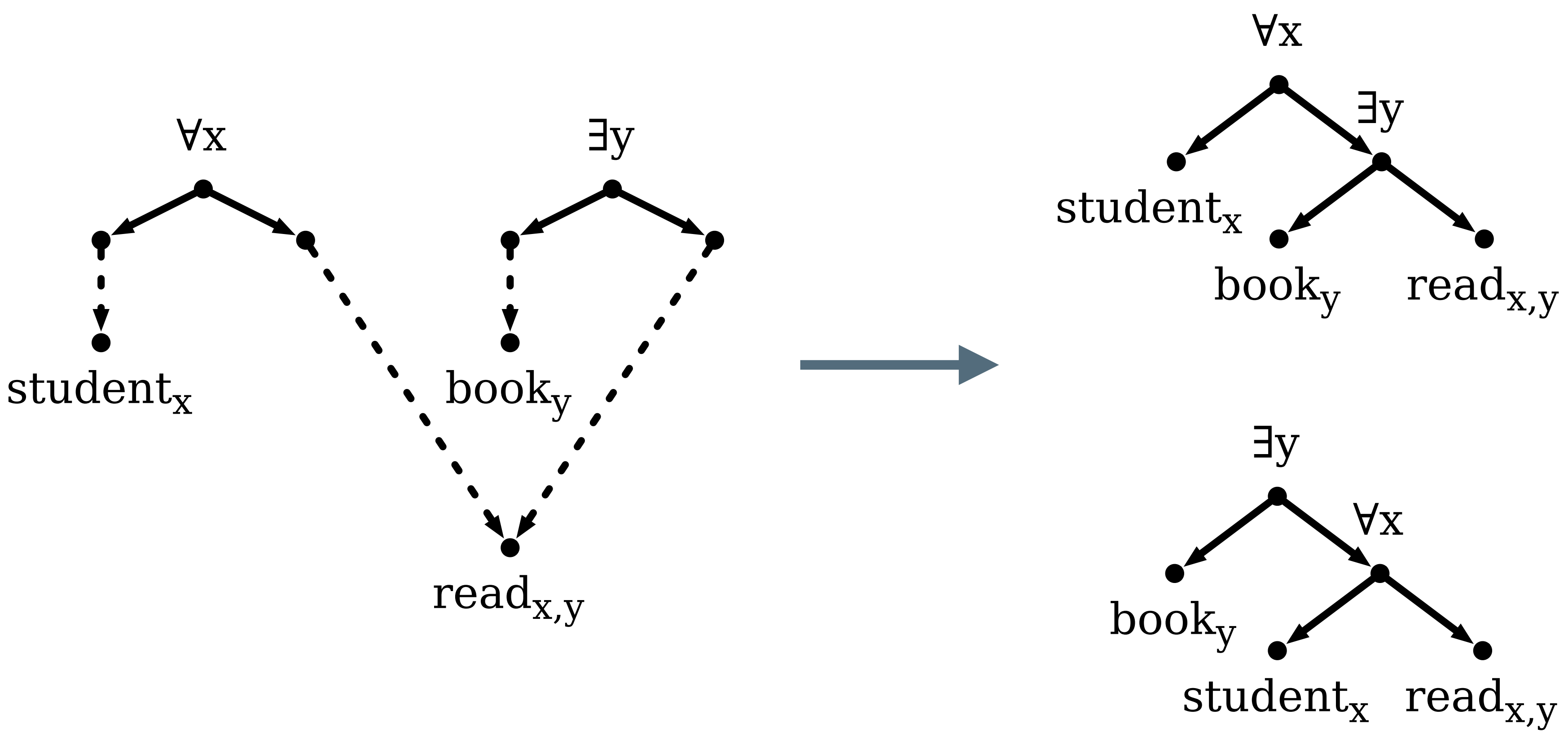
Computational Linguistics

Algorithms for Scope Underspecification

Dietrich Klakow & Stefan Thater
FR 4.7 Allgemeine Linguistik (Computerlinguistik)
Universität des Saarlandes

Summer 2014

What this lecture is about



Overview

- Some basic assumptions about sentence meaning
- Scope ambiguities
- Modelling scope ambiguities with dominance graphs
- An algorithm for solving dominance graphs

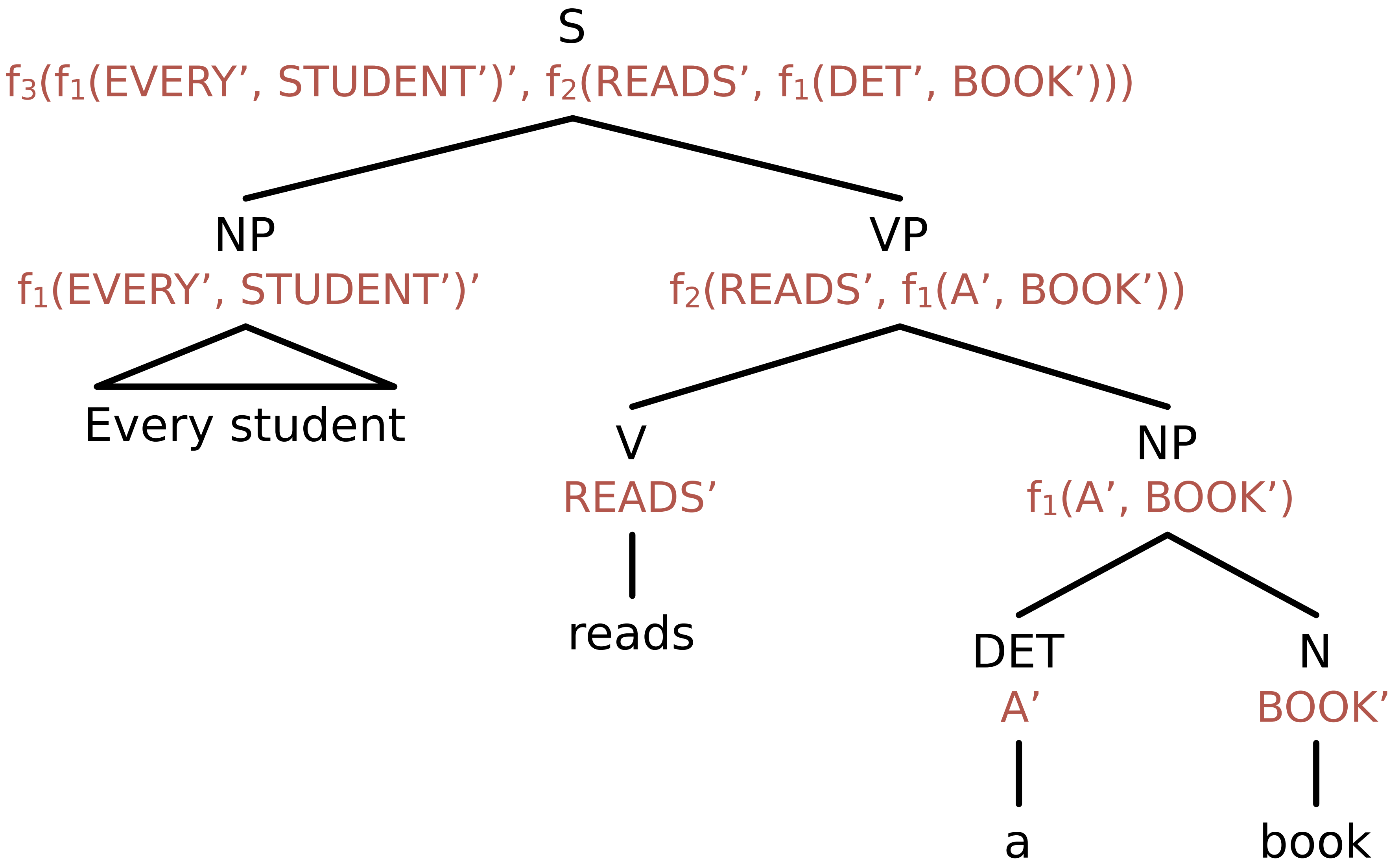
Sentence meaning – Assumptions

- **Truth-functional interpretation:** The meaning of a declarative sentence is given by its truth conditions
- ⇒ we can represent the meaning of natural language sentences by logical formula that “capture” the truth-conditions of the original sentence.
- *Every student works* $\mapsto \forall x(\text{student}'(x) \rightarrow \text{work}'(x))$

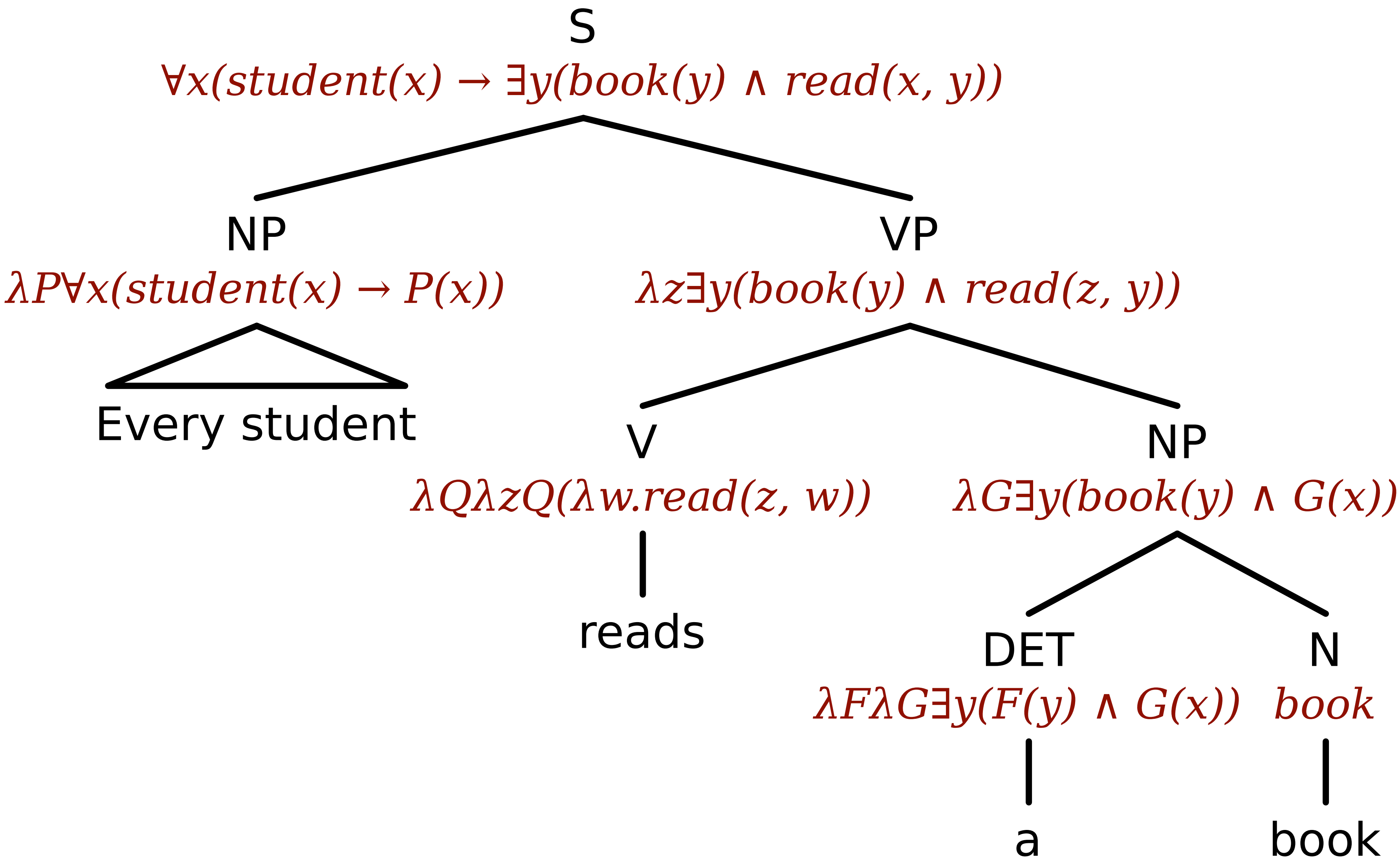
Sentence meaning – Assumptions

- **Compositionality:** The meaning of a complex expression is a function of the meanings of its parts and of the syntactic rules by which they are combined
- **Compositional semantic construction** based on the syntactic tree of the natural language expression
  - The semantic lexicon assigns meaning representations to lexical (leaf) nodes of the syntax tree.
  - The semantic representation of an inner node is computed by combining the representations of its child nodes.

Compositional Semantics Construction



Compositional Semantics Construction



Compositional Semantics Construction

- Compositionality: The meaning of a complex expression is **a function** of the meanings of its parts and of the syntactic rules by which they are combined
- **⇒ Every syntax tree is mapped to a unique semantic representation**

Ambiguities

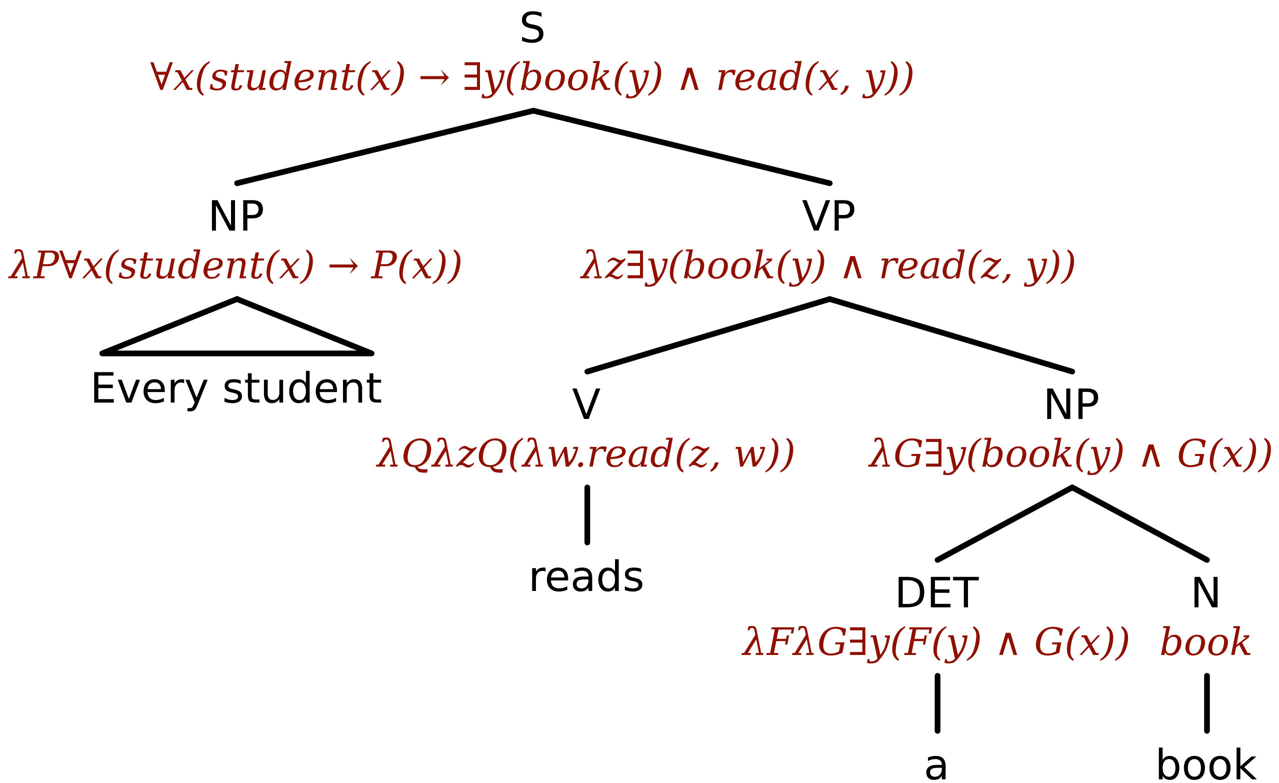
- Natural language is ambiguous: a sentence can have more than one interpretation (“reading”).
- **Lexical ambiguities**
  - *Iraqi head seeks arms*
- **Structural ambiguities**
  - *Enraged cow injures farmer with axe*
  - *The salesman sold the dog biscuits*
  - *Every student reads a book*

Scope Ambiguities

- Scope ambiguities can arise when a sentence contains two or more quantifiers and/or other scope-taking operators (negations, modal expressions, etc.)
- *Every student reads a book*
  - $\forall x(\text{student}'(x) \rightarrow \exists y(\text{book}'(y) \wedge \text{read}'(x, y)))$ [∀∃]
  - $\exists y(\text{book}'(y) \wedge \forall x(\text{student}'(x) \rightarrow \text{read}'(x, y)))$ [∃∀]

Scope Ambiguities – Problem #1

- The approach outlined before will give us just one reading (the “surface scope reading”)
- **Problem:** How to compute the ∃∀ reading?



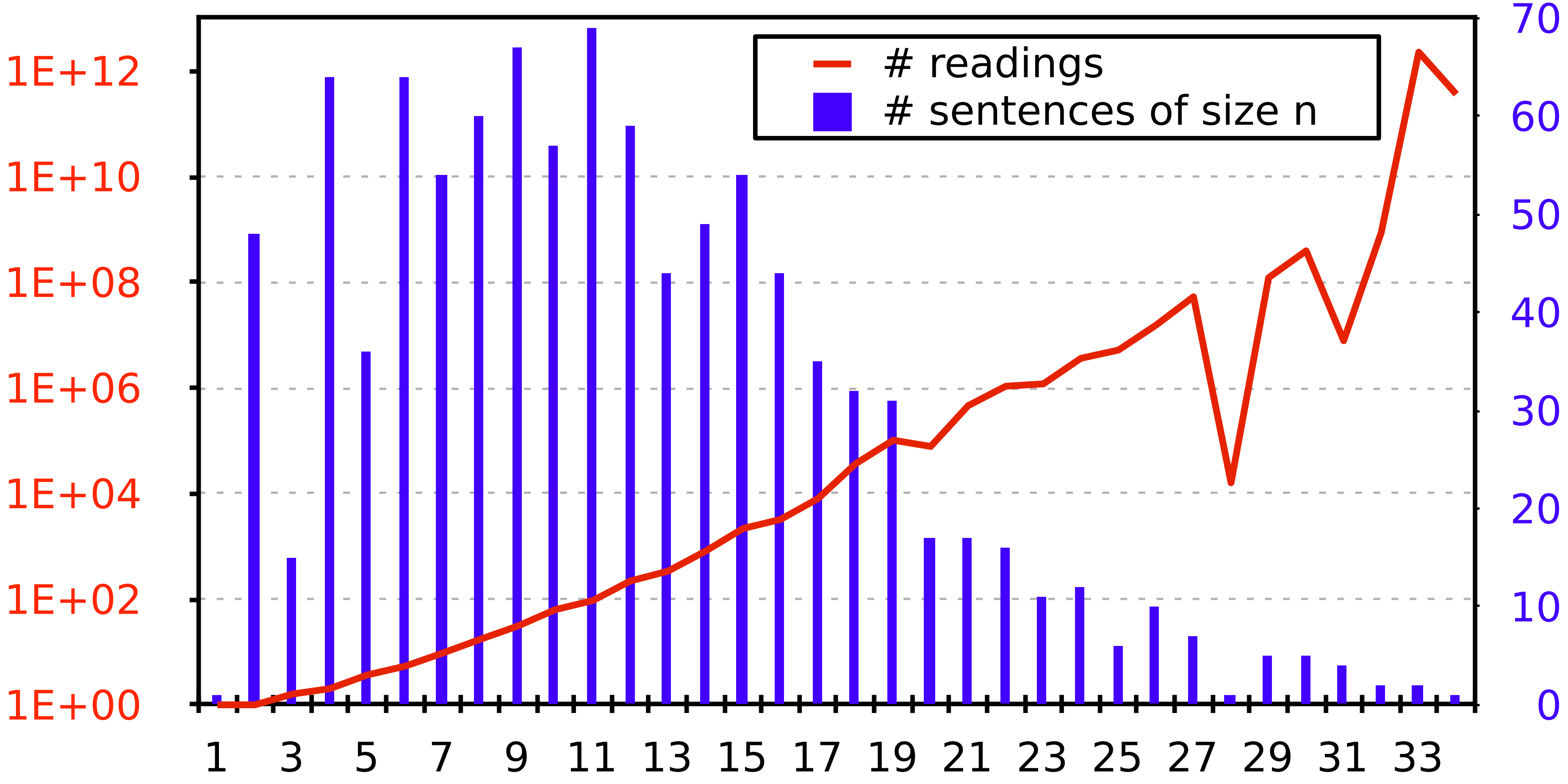
Scope Ambiguities – Problem #2

- **Combinatorial explosion of readings:** The number of readings of a sentence can grow exponentially in the number of scope-taking operators it contains.
- *Every student reads a book* ⇒ 2 readings
- *Every student reads a book about an interesting topic*
  ⇒ 5 readings (some of which are logically equivalent)
- *We quickly put up the tents in the lee of a small hillside and cook for the first time in the open*
  ⇒ 480 readings* (most of which are logically equivalent)

* according to the English Resource Grammar

Scope Ambiguities – Problem #2

- Number of readings in a “real-live” corpus according to the English Resource Grammar (Koller &al., ACL 2008)



Coping with scope – Options

(1) Ignore scope ambiguities

- for instance, always compute the “surface scope” reading

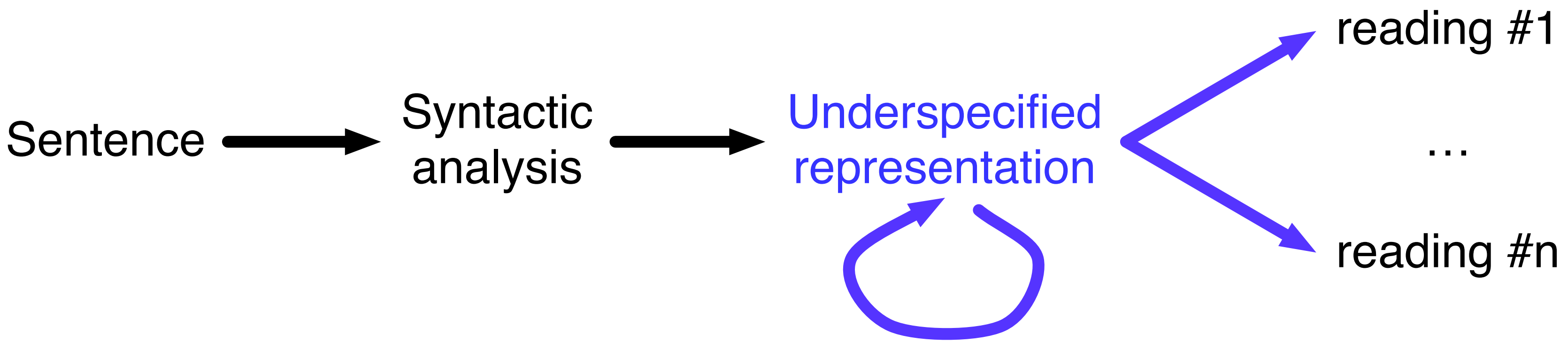
(2) Enumerate all readings and then select the “right” one

- we need more complex semantics construction rules and a method to choose the “right” reading
- computationally very expensive, since sentences can easily have millions of readings

(3) Use scope underspecification

Scope Underspecification

- Don’t explicity enumerate readings
- Instead, represent all readings of a sentence by a single compact underspecifed representation (USR).
- The individual readings can be enumerated from the underspecified representation if needed (this lecture).
- We can perform inferences directly on the level of underspecified representations (next lecture).



A notational change

- *Every student reads a book*
  - ∀x(student(x), ∃y(book(y), read(x, y)))
  - ∃y(book(y), ∀x(student(x), read(x, y)))
- Abbreviations:
  - ∀x(X, Y) abbreviates ∀x(X → Y)
  - ∃x(X, Y) abbreviates ∃x(X ∧ Y)

Readings = Trees

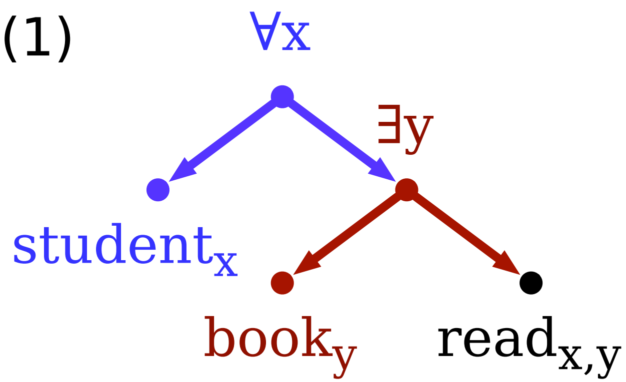
- *Every student reads a book*

  (1) ∀x(student(x), ∃y(book(y), read(x, y)))

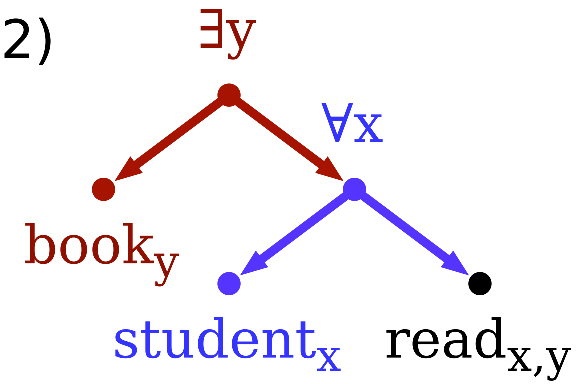
  (2) ∃y(book(y), ∀x(student(x), read(x, y)))

- Represent readings as trees:

(1)

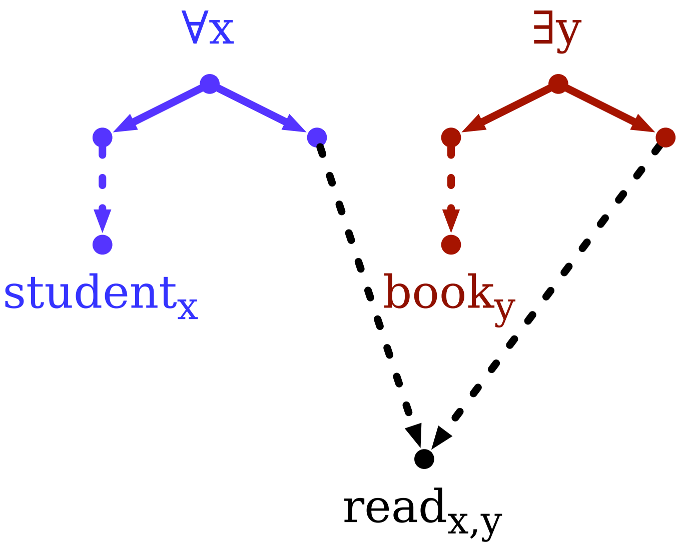


(2)



Dominance Graphs (informal)

- Dominance graphs consist of
  - **“tree fragments”** connected by
  - **dominance edges**

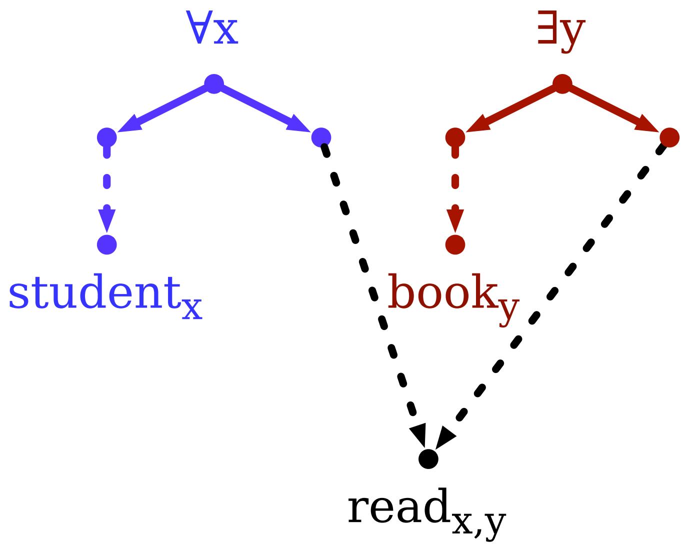


- Tree fragments (solid lines) specify the “semantic material” that is common to all readings
- Dominance edges (dotted lines) specify constraints: The upper node must dominate the lower one.
- Subclass of “normal” dominance graphs: dominance edges always go from leaves (“holes”) to roots.

Normal Dominance Graphs

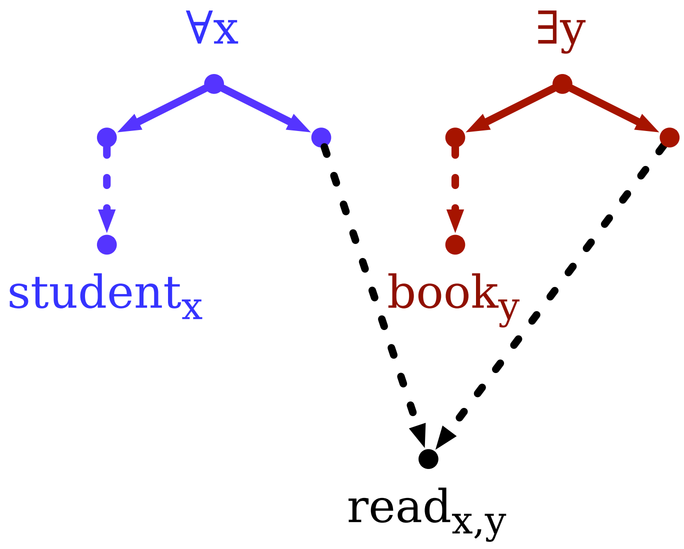
- **A normal dominance graph** is a graph G = ⟨V, E ⊎ D⟩ such that
  - the subgraph ⟨V, E⟩ is a collection of node disjoint trees where the height of each tree is ≤ 1

    we call the roots in ⟨V, E⟩ **roots** and all other nodes **holes**
  - if ⟨v_1, v_2⟩ ∈ D, then v_1 is a hole and v_2 a root
  - every hole has at least one outgoing dominance edge.



Normal Dominance Graphs

- **A labelled dominance graph** is a graph ⟨V, E ⊎ D, L⟩ such that
  - ⟨V, E ⊎ D⟩ is a (normal) dominance graph and
  - L is a labelling function that assigns a node *v* a label with arity *n* iff *v* is a root with *n* outgoing tree edges



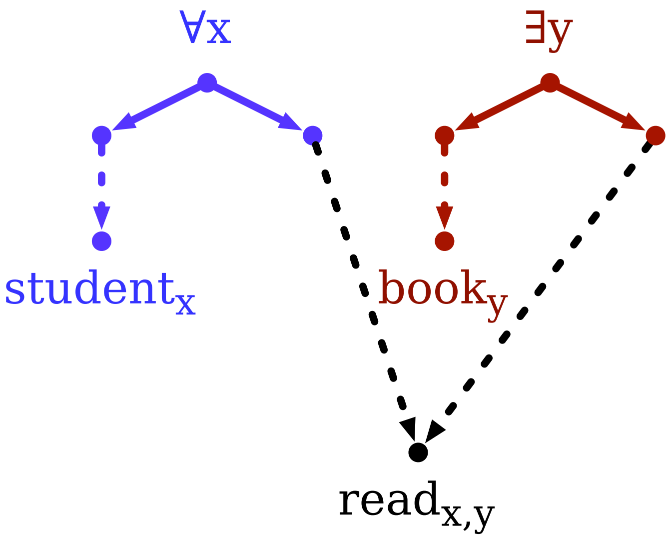
Solved Forms

- A dominance graph G **is in solved form** if G is a forest
- Let $G = \langle V, E \uplus D \rangle$ and $G' = \langle V, E \uplus D' \rangle$

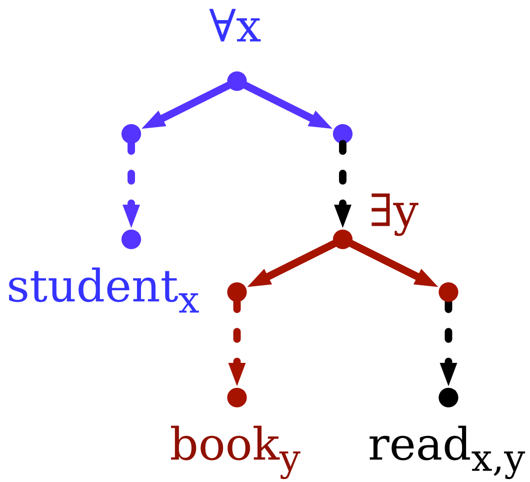
  We say that G' **is a solved form of** G if G' is in solved form and the reachability relation of G' extends that of G

  That is: whenever v_1 and v_2 are connected by some dominance edge in G', there must be a directed path from v_1 to v_2 in G.
- Note that dominance graphs and their solved forms differ only in their sets of dominance edges.
- Note also that the solved forms of connected dominance graphs are always trees.

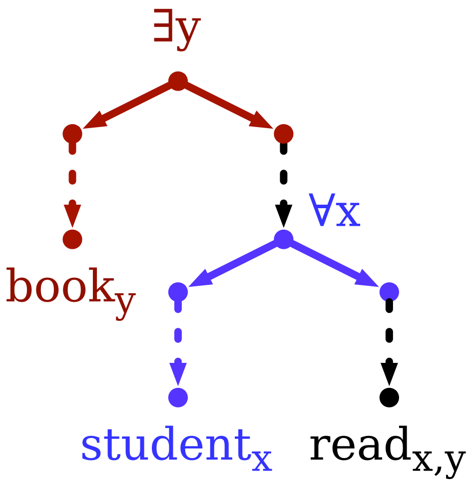
Solved Forms – Example



dominance graph

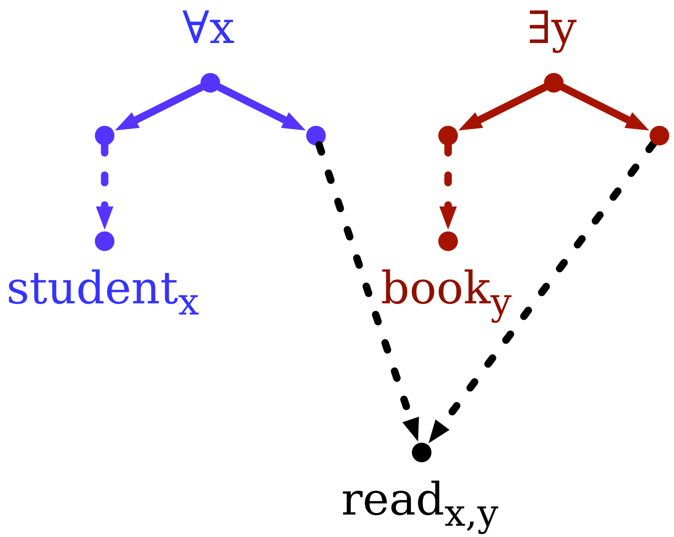


solved form #1

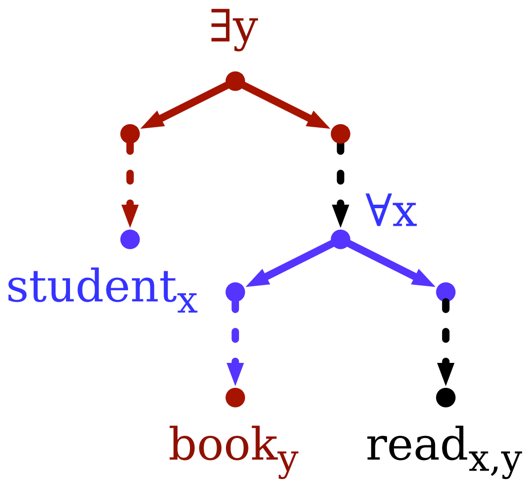


solved form #2

Not a solved form of



dominance graph G



not a solved form of G

Computational Questions

- **The solvability problem:**
  Does a given dominance graph have any solved forms?
- **The enumeration problem:**
  Given a dominance graph, enumerate all its (minimal) solved forms.
- We will discuss the algorithm by Bodirsky &al. (2004)
- To keep things simple, we restrict the presentation to *connected* dominance graphs.

(Bodirsky &al., 2004)

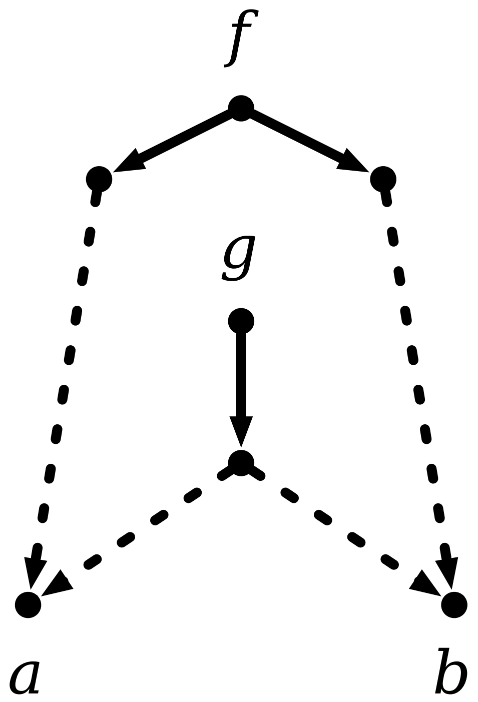
Solving dominance graphs

- The algorithm of Bodirsky &al. (2004) constructs a solved form of a dominance graph G as follows:
  1. nondeterministically choose a “free fragment” F from G
  2. remove F from G; this decomposes the graph G into weakly connected components G_1, …, G_k
  3. recursively compute a solved form for G_1, …, G_k
  4. attach the solved form of G_i under the corresponding hole of the free fragment F (for $1 \leq i \leq k$)

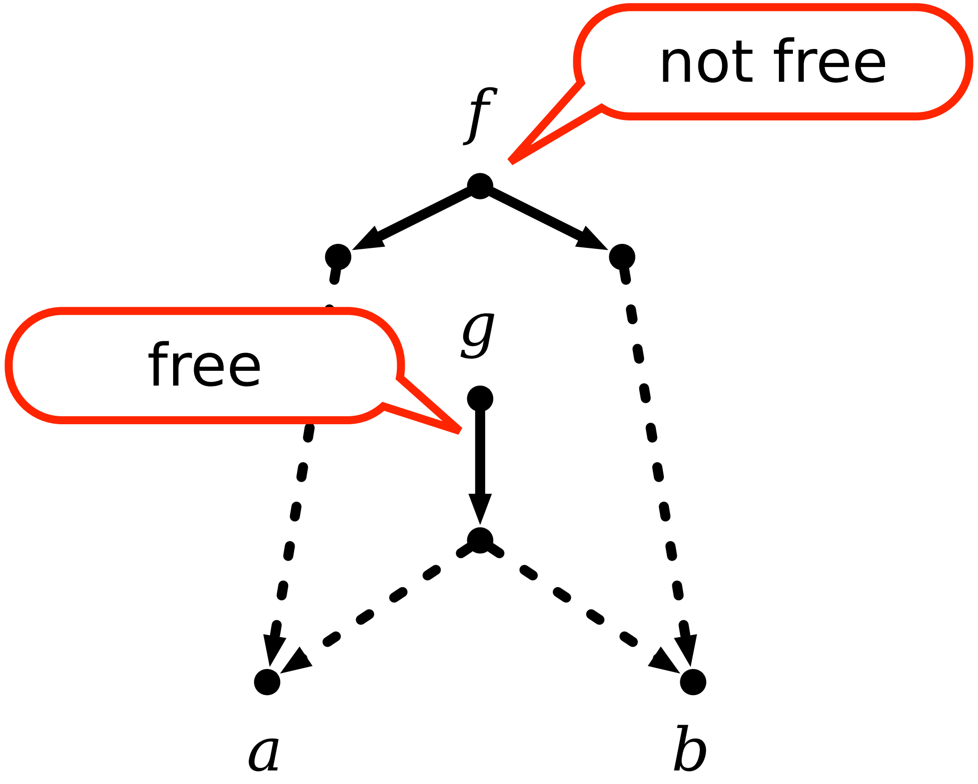
Free Fragments

- The workhorse of the algorithm is the notion of a “free fragment.”
- We say that a fragment **F is free in a** normal dominance graph **G** iff
  - the root of F has no incoming dominance edges, and
  - no distinct holes of F are connected by an undirected path in the graph G’ obtained from G by removing the root of F
- It can be shown that the following statements are equivalent if G is solvable (normal) dominance graph:
  - F is a free fragment in G
  - G has a solved form with top-most fragment F

Free Fragments: An Example



Free Fragments: An Example



(Bodirsky &al., 2004)

Solving dominance graphs

solve(G = ⟨V, E ⊎ D⟩) = (*)

choose a free fragment F of G **else fail**

let G_1, …, G_k be the WCCs of G\F

let $\langle V_i, E_i \uplus D_i \rangle = solve(G_i)$

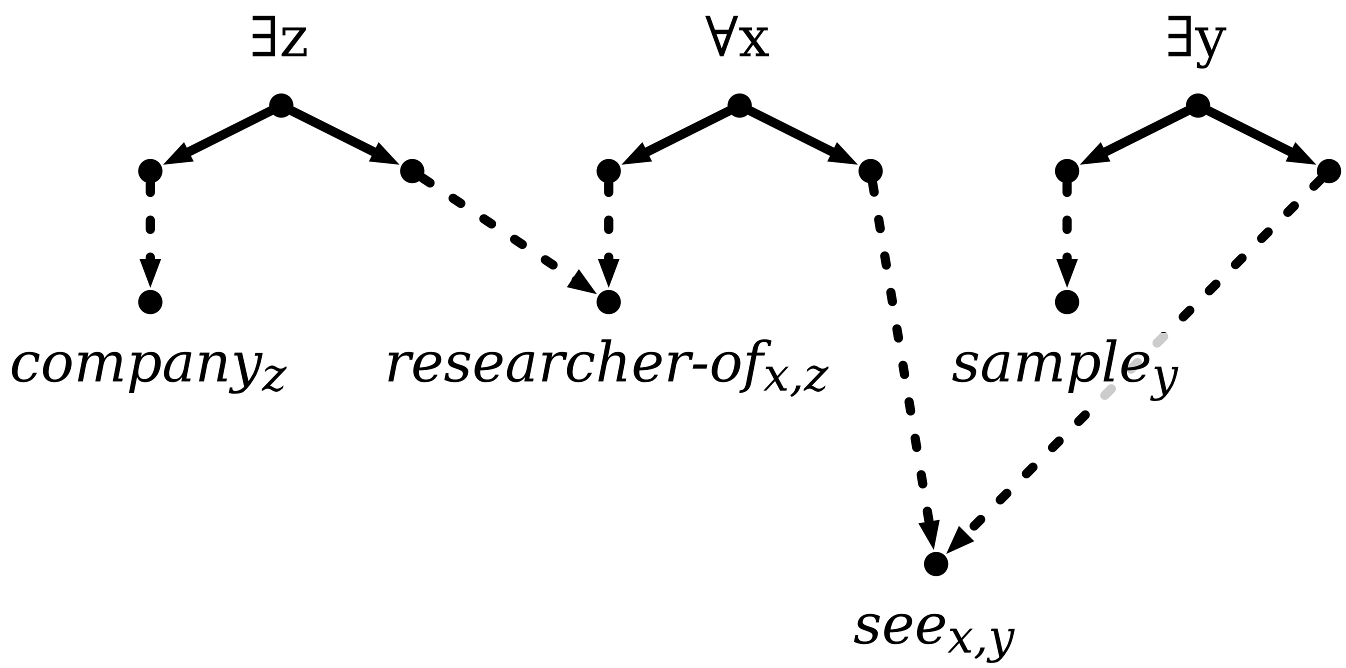
return $\langle V, E \uplus D_1 \uplus \cdots \uplus D_k \uplus D' \rangle$

where D’ are dominance edges that connect the holes of F with the solved form of one of the corresponding G_i

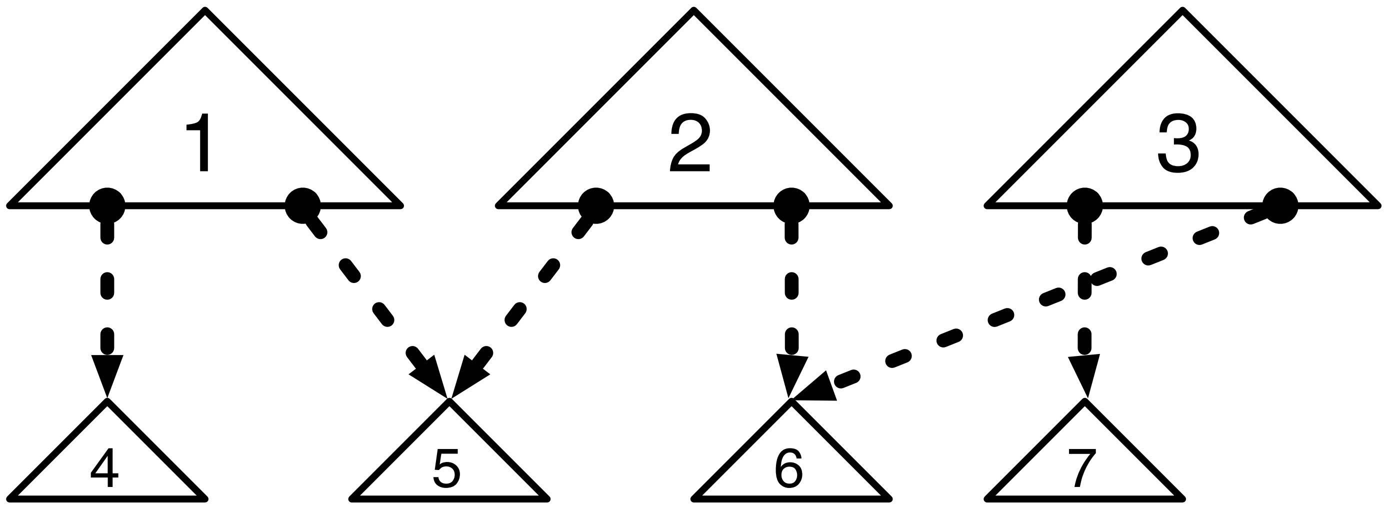
(*) slightly simplified version, works only for connected normal dominance graphs

An Example

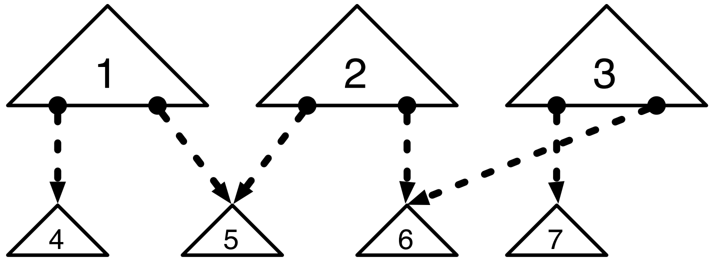
- Underspecified representation for the sentence "every researcher of a company sees a sample:"



An Example

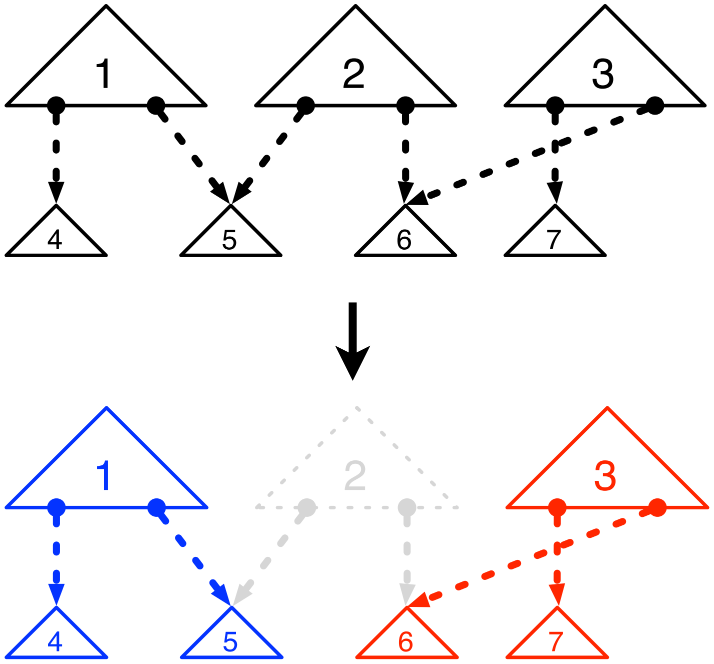


An Example



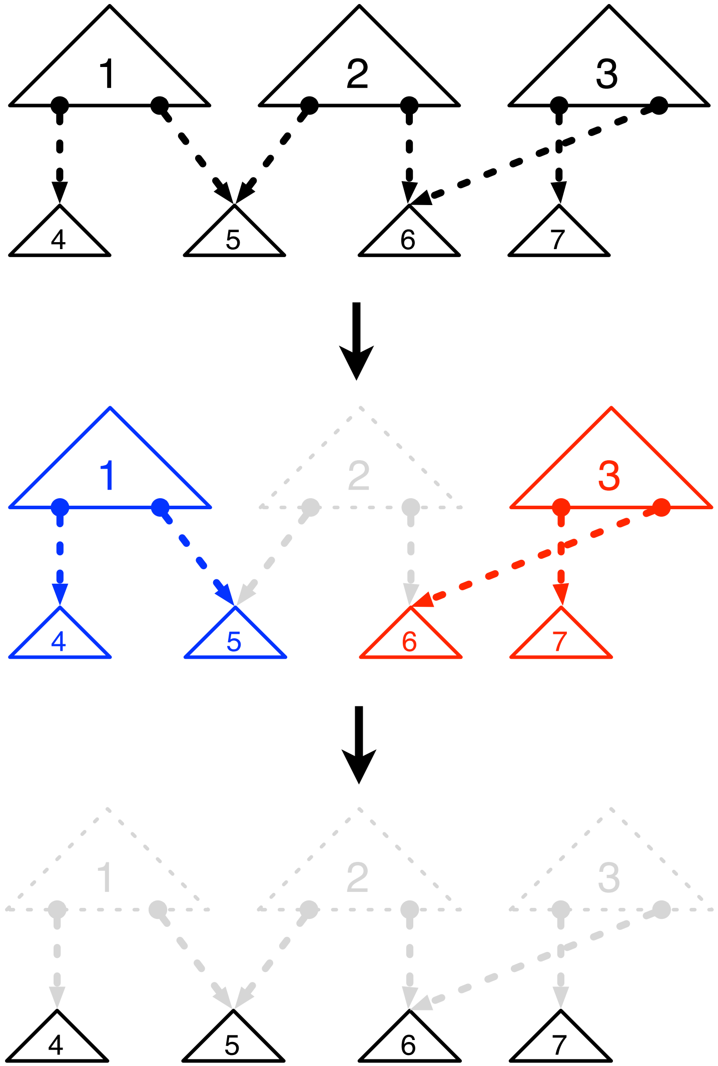
subgraph	free	wccs

An Example



subgraph	free	wccs
{1,...,7}	2	{1,4,5}, {3,6,7}

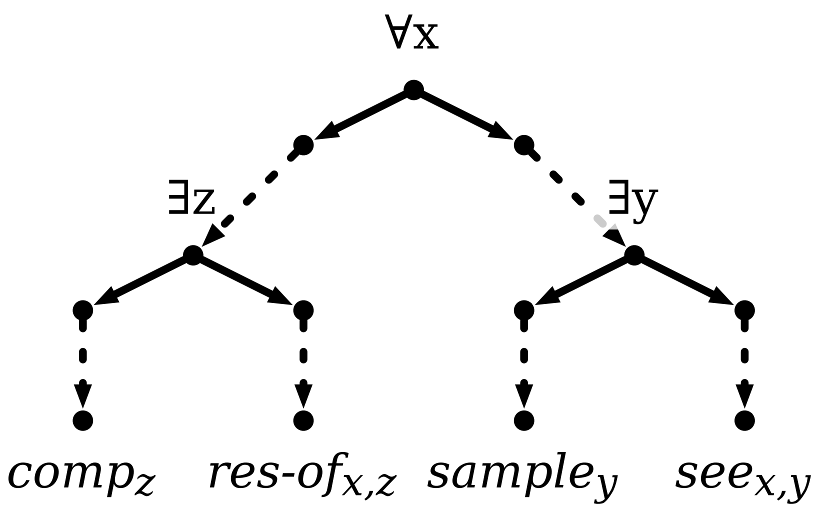
An Example



subgraph	free	wccs
{1,...,7}	2	{1,4,5}, {3,6,7}
{1,4,5}	1	{4}, {5}
{3,6,7}	3	{6}, {7}

The final solved form

- ∀x(∃z(comp(z), res-of(x,z)), ∃y(sample(y), see(x, y)))



Properties of the solver

- It can be shown that the following statements are equivalent:
  - solve(G) fails for some nondeterministic choice
  - G is not solvable
  - solve(G) fails for all nondeterministic choices

Properties of the solver

- We can test whether a fragment is free in time O(n + m)
  - where n is the number of nodes and
  - m the number of edges in a dominance graph G.
- The overall running time of solve(G) is in O(n · (n + m)) per solved-form.

Literature

- Alexander Koller, Manfred Pinkal, and Stefan Thater. Scope Underspecification with Tree Descriptions: Theory and Practice [PDF]. In: Resource Adaptive Cognitive Processes. Ed. by Matthew Crocker and Jörg Siekmann. Cognitive Technologies Series. Berlin: Springer. 2009.
- Manuel Bodirsky, Denys Duchier, Joachim Niehren, and Sebastian Miele (2004). A new algorithm for normal dominance constraints [PDF]. In the proceedings of the Symposium on Discrete Algorithms (SODA04), 59-67.