SPUD: Integrierte Satzplanung & Realisierung

Proseminar “Generierung”

Alexander Koller

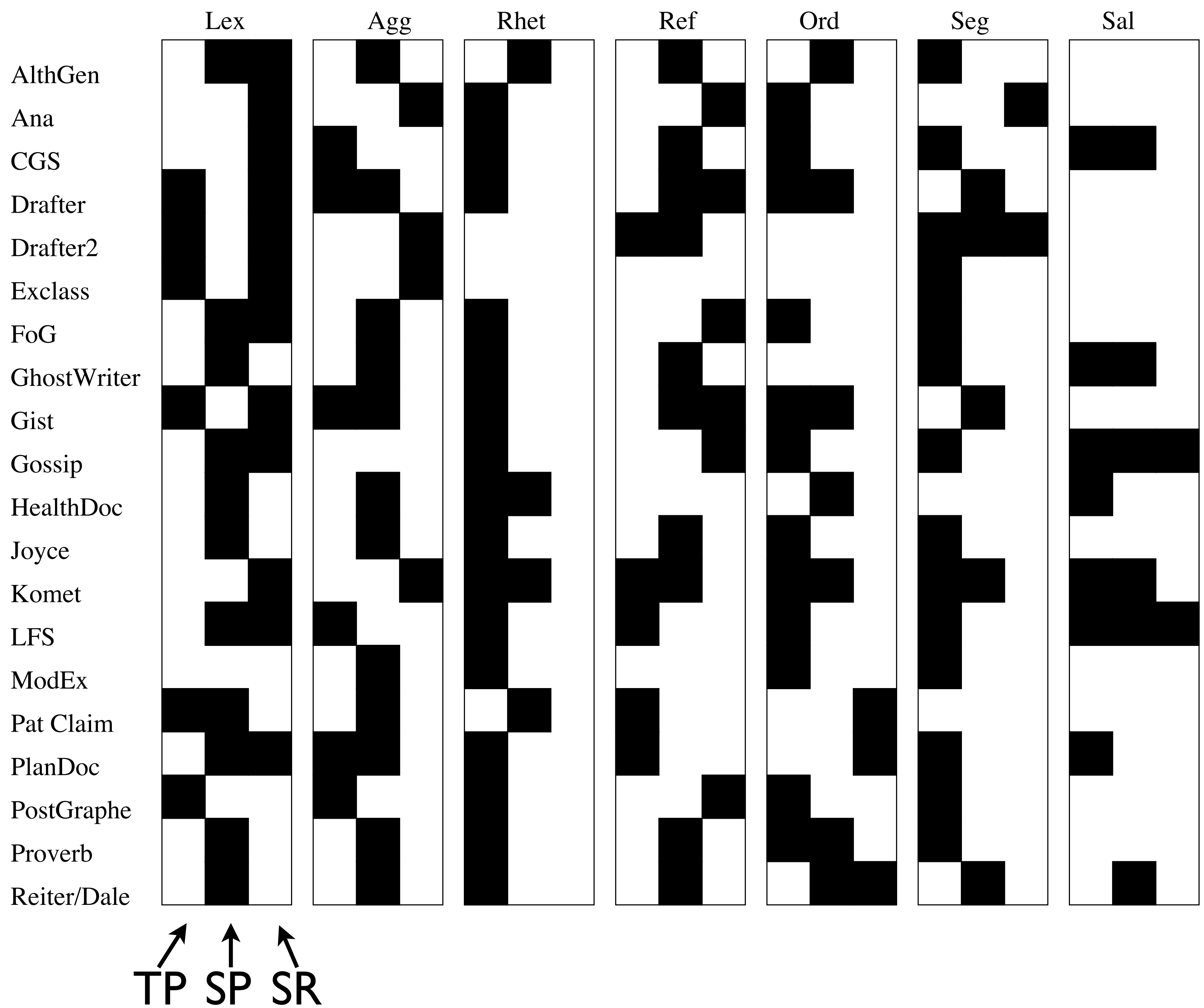
21. Januar 2011

The NLG pipeline

- Most generation systems go through the "pipeline" sequentially:
  - text planning → sentence planning → surface realization
- In terms of "goals" and "choices": Make all choices of a certain type first.
- What is sentence planning?
  - somewhat artificial concept

Tasks vs. pipeline stages

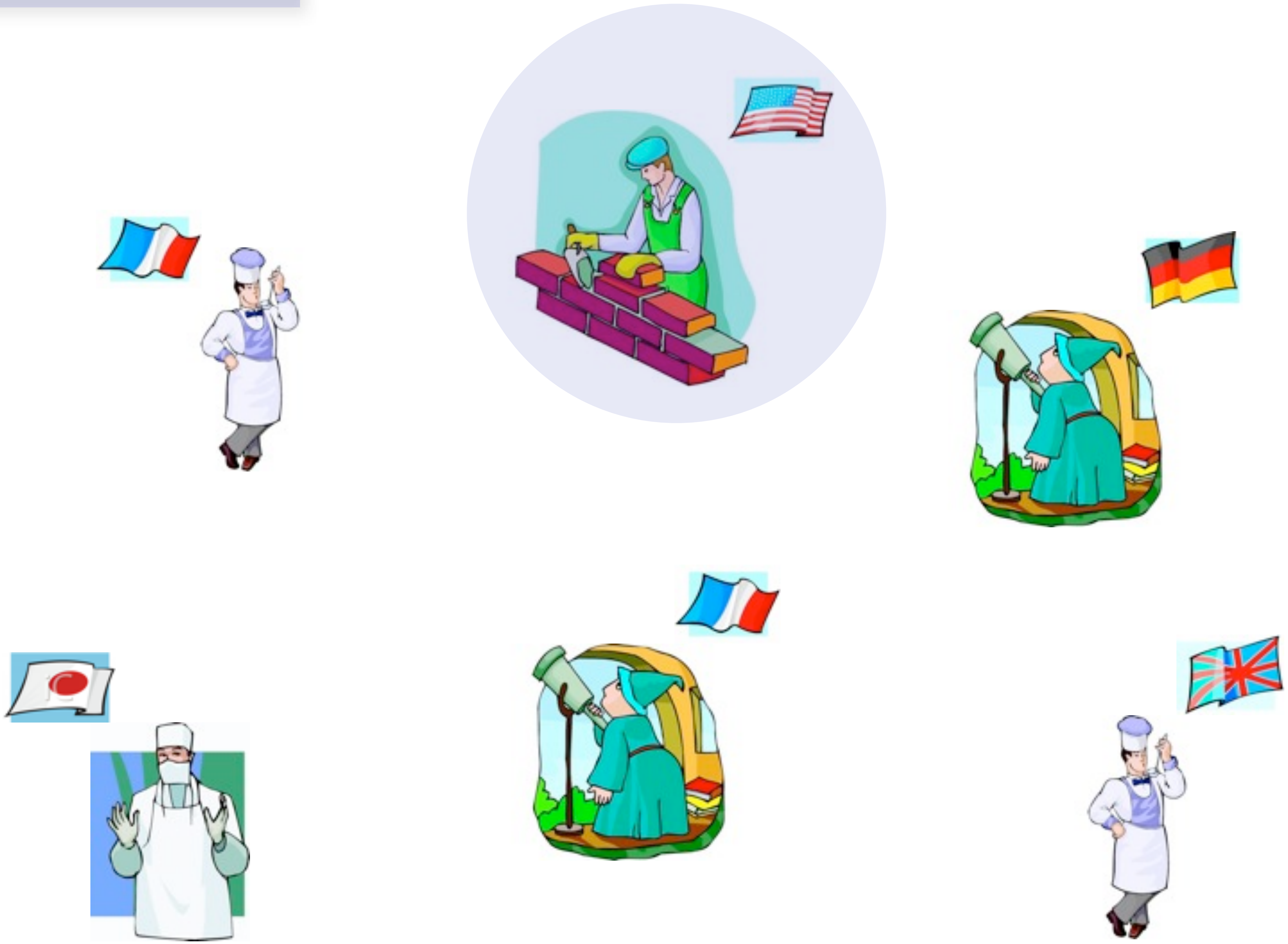
(Mellish et al. 06, RAGS report)



Grammar in RE generation

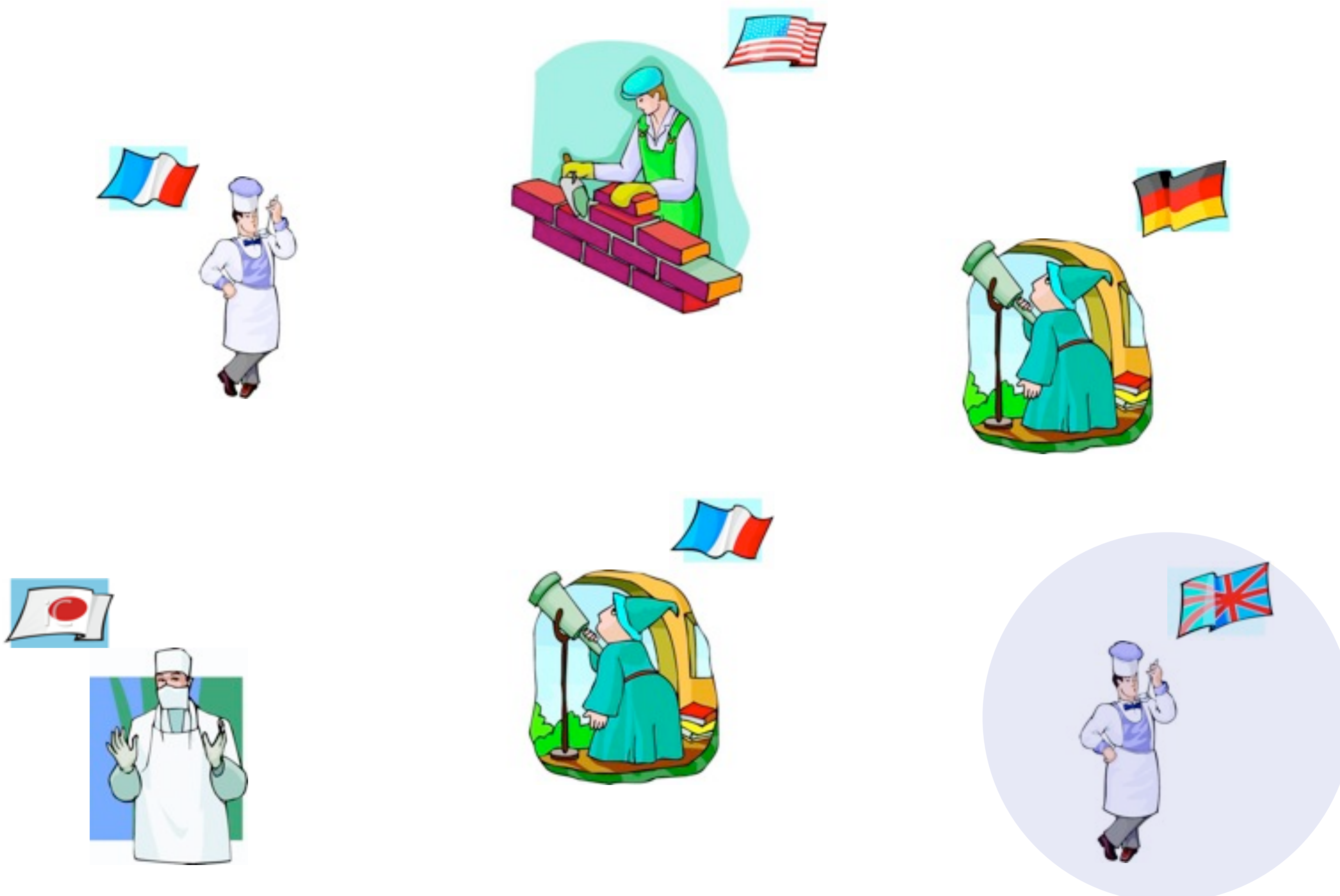
Grammar in RE generation

the bricklayer



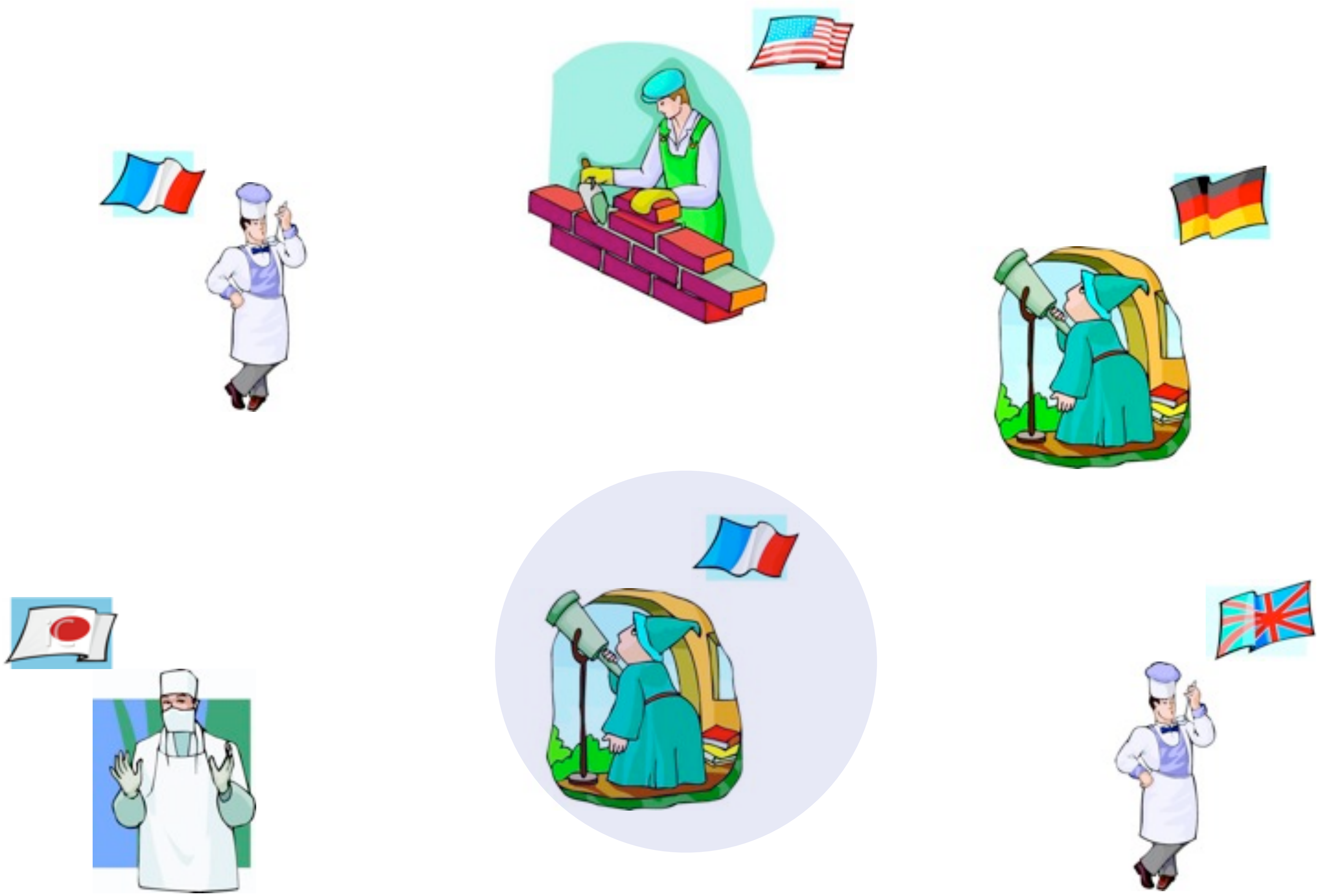
Grammar in RE generation

the Englishman



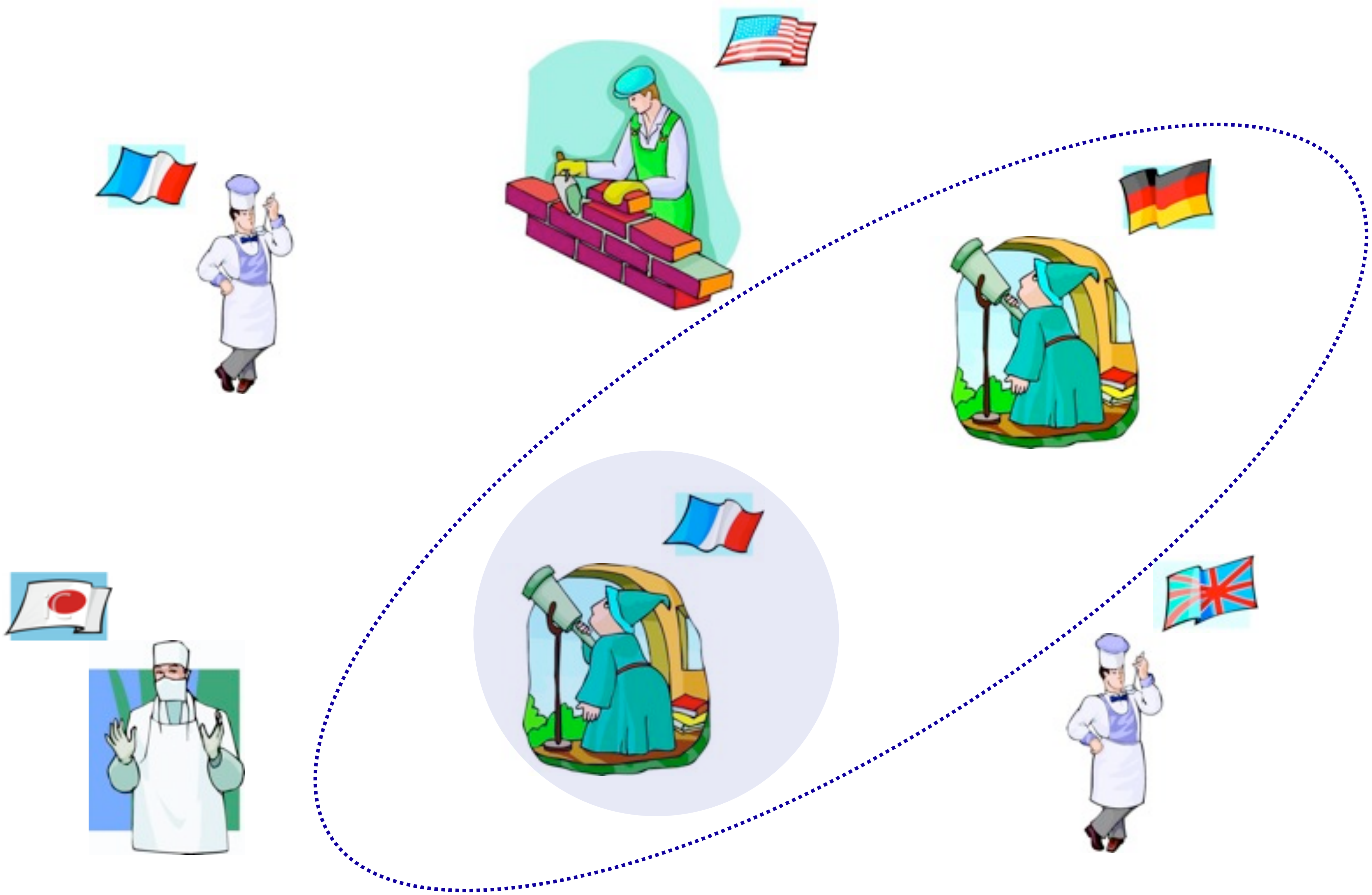
Grammar in RE generation

the NOUN



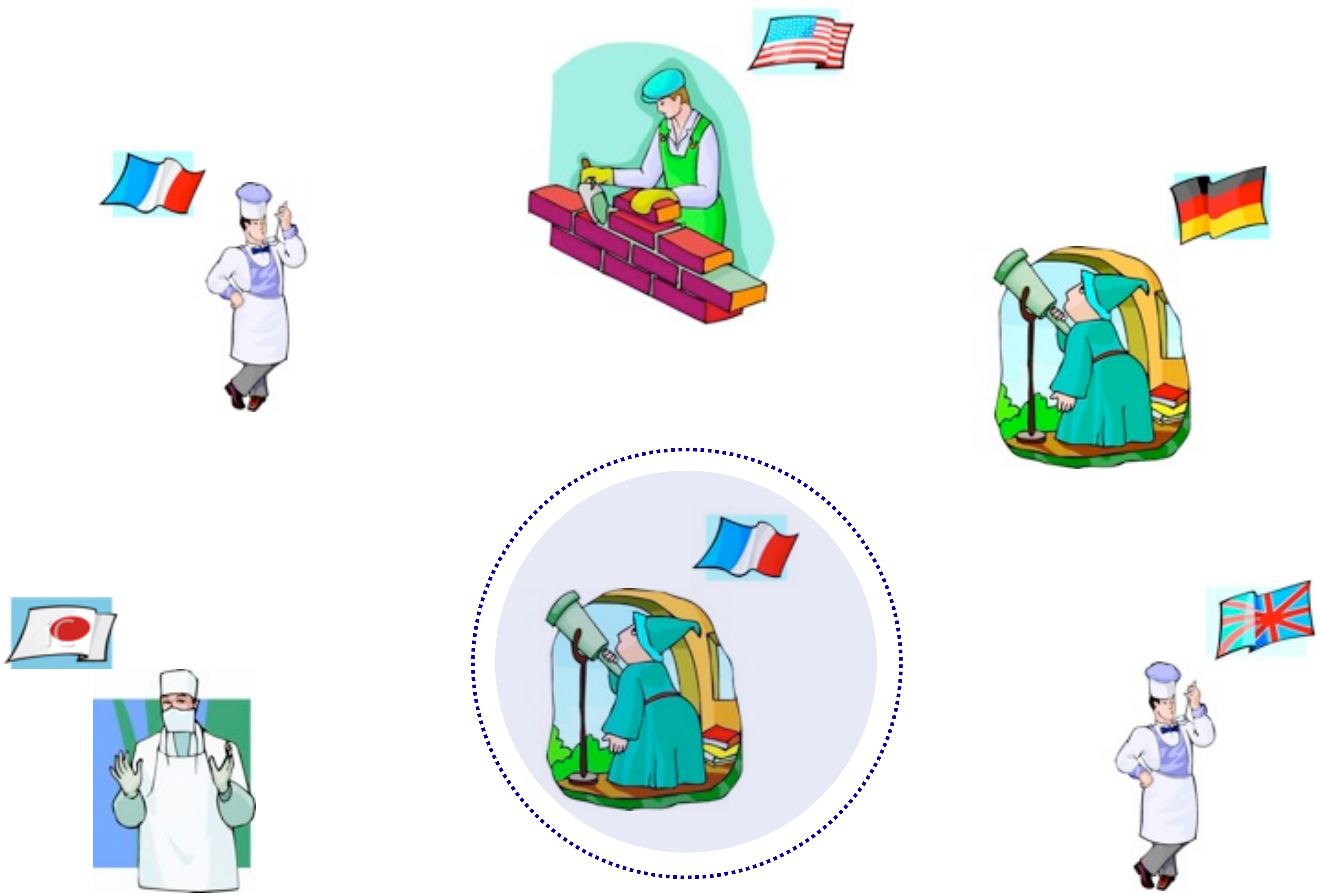
Grammar in RE generation

the ADJECTIVE astronomer



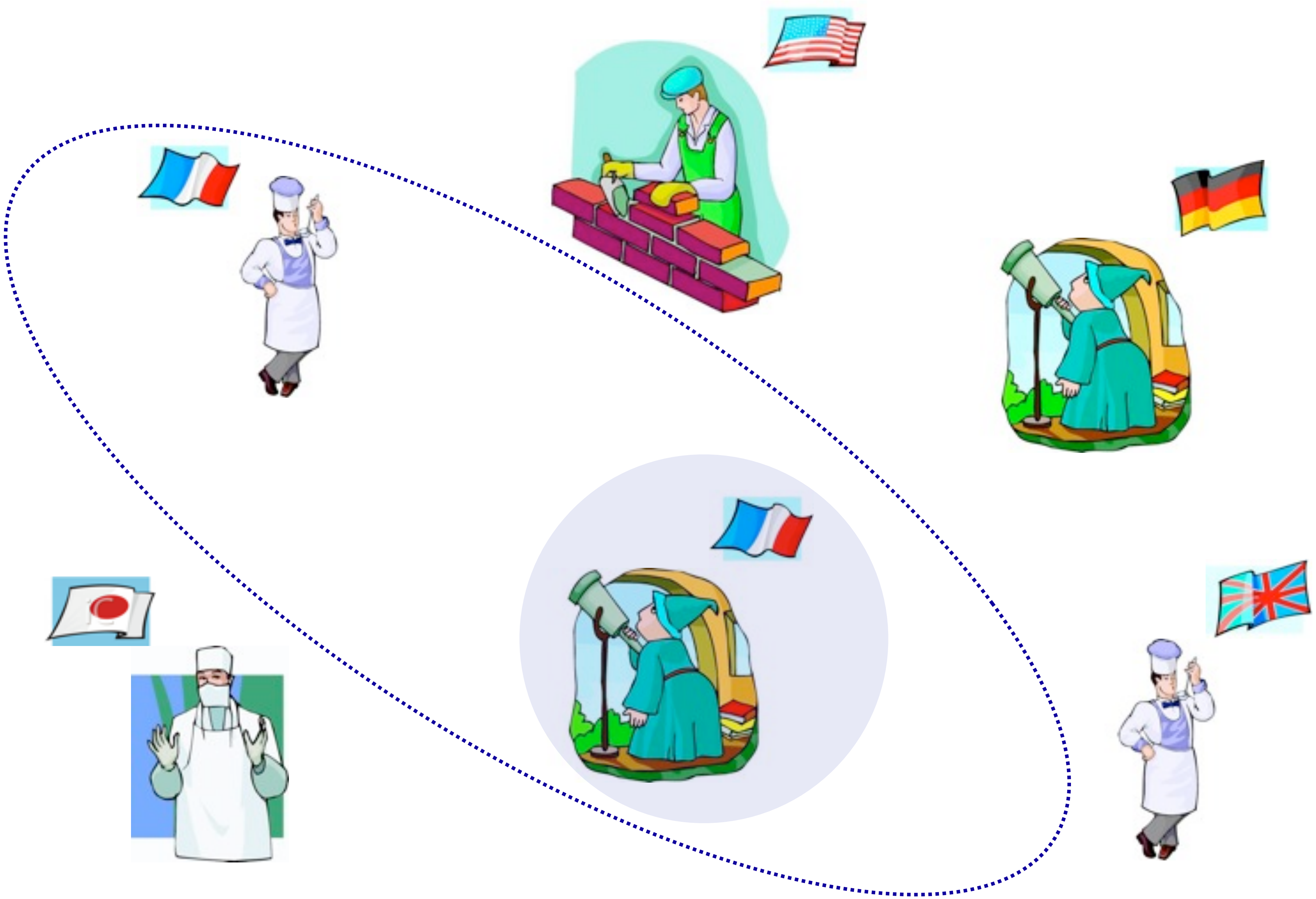
Grammar in RE generation

the French astronomer ✓



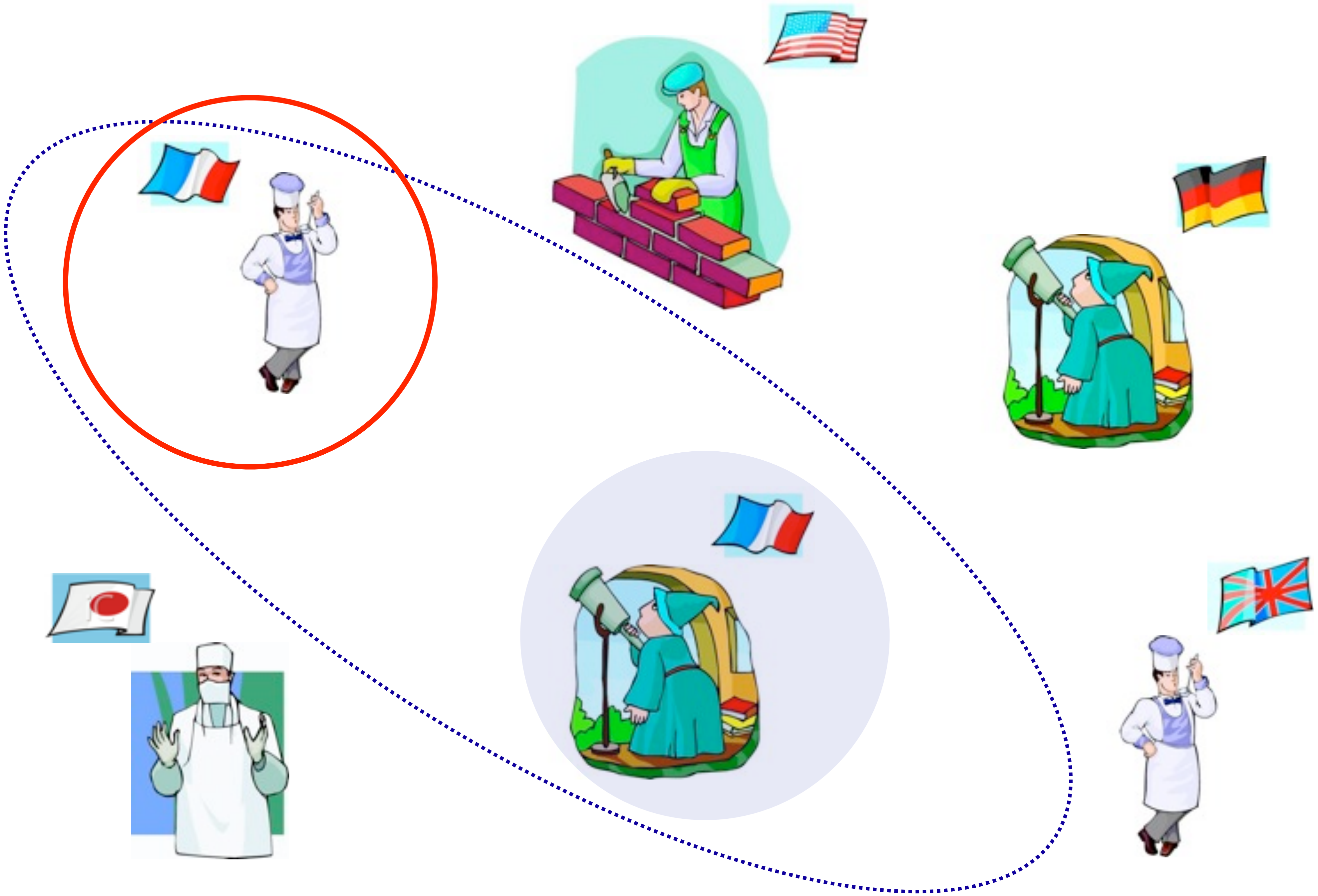
Grammar in RE generation

the ADJECTIVE Frenchman



Grammar in RE generation

the ??? Frenchman



RE generation vs. realization

the bricklayer and the doctor

bricklayer ⊔ doctor

?

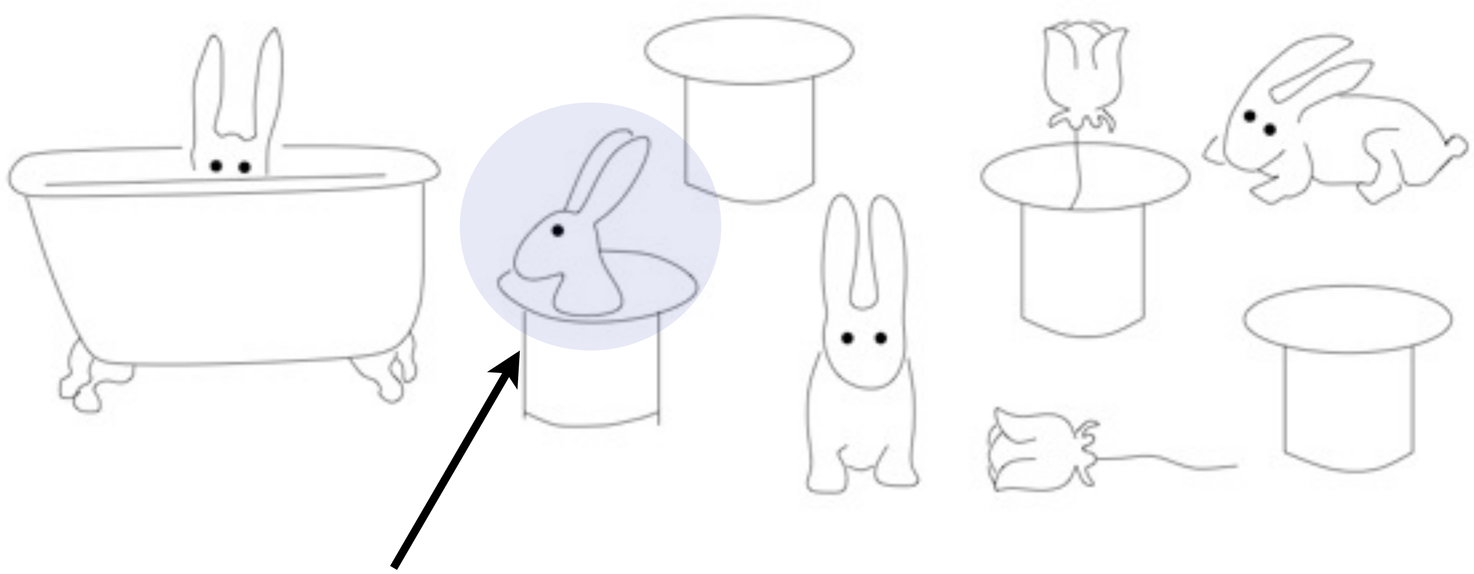
the non-Europeans

¬ european

Interacting REs

(Stone & Webber 98)

- Referring expressions can constrain each other:



What do you call this guy?

Lessons from the example

- Referring expressions generation
  - is typically subsumed under sentence planning
  - requires access to grammatical resources
  - but theoretically, only realizer should see grammar

- RE that looks good to sentence planner might be bad or impossible for realizer.

- How to keep SP and SR resources synchronized?

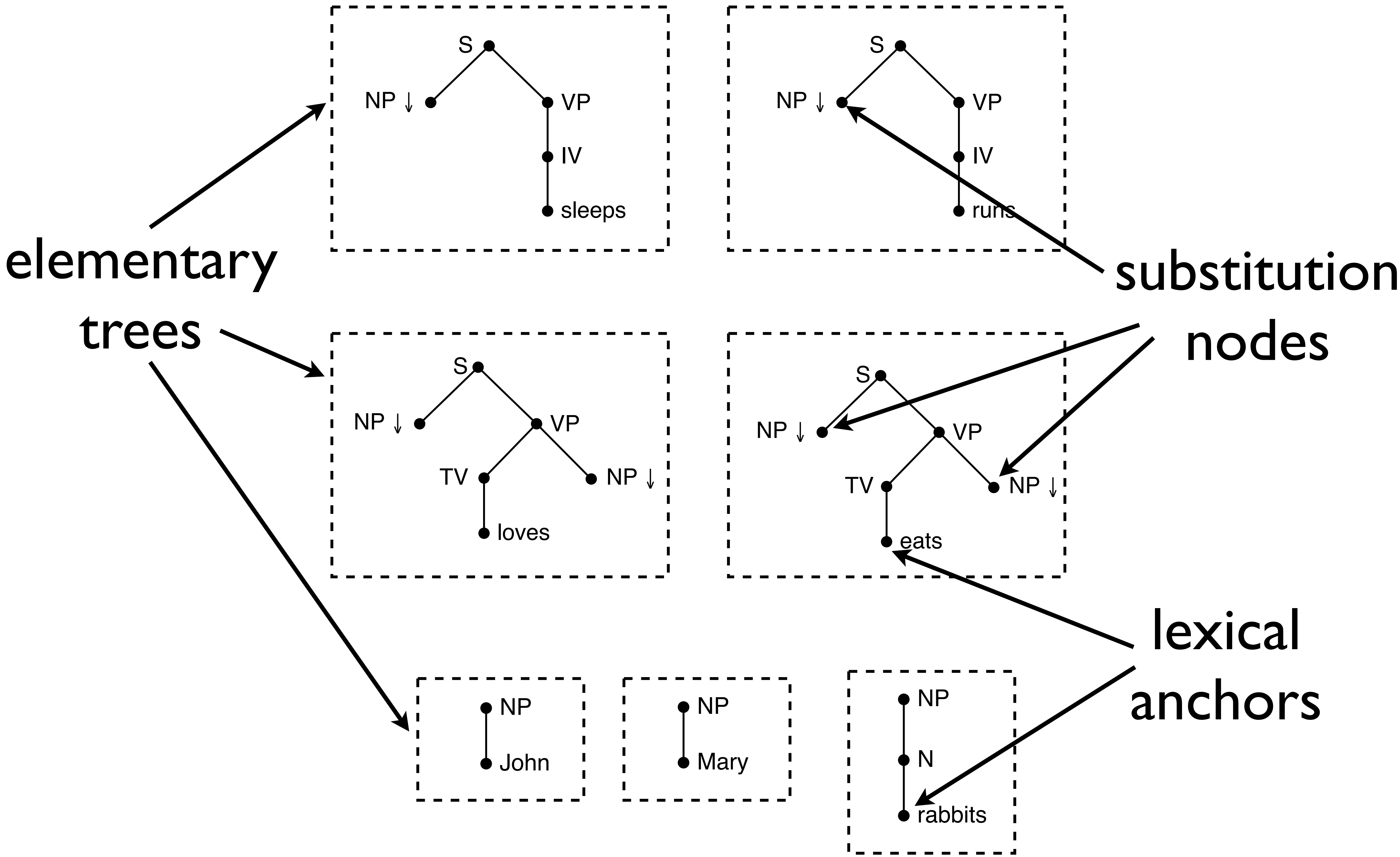
Today

- Separation between sentence planning and surface realization is artificial.
- Tree-Adjoining Grammars (TAG).
- SPUD: Integrated sentence planning and surface realization based on TAG.

Tree-adjoining grammars

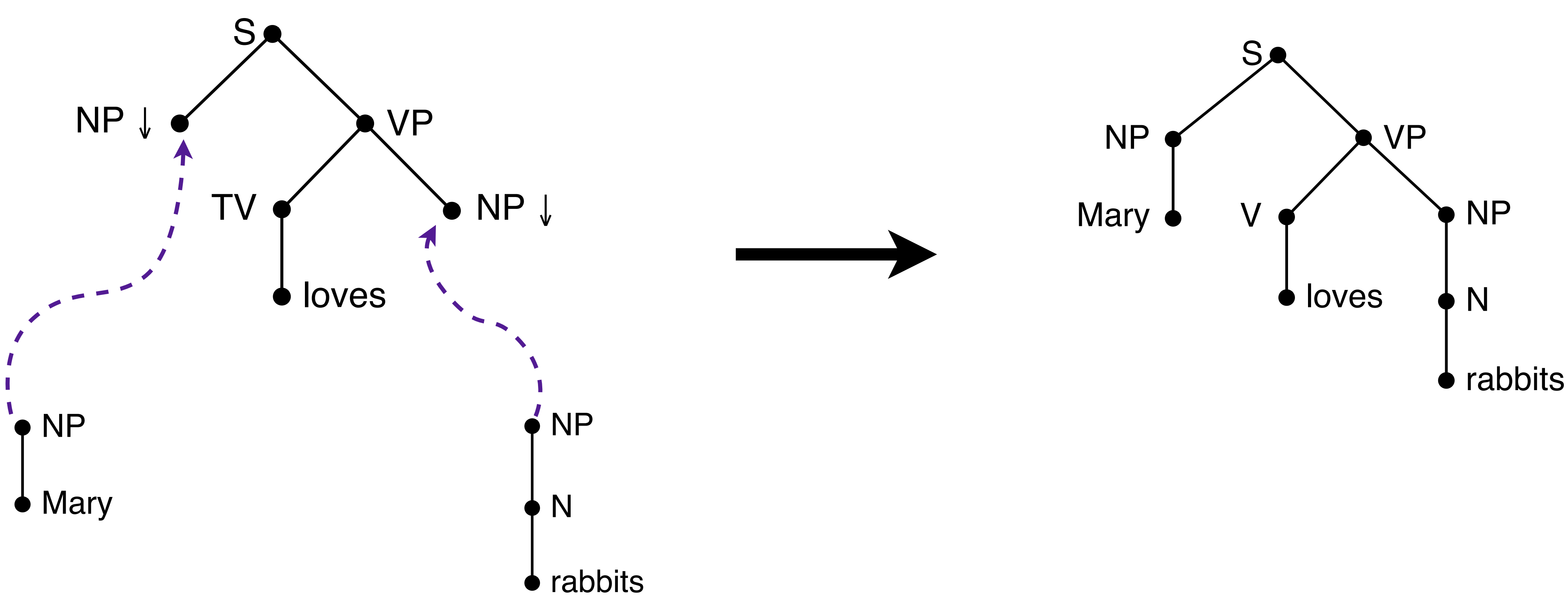
- Lexicalized grammar formalism, invented by Aravind Joshi.
- Idea: Build syntactic derivation by combining elementary trees by substitution and adjunction.
- Goes beyond context-free expressive power.

TAG: A grammar, part 1

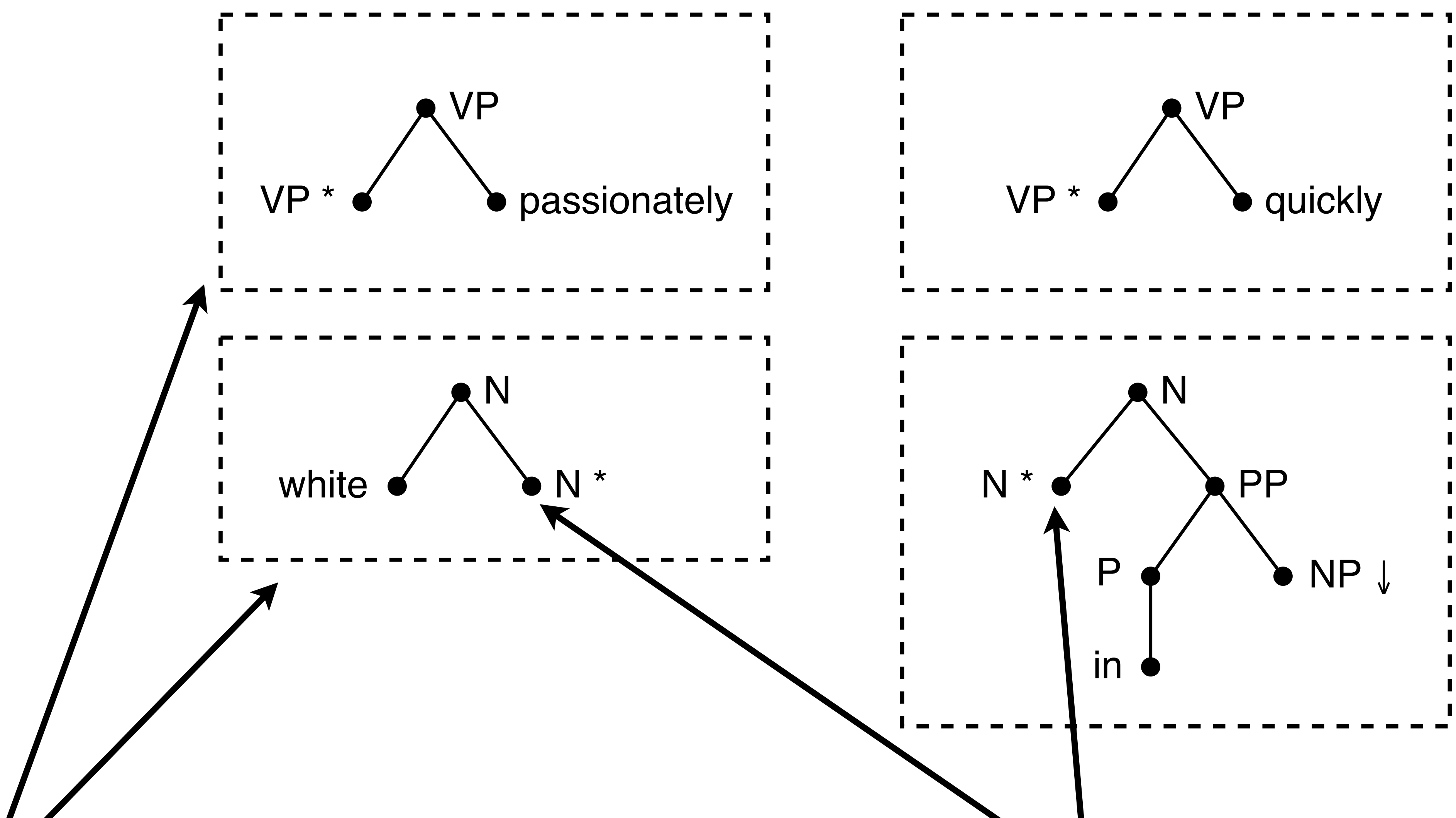


Substitution

- Elementary trees can be combined by substituting a substitution node with another elementary tree:



Grammar, Part 2

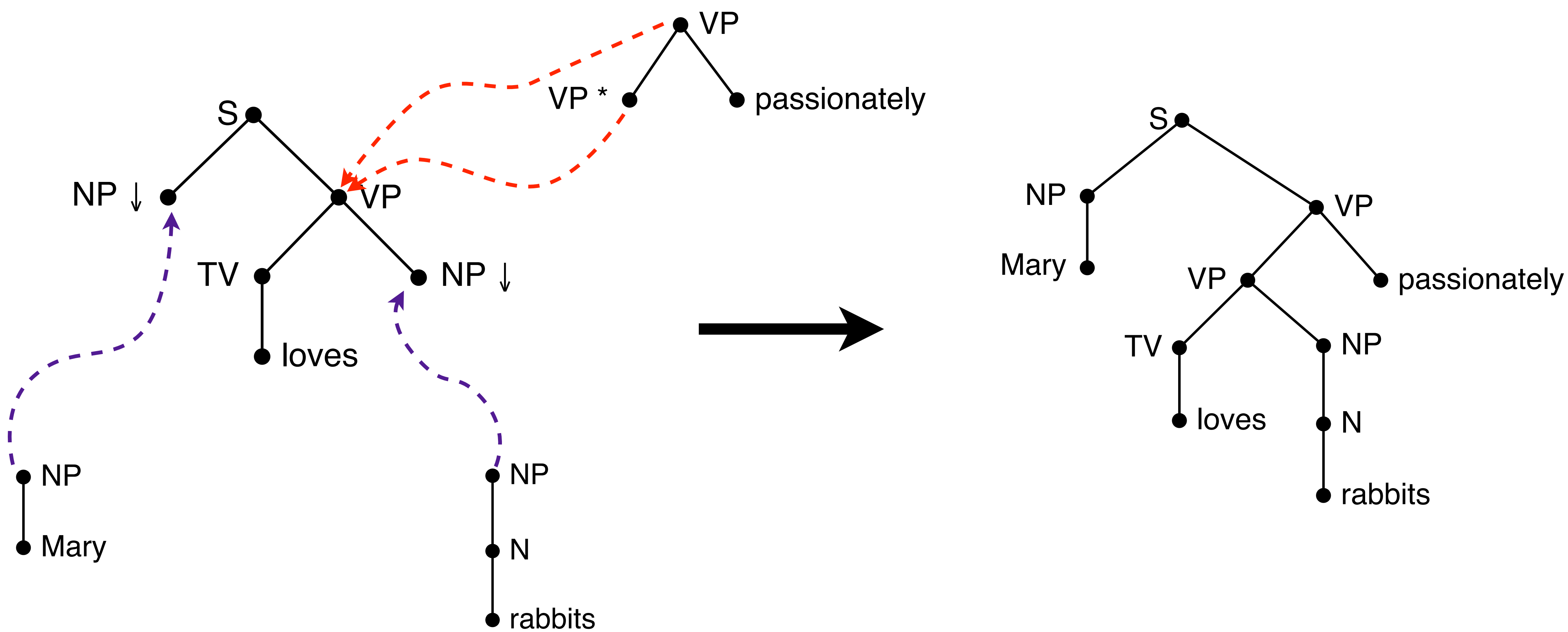


elementary trees like these are called auxiliary trees

because they have foot nodes

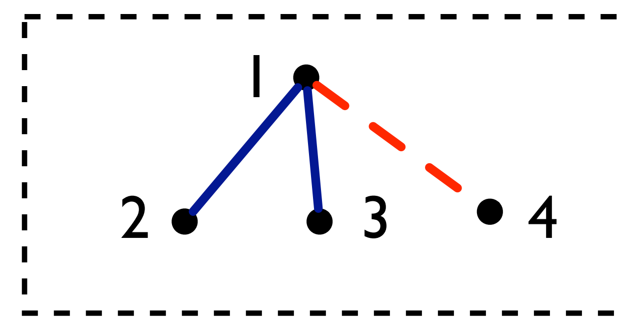
Adjunction

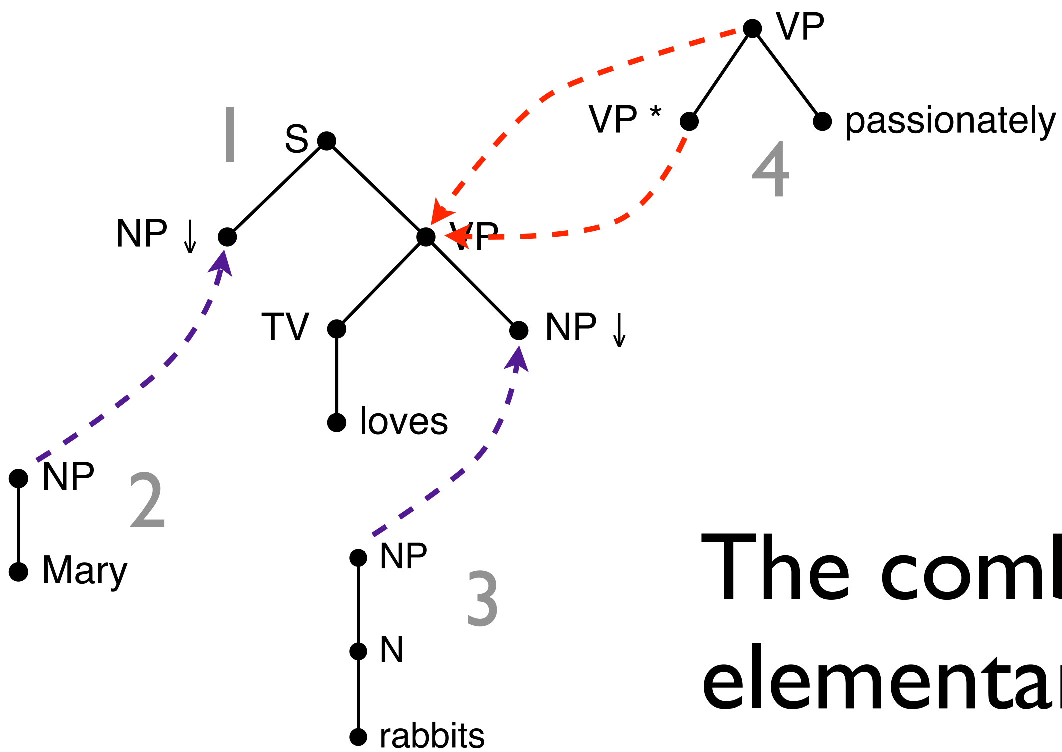
- Auxiliary trees can be adjoined into nodes of other elementary trees.



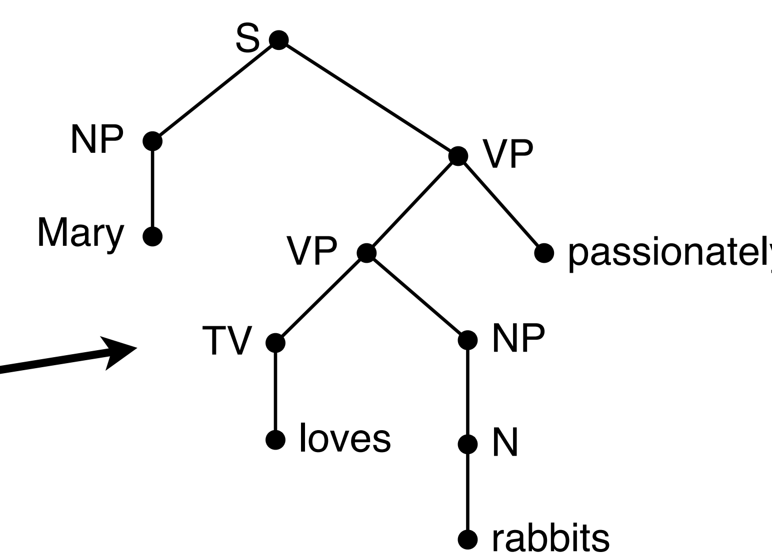
Derivation trees

- Record the structure of a TAG derivation in a derivation tree:



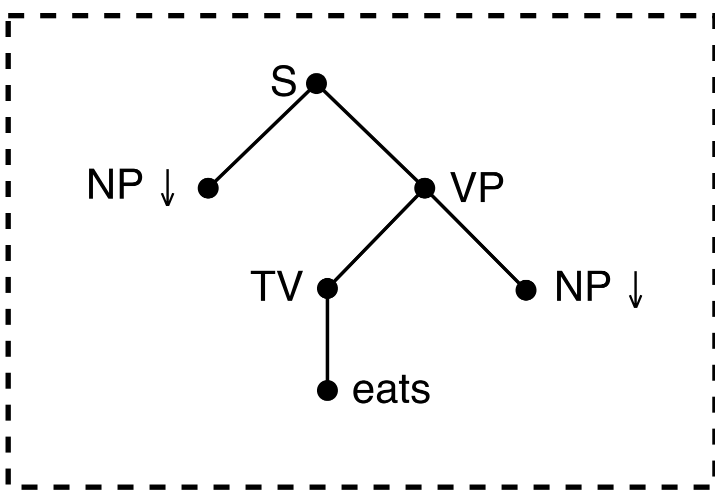


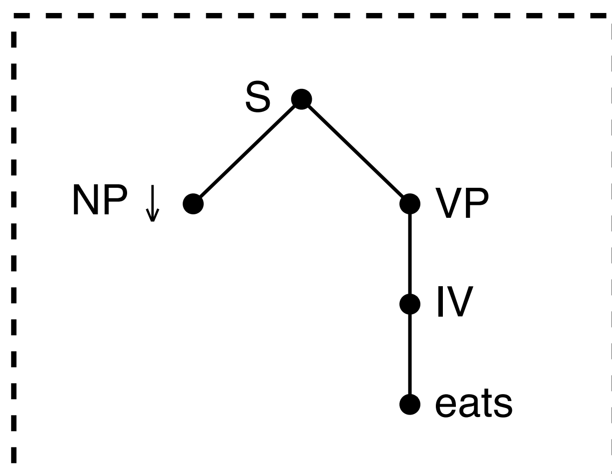
The combination of the elementary trees is called the derived tree:

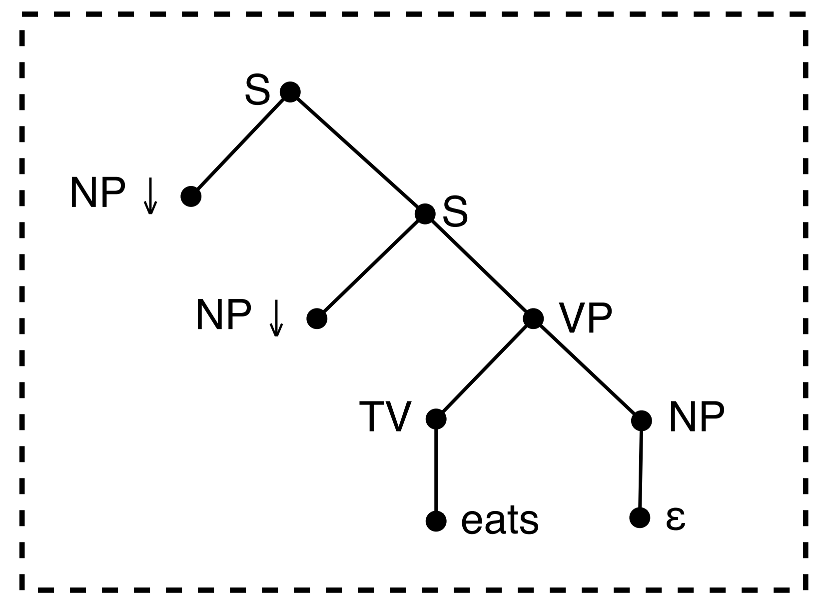


Lexical ambiguity in TAG

- Words in TAG highly ambiguous because e-trees contain so much grammatical context:





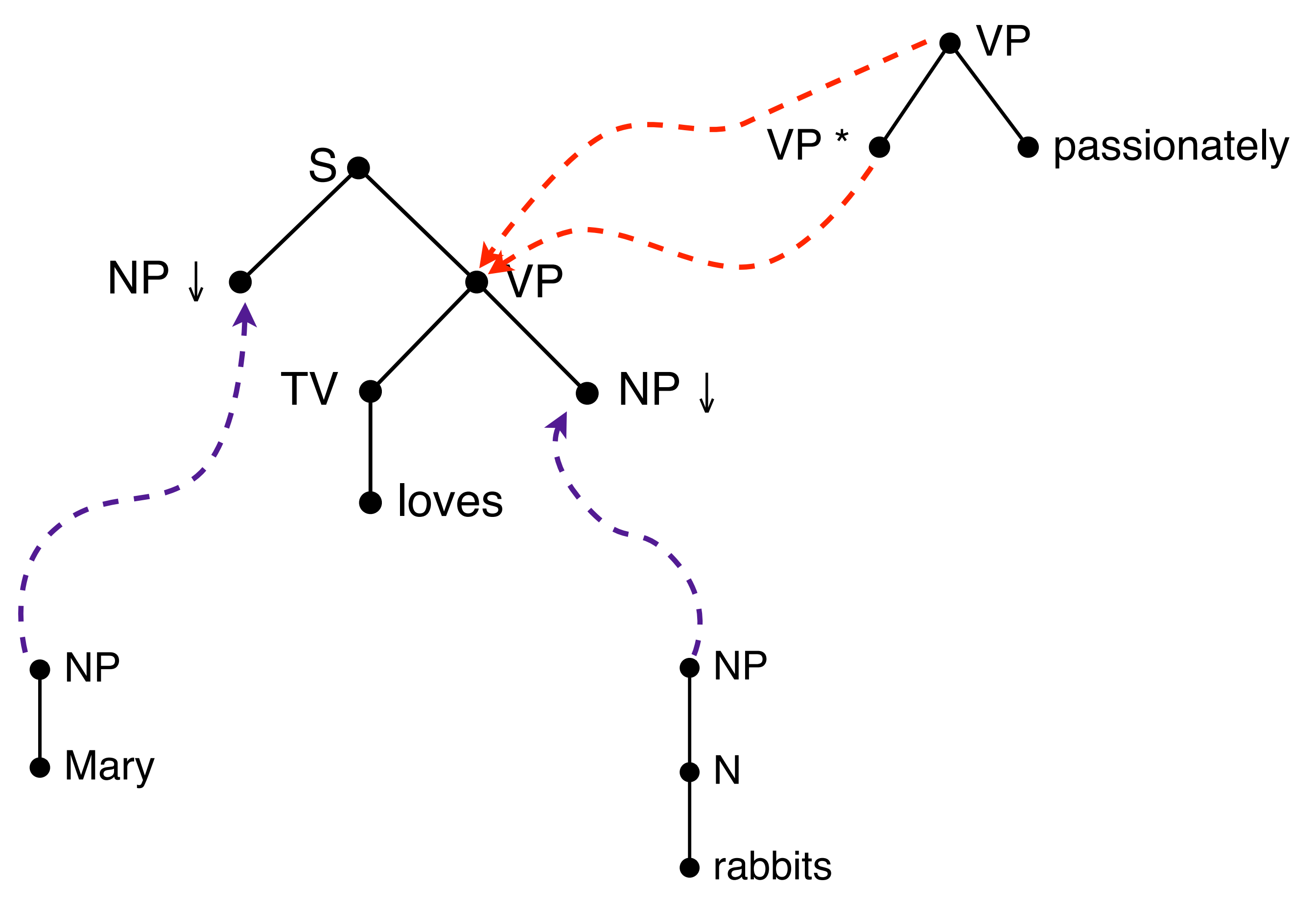


TAG: Summary

- Spell out grammatical use of each word in an elementary tree ("extended domain of locality").
- Two-sided adjunction makes TAG more expressive than context-free grammars.
- "Mildly context-sensitive" grammar formalism; can be parsed in time $O(n^6)$.

TAG in generation

- NLG is about goals and choices.
- In TAG: choice = selection of e-tree to add to the current derivation, top-down.

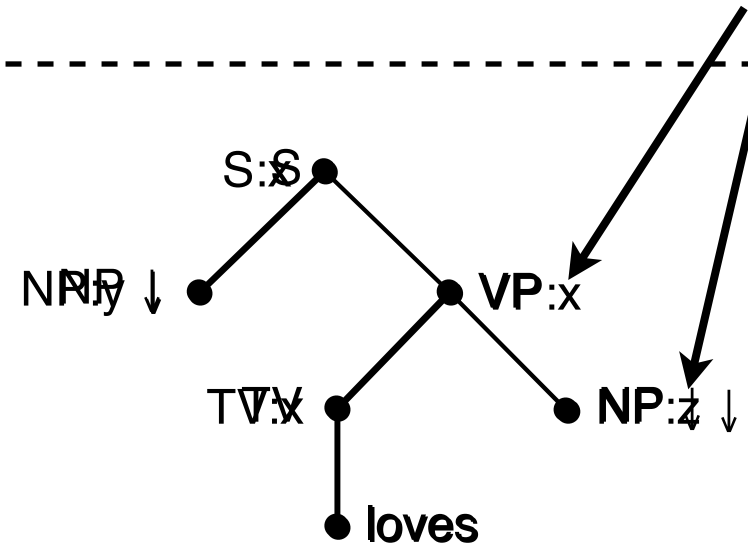


SPUD

- SPUD (Matthew Stone, late 90's):
  - equip e-trees with semantic and pragmatic information
  - use this information to drive top-down TAG generation
  - heuristic search for a complete derivation
- This solves (some) sentence planning and realization at the same time.
- Different versions: Stone & Doran, ACL 98; Stone et al., Computational Intelligence 03.

SPUD: Lexicon entries

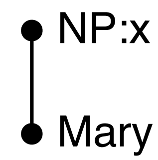
semantic indices



semantic content: {loves(x,y,z)}

semantic condition: {animate(y)}

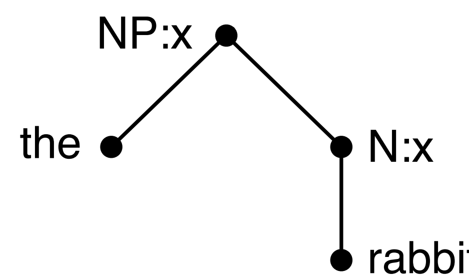
pragmatic condition: nothing



sem. cont.: {name(x,Mary)}

sem. cond.: nothing

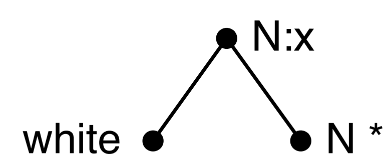
prag. cond.: nothing



sem. cont.: {rabbit(x)}

sem. cond.: nothing

prag. cond.: {unique-id(x), discourse-old(x)}

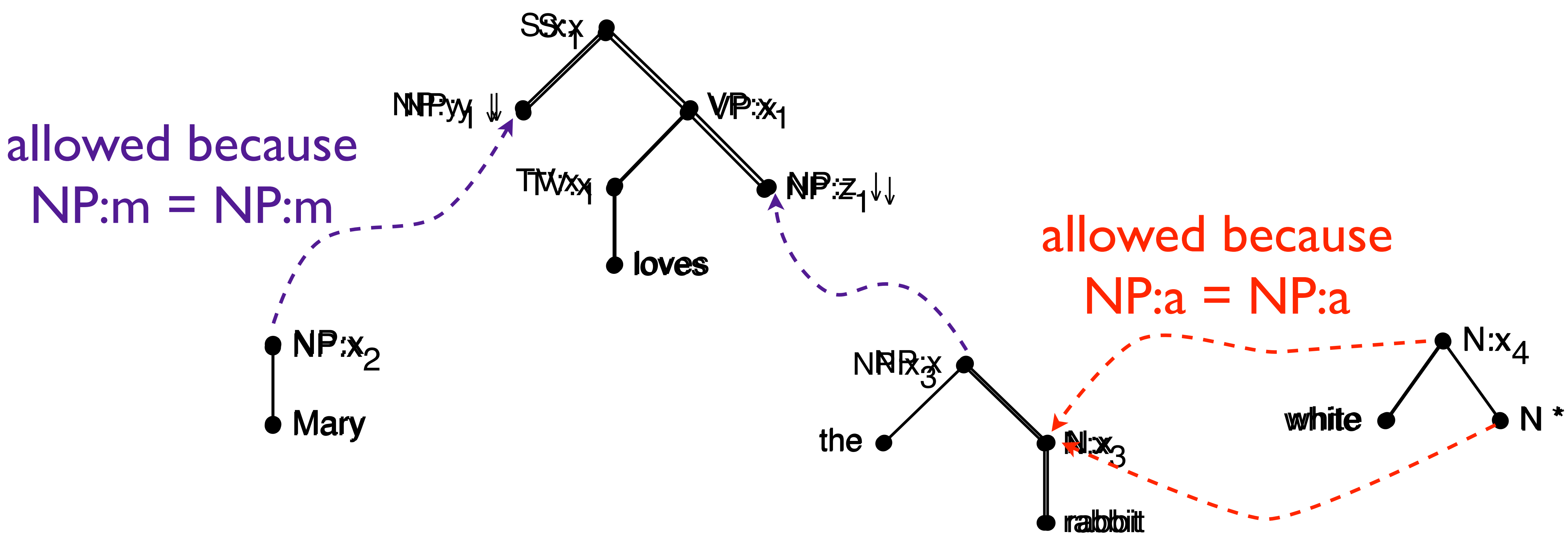


sem. cont.: {white(x)}

sem. cond.: nothing

prag. cond.: nothing

Combining tree instances



substitution: $\{e/x_1, m/y_1, m/x_2, a/z_1, a/x_3, a/x_4\}$

SPUD: Knowledge base

Semantic information:

a

b

{loves(e,m,a), name(m,Mary), rabbit(a), rabbit(b), white(a), brown(b), ...}

Pragmatic information:

{discourse-new(a), unique-id(a), ...}

Communicative goal: {communicate “loves(e,m,a)”}

Root node specification: generate an S:e

The SPUD search algorithm

- States consist of:
  - list of unsatisfied communicative goals
  - TAG derivation tree
  - substitution for semantic indices
  - "constraint network" to keep track of REs
- Initial state:
  - initial communicative goal
  - empty derivation, starting with an open substitution node according to the root node specification
  - empty substitution

The SPUD search algorithm

- Search step:
  - choose a new elementary tree from the lexicon
  - choose a substitution for the indices of this tree
  - if tree instance can be added to the derivation and the semantic and pragmatic conditions are satisfied, then add it to the derivation and update search state
- Repeat this step until in a goal state, i.e.:
  - all communicative goals expressed
  - derivation is grammatically complete
  - all REs are unique

The search heuristic

- In each step, select the single e-tree that is best according to the following criteria:
  - contribution to communicative goals
  - ambiguity of referring expressions
  - salience of individuals that are selected for indices
  - number of remaining flaws in syntactic derivation
  - specificity of semantic content
- SPUD uses a greedy search strategy:
  It never backtracks.

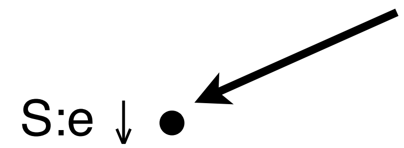
An example

KB: a  b  loves(e,m,a), name(m,Mary), d-new(a), unique-id(a)

CG: loves(e,m,a)

Root: S:e

distractors:



substitution: { }

An example

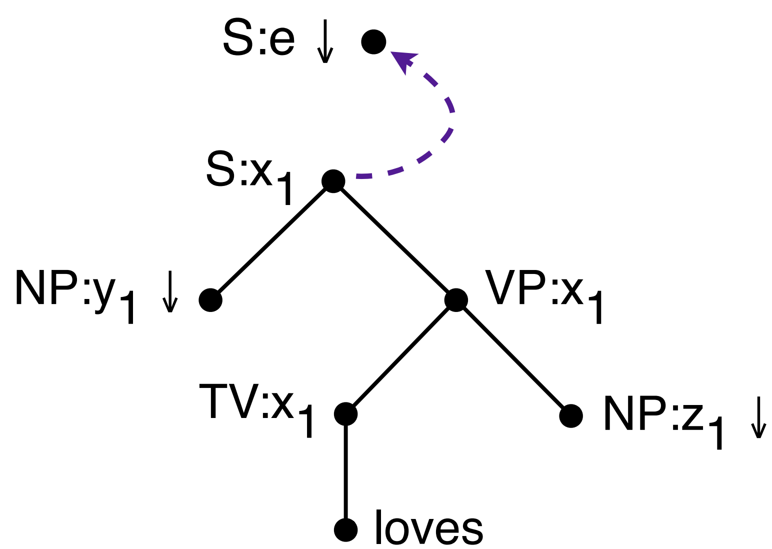
KB: 

loves(e,m,a), name(m,Mary), d-new(a), unique-id(a)

CG: loves(e,m,a) ✓

Root: S:e

distractors: y_1: {a,b,e}
z_1: {m,b,e}



semantic content: {loves(e,m,a)}

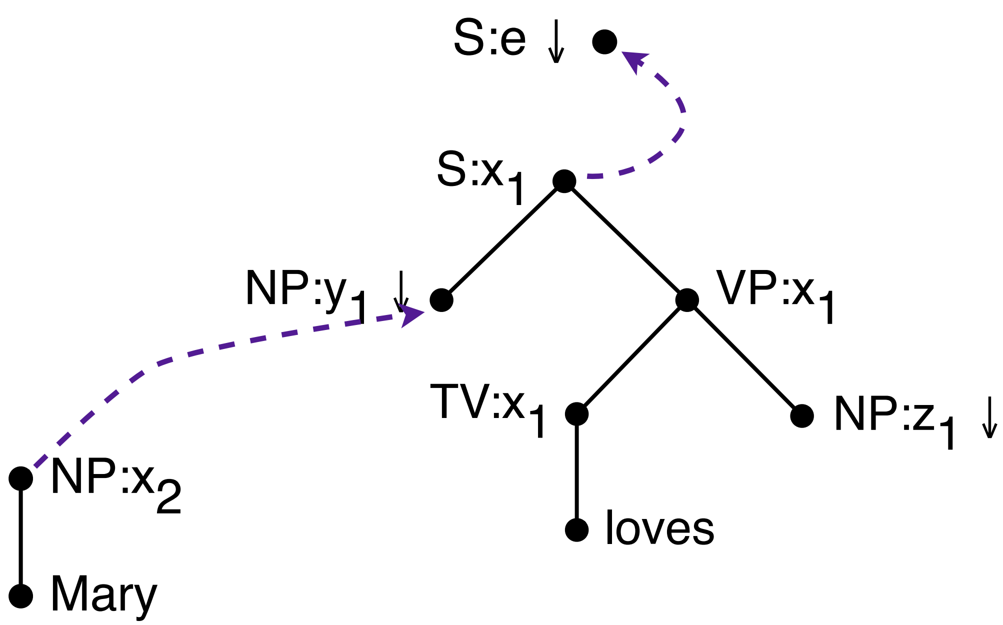
pragmatic condition: nothing

substitution: {e/x_1, m/y_1, a/z_1}

An example

KB: a  b  loves(e,m,a), name(m,Mary), d-new(a), unique-id(a)

CG: loves(e,m,a) ✓ Root: S:e distractors: y_1: {a,b,e}
z_1: {m,b,e}



sem. cont.: {name(m,Mary)}
prag. cond.: nothing

substitution: $\{e/x_1, m/y_1, a/z_1\}m/x_2\}$

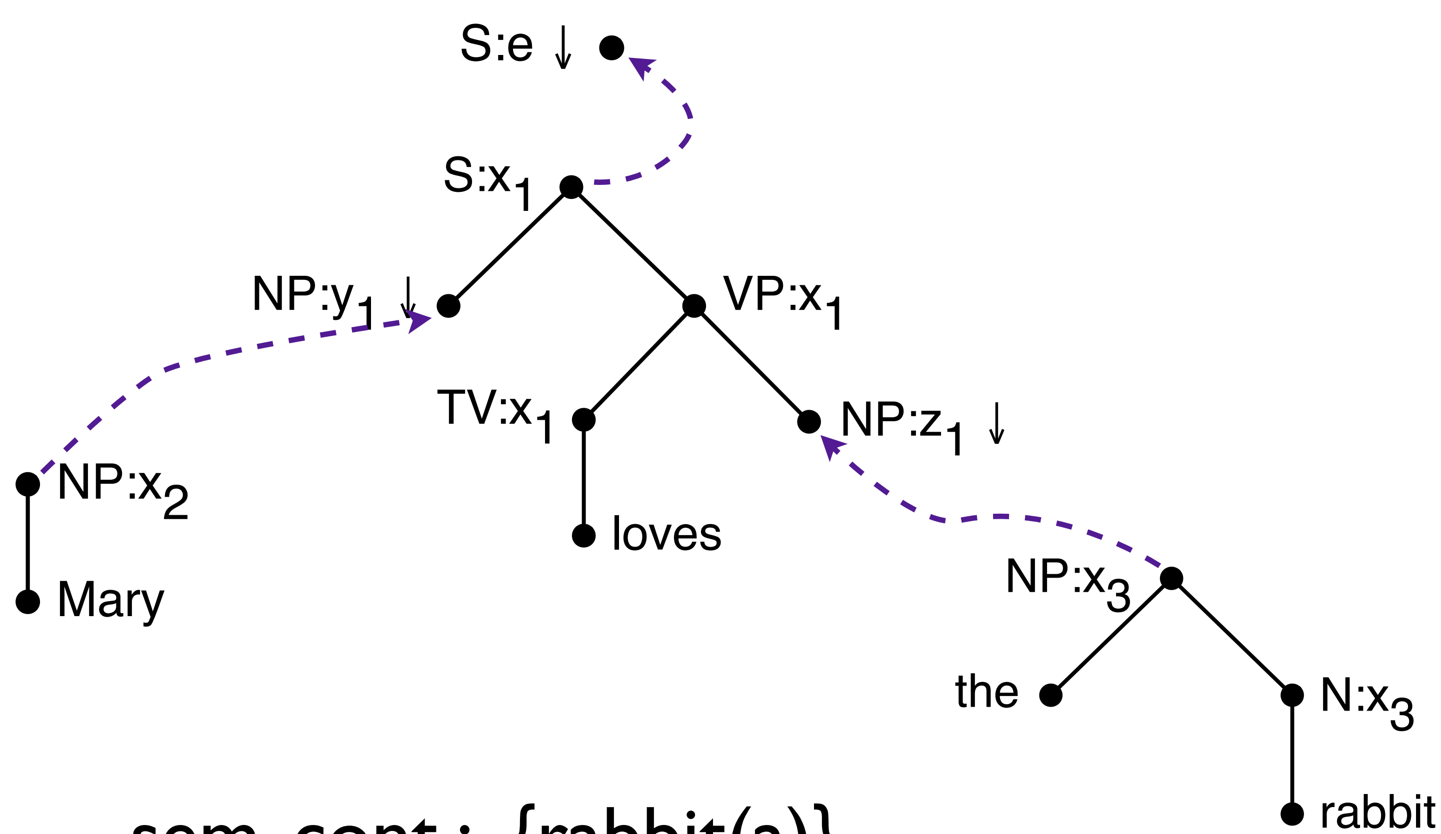
An example

KB: a  b  loves(e,m,a), name(m,Mary), d-new(a), unique-id(a)

CG: loves(e,m,a) ✓

Root: S:e

distractors: y_1: unique
z_1: {m,b,e}



sem. cont.: {rabbit(a)}

prag. cond.: {unique-id(a), discourse-new(a)}

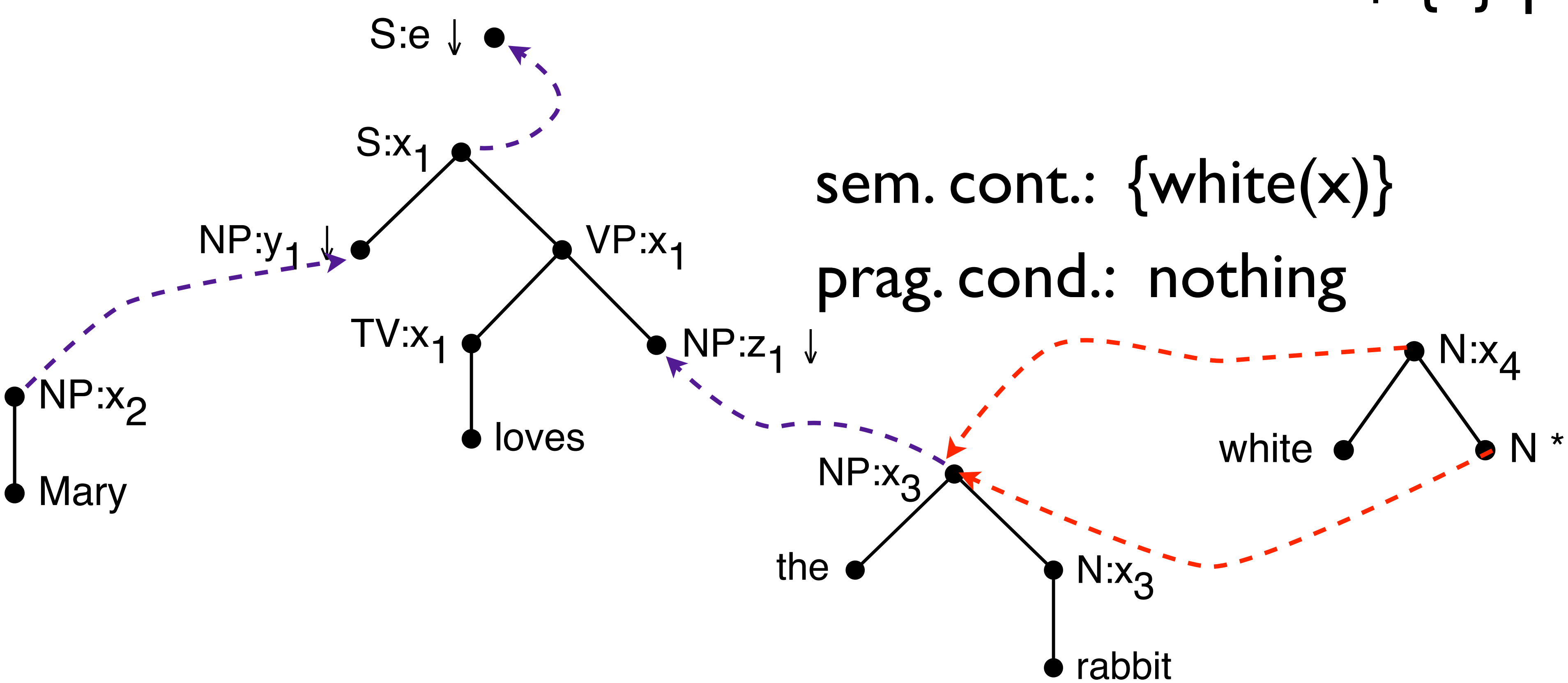
substitution: {e/x_1, m/y_1, a/z_1, m/x_2}, a/x_3}

An example

KB: a  b  loves(e,m,a), name(m,Mary), d-new(a), unique-id(a)

CG: loves(e,m,a) ✓ Root: S:e distractors: y_1: unique

z_1: {b} unique



sem. cont.: {white(x)}

prag. cond.: nothing

substitution: {e/x_1, m/y_1, a/z_1, m/x_2, a/x_3, a/x_4}

SPUD captures SR + SP

- Surface realization:
  - derivation is grammatical according to TAG grammar
  - semantic content subsumes communicative goal
  - semantic content of sentence is supported by the knowledge base

- Very "semantic" approach to surface realization: Input consists of semantic representations, not abstract syntax specifications.

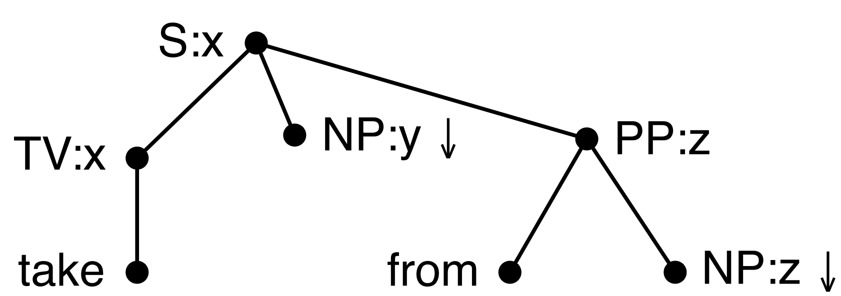
SPUD captures SR + SP

- Referring expressions ($\subset$ sentence planning):
  - definite descriptions are required to be unique
- Integration with realization
  - avoids problems mentioned in the introduction

Interacting REs

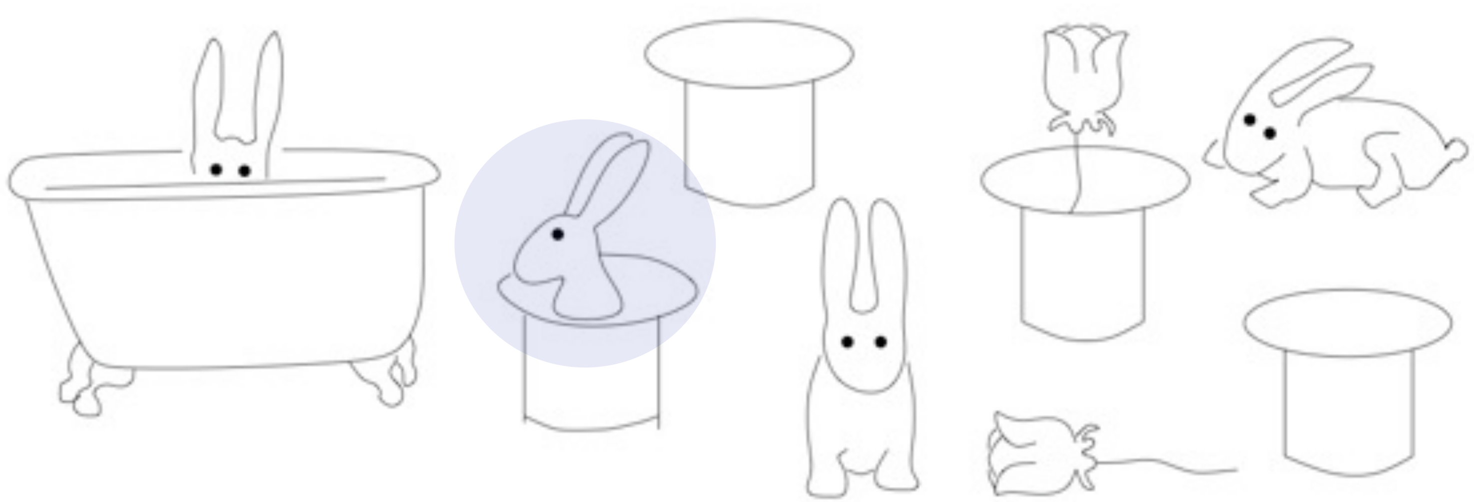
(Stone & Webber 98)

- Use semantic conditions to generate very succinct referring expressions:
  - "Take the rabbit from the hat"



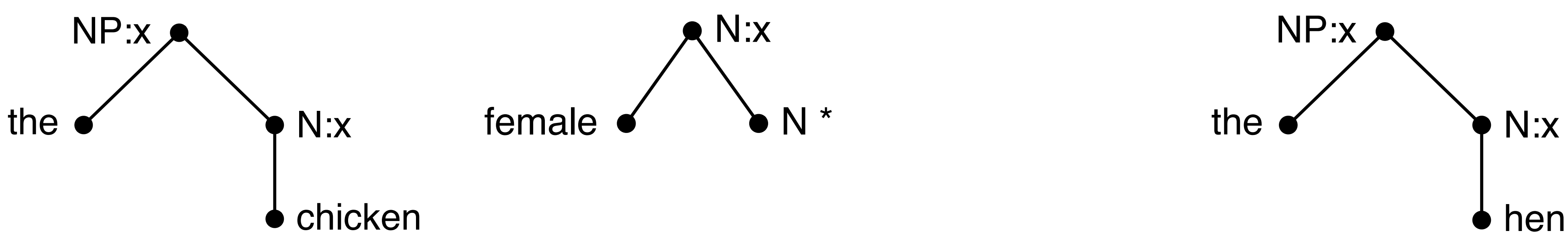
sem. cont.: {take(x,y,z)}

sem.cond.: {in(y,z)}



SPUD captures SR + SP

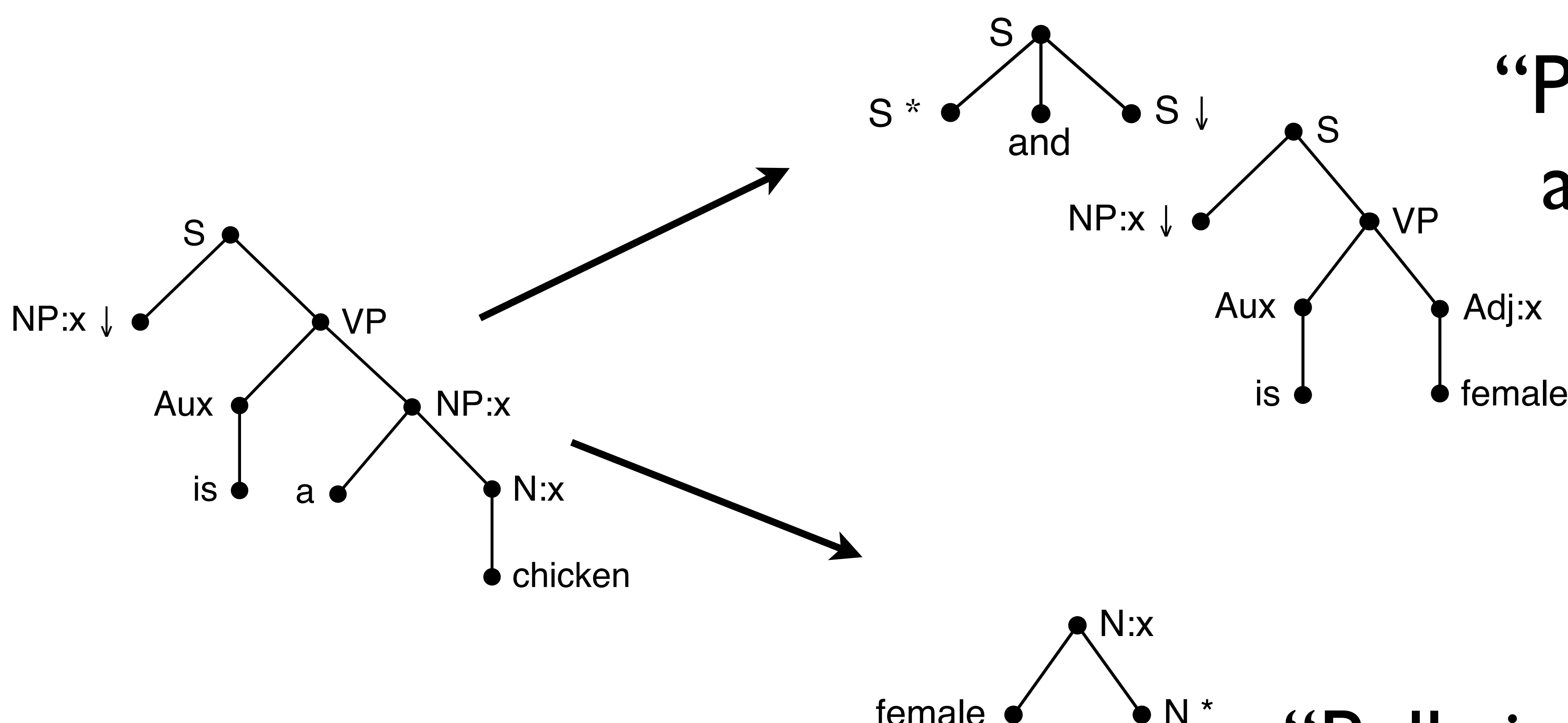
- Lexical choice (⊂ sentence planning):
  - encode with lexical ambiguity and semantic content



{chicken(x)} {female(x)} {chicken(x), female(x)}

SPUD captures SR + SP

- Aggregation (sometimes ⊂ sentence planning):
  - encode with lexical ambiguity and semantic content
  - aggregated and non-aggregated version are equally allowed



"Polly is a chicken and Polly is female"

"Polly is a female chicken."

Summary

- Separation of sentence generation into sentence planning and realization is somewhat artificial and dangerous.

- SPUD: Perform SP and SR in one step by
  - adding semantic + pragmatic info to TAG e-trees
  - performing greedy search for derivation based on syntactic, semantic, and pragmatic information