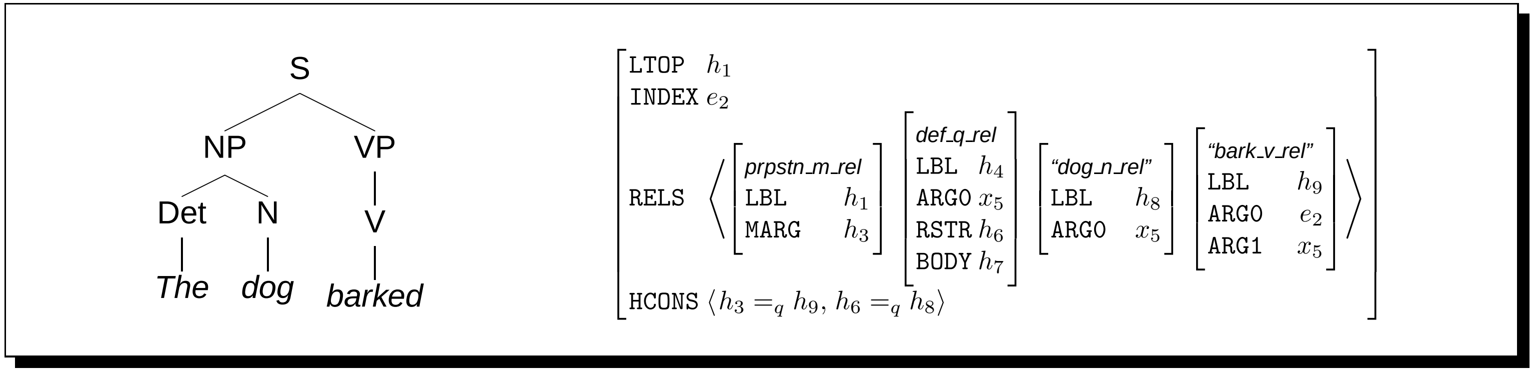


Algorithms for AI and NLP (INF4820 — Parsing)

S $\longrightarrow$ *NP VP; NP* $\longrightarrow$ *Det N; VP* $\longrightarrow$ *V NP*

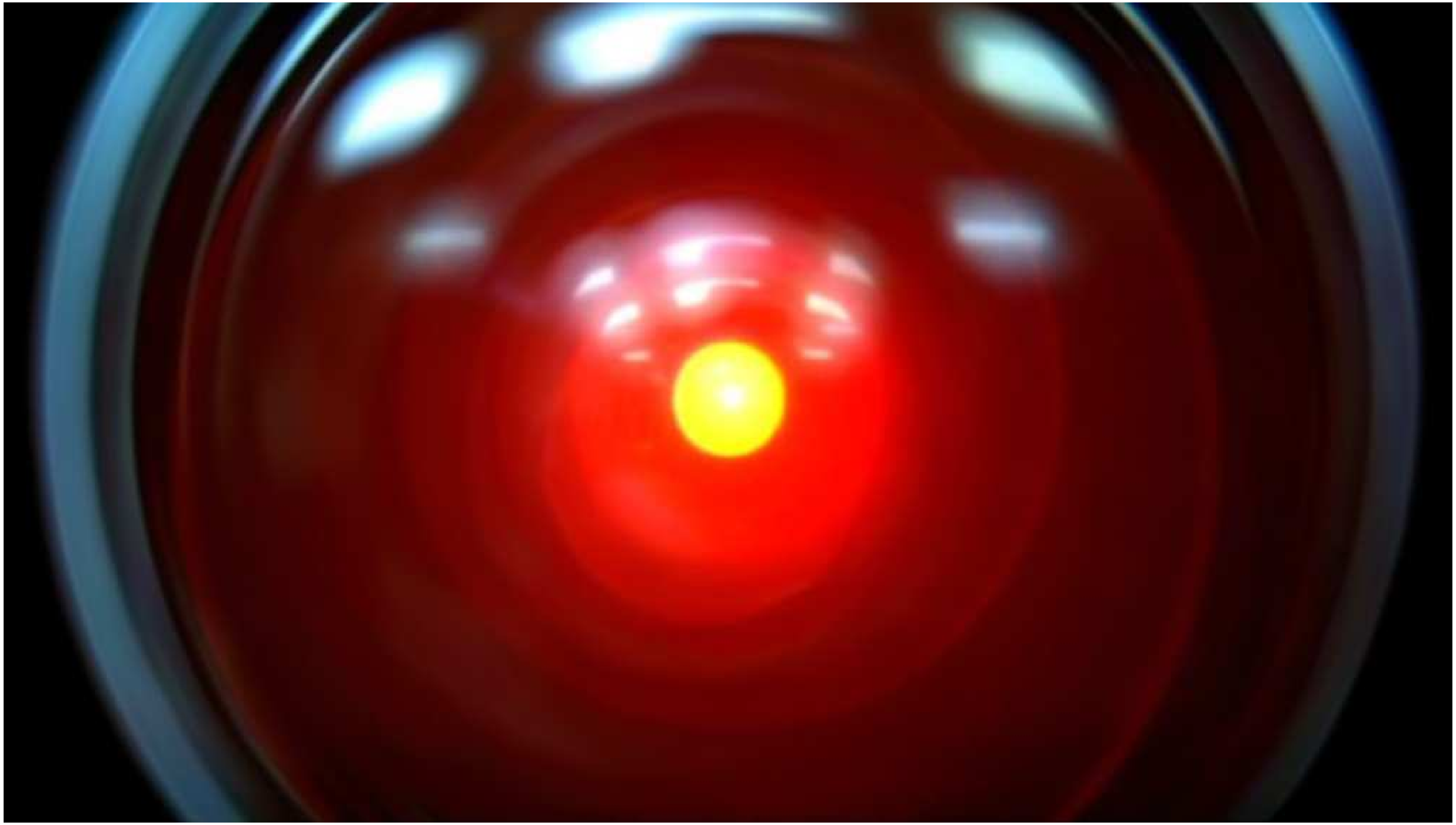
Stephan Oepen and Jan Tore Lønning

Universitetet i Oslo

{ oe | jtl }@ifi.uio.no

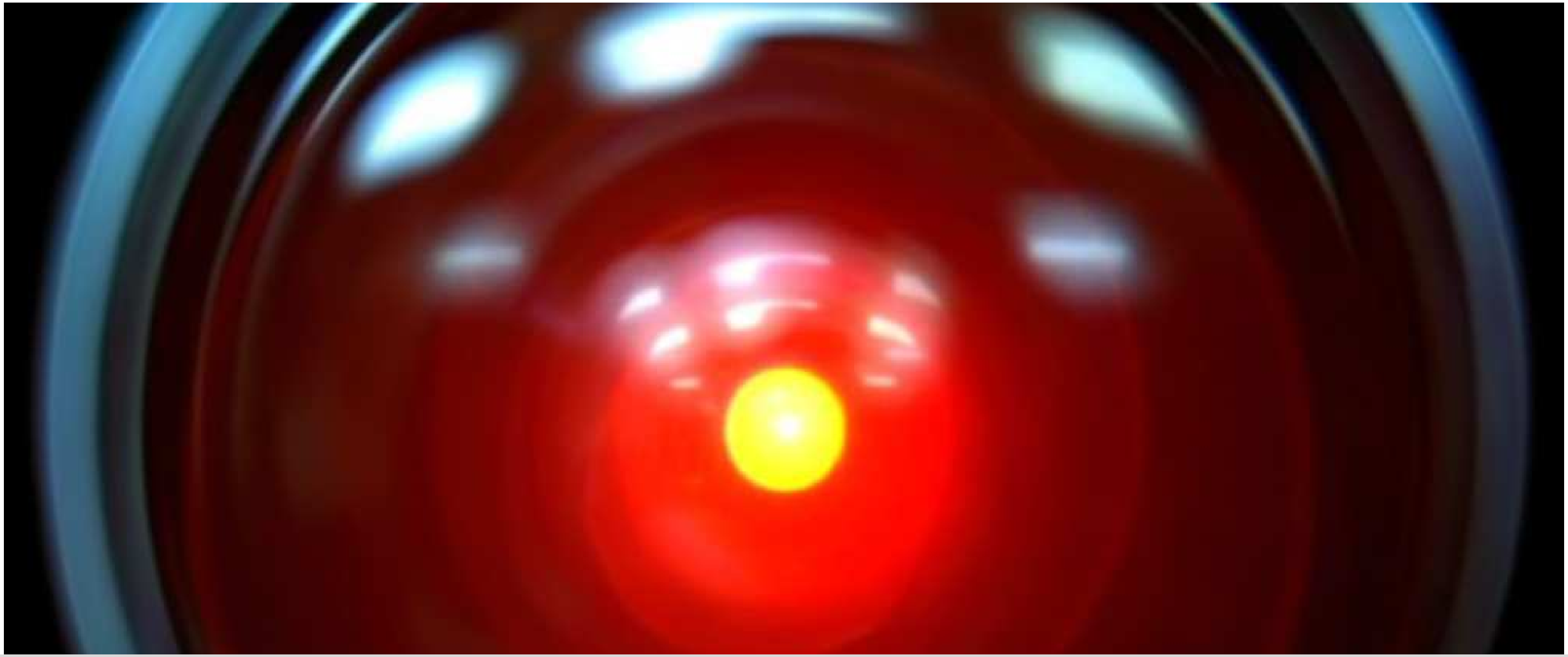
So, What Actually is Language Technology?

So, What Actually is Language Technology?



(2001: A Space Odyssey; HAL 9000; 1968)

So, What Actually is Language Technology?



→ (young) interdisciplinary science: language, cognition, computation;

→ (again) culturally and commercially relevant for 'knowledge society'.

No, Really, What is Computational Linguistics?

... teaching computers our language. (Alien Researcher, 2000)

No, Really, What is Computational Linguistics?

... teaching computers our language. (Alien Researcher, 2000)

We Understand™. Unlike other solutions based on keyword or phrase recognition, YY Software's product actually understands customer e-mails and Web interaction. (Start-Up Marketing Blurb, 2000)

No, Really, What is Computational Linguistics?

... teaching computers our language. (Alien Researcher, 2000)

We Understand™. Unlike other solutions based on keyword or phrase recognition, YY Software's product actually understands customer e-mails and Web interaction. (Start-Up Marketing Blurb, 2000)

... the scientific study of human language—specifically of the system of rules and the ways in which they are used in communication—using mathematical models and formal procedures that can be realized and validated using computers; a cross-over of many disciplines. (Stanford Linguistics Professor, 1980s)

No, Really, What is Computational Linguistics?

... teaching computers our language. (Alien Researcher, 2000)

We Understand™. Unlike other solutions based on keyword or phrase recognition, YY Software's product actually understands customer e-mails and Web interaction. (Start-Up Marketing Blurb, 2000)

... the scientific study of human language—specifically of the system of rules and the ways in which they are used in communication—using mathematical models and formal procedures that can be realized and validated using computers; a cross-over of many disciplines. (Stanford Linguistics Professor, 1980s)

... a sub-discipline of our Artificial Intelligence programme.

(MIT CS Professor, 1970s)

Families of Language Processing Tasks

Speech Recognition and Synthesis

Summarization & Text Simplification

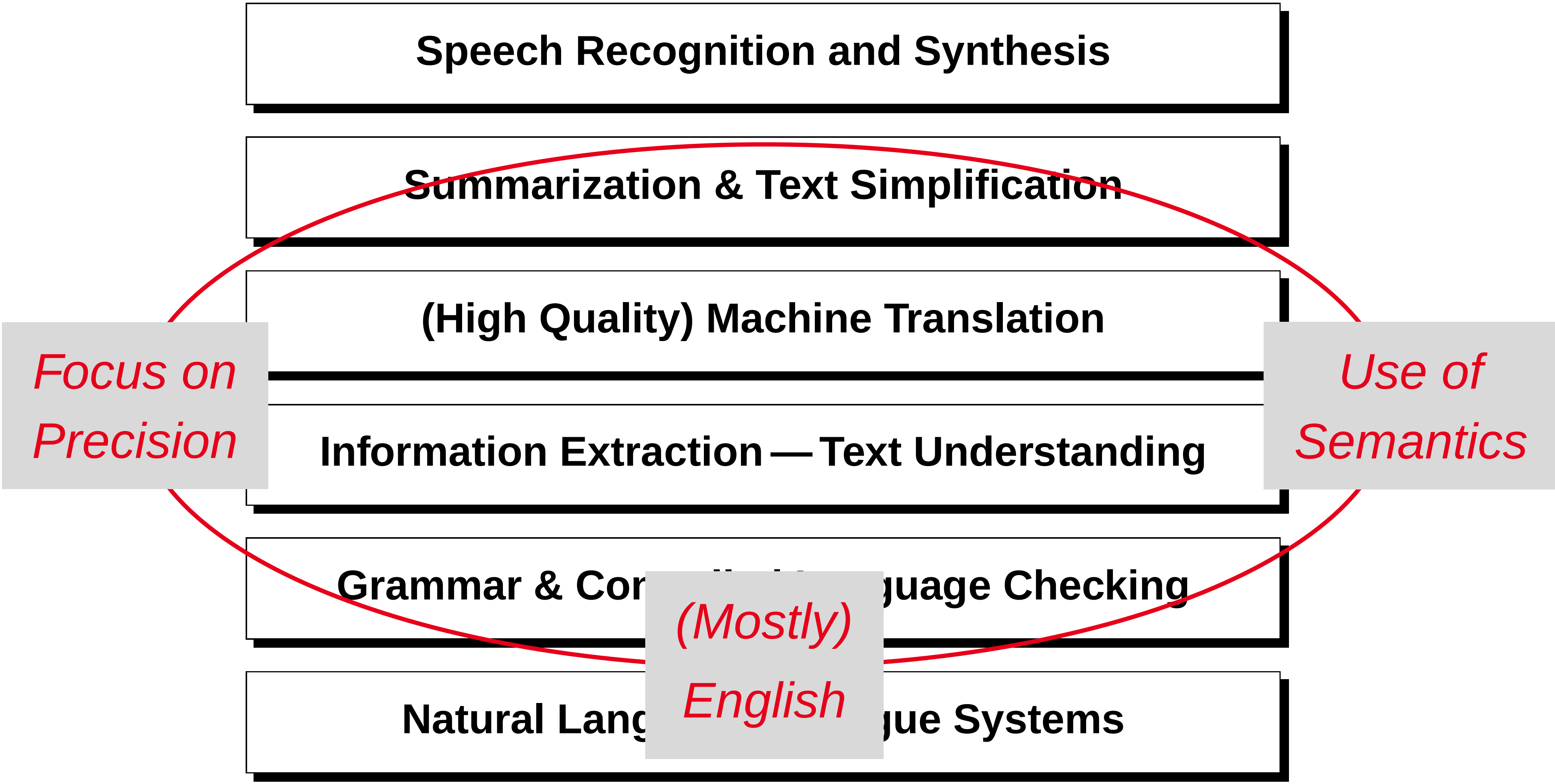
(High Quality) Machine Translation

Information Extraction — Text Understanding

Grammar & Controlled Language Checking

Natural Language Dialogue Systems

Families of Language Processing Tasks



What Makes Natural Language a Hard Problem?

```
< Den andre veien mot Bergen er kort. --- 12 x 30 x 25 = 25
> The other path towards Bergen is short. {0.58} (1:1:0).
> The other road towards Bergen is short. {0.56} (1:0:0).
> The second road towards Bergen is short. {0.55} (2:0:0).
> That other path towards Bergen is a card. {0.54} (0:1:0).
> That other road towards Bergen is a card. {0.54} (0:0:0).
> The second path towards Bergen is short. {0.51} (2:1:0).
> The other road against Bergen is short. {0.48} (1:2:0).
> The second road against Bergen is short. {0.48} (2:2:0).
  ...
> Short is the other street towards Bergen. {0.33} (1:4:0).
> Short is the second street towards Bergen. {0.33} (2:4:0).
  ...
```

What Makes Natural Language a Hard Problem?

```
< Den andre veien mot Bergen er kort. --- 12 x 30 x 25 = 25
> The other path towards Bergen is short. {0.58} (1:1:0).
> The other road towards Bergen is short. {0.56} (1:0:0).
> The second road towards Bergen is short. {0.55} (2:0:0).
> That other path towards Bergen is a card. {0.54} (0:1:0).
> That other road towards Bergen is a card. {0.54} (0:0:0).
> The second path towards Bergen is short. {0.51} (2:1:0).
> Th                                                     ).
> Th                                                     )).
  ...
> Sh                                                     0).
> Sh                                                     4:0).
  ...
```

Scraped Off the Internet

The other way to Bergen is short.

the road to the other bergen is short .

Den other roads against Boron Gene are short.

Other one autobahn against Mountains am abrupt.

More, and More, and More Ambiguity

Speech Recognition

<table>
<tr><td colspan="2">its</td><td>hard</td><td>to</td><td>wreck</td><td>a</td><td>nice</td><td>beach</td></tr>
<tr><td>it</td><td>’s</td><td>hard</td><td>to</td><td colspan="3">recognize</td><td>speech</td></tr>
</table>

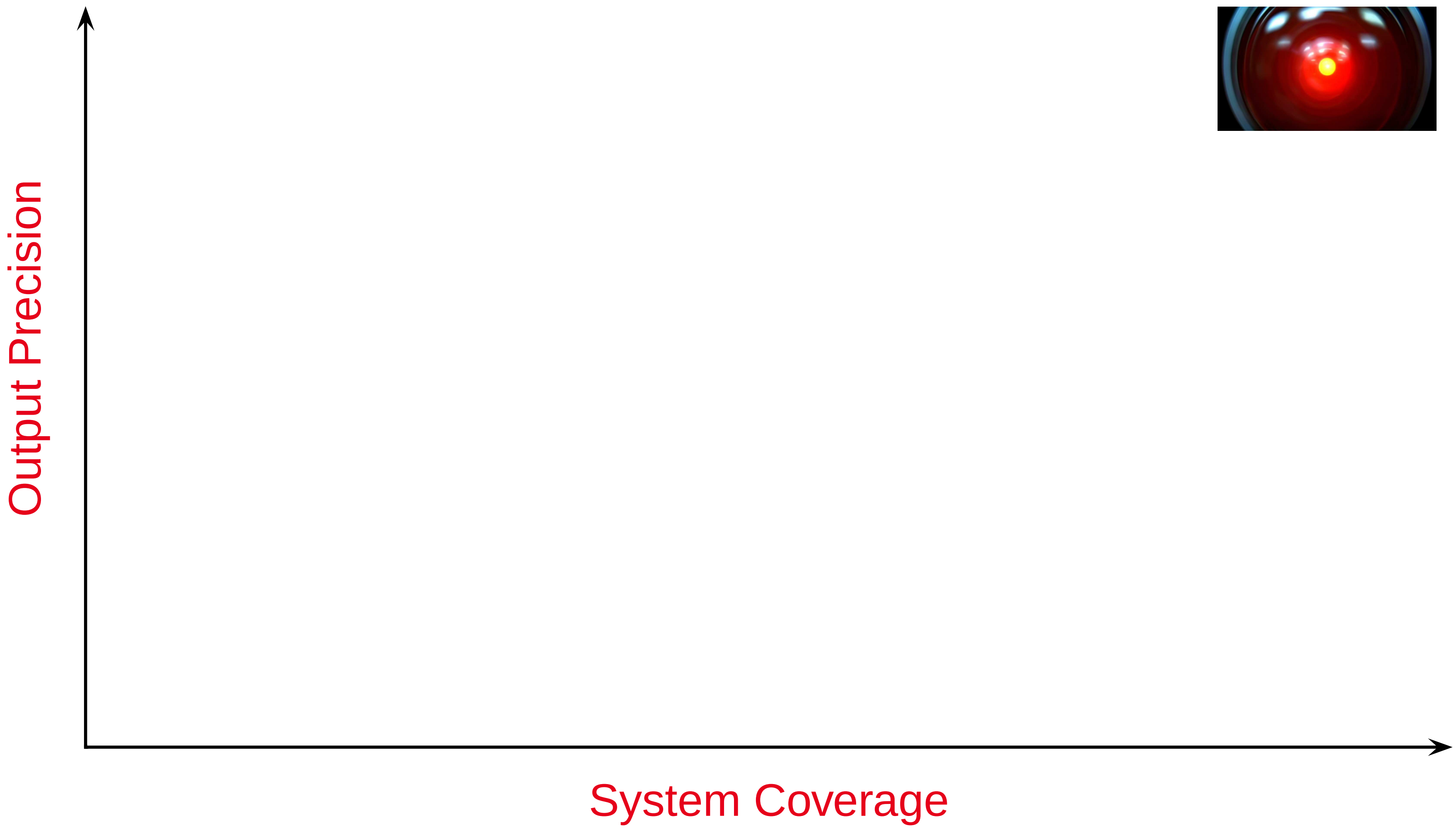
Morphology

- *fisker*   *fisk* $_{\mathrm{N}}$ $+$ plural vs. *fiske* $_{\mathrm{V}}$ $+$ present vs. *fisker* $_{\mathrm{N}}$ $+$ singular;
- *brus-automat* vs. *bru-sau-tomat*; *vinduene* vs. *vin-duene*; et al.

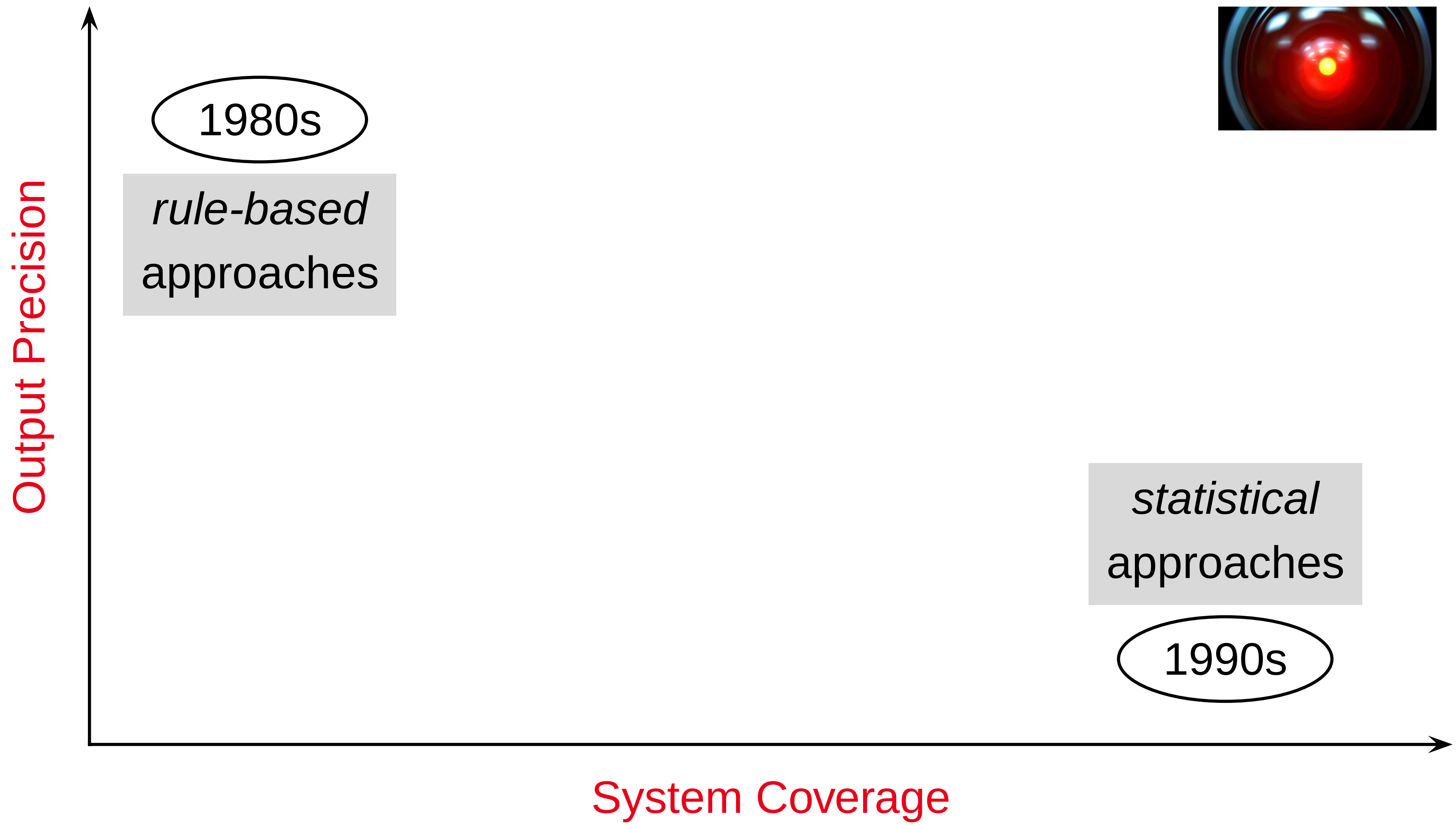
Semantics

- *All Norwegians speak two languages.* $\exists l_1, l_2 \forall n \ldots$ vs. $\forall n \exists l_1, l_2 \ldots$

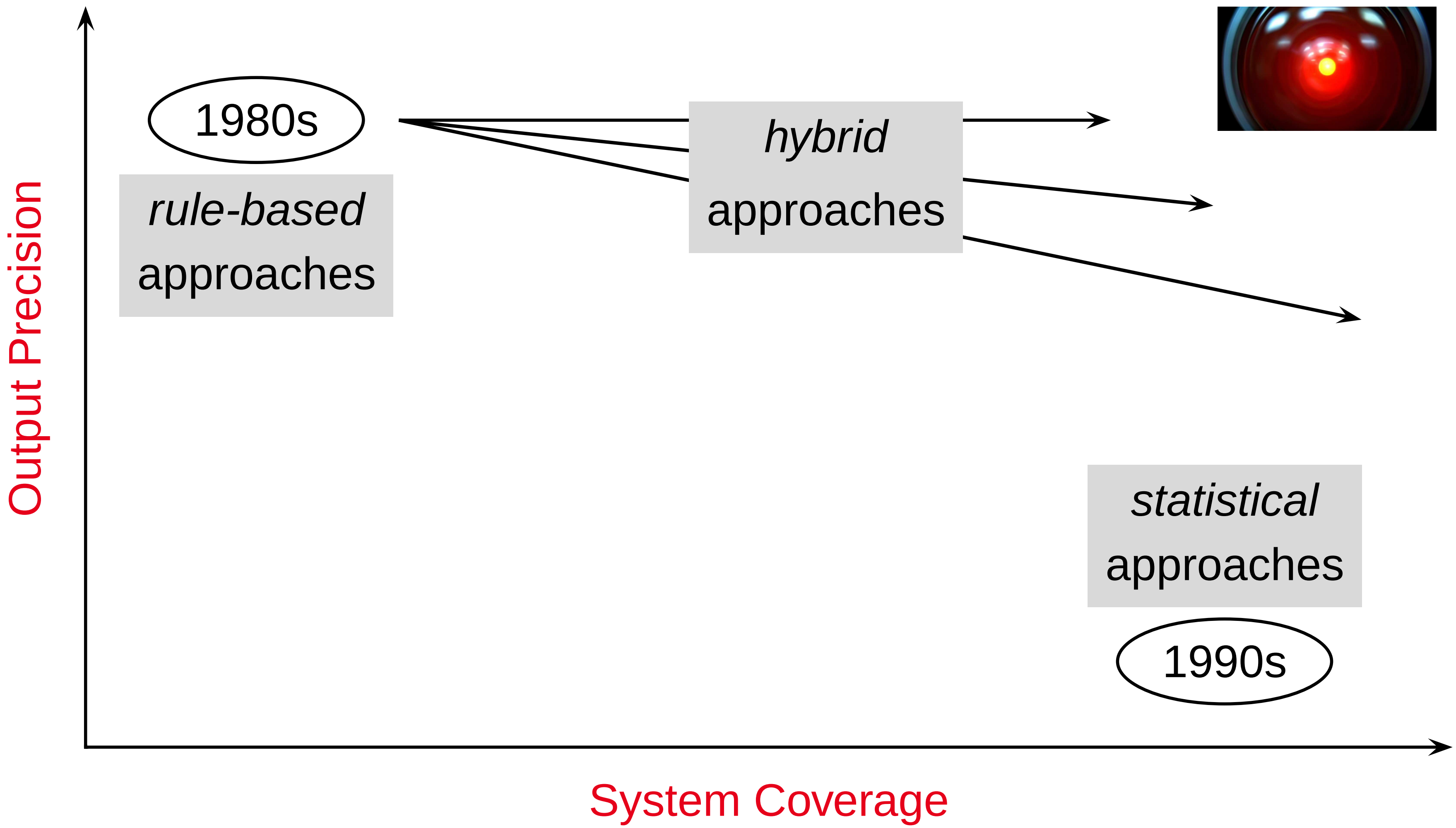
The Holy Grail: Balancing Coverage and Precision



The Holy Grail: Balancing Coverage and Precision



The Holy Grail: Balancing Coverage and Precision



A Tool Towards Understanding: (Formal) Grammar

Wellformedness

- *Kim was happy because ____ passed the exam.*
- *Kim was happy because ____ final grade was an A.*
- *Kim was happy when she saw ____ on television.*

A Tool Towards Understanding: (Formal) Grammar

Wellformedness

- *Kim was happy because ____ passed the exam.*
- *Kim was happy because ____ final grade was an A.*
- *Kim was happy when she saw ____ on television.*

Meaning

- *Kim gave Sandy the book.*
- *Kim gave the book to Sandy.*
- *Sandy was given the book by Kim.*

A Tool Towards Understanding: (Formal) Grammar

Wellformedness

- *Kim was happy because ____ passed the exam.*
- *Kim was happy because ____ final grade was an A.*
- *Kim was happy when she saw ____ on television.*

Meaning

- *Kim gave Sandy the book.*
- *Kim gave the book to Sandy.*
- *Sandy was given the book by Kim.*

Ambiguity

- *Kim saw the astronomer with the telescope.*
- *Have her report on my desk by Friday!*

A Grossly Simplified Example

The Grammar of Spanish

$S \rightarrow$ NP VP
VP $\rightarrow$ V NP
VP $\rightarrow$ VP PP
PP $\rightarrow$ P NP
NP $\rightarrow$ "nieve"
NP $\rightarrow$ "Juan"
NP $\rightarrow$ "Oslo"
V $\rightarrow$ "amó"
P $\rightarrow$ "en"

Juan amó nieve en Oslo

A Grossly Simplified Example

The Grammar of Spanish

S → NP VP

VP → V NP

VP → VP PP

PP → P NP

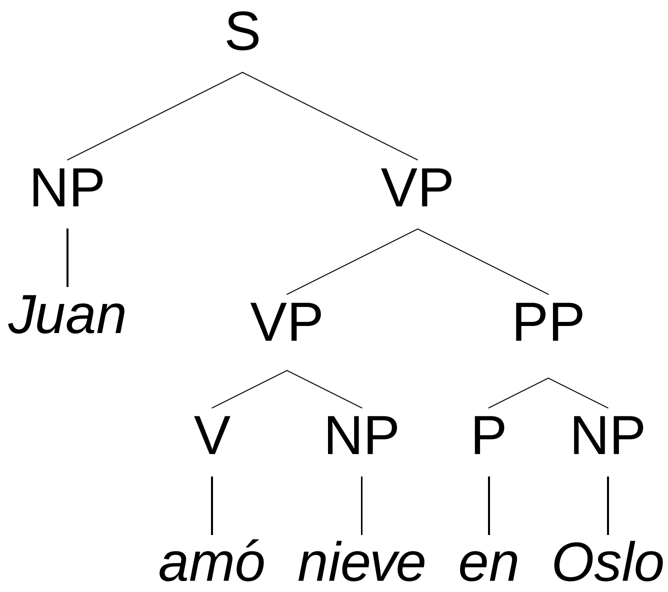
NP → “nieve”

NP → “Juan”

NP → “Oslo”

V → “amó”

P → “en”

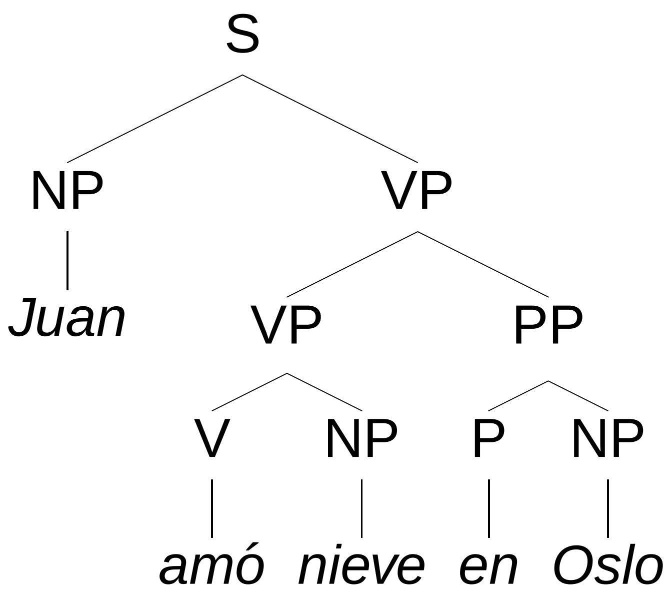


Juan amó nieve en Oslo

A Grossly Simplified Example

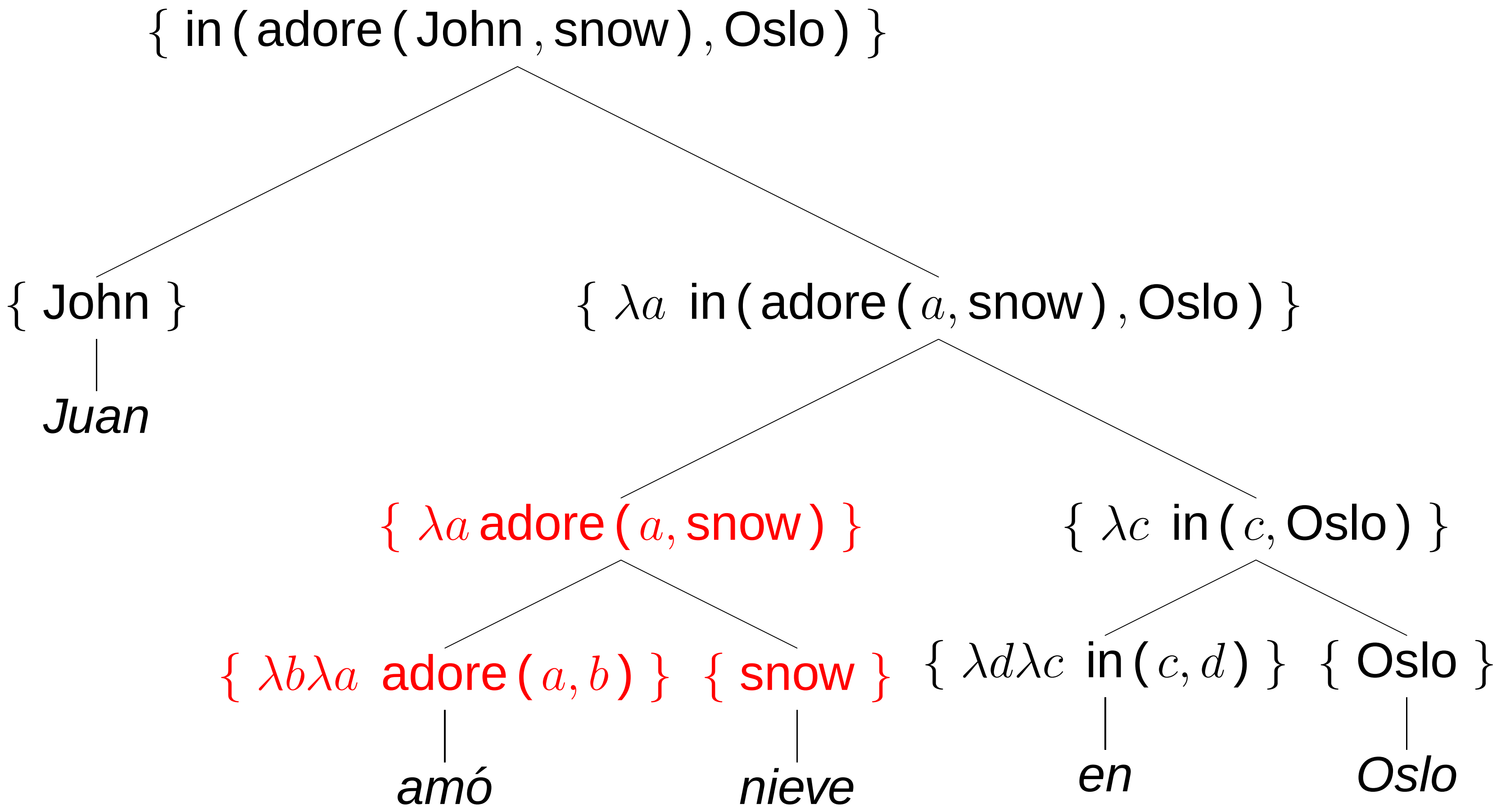
The Grammar of Spanish

S $\rightarrow$ NP VP	{ VP (NP) }
VP $\rightarrow$ V NP	{ V (NP) }
VP $\rightarrow$ VP PP	{ PP (VP) }
PP $\rightarrow$ P NP	{ P (NP) }
NP $\rightarrow$ “nieve”	{ snow }
NP $\rightarrow$ “Juan”	{ John }
NP $\rightarrow$ “Oslo”	{ Oslo }
V $\rightarrow$ “amó”	{ $\lambda b \lambda a$ adore (a, b) }
P $\rightarrow$ “en”	{ $\lambda d \lambda c$ in (c, d) }



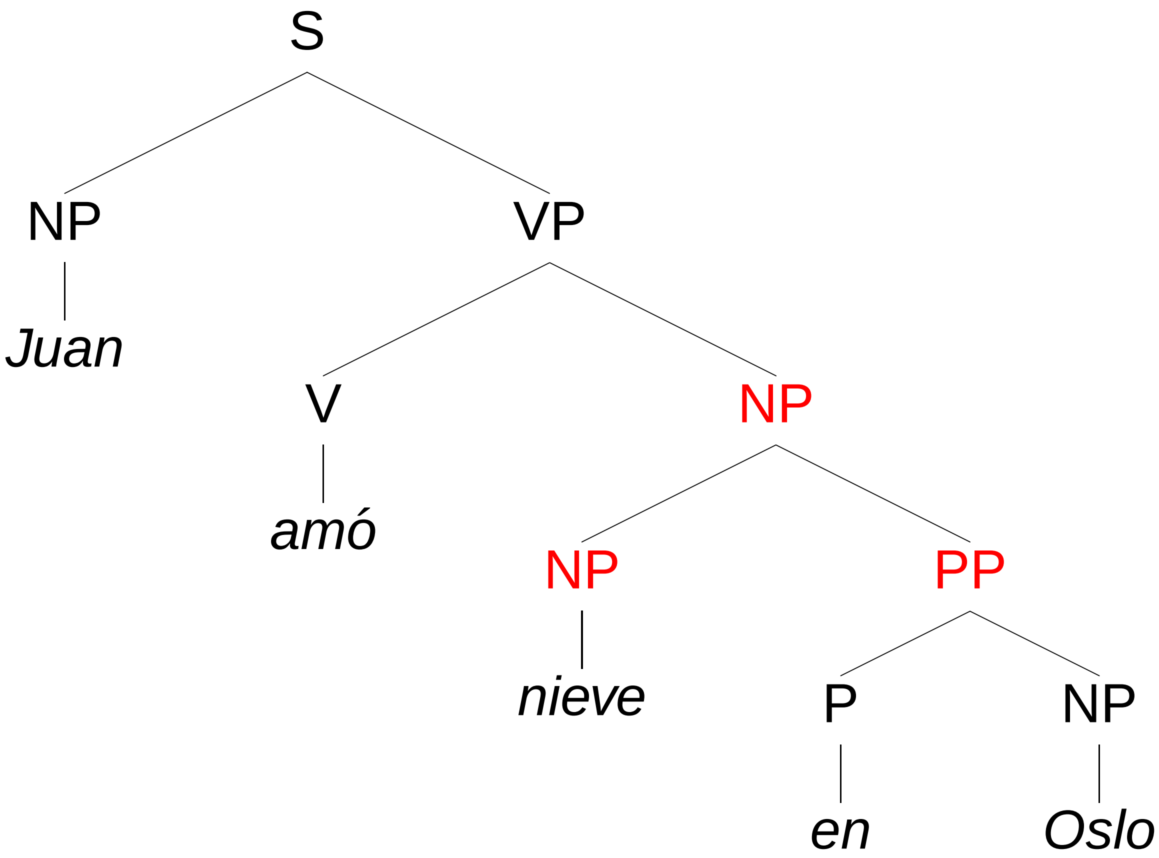
Juan amó nieve en Oslo

Meaning Composition (Grossly Simplified, Still)



VP $\rightarrow$ V NP { V (NP) }

Another Interpretation — Structural Ambiguity



NP → NP PP { PP (NP) }

Mildly Mathematically: Context-Free Grammars

- Formally, a *context-free grammar* (CFG) is a quadruple: $\langle C, \Sigma, P, S \rangle$
- C is the set of categories (aka *non-terminals*), e.g. $\{\text{S}, \text{NP}, \text{VP}, \text{V}\}$;
- Σ is the vocabulary (aka *terminals*), e.g. $\{\text{Kim}, \text{snow}, \text{saw}, \text{in}\}$;
- P is a set of category rewrite rules (aka *productions*), e.g.

S $\rightarrow$ NP VP
VP $\rightarrow$ V NP
NP $\rightarrow$ Kim
NP $\rightarrow$ snow
V $\rightarrow$ saw

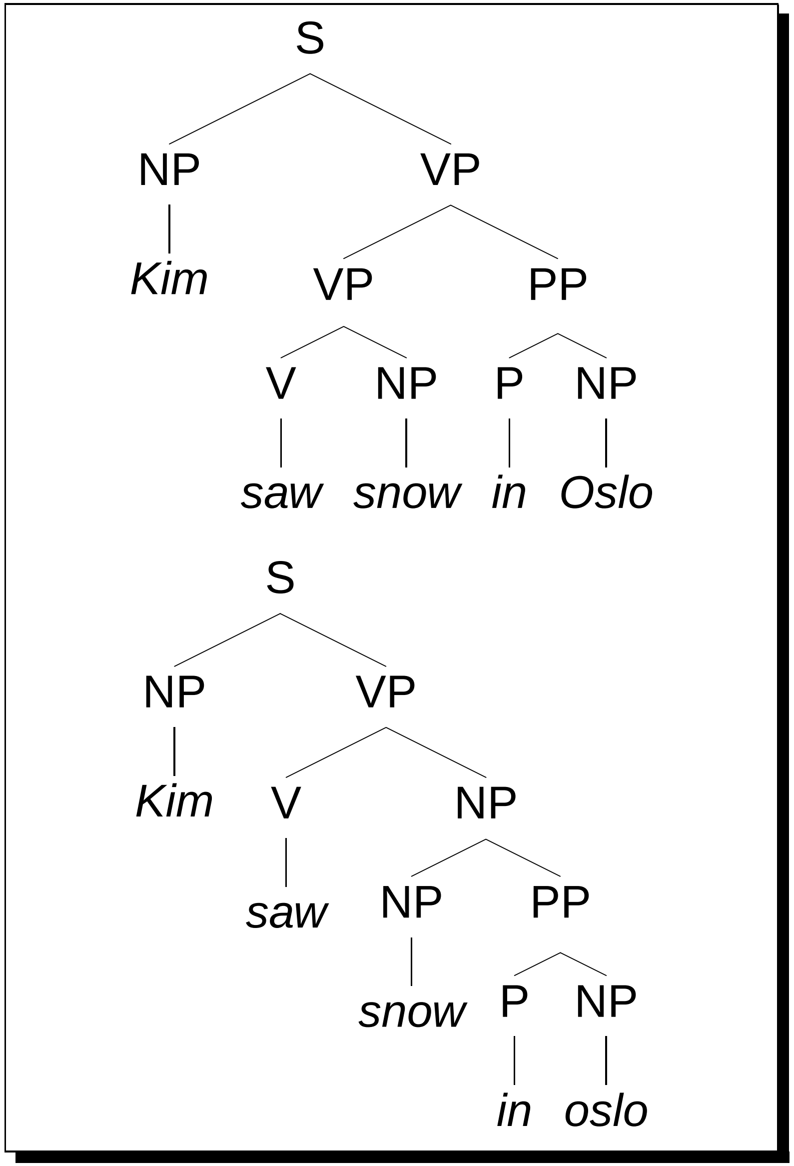
- $S \in C$ is the *start symbol*, a filter on complete ('sentential') results;
- for each rule '$\alpha \rightarrow \beta_1, \beta_2, ..., \beta_n$' $\in P$: $\alpha \in C$ and $\beta_i \in C \cup \Sigma$; $1 \leq i \leq n$.

Parsing: Recognizing the Language of a Grammar

S → NP VP
VP → V | V NP | VP PP
NP → NP PP
PP → P NP
NP → Kim | snow | Oslo
V → saw
P → in

All Complete Derivations

- are rooted in the start symbol S;
- label internal nodes with categories $\in C$, leafs with words $\in \Sigma$;
- instantiate a grammar rule $\in P$ at each local subtree of depth one.



A Simple-Minded Parsing Algorithm

Control Structure

- top-down: given a parsing goal α, use all grammar rules that rewrite α;
- successively instantiate (extend) the right-hand sides of each rule;
- for each β_i in the RHS of each rule, recursively attempt to parse β_i;
- termination: when α is a prefix of the input string, parsing succeeds.

(Intermediate) Results

- Each result records a (partial) tree and remaining input to be parsed;
- complete results consume the full input string and are rooted in S;
- whenever a RHS is fully instantiated, a new tree is built and returned;
- all results at each level are combined and successively accumulated.

A Recursive Descent Parser

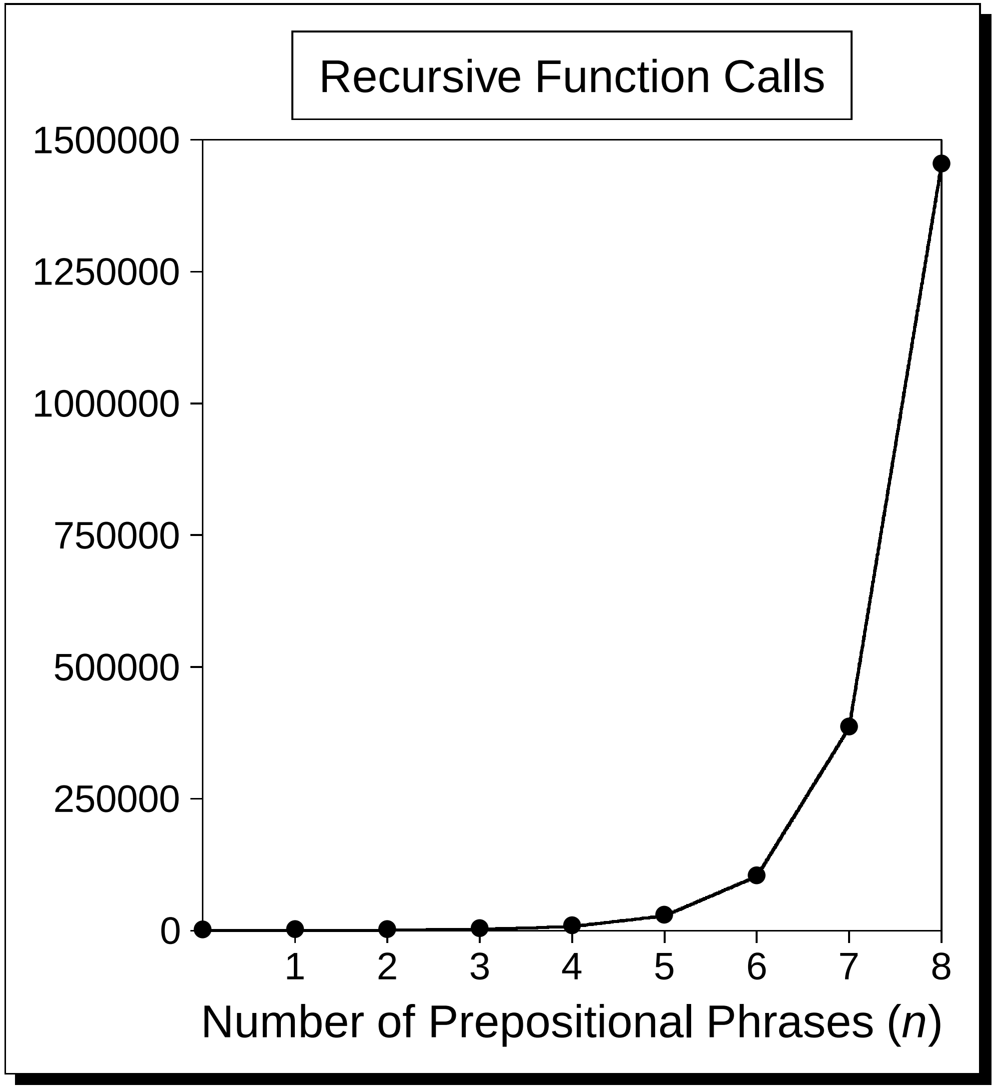
```
(defun parse (input goal)
  (if (equal (first input) goal)
    (list (make-state :tree (first input) :input (rest input)))
    (loop
        for rule in (rules-rewriting goal)
        append (instantiate (rule-lhs rule) nil (rule-rhs rule) input))))
```

```
(defun instantiate (lhs analyzed unanalyzed input)
  (if (null unanalyzed)
    (list (make-state :tree (make-tree :root lhs :daughters analyzed)
                      :input input))
    (loop
        for parse in (parse input (first unanalyzed))
        append (instantiate
                lhs
                (append analyzed (list (state-tree parse)))
                (rest unanalyzed)
                (state-input parse)))))
```

A Closer Look at the Calling Sequence

```
SSP(18): (parse '(kim adored snow) 's)
parse(): input: (KIM ADORED SNOW); goal: S
  parse(): input: (KIM ADORED SNOW); goal: NP
    parse(): input: (KIM ADORED SNOW); goal: KIM
    parse(): input: (KIM ADORED SNOW); goal: SANDY
    parse(): input: (KIM ADORED SNOW); goal: SNOW
  parse(): input: (ADORED SNOW); goal: VP
    parse(): input: (ADORED SNOW); goal: V
      parse(): input: (ADORED SNOW); goal: LAUGHED
      parse(): input: (ADORED SNOW); goal: ADORED
    parse(): input: (ADORED SNOW); goal: V
      parse(): input: (ADORED SNOW); goal: LAUGHED
      parse(): input: (ADORED SNOW); goal: ADORED
    parse(): input: (SNOW); goal: NP
      ...
```

Quantifying the Complexity of the Parsing Task



Kim adores snow (in Oslo)^n

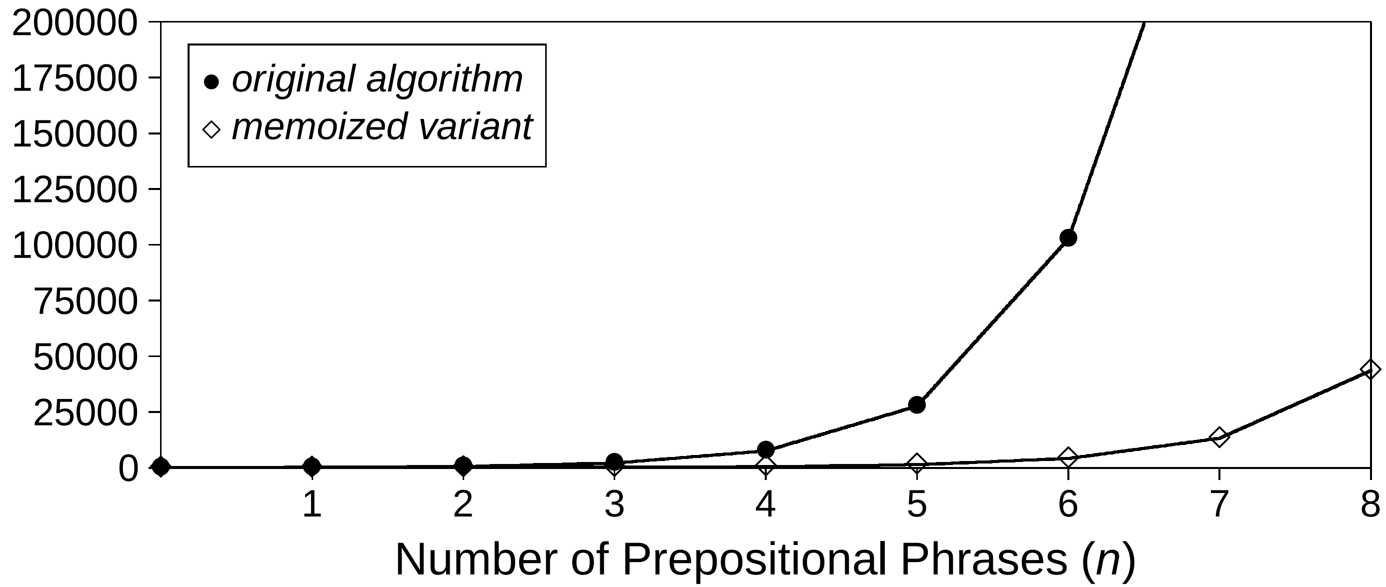
n	**trees**	**calls**
0	1	46
1	2	170
2	5	593
3	14	2,093
4	42	7,539
5	132	27,627
6	429	102,570
7	1430	384,566
8	4862	1,452,776
⋮	⋮	⋮

Memoization: Remember Earlier Results

Dynamic Programming

- The function call `(parse (adored snow) V)` executes two times;
- *memoization* — record `parse()` results for each set of arguments;

→ requires abstract data type, efficient indexing on *input* and *goal*.



Top-Down vs. Bottom-Up Parsing

Top-Down (Goal-Oriented)

- Left recursion (e.g. a rule like 'VP $\rightarrow$ VP PP') causes infinite recursion;
- grammar conversion techniques (eliminating left recursion) exist, but will typically be undesirable for natural language processing applications;

$\rightarrow$ assume bottom-up as basic search strategy for remainder of the course.

Bottom-Up (Data-Oriented)

- unary (left-recursive) rules (e.g. 'NP $\rightarrow$ NP') would still be problematic;
- lack of parsing goal: compute all possible derivations for, say, the input *adores snow*; however, it is ultimately rejected since it is not sentential;
- availability of partial analyses desirable for, at least, some applications.

A Bottom-Up Variant (1 of 2)

- Work upwards from string; successively combine words or phrases into larger phrases;
- use all grammar rules that have the (currently) next input word as β_1 in their RHS;
- recursively attempt to instantiate the remaining part of each rule RHS (β_i; $2 \leq i \leq n$);
- when a rule $\alpha \rightarrow \beta_i^+$ has been completely instantiated, attempt all rules starting in α;
- for each (remaining) input (suffix), derive all trees that span a prefix or all of the input.

```
(defun parse (input)
  (when input
    (loop
        for rule in (rules-starting-in (first input))
        append (instantiate (rule-lhs rule)
                            (list (first (rule-rhs rule)))
                            (rest (rule-rhs rule))
                            (rest input)))))
```

A Bottom-Up Variant (2 of 2)

```
(defun instantiate (lhs analyzed unanalyzed input)
  (if (null unanalyzed)
    (let ((tree (make-tree :root lhs :daughters analyzed)))
      (cons (make-state :tree tree :input input)
            (loop
                for rule in (rules-starting-in lhs)
                append
                  (instantiate (rule-lhs rule)
                               (list tree)
                               (rest (rule-rhs rule))
                               input))))
    (loop
        for state in (parse input)
        when (equal (tree-root (state-tree state))
                    (first unanalyzed))
        append (instantiate lhs
                            (append analyzed (list (state-tree state)))
                            (rest unanalyzed)
                            (state-input state)))))
```