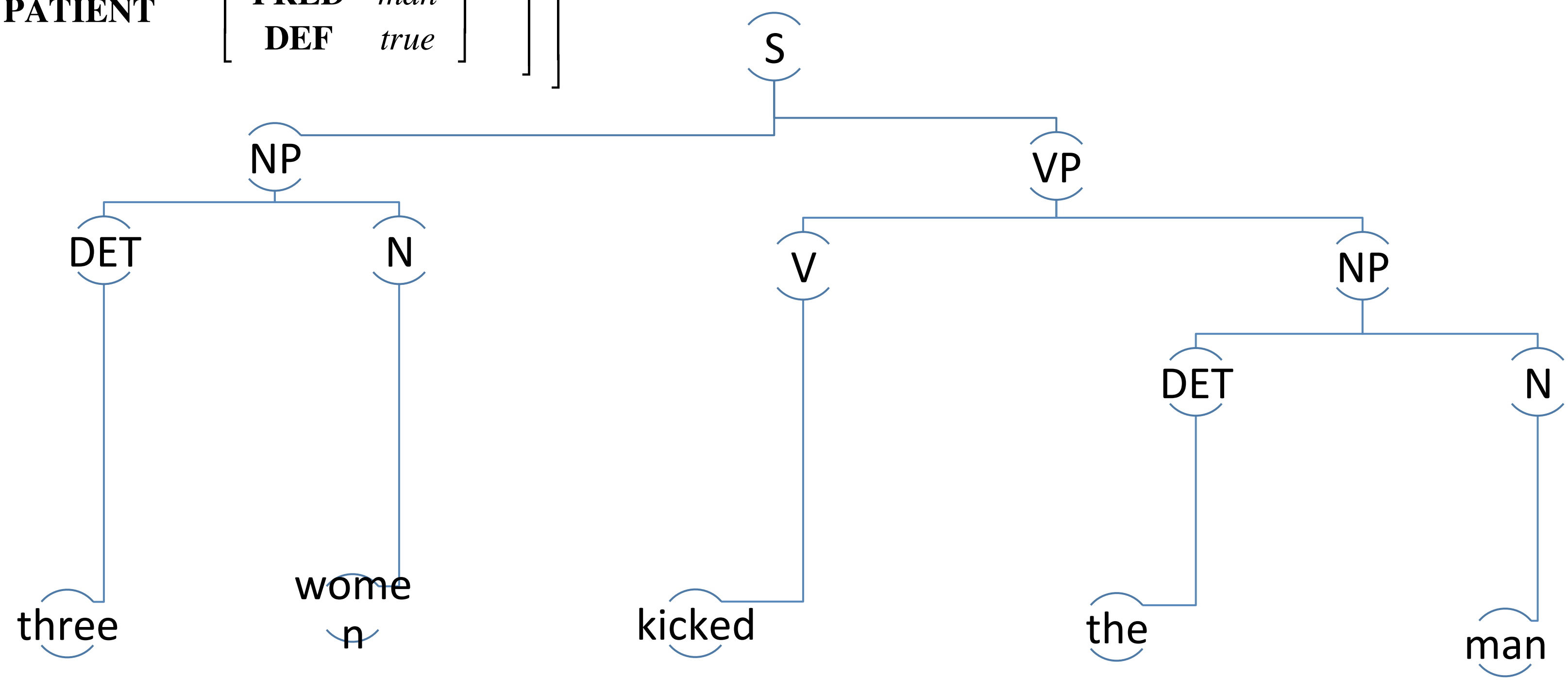
5. Evaluating realisers

Part 1

CHOICES IN REALISATION

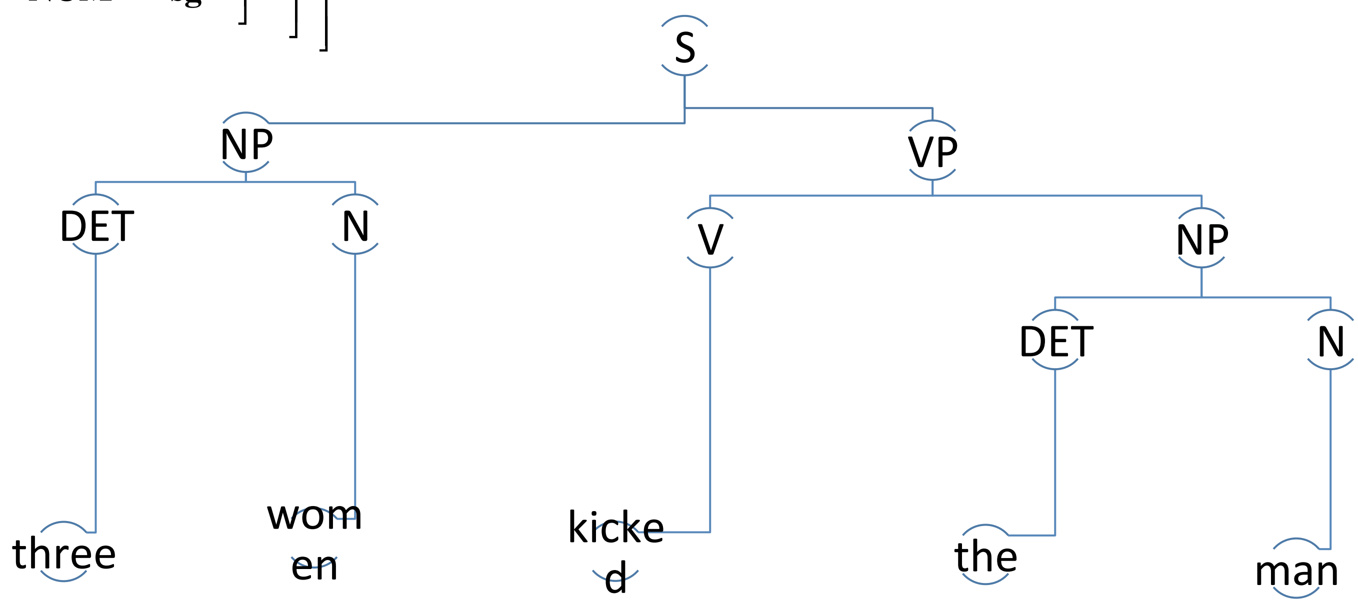
Input - Output

$$
\begin{bmatrix}
\textit{Event} & & \\
\textbf{TYPE} & \textit{declarative} & \\
\textbf{PRED} & \textit{kick} & \\
\textbf{TENSE} & \textit{past} & \\
\textbf{ARGS} & \begin{bmatrix} \textbf{AGENT} & \begin{bmatrix} \textbf{PRED} & \textit{woman} \\ \textbf{QUANT} & 3 \end{bmatrix} \\ \textbf{PATIENT} & \begin{bmatrix} \textbf{PRED} & \textit{man} \\ \textbf{DEF} & \textit{true} \end{bmatrix} \end{bmatrix} &
\end{bmatrix}
$$



Input – Output (Take 2)

$$
\begin{bmatrix}
\textit{Event} & \\
\textbf{TYPE} & \textit{declarative} \\
\textbf{VERB} & \textit{kick} \\
\textbf{TENSE} & \textit{past} \\
\textbf{ARGS} & \begin{bmatrix}
\textbf{SUBJECT} & \begin{bmatrix} \textbf{NOUN} & \textit{woman} \\ \textbf{DET} & 3 \\ \textbf{NUM} & \textbf{pl} \end{bmatrix} \\
\textbf{OBJECT} & \begin{bmatrix} \textbf{NOUN} & \textit{man} \\ \textbf{DET} & \textit{the} \\ \textbf{NUM} & \textbf{sg} \end{bmatrix}
\end{bmatrix}
\end{bmatrix}
$$



How specific is the input?

$$
\left[\begin{array}{ll}
\multicolumn{2}{l}{\textit{Event}} \\
\textbf{TYPE} & \textit{declarative} \\
\textbf{VERB} & \textit{kick} \\
\textbf{TENSE} & \textit{past} \\
\textbf{ARGS} & \left[\begin{array}{ll}
\textbf{SUBJECT} & \left[\begin{array}{ll} \textbf{NOUN} & \textit{woman} \\ \textbf{DET} & 3 \\ \textbf{NUM} & \textbf{pl} \end{array}\right] \\
\textbf{OBJECT} & \left[\begin{array}{ll} \textbf{NOUN} & \textit{man} \\ \textbf{DET} & \textit{the} \\ \textbf{NUM} & \textbf{sg} \end{array}\right]
\end{array}\right]
\end{array}\right]
\qquad
\left[\begin{array}{ll}
\multicolumn{2}{l}{\textit{Event}} \\
\textbf{TYPE} & \textit{declarative} \\
\textbf{PRED} & \textit{kick} \\
\textbf{TENSE} & \textit{past} \\
\textbf{ARGS} & \left[\begin{array}{ll}
\textbf{AGENT} & \left[\begin{array}{ll} \textbf{PRED} & \textit{woman} \\ \textbf{QUANT} & 3 \end{array}\right] \\
\textbf{PATIENT} & \left[\begin{array}{ll} \textbf{PRED} & \textit{man} \\ \textbf{DEF} & \textit{true} \end{array}\right]
\end{array}\right]
\end{array}\right]
$$

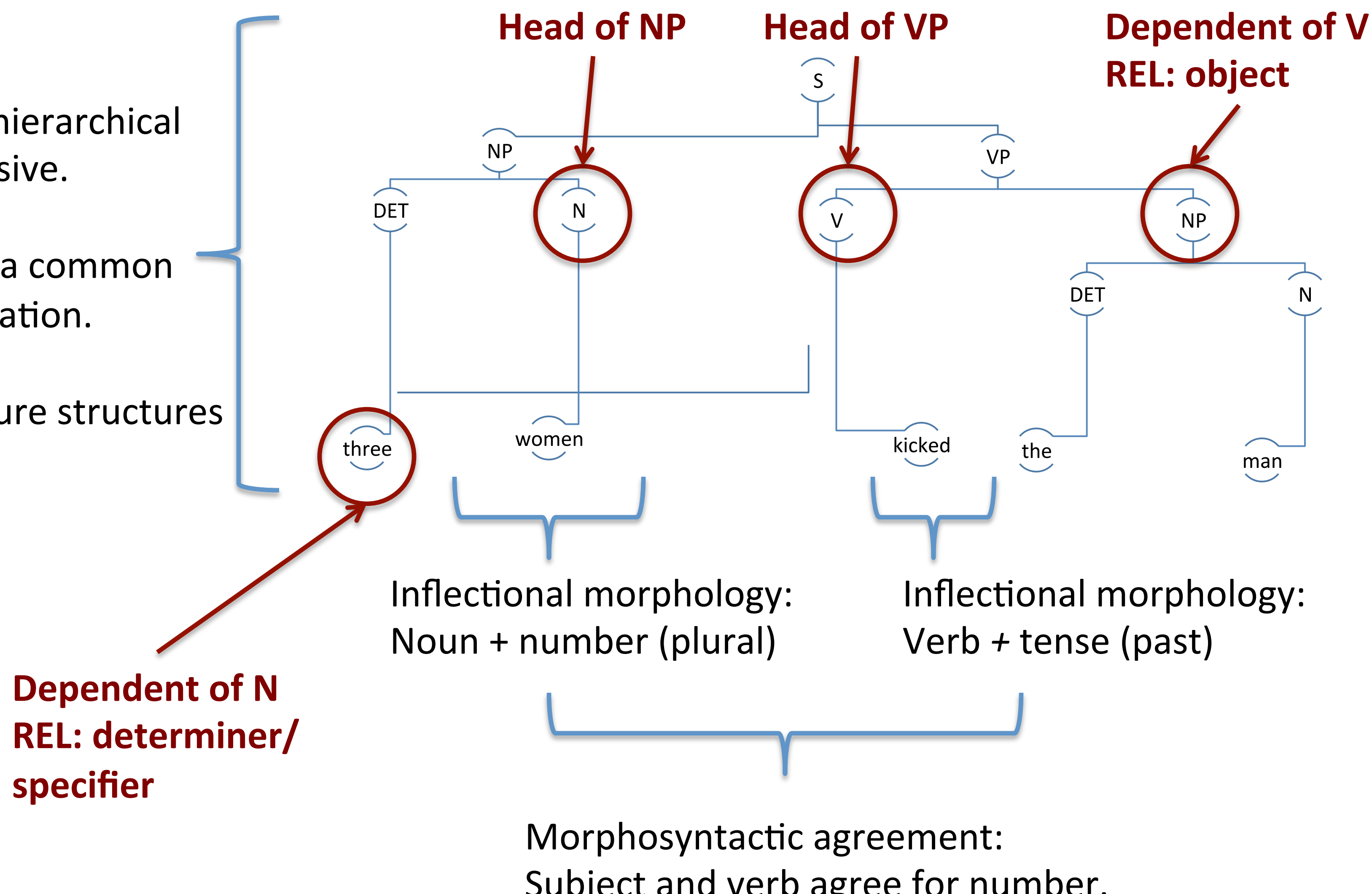
- Input differs wrt:
  - Specification of argument roles (functional, semantic).
  - Specification of function words (e.g. DET)
  - Specification of morphological features (e.g. NUM)

Constituency-based representation

Syntax is hierarchical and recursive.

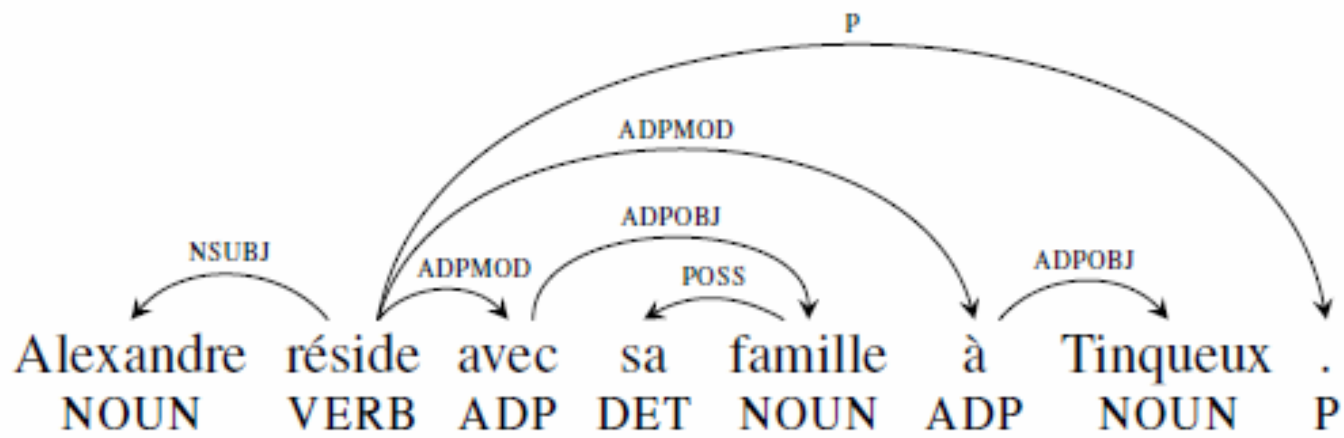
Trees are a common representation.

Also, feature structures (graphs).



Dependency-based representation

- Example French sentence (from the Universal Dependency Treebank; McDonald et al 2013):



Knowledge sources

- Lexical knowledge:
  - NB: Not all systems have a separate lexicon!
  - Words + morphological rules
  - Crucially: **exceptions to the rules**
  - English:
    - *be* → was, were
    - *eat* → *ate, ate*

Many verbs in English do not change in the plural. This is the exception, not the rule.

Knowledge sources

- Syntactic knowledge:
  - Knowledge of the right constituent order.
    - English:
      - S → $NP_{[subj]}$ VP
      - VP → V $NP_{[obj]}$
      - *Three women kicked the man*
  - But several options become possible if, e.g., pragmatic factors come into play:
    - *It was three women who kicked the man*. (it-cleft)
    - *The man was kicked by three women.* (passive)

Not all rules are easy to state

- Modifier ordering in English (cf. Maalouf 2002; Mitchell 2009):
  - *the large green ball*
  - *?the green large ball*
- Example from Penn Treebank (after Callaway 2005):
  - *a $100 million Oregon general obligation veterans' tax note issue.*

Not all rules are easy to state

- Adverbial modifiers apparently can be placed anywhere:
  - *For now, I'll wait and see.*
  - *I'll wait and see for now.*
  - *I'll wait for now and see.*
- But (ex cited by Rajkumar & White 2014):
  - *Separately, the Federal Energy Regulatory Commission turned down* ***for now*** *a request by Northeast seeking approval of its possible purchase of PS of New Hampshire. (WSJ0013.16).*
  - *Separately, the Federal Energy Regulatory Commission turned down a request by Northeast seeking approval of its possible purchase of PS of New Hampshire* ***for now****. (WSJ0013.16).*

Not all rules are easy to state

- When should "that" be used? Is it always optional?
  - *He said he'd go.*
  - *He said **that** he'd go.*

- But consider (cf Rajkumar & White 2014):
  - *He [said **that** [for the second month in a row, food processors reported a shortage of non- fat dry milk]]. (WSJ0036.61)*
  - *He [[said for the second month in a row], food processors reported a shortage of nonfat dry milk].*

The problem of (unintended) ambiguity

- A choice can determine whether the output is ambiguous or not.
  - We saw this in the case of “that”.

- Modifier and determiner repetition:
  - *He shot the young lions and horses.*
  - *He shot the young lions and the horses.*
  - *He escorted the old men and horses.*
  - (Cf. Khan et al 2012, for experimental work)

By way of a revised characterisation

- Input:
  - A “sentence plan”
  - Unordered.
  - Uninflected.
  - **Possibly including pragmatic and other info**.
    - But this depends on the input specification.

- Task:
  - Map this to a syntactic structure **which communicates the info and conveys the pragmatic intentions.**
    - Ideally, **avoid ambiguity** in the process.
    - Ideally, **respect linearisation rules**.
  - Apply morphological rules
  - Render as a string

Part 2

A TYPOLOGY OF REALISERS

Grammar-based realisers

- Include explicit grammatical rules.
  - Rules are hand-written or extracted automatically from parsed corpora (treebanks).
  - Represent linguistic choices as choices between rules (also locally).
  - Often, choose among alternatives on a "global" level: which is the best sentence to realise a given input?
  - Typically use a chart parsing algorithm (on which, more later).

Grammar-based realisers

1. Symbolic
   - May entertain multiple hypotheses.
   - Output represents the “best” choice based on the rule-base and the algorithm.
2. Overgenerate-and-rank
   - Generate multiple outputs.
   - Rank these outputs using a statistical model.
   - “Best choice” = the one ranked highest by the model, given its features.

Rules for Grammar-Based Realisers

- Rules may come from:
  1. Hard labour (i.e. hand-coding)
  2. Extraction from parsed corpora (treebanks)
     - E.g. Hockenmaier & Steedman (2007): conversion of the Penn Treebank into structures based on Combinatory Categorial Grammar.
       – Used with the OpenCCG Realiser.
     - E.g. Callaway (2003): conversion of the Penn Treebank into inputs for FUF/SURGE.

Extracting input from treebank

- Penn treebank input (Callaway 2003):

```
(S (PP (IN Without)                                (
       (NP (NNP GM)))
   (, ,)
   (NP-SBJ
       (NP (JJ overall) (NNS sales))
       (PP (IN for)
           (NP (DT the) (JJ other)
               (NNP U.S.) (NNS automakers))))
   (VP (VBD were)
       (ADJP-PRD (RB roughly) (JJ flat)
           (PP (IN with)
               (NP (CD 1989) (NNS results ")))))))
```

Extracting input from treebank

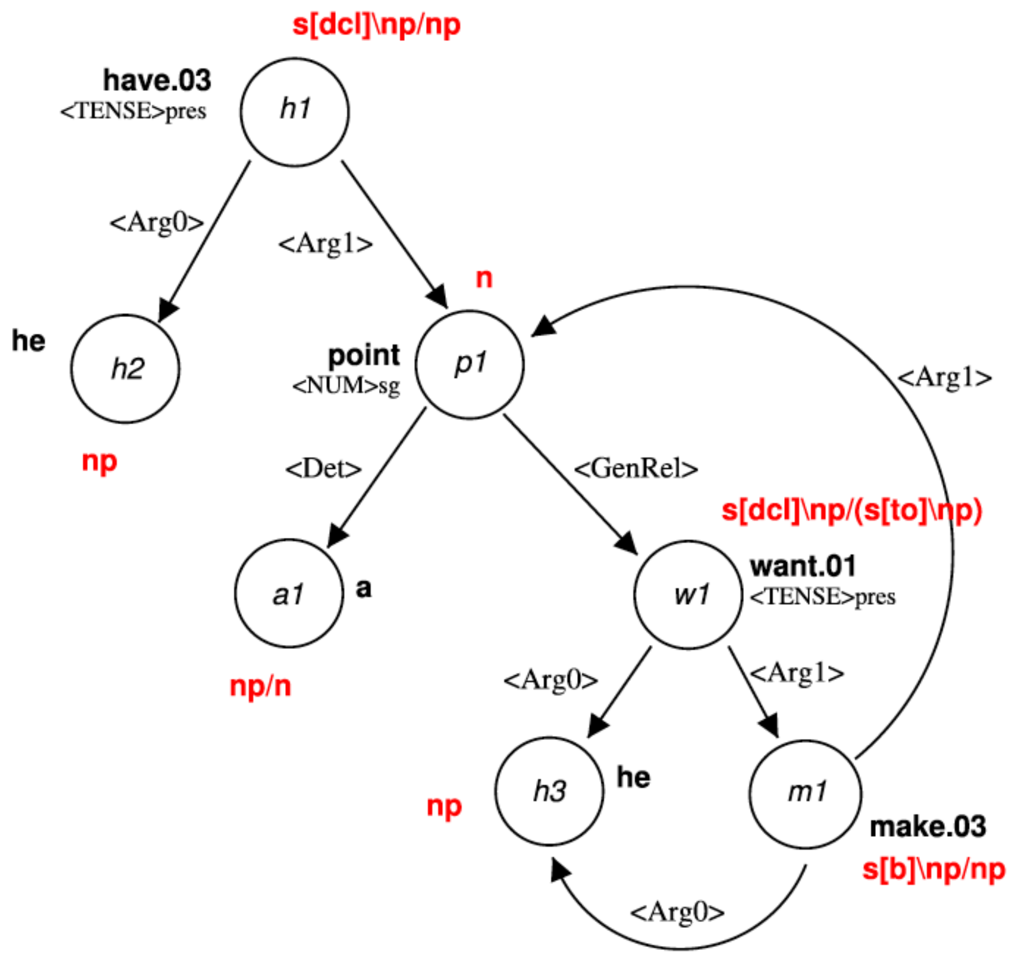
- Conversion into feature structure for FUF/SURGE (Callaway, 2003)
- Note: this input is **unordered.**

```
((cat clause)
 (circum ((accompaniment ((cat pp) (position front) (accomp-polarity -)
                          (np ((cat proper) (lex "GM")))))))
 (process ((type ascriptive) (tense past)))
 (participants ((carrier ((cat common) (lex "sale") (number plural)
                          (describer ((cat adj) (lex "overall"))))
                          (qualifier ((cat pp) (prep ((lex "for")))
                                      (np ((cat common) (lex "automaker") (definite yes)
                                           (number plural) (status different)
                                           (classifier ((cat proper) (lex "U.S."))))))))
                (attribute ((cat ap) (lex "flat") (modifier ((cat adv) (lex "roughly")))
)                           (qualifier ((cat pp) (prep ((lex "with")))
                                        (np ((cat common) (lex "result") (number plural)
                                             (classifier ((cat date) (year 1989))))))))))
```

NB: Aim here is to use the input from the Penn Treebank to regenerate the original sentence.
NOT to induce a grammar for FUF/SURGE (which is hand-coded).

Compare to input for OpenCCG

He has a point he wants to make.
(Rajkumar & White 2013)



This input is also derived from the Penn Treebank.
Rules for OpenCCG can also be induced from the treebank.

Grammar-based, hand-crafted

- Grammar-based, using hand-crafted rules:
  - FUF/SURGE (Elhadad & Robin, 1996)
    - Based on Functional Unification Grammar.
    - Models choices using unification of feature structures.
    - Among the best coverage and BLEU score (Callaway, 2005).
  - KPML (Bateman, 1997)
    - Based on Systemic-Functional Grammar.
    - Models choices as paths through a graph (systemic network).
    - Systemic networks include a combination of grammatical and pragmatically-motivated choice points.

Grammar-based + reranking

- HALOGEN (Langkilde 2000, 2002)
  - Earliest example of ranking model for NLG.
  - Grammar is relatively "theory-neutral", relatively few rules.
  - Ranking based on n-gram models.
- OpenCCG (White et al 2007)
  - Based on Combinatory Categorial Grammar
  - Grammar induced from the Penn Treebank (Hockenmaier & Steedman 2007).
  - Ranking (in current version) based on a variety of features, not just n-grams.

Classification-based approaches

- Use multiple classifiers (chained) to make decisions at a local level.
- Example: Fillipova & Strube (2009):
  - Input: Constituents (subject, object, adverb…)

1. Use a classifier to identify the first constituent.
2. Use a second classifier to sort the remainder.
3. Insert verb after the subject (wherever it happens to be).

Things to note

- How detailed the input is…
- What the input includes (semantics, morpho-syntax, pragmatic info)…

- … all depends on your theory of grammar.

Part 3

AN OVERVIEW OF THE BASIC ALGORITHM

Many ways to say the same thing

- Example input: ORDER(eat(you,chicken))
  - → Eat chicken!
  - → It is required that you eat chicken!
  - → It is required that you eat poulet!
  - → Poulet should be eaten by you.
  - → You should eat chicken/chickens.
  - → Chicken/Chickens should be eaten by you.

- Of course, this is assuming we have a non-trivial grammar that supports multiple choices.

Chart Generation

- Historically, chart algorithms were developed for parsing.
  - Proposed as solutions for realisation by Shieber (1988) and Kay (1996).
  - Exposition here follows Kay (1996).

The basic idea

- The algorithm schema requires two main components:
  - **Agenda**: holds bits of input, in some order
  - **Chart**: holds (partial) outputs
  - **Algorithm**:
    - Removes items from the Agenda and puts them on the Chart
    - Merges items on the chart
    - Places new items back on the Agenda
  - Major pro:
    - Allows us to entertain multiple realisation hypotheses while minimising extra work.

Example (Kay 1996)

A very simple input semantics (very flat!):

- r: run(r), past(r), fast(r), arg1(r,j), name(j, John)
- “There’s a run event whose agent is John and the event was fast.”
- Our ‘r’ and ‘j’ are constants.

A very simple lexicon:

Word	Category	Semantics
John	np(x)	x: name(x, John)
ran	vp(x,y)	x: run(x), argl (x, y), past(x)
fast	adv(x)	x: fast(x)
quickly	adv(x)	x: fast(x)

Example (Kay 1996)

A very simple set of grammar rules:

- s(x) → np(y), vp(x,y)
- vp(x) → vp(x) adv(x)

r: run(r), past(r), fast(r), arg1(r,j), name(j, John)

s(x) → np(y), vp(x,y)
vp(x) → vp(x) adv(x)

AGENDA

Word	Category	Semantics
John	np(x)	x: name(x, John)
ran	vp(x,y)	x: run(x), argl (x, y), past(x)
fast	adv(x)	x: fast(x)
quickly	adv(x)	x: fast(x)

→

CHART

Word	Category	Semantics

r: run(r), past(r), fast(r), arg1(r,j), name(j, John)

s(x) → np(y), vp(x,y)
vp(x) → vp(x) adv(x)

AGENDA

Word	Category	Semantics
ran	vp(x,y)	x: run(x), argl (x, y), past(x)
fast	adv(x)	x: fast(x)
quickly	adv(x)	x: fast(x)

→

CHART

Word	Category	Semantics
John	np(x)	x: name(x, John)

r: run(r), past(r), fast(r), arg1(r,j), name(j, John)

s(x) → np(y), vp(x,y)
vp(x) → vp(x) adv(x)

AGENDA

Word	Category	Semantics
fast	adv(x)	x: fast(x)
quickly	adv(x)	x: fast(x)

CHART

Word	Category	Semantics
John	np(x)	x: name(x, John)
ran	vp(x,y)	x: run(x), argl (x, y), past(x)

Can merge these on the basis of the s(x) rule.
John ran

r: run(r), past(r), fast(r), arg1(r,j), name(j, John)

s(x) → np(y), vp(x,y)

vp(x) → vp(x) adv(x)

AGENDA

Word	Category	Semantics
fast	adv(x)	x: fast(x)
quickly	adv(x)	x: fast(x)
John ran	S(r)	r:run(r), past(r), arg1(r,j), name(j, john)

CHART

Word	Category	Semantics
John	np(x)	x: name(x, John)
ran	vp(x,y)	x: run(x), argl (x, y), past(x)

Not done yet! Haven't covered all of input semantics.

r: run(r), past(r), fast(r), arg1(r,j), name(j, John)

s(x) → np(y), vp(x,y)
vp(x) → vp(x) adv(x)

AGENDA

Word	Category	Semantics
quickly	adv(x)	x: fast(x)
John ran	S(r)	r:run(r), past(r), arg1(r,j), name(j, john)

CHART

Word	Category	Semantics
John	np(x)	x: name(x, John)
ran	vp(x,y)	x: run(x), argl (x, y), past(x)
fast	adv(x)	x: fast(x)

Can merge on the basis of the vp(x) rule.
ran fast

r: run(r), past(r), fast(r), arg1(r,j), name(j, John)

s(x) → np(y), vp(x,y)

vp(x) → vp(x) adv(x)

AGENDA

Word	Category	Semantics
quickly	adv(x)	x: fast(x)
John ran	S(r)	r:run(r), past(r), arg1(r,j), name(j, john)
ran fast	vp(x)	r: run(r), past(r), fast(r), argl(r,j)

→

CHART

Word	Category	Semantics
John	np(x)	x: name(x, John)
ran	vp(x,y)	x: run(x), argl (x, y), past(x)
fast	adv(x)	x: fast(x)

r: run(r), past(r), fast(r), arg1(r,j), name(j, John)

s(x) → np(y), vp(x,y)
vp(x) → vp(x) adv(x)

AGENDA

Word	Category	Semantics
John ran	S(r)	r:run(r), past(r), arg1(r,j), name(j, john)
ran fast	vp(x)	r: run(r), past(r), fast(r), argl(r,j)

CHART

Word	Category	Semantics
John	np(x)	x: name(x, John)
ran	vp(x,y)	x: run(x), argl (x, y), past(x)
fast	adv(x)	x: fast(x)
quickly	adv(x)	x: fast(x)

Can merge on the basis of the vp(x) rule.
ran quickly

r: run(r), past(r), fast(r), arg1(r,j), name(j, John)

s(x) → np(y), vp(x,y)
vp(x) → vp(x) adv(x)

AGENDA

Word	Category	Semantics
John ran	S(r)	r:run(r), past(r), arg1(r,j), name(j, john)
ran fast	vp(x)	r: run(r), past(r), fast(r), argl(r,j)
ran quickly	Vp(x)	r: run(r), past(r), fast(r), argl(r,j)

CHART

Word	Category	Semantics
John	np(x)	x: name(x, John)
ran	vp(x,y)	x: run(x), argl (x, y), past(x)
fast	adv(x)	x: fast(x)
quickly	adv(x)	x: fast(x)

r: run(r), past(r), fast(r), arg1(r,j), name(j, John)

s(x) $\rightarrow$ np(y), vp(x,y)
vp(x) $\rightarrow$ vp(x) adv(x)

AGENDA

Word	Category	Semantics
ran fast	vp(x)	r: run(r), past(r), fast(r), argl(r,j)
ran quickly	Vp(x)	r: run(r), past(r), fast(r), argl(r,j)

CHART

Word	Category	Semantics
John	np(x)	x: name(x, John)
ran	vp(x,y)	x: run(x), argl (x, y), past(x)
fast	adv(x)	x: fast(x)
quickly	adv(x)	x: fast(x)
John ran	S(r)	r:run(r), past(r), arg1(r,j), name(j, john)

r: run(r), past(r), fast(r), arg1(r,j), name(j, John)

s(x) → np(y), vp(x,y)
vp(x) → vp(x) adv(x)

AGENDA

Word	Category	Semantics
ran quickly	Vp(x)	r: run(r), past(r), fast(r), argl(r,j)

Can merge these on the basis of the s(x) rule.
John ran fast

CHART

Word	Category	Semantics
John	np(x)	x: name(x, John)
ran	vp(x,y)	x: run(x), argl (x, y), past(x)
fast	adv(x)	x: fast(x)
quickly	adv(x)	x: fast(x)
John ran	S(r)	r:run(r), past(r), arg1(r,j), name(j, john)
ran fast	vp(x)	r: run(r), past(r), fast(r), argl(r,j)

r: run(r), past(r), fast(r), arg1(r,j), name(j, John)

s(x) → np(y), vp(x,y)
vp(x) → vp(x) adv(x)

AGENDA

Word	Category	Semantics
ran quickly	Vp(x)	r: run(r), past(r), fast(r), argl(r,j)
John ran fast	S(x)	Complete!

CHART

Word	Category	Semantics
John	np(x)	x: name(x, John)
ran	vp(x,y)	x: run(x), argl (x, y), past(x)
fast	adv(x)	x: fast(x)
quickly	adv(x)	x: fast(x)
John ran	S(r)	r:run(r), past(r), arg1(r,j), name(j, john)
ran fast	vp(x)	r: run(r), past(r), fast(r), argl(r,j)

r: run(r), past(r), fast(r), arg1(r,j), name(j, John)

s(x) → np(y), vp(x,y)
vp(x) → vp(x) adv(x)

AGENDA

Word	Category	Semantics
John ran fast	S(x)	Complete!

CHART

Word	Category	Semantics
John ran	S(r)	r:run(r), past(r), arg1(r,j), name(j, john)
ran fast	vp(x)	r: run(r), past(r), fast(r), argl(r,j)
ran quickly	Vp(x)	r: run(r), past(r), fast(r), argl(r,j)

Can merge these on the basis of the s(x) rule.

John ran quickly

r: run(r), past(r), fast(r), arg1(r,j), name(j, John)

s(x) → np(y), vp(x,y)
vp(x) → vp(x) adv(x)

AGENDA

Word	Category	Semantics
John ran fast	S(r)	Complete!
John ran quickly	S(r)	Complete

Done! Have covered all the input semantics.
NB: Two realisations produced.

CHART

Word	Category	Semantics
John ran	S(r)	r:run(r), past(r), arg1(r,j), name(j, john)
ran fast	vp(x)	r: run(r), past(r), fast(r), argl(r,j)
ran quickly	Vp(x)	r: run(r), past(r), fast(r), argl(r,j)

Some things to notice

- We implicitly restricted the process so that the application of a rule can only cover a given part of the input once.
- Avoids things like:
  - *ran fast quickly*

Word	Category	Semantics
John ran	S(r)	r:run(r), past(r), arg1(r,j), name(j, john)
ran fast	vp(x)	r: run(r), past(r), fast(r), argl(r,j)
ran quickly	vp(x)	r: run(r), past(r), fast(r), argl(r,j)

Some further details

- In this rough outline:
  - Every time we put an edge on the chart, we consider **whether it can somehow interact with any other edge of the chart.**
- Solution:
  - Only consider interactions where edges have “open positions” in which the current edge can be slotted in.
    - The notion of “active” edges.

Some further details

- Consider:
    - NP → DET AP N
    - AP → Adj*
    - *The tall, dark, handsome man*
    - Multiple applications of the NP modification rule. Several orderings possible.
    - Each ordering could end up being merged with the NP!
    - Exponential!
- Solution:
  – Keep track of which entities in the input have been covered by the edges on the chart.
  – Only construct maximal edges when merging:
    - Don't build *the tall dark man* if that leaves out *handsome*.

Part 4

OVERGENERATION AND RANKING

Nitrogen and HALogen

- Pioneering realisation systems with wide coverage (i.e. handle many phenomena of English grammar)
- Based on overgeneration/ranking
- HALogen (Langkilde-Geary 2002) is a successor to Nitrogen (Langkilde & Knight 1998)
  - main differences:
    - representation data structure for possible realisation alternatives
    - HALogen handles more grammatical features

Structure of HALogen

Symbolic Generator

- Rules to map input representation to syntactic structures
- Lexicon
- Morphology

→ multiple outputs represented in a “forest”

↓

Statistical ranker

- n-gram model (from Penn Treebank)

← *best sentence*

HALogen Input

Grammatical specification

```
(e1 /  eat
      :subject (d1 / dog)
      :object (b1 / bone
              :premod(m1 / meaty))
      :adjunct(t1 / today))
```

Semantic specification

```
(e1 /  eat
      :agent (d1 / dog)
      :patient (b1 / bone
              :premod(m1 / meaty))
      :temp-loc(t1 / today))
```

- Labeled feature-value representation specifying properties and relations of domain objects (e1, d1, etc)
- Recursively structured
- Order-independent
- Can be either grammatical or semantic (or mixture of both)
  – recasting mechanism maps from one to another

HALogen base generator

- Consists of about 255 hand-written rules
- Rules map an input representation into a packed set of possible output expressions.
  - Each part of the input is recursively processed by the rules, until only a string is left.
- Types of rules:
  1. recasting
  2. ordering
  3. filling
  4. morphing

Recasting

- Map semantic input representation to one that is closer to surface syntax.

Semantic specification
(e1 / eat
:patient (b1 / bone
:premod(m1 / meaty))
:temp-loc(t1 / today)
:agent (d1 / dog))

IF relation = :agent
AND sentence is not passive
THEN map relation to :subject

Grammatical specification
(e1 / eat
:object (b1 / bone
:premod(m1 / meaty))
:adjunct(t1 / today)
:subject (d1 / dog))

Ordering

- Assign a linear order to the values in the input.

Grammatical specification

```
(e1 /  eat
        :object (b1 / bone
          :premod(m1 / meaty))
        :adjunct(t1 / today)
        :subject (d1 / dog))
```

Put subject first unless sentence is passive.
Put adjuncts sentence-finally.

Grammatical specification + order

```
(e1 /  eat
        :subject (d1 / dog)
        :object (b1 / bone
          :premod(m1 / meaty))
        :adjunct(t1 / today))
```

Filling

- If input is under-specified for some features, add all the possible values for them.
  - NB: this allows for different degrees of specification, from minimally to maximally specified input.
  - Can create multiple “copies” of same input

Grammatical specification + order

```
(e1 /  eat
      :subject (d1 / dog)
      :object (b1 / bone
        :premod(m1 / meaty))
      :adjunct(t1 / today))
```

+:TENSE (past)

+:TENSE (present)

Morphing

- Given the properties of parts of the input, add the correct inflectional features.

```
Grammatical specification + order
(e1 /  eat
        :tense(past)
        :subject (d1 / dog)
        :object (b1 / bone
          :premod(m1 / meaty))
        :adjunct(t1 / today))
```

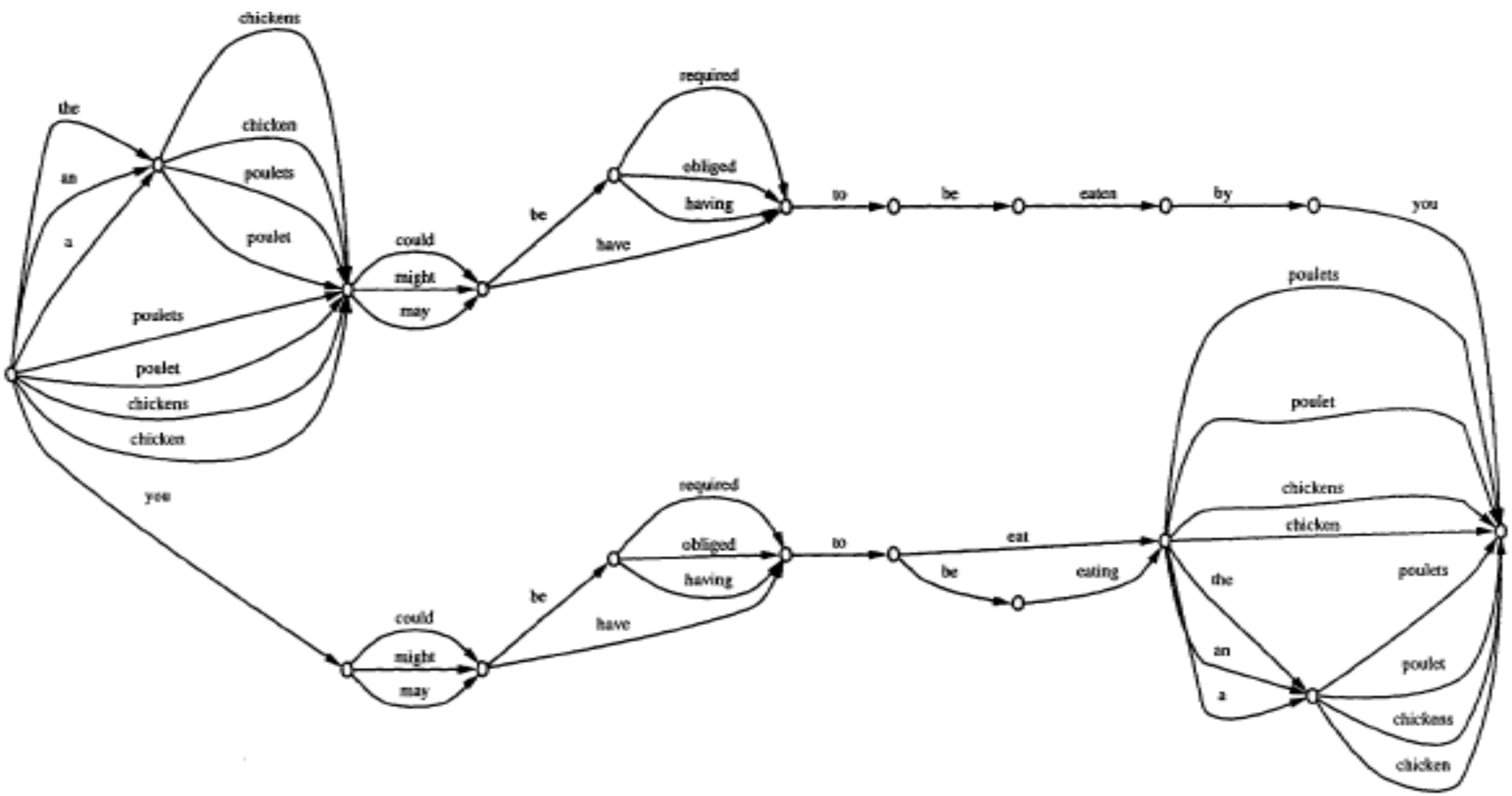
```
Grammatical specification + order
(e1 /  ate
        :subject (d1 / dog)
        :object (b1 / bone
          :premod(m1 / meaty))
        :adjunct(t1 / today))
```

The output of the base generator

- Problem:
  - a single input may have literally hundreds of possible realisations after base generation
  - these need to be represented in an efficient way to facilitate search for the best output

- Options:
  - word lattice
  - forest of trees

Option 1: lattice structure (Langkilde-Geary 2000)



"You may have to eat chicken": 576 possibilities!

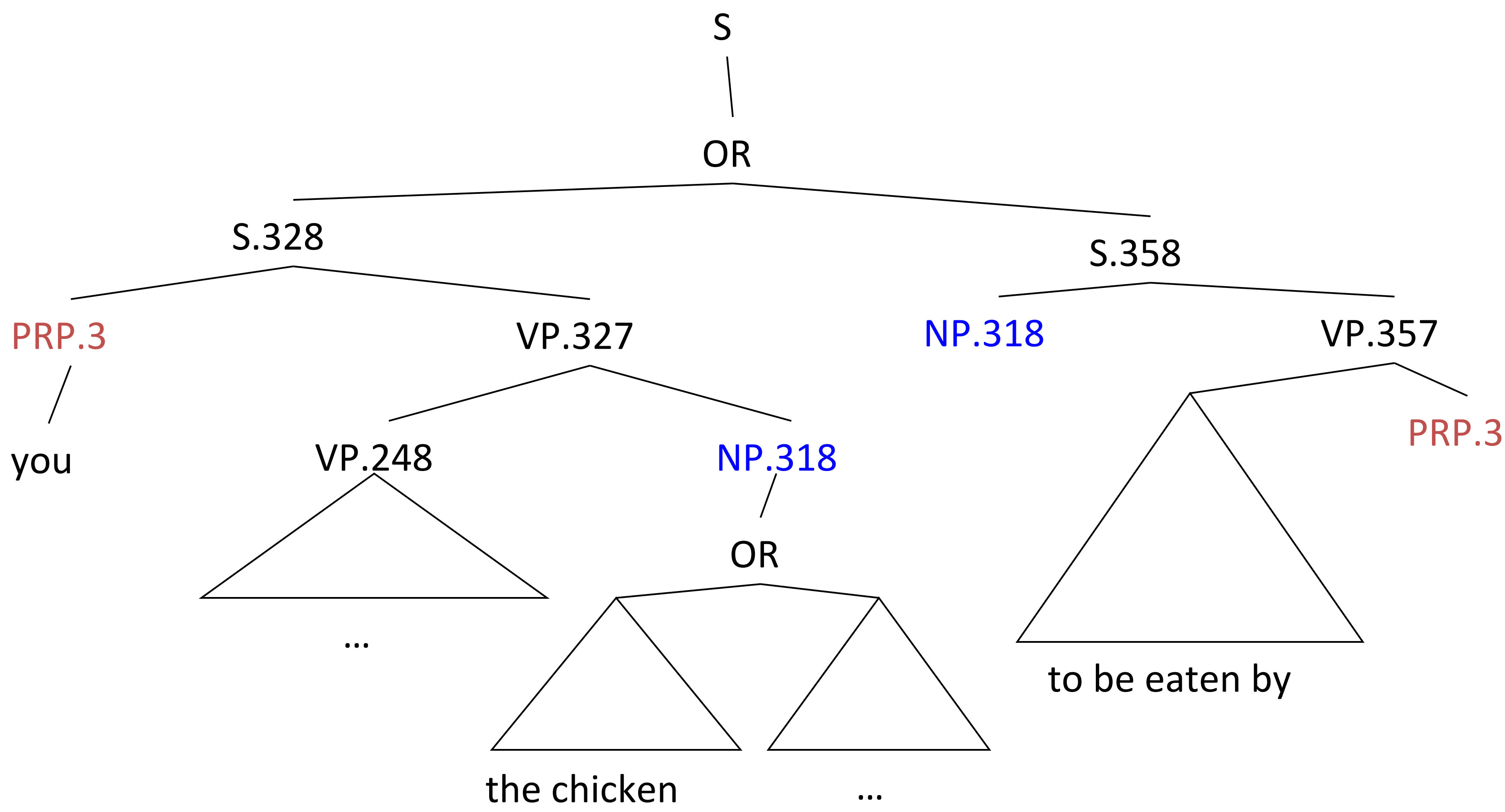
Properties of lattices

- In a lattice, a complete left-right path represents a possible sentence.

- Lots of duplication!
  - e.g. the same word “chicken” occurs multiple times
  - ranker will be scoring the same substring more than once

- In a lattice path, every word is dependent on all other words.
  - can’t model local dependencies

Option 2: Forests (Langkilde-Geary '00,'02)



Properties of forests

- Efficient representation:
  - each individual constituent represented only once, with pointers
  - ranker will only compute a partial score for a subtree once
  - several alternatives represented by disjunctive (“OR”) nodes
- Equivalent to a non-recursive context-free grammar
  - S.469 $\rightarrow$ S.328
  - S.469 $\rightarrow$ S.358
  - …

Statistical ranking

- Uses n-gram language models to choose the best realisation *r*:

$$
\begin{aligned}
r_{best} &= \arg\max_{r \in forest} \prod_{i=1}^{n} P(w_i \mid w_1 \ldots w_{i-1}) \\
&= \arg\max_{r \in forest} \prod_{i=1}^{n} P(w_i \mid w_{i-1}) \, [\text{Markov assumption}]
\end{aligned}
$$

Performance of HALogen

Minimally specified input frame (bigram model):

- *It would sell its fleet age of Boeing Co. 707s because of maintenance costs increase the company announced earlier.*

Minimally specified input frame (trigram model):

- *The company earlier announced it would sell its fleet age of Boeing Co. 707s because of the increase maintenance costs.*

Almost fully specified input frame:

- *Earlier the company announced it would sell its aging fleet of Boeing Co. 707s because of increased maintenance costs.*

Observations

- The usual issues with n-gram models apply:
  - bigger *n* → better output, but more data sparseness

- Domain dependent
  - relatively easy to train, assuming corpus in the right format

Beyond n-grams?

- N-gram models rank purely based on word sequences.
- Recent work has begun to consider factoring in other features during re-ranking.
  - This takes us beyond simple language models.
  - Consider factored language models, for example.

Example: Distance-based features

- Recall:
  - *Separately, the Federal Energy Regulatory Commission turned down* ***for now*** *a request by Northeast seeking approval of its possible purchase of PS of New Hampshire. (WSJ0013.16).*
  - *Separately, the Federal Energy Regulatory Commission turned down a request by Northeast seeking approval of its possible purchase of PS of New Hampshire* ***for now****. (WSJ0013.16).*
- Distance-based features can cause a ranker to prefer outputs where the modifier is closer to the host.

Example: Agreement features

- If input is underspecified w.r.t. inflection, we would like to enable our ranker to prefer sentences where subject-verb agreement is correct.
  - E.g. agreement based on animacy and number.
  - Can be compromised by distance (e.g. with a WH-clause between subject NP and Verb).
  - N-gram models can miss this.
    - ***The car, which was bought by the manager, was/were damaged.***
    - ***The people who/which/that bought cigarettes…***

But wait..

- Why not put these directly in the grammar?
  – Grammar is then guaranteed to only overgenerate with correct alternatives.
  – Ranking can proceed as normal.

- The main problems:
  – Not all rules are easy to specify (cf. modifier ordering);
  – Some rules have a lot of exceptions, sub-regularities etc.

Part 6

REALISATION ENGINES

A realisation engine

- Unlike a realiser, a realisation engine is simply a software library which:
  - Performs linearisation
  - Performs morphological inflection
  - i.e. generates correct syntactic structures

  BUT:

  - Leaves the choices up to the user/engineer.

SimpleNLG

- SimpleNLG (Gatt & Reiter 2009):
  - Developed at Aberdeen
  - Java API to generate English sentences
  - Versions now exist for French (Vaudry & Lapalme 2013), German (Bollman 2011), Brazilian Portugese (de Oliveira & Sripada 2014).

- Features:
  - No input specification.
  - No choice-making behaviour, except for basic linearisation and inflection decisions.
  - Allows mixture of canned text and syntax.
  - Theory-neutral (except for definition of phrase and word types).
  - Reasonable coverage (but not formally evaluated).

SimpleNLG Example

- Target sentence: ***Once upon a time there was a cat.***

```
SPhraseSpec s = this.phraseFactory.createClause();
s.setSubject("there");
VPPhraseSpec vp = this.phraseFactory.createVerbPhrase("be");
NPPhraseSpec np = this.phraseFactory.createNounPhrase("a", "cat");
vp.setComplement(np);
s.setVerbPhrase(vp);
s.setFeature(Feature.TENSE, Tense.PAST);
StringElement string  = new StringElement(“Once upon a time”);
s.setFrontModifier(string);
```

Why bother?

- Often, developers of NLG systems are interested in other parts of the NLG process.
  - I.e. don't want to bother with a sophisticated realisation component.

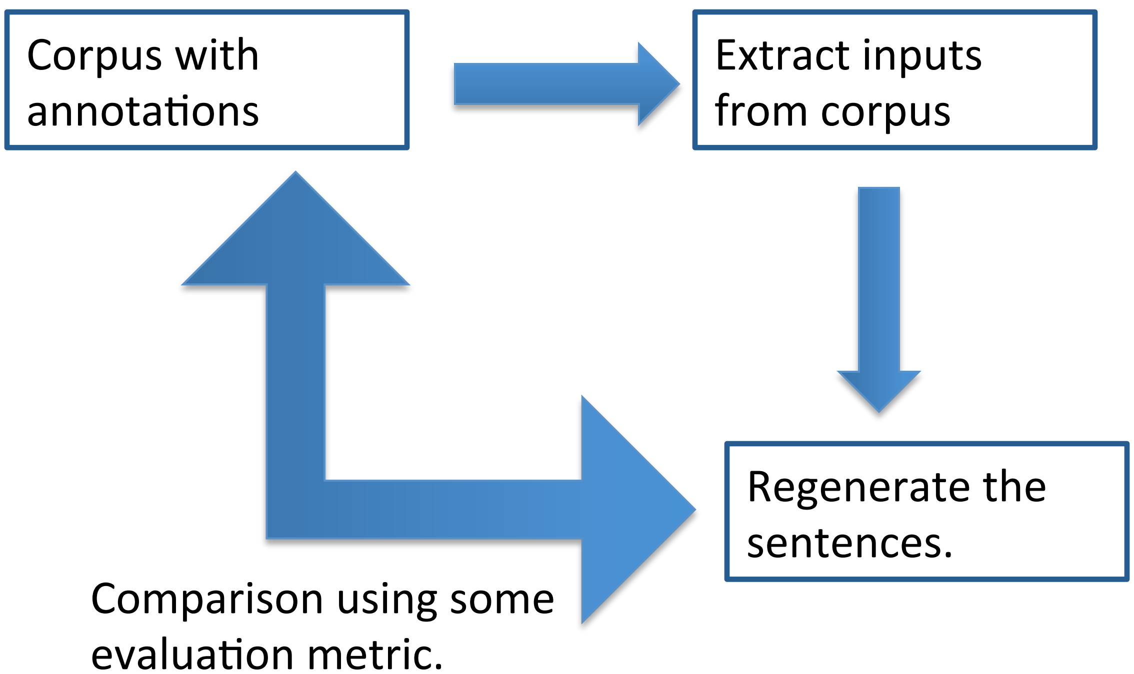
- Full control
  - The fact that it's theory-neutral helps.

- Simplicity
  - Used by quite a large community, accessible to non-linguists.
  - Used also by individuals interested in using an NL front-end, but not really doing "full-fledged" NLG.

Part 6

EVALUATING REALISERS

The typical evaluation setup



Evaluation metrics

- Coverage:
  - How much of the corpus does the realiser manage to re-generate?
    - What proportion does it regenerate exactly?
    - What proportion does it have no output for?
- String overlap:
  - Simple String Accuracy
  - BLEU
  - Both of these give average scores over the test set.

Looking under the hood

- Simple string-based averages don't tell us what it is exactly that is going wrong.

- Callaway (2005):
  - Exhaustive analysis of errors made by FUF/SURGE against the Penn Treebank.
  - Very high coverage, highest BLEU score recorded to date.
  - Errors arise from a variety of sources:
    - Errors in the corpus annotation.
    - Errors during transformation (extraction of inputs from corpus)
    - Errors of syntax (problems with rules)
    - …

Current Frontiers in Realisation

- Improving realisers by taking into account more linguistic features.
  - Re-ranking, or grammar engineering?
- Multilinguality:
  - Most realisation work done on English. Other languages have very different (sometimes more complex challenges).
  - Problem: corpora from which to induce grammars, train re-rankers.

Some final observations

- Traditionally, realisation is viewed as the final stage of NLG.
- However, lexicalisation, aggregation etc are often thought of as sub-tasks of realisation.

- As statistical models get more sophisticated, we see realisation also working with:
  - Lexical features
  - Pragmatic features
  - Information structure

References

- Bateman, J. A. (1997). Enabling technology for multilingual natural language generation: the KPML development environment. Natural Language Engineering, 3(1), 15–55. doi:10.1017/S1351324997001514
- Callaway, C. (2003). Evaluating coverage for large symbolic NLG grammars. In Proceedings of the 18th international joint conference on Artificial intelligence (IJCAI'03) (pp. 811–816). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.169.7097&rep=rep1&type=pdf
- Callaway, C. B. (2005). The Types and Distributions of Errors in a Wide Coverage Surface Realizer Evaluation. In Proceedings of the 10th European Workshop on Natural Language Generation (ENLG'05) (pp. 162–167). Aberdeen, UK: Association for Computational Linguistics.
- Elhadad, M., & Robin, J. (1996). An overview of SURGE: A reusable comprehensive syntactic realization component. In Procedings of the 8th International Natural Language Generation Workshop (IWNLG'98) (pp. 1–4). Sussex, UK: Association for Computational Linguistics. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.43.5187&rep=rep1&type=pdf
- Filippova, K., & Strube, M. (2009). Tree linearization in English: Improving language model based approaches. Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT'09), 33(June), 225–228. Retrieved from http://portal.acm.org/citation.cfm?id=1620915

References

- Gatt, A., & Reiter, E. (2009). SimpleNLG: A realisation engine for practical applications. In Proceedings of the 12th European Workshop on Natural Language Generation (ENLG'09) (pp. 90–93). Athens, Greece: Association for Computational Linguistics.
- Hockenmaier, J., & Steedman, M. (2007). CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. Computational Linguistics, 33(3), 355–396. doi:10.1162/coli.2007.33.3.355
- Kay, M. (1996). Chart Generation. In Proceedings of the 34th annual meeting of the Association for Computational Linguistics (ACL'96) (pp. 200–204). Santa Cruz, CA: Association for Computational Linguistics.
- Khan, I. H., Deemter, K. Van, & Ritchie, G. (2012). Managing Ambiguity in Reference Generation: The Role of Surface Structure. Topics in Cognitive Science, 4(2), 211–231. doi:10.1111/j.1756-8765.2011.01167.x
- Langkilde, I. (2000). Forest-based statistical sentence generation. In Proceedings of the 6th Applied Natural Language Processing Conference and the 1st Meeting of the North American Chapter of the Association of Computational Linguistics (ANLP-NAACL'00) (pp. 170–177). Seattle, WA: Association for Computational Linguistics.
- Langkilde, I., & Knight, K. (1998). Generation that exploits corpus-based statistical knowledge. In Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (ACL/COLING'98) (pp. 704–710). doi:10.3115/980845.980963
- Langkilde-Geary, I., & Knight, K. (2002). HALogen Statistical Sentence Generator. In Proceedings of the ACl 2002 Demnonstrations Session (pp. 102–103). Philadelphia, PA: Association for Computational Linguistics.
- Malouf, R. (2000). The order of prenominal adjectives in natural language generation. In Proceedings of the 38th Annual Meeting on Association for Computational Linguistics (ACL'00) (pp. 85–92). Morristown, NJ, USA: Association for Computational Linguistics. doi:10.3115/1075218.1075230
- Mitchell, T. M., Shinkareva, S. V, Carlson, A., Chang, K.-M., Malave, V. L., Mason, R. a, & Just, M. A. (2008). Predicting human brain activity associated with the meanings of nouns. Science, 320(5880), 1191–5. doi:10.1126/science.1152876
- Rajkumar, R., & White, M. (2014). Better Surface Realization through Psycholinguistics. Language and Linguistics Compass, 8(10), 428–448.
- White, M., Rajkumar, R., & Martin, S. (2007). Towards Broad Coverage Surface Realization with CCG. In Proceedings of the Workshop on Using Corpora for NLG: Language Generation and Machine Translation (UCNLG +MT}.