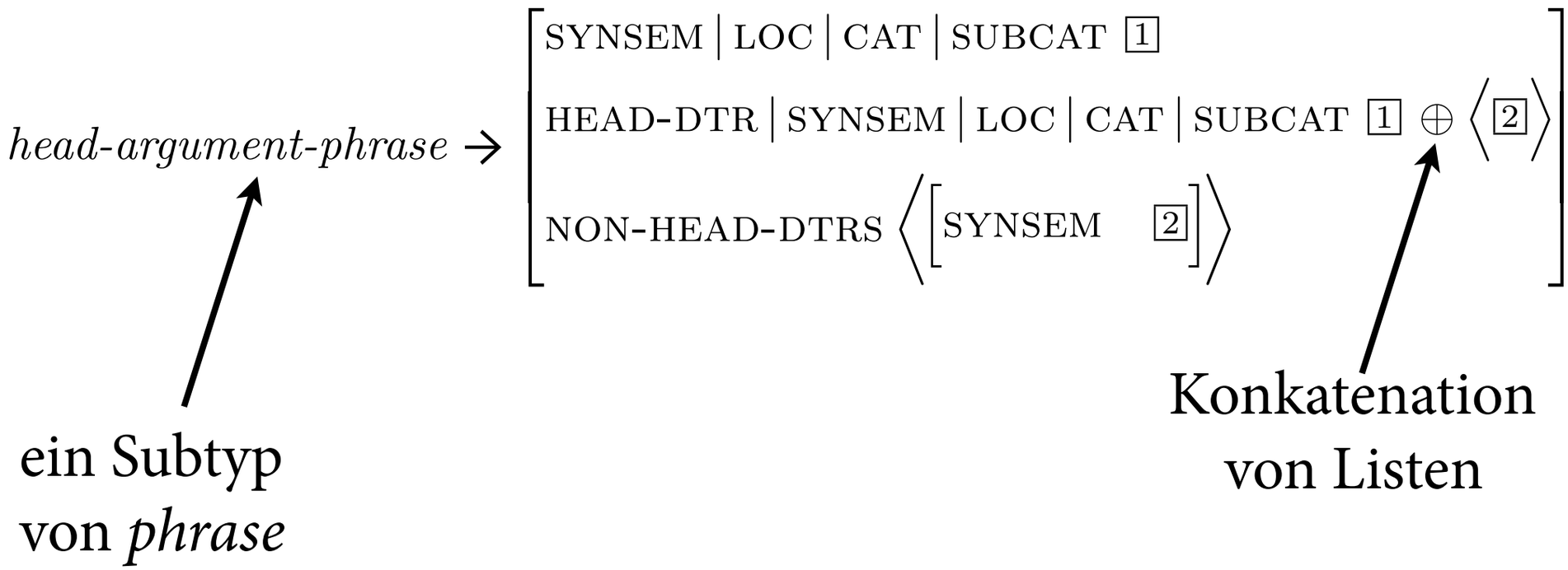
Parsing von unifikationsbasierten Grammatikformalismen

Vorlesung “Grammatikformalismen”
Alexander Koller

18. Juli 2017

Das Kopf-Argument-Schema

- Wichtigstes Schema: Kombination von Kopf mit Argument mit dem *Kopf-Argument-Schema*.



Kopf + Argument

$$\textit{head-arg-phr} \rightarrow \begin{bmatrix} \text{SYNSEM} \mid \text{LOC} \mid \text{CAT} \mid \text{SUBCAT } \boxed{1} \\ \text{HEAD-DTR} \mid \text{SYNSEM} \mid \text{LOC} \mid \text{CAT} \mid \text{SUBCAT } \boxed{1} \oplus \left\langle \boxed{2} \right\rangle \\ \text{NON-HEAD-DTRS} \left\langle \begin{bmatrix} \text{SYNSEM} & \boxed{2} \end{bmatrix} \right\rangle \end{bmatrix}$$

$$\begin{bmatrix} \text{PHON} & \text{‘ein Käsebrot’} \\ \text{SYNSEM} \mid \text{LOC} \mid \text{CAT} \mid \text{SUBCAT} & \langle\rangle \\ \text{HEAD-DTR} & \textit{word}\left[\begin{bmatrix} \text{PHON} & \text{‘Käsebrot’} \\ \text{SYNSEM} \mid \text{LOC} \mid \text{CAT} & \textit{cat}\left[\begin{bmatrix} \text{HEAD} & \textit{noun}\begin{bmatrix} \text{GEND} & \text{neut} \\ \text{NUM} & \text{sg} \end{bmatrix} \\ \text{SUBCAT} & \left\langle \boxed{2}\ \textsc{Det}\left[\textit{sg}\right] \right\rangle \end{bmatrix}\right] \end{bmatrix}\right] \\ \text{NON-HEAD-DTRS} & \left\langle \textit{word}\left[\begin{bmatrix} \text{PHON} & \text{‘ein’} \\ \text{SYNSEM} & \textit{synsem}\left[\boxed{2}\begin{bmatrix} \text{LOC} \mid \text{CAT} & \textit{cat}\left[\begin{bmatrix} \text{HEAD} & \textit{det}\begin{bmatrix} \text{NUM} & \text{sg} \end{bmatrix} \\ \text{SUBCAT} & \langle\rangle \end{bmatrix}\right] \end{bmatrix}\right] \end{bmatrix}\right] \right\rangle \\ \textit{head-arg-phr} & \end{bmatrix}$$

$$\textit{word}\left[\begin{bmatrix} \text{PHON} & \text{‘ein’} \\ \text{SYNSEM} \mid \text{LOC} \mid \text{CAT} & \textit{cat}\left[\begin{bmatrix} \text{HEAD} & \textit{det}\begin{bmatrix} \text{NUM} & \text{sg} \end{bmatrix} \\ \text{SUBCAT} & \langle\rangle \end{bmatrix}\right] \end{bmatrix}\right]$$

$$\textit{word}\left[\begin{bmatrix} \text{PHON} & \text{‘Käsebrot’} \\ \text{SYNSEM} \mid \text{LOC} \mid \text{CAT} & \textit{cat}\left[\begin{bmatrix} \text{HEAD} & \textit{noun}\begin{bmatrix} \text{GEND} & \text{neut} \\ \text{NUM} & \text{sg} \end{bmatrix} \\ \text{SUBCAT} & \left\langle \textsc{Det}\left[\textit{sg}\right] \right\rangle \end{bmatrix}\right] \end{bmatrix}\right]$$

Expressivität

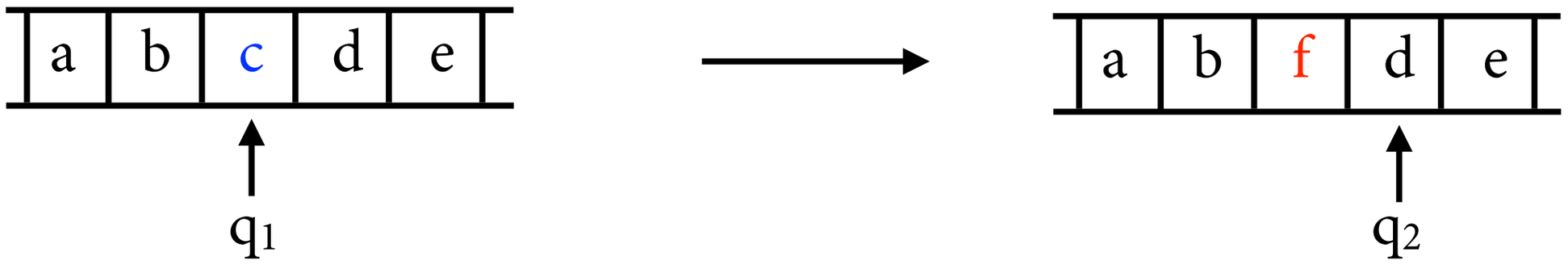
- Wir haben uns überzeugt, dass HPSG linguistische Phänomene sehr elegant beschreiben kann.

- Welchen Preis in formaler Expressivität bezahlen wir dafür?

Chomsky-Hierarchie

Typ	Grammatiken	Automaten
0	Typ-0-Grammatiken	Turingmaschinen
1	kontextsensitive	linear platzbeschränkte Turingmaschinen
2	kontextfreie	Kellerautomaten
3	reguläre	endliche Automaten

Codierung von Turing-Maschinen



$$\begin{bmatrix} \text{LEFT} & \langle b, a \rangle \\ \text{RIGHT} & \langle c, d, e \rangle \\ \text{STATE} & q_1 \end{bmatrix} \longrightarrow \begin{bmatrix} \text{LEFT} & \langle f, b, a \rangle \\ \text{RIGHT} & \langle d, e \rangle \\ \text{STATE} & q_2 \end{bmatrix}$$

Codierung von Turing-Maschinen

$$\begin{bmatrix} \text{LEFT} & \langle b, a \rangle \\ \text{RIGHT} & \langle c, d, e \rangle \\ \text{STATE} & q_1 \end{bmatrix} \longrightarrow \begin{bmatrix} \text{LEFT} & \langle f, b, a \rangle \\ \text{RIGHT} & \langle d, e \rangle \\ \text{STATE} & q_2 \end{bmatrix}$$

Regelschema (vereinfacht):

$$rstep \rightarrow \begin{bmatrix} \text{LEFT} & \begin{bmatrix} \text{FIRST} & f \\ \text{REST} & \boxed{1} \end{bmatrix} \\ \text{RIGHT} & \boxed{2} \\ \text{STATE} & q_2 \\ \text{DTR} & \begin{bmatrix} \text{LEFT} & \boxed{1} \\ \text{RIGHT} & \begin{bmatrix} \text{FIRST} & c \\ \text{REST} & \boxed{2} \end{bmatrix} \\ \text{STATE} & q_1 \end{bmatrix} \end{bmatrix}$$

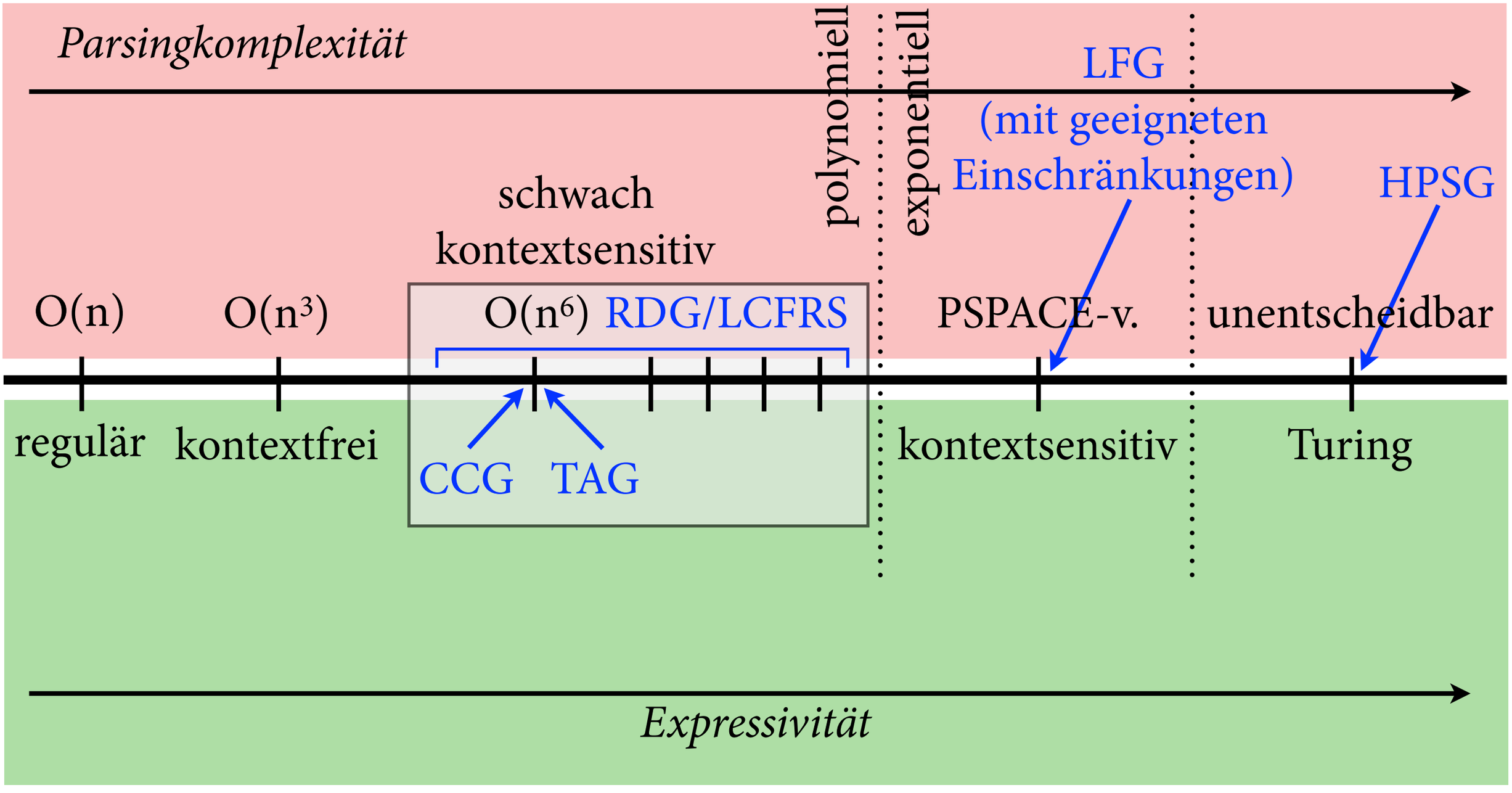
Codierung: Grundidee

- Kann Berechnung einer Turingmaschine mit zwei Stacks simulieren.

- In HPSG-Featurestrukturen kann man zwei Stacks mit zwei listenwertigen Features darstellen.
  - geht in TAG, CCG nicht:
    dort endliche Wertebereiche für Features

- Daher ist Wortproblem von HPSG unentscheidbar.

Fazit


Parsingkomplexität
polynomiell
exponentiell
LFG
(mit geeigneten
Einschränkungen)
HPSG
schwach
kontextsensitiv
O(n)
O(n³)
O(n⁶)
RDG/LCFRS
PSPACE-v.
unentscheidbar
regulär
kontextfrei
CCG
TAG
kontextsensitiv
Turing
Expressivität

Parsing mit Featurestrukturen

- Grammatikalität von HPSG-Analysen bisher über Constraints definiert:
  - FS mit Subtyp von *sign* und korrektem Wert von PHON
  - alle Knoten sort-resolved und wohlgeformt (d.h. jeder Knoten durch Lexikoneintrag oder spezifischen Typ von Phrase motiviert)
- Heute: CKY-artiger Parser für HPSG.
- Grundideen übertragen sich auf andere unifikationsbasierte Formalismen, z.B. LFG.

Grundidee

- Parsingschema für Bottom-Up-Parser:

$$\frac{[i,j,\ F_1] \qquad [j,k,\ F_2]}{[i,k,\ C(\tau) \sqcup [\text{hdtr } F_1] \sqcup [\text{nhdtr } F_2]]}\ \tau$$

  - Einbettende Features hdtr und nhdtr hängen von Phrasentyp τ ab.

  - FS C(τ) kommt aus Typconstraint für τ. Unifikation kann fehlschlagen; dann Regel nicht anwendbar.

Grundidee

- Parsingschema für Bottom-Up-Parser:

$$\frac{[i,j,\ F_1] \qquad [j,k,\ F_2]}{[i,k,\ C(\tau) \sqcup [\text{hdtr } F_1] \sqcup [\text{nhdtr } F_2]]}\ \tau$$

FS F_2 ist Analyse von Substring j-k

- Einbettende Features hdtr und nhdtr hängen von Phrasentyp τ ab.

- FS C(τ) kommt aus Typconstraint für τ. Unifikation kann fehlschlagen; dann Regel nicht anwendbar.

Grundidee

- Parsingschema für Bottom-Up-Parser:

$$\frac{[i,j,\ F_1] \qquad [j,k,\ F_2]}{[i,k,\ C(\tau) \sqcup [\text{hdtr } F_1] \sqcup [\text{nhdtr } F_2]]}\ \tau$$

FS F_2 ist Analyse von Substring j-k

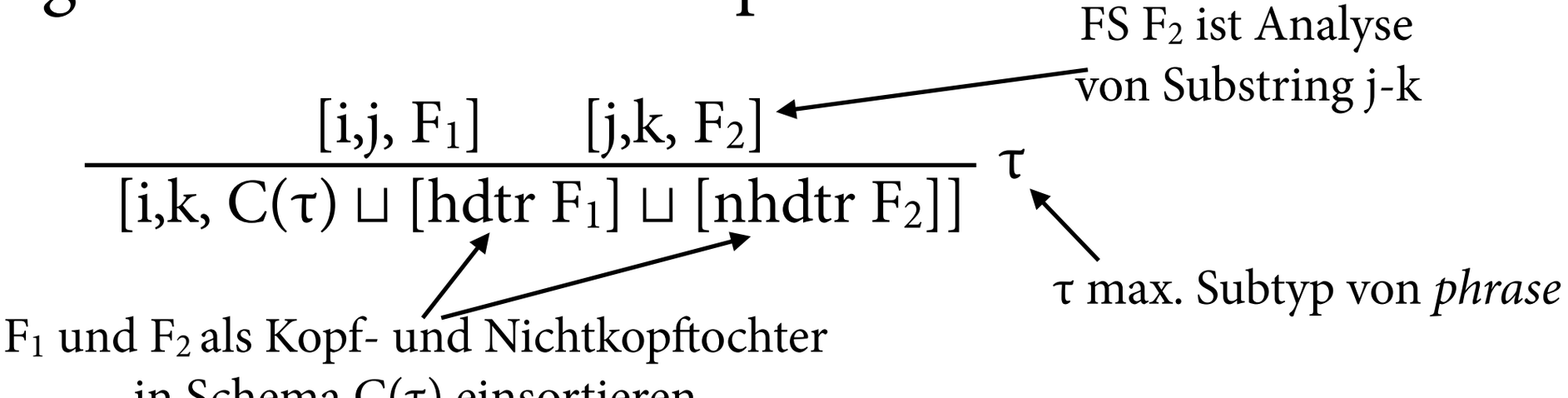
τ max. Subtyp von *phrase*

- Einbettende Features hdtr und nhdtr hängen von Phrasentyp τ ab.

- FS $C(\tau)$ kommt aus Typconstraint für τ. Unifikation kann fehlschlagen; dann Regel nicht anwendbar.

Grundidee

- Parsingschema für Bottom-Up-Parser:



- Einbettende Features hdtr und nhdtr hängen von Phrasentyp τ ab.

- FS C(τ) kommt aus Typconstraint für τ. Unifikation kann fehlschlagen; dann Regel nicht anwendbar.

Beispiel

ein Käsebrot	
isst	

isst

ein Käsebrot

Beispiel

	ein Käsebrot
isst	$\begin{bmatrix} word \\ \begin{bmatrix} \text{PHON} & \text{‘isst’} \\ \text{SYNSEM} \mid \text{LOC} \mid \text{CAT} & \begin{bmatrix} cat \\ \begin{bmatrix} \text{HEAD} & \begin{bmatrix} verb \\ \text{NUM} \quad \text{sg} \end{bmatrix} \\ \text{SUBCAT} & \left\langle \text{NP}\left[nom\right], \text{NP}\left[acc\right] \right\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$	
	isst	ein Käsebrot

Beispiel

ein Käsebrot

isst

	$\textit{head-arg-phr}\left[\begin{array}{ll}\textsc{phon} & \text{`ein Käsebrot'} \\ \textsc{synsem}\,\vert\,\textsc{loc}\,\vert\,\textsc{cat} & \textit{cat}\left[\begin{array}{ll}\textsc{head} & \textit{noun}\left[\begin{array}{ll}\textsc{gend} & \textsc{neut} \\ \textsc{num} & \textsc{sg}\end{array}\right] \\ \textsc{subcat} & \langle\rangle\end{array}\right]\end{array}\right]$
$\textit{word}\left[\begin{array}{ll}\textsc{phon} & \text{`isst'} \\ \textsc{synsem}\,\vert\,\textsc{loc}\,\vert\,\textsc{cat} & \textit{cat}\left[\begin{array}{ll}\textsc{head} & \textit{verb}\left[\textsc{num}\quad \textsc{sg}\right] \\ \textsc{subcat} & \left\langle \textsc{np}\left[\textit{nom}\right], \textsc{np}\left[\textit{acc}\right]\right\rangle\end{array}\right]\end{array}\right]$	

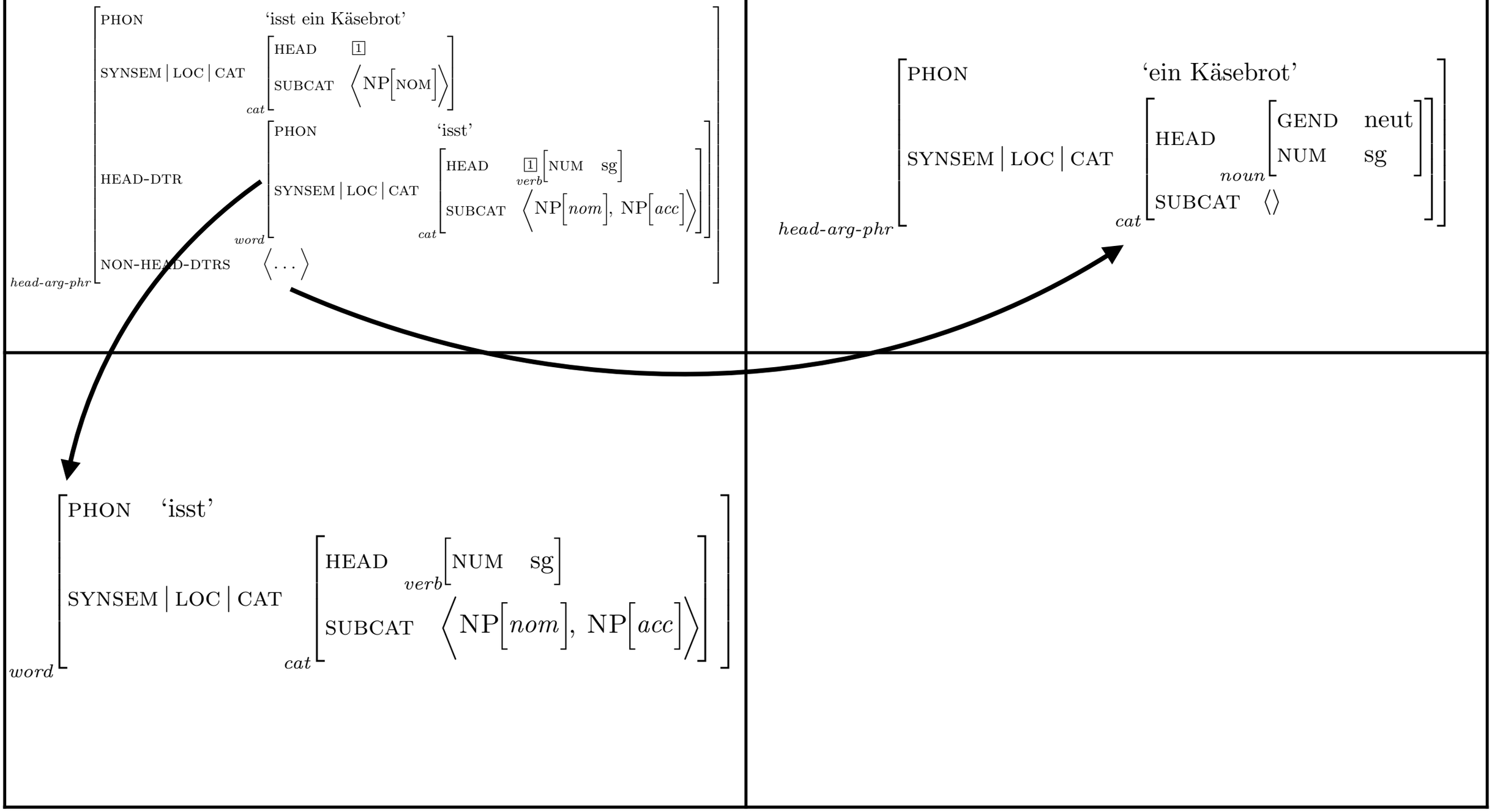
isst

ein Käsebrot

Beispiel

ein Käsebrot

isst



isst

ein Käsebrot

Algorithmische Fragen

- *Gleichheit testen:* Wenn ich [i,k,F] und [i,k,G] ableiten kann, muss ich effizient überprüfen, ob F und G die gleiche Featurestruktur sind.
  - Wenn nein, muss neues Item auf Agenda eingetragen werden.

- *Unifikation:* Muss $C(\tau) \sqcup [\text{hdtr } F_1] \sqcup [\text{nhdtr } F_2]$ effizient ausrechnen.

- Mindestens 90% der Laufzeit eines unifikations-basierten Parsers gehen in diese zwei Funktionen.

Subsumption von TFS

- TFS F *subsumiert* TFS G ($F \sqsubseteq G$), wenn alle Informationen in F auch in G vorhanden sind:
  - Pfade: $F(\pi)$ definiert $\Rightarrow$ $G(\pi)$ definiert
  - Reentrancy: $F(\pi) = F(\pi') \Rightarrow G(\pi) = G(\pi')$
  - Subtypen: $\theta_F(F(\pi)) \sqsubseteq \theta_G(G(\pi))$
- *Unifikation:* $F = F_1 \sqcup F_2$ ist FS (falls existiert), so dass
  - subsumiert beide: $F_1 \sqsubseteq F$, $F_2 \sqsubseteq F$
  - erfindet nichts dazu: für alle F' mit $F_1 \sqsubseteq F'$, $F_2 \sqsubseteq F'$ gilt $F \sqsubseteq F'$.

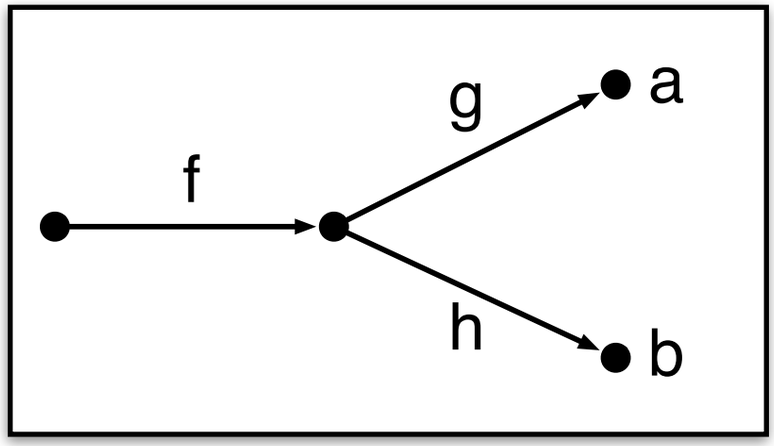
Subsumption testen

- Algorithmus von Malouf et al. 2000: Gegeben F und G, teste gleichzeitig, ob $F \sqsubseteq G$ und ob $G \sqsubseteq F$.
  - Spezialfall: $F = G$ gdw. $F \sqsubseteq G$ und $G \sqsubseteq F$.

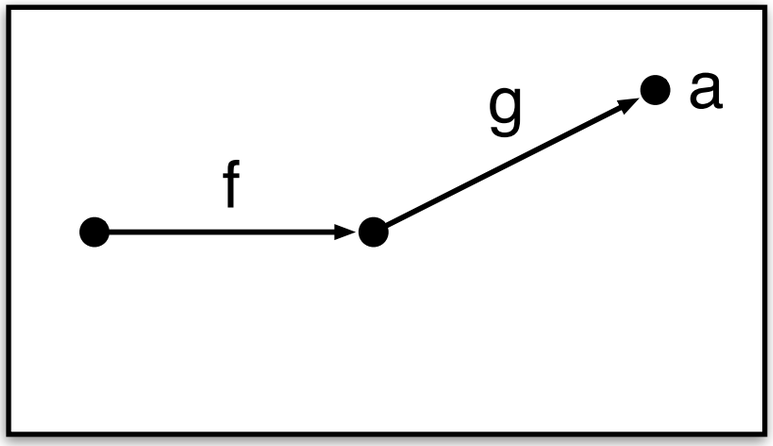
- Traversiere beide FS rekursiv.
  - Speichere für jeden Knoten u in F einen *Pointer* zu einem Knoten v in G, und umgekehrt.
  - Erkenne $F \not\sqsubseteq G$, falls: u mehr Features hat als v; Typen von u, v nicht zusammenpassen; u beim zweiten Besuch schon einen Pointer zu einem Knoten $\neq$ v hatte (und umgekehrt).

Beispiel

F_1



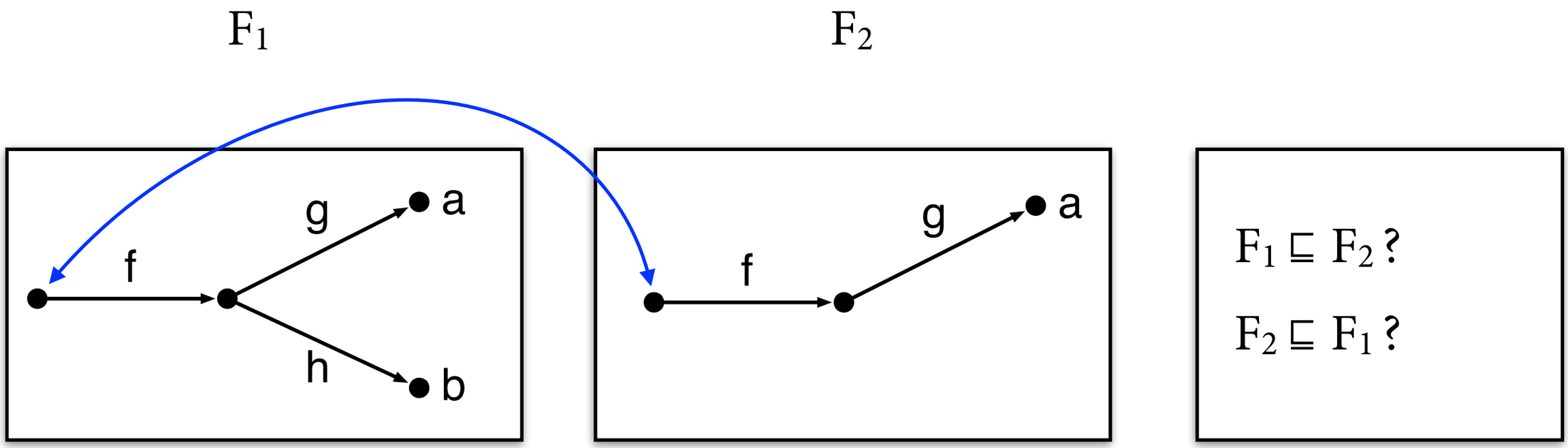
F_2



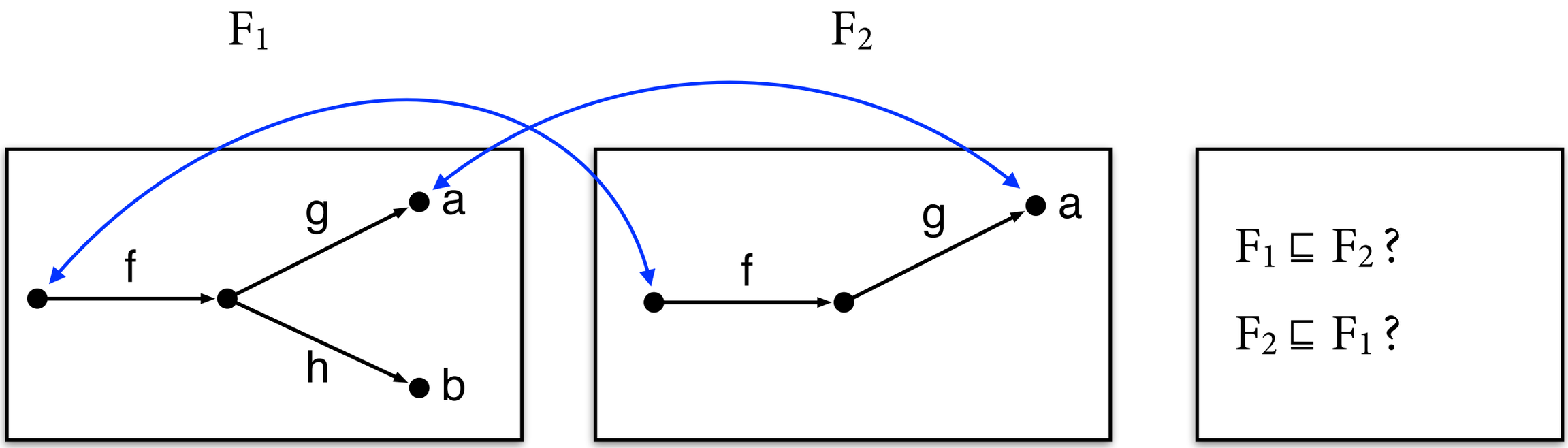
$F_1 \sqsubseteq F_2$?

$F_2 \sqsubseteq F_1$?

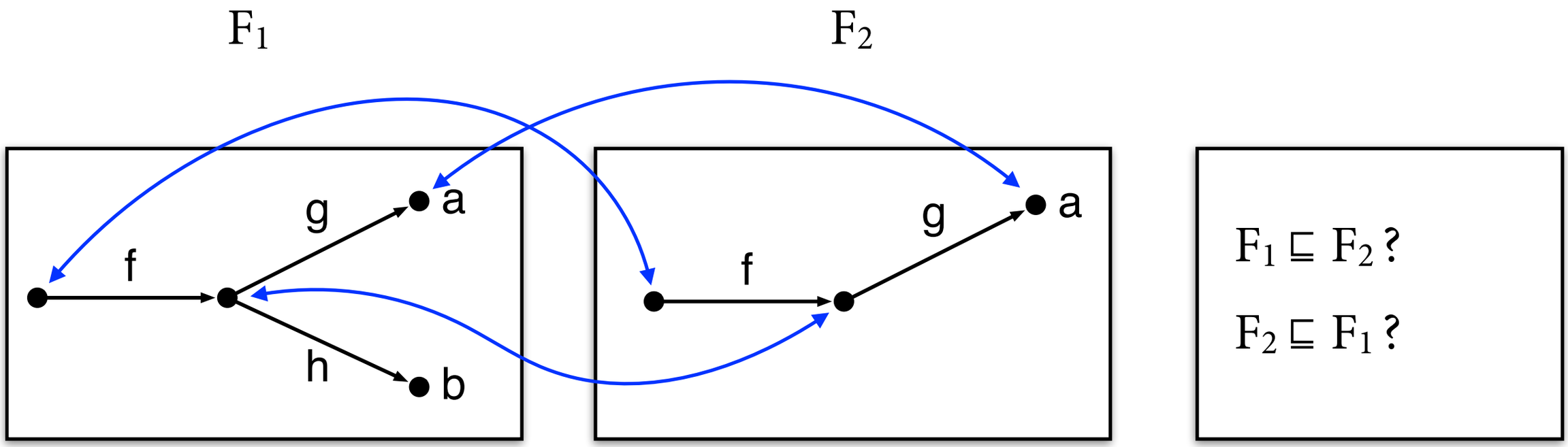
Beispiel



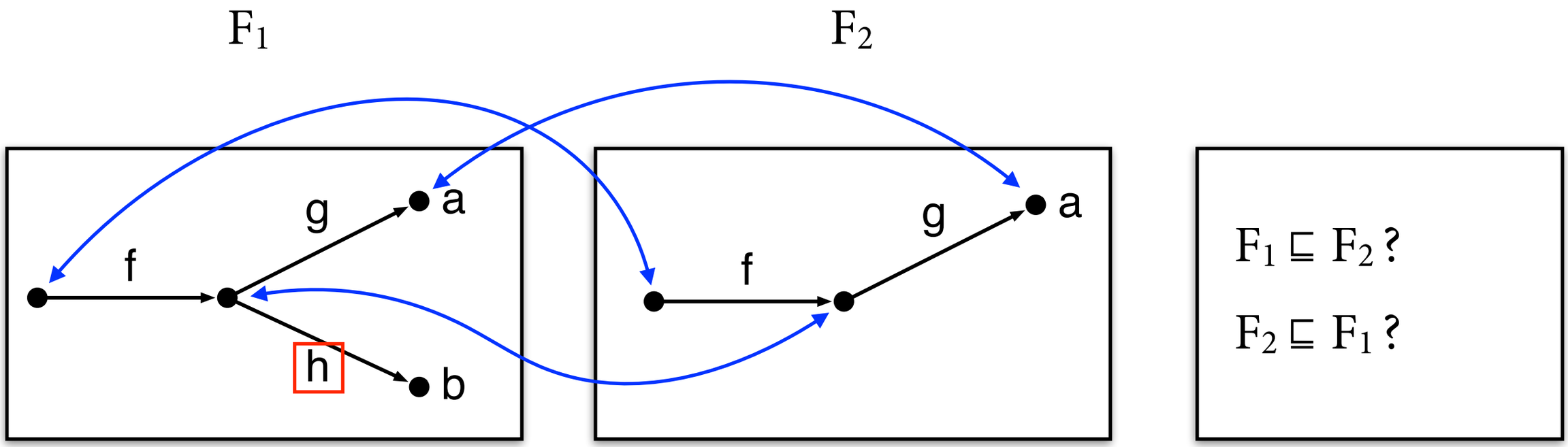
Beispiel



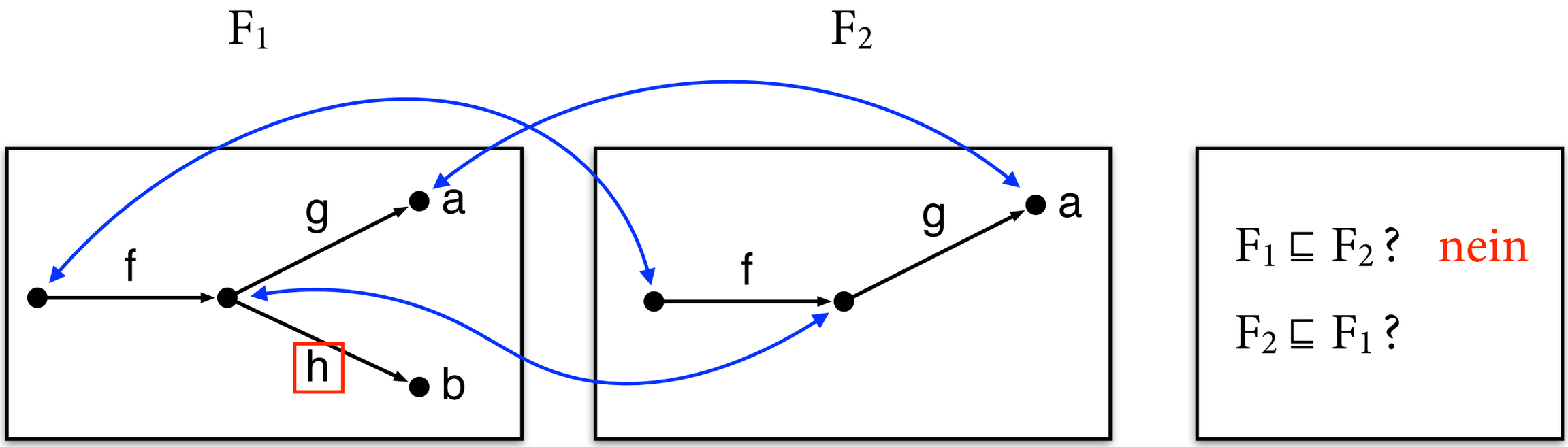
Beispiel



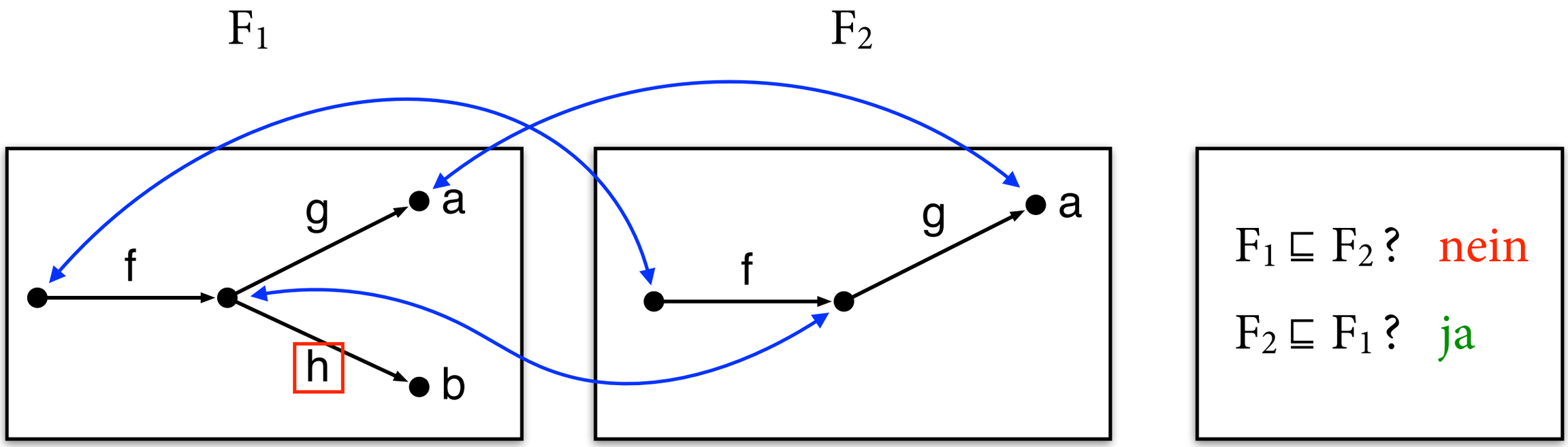
Beispiel



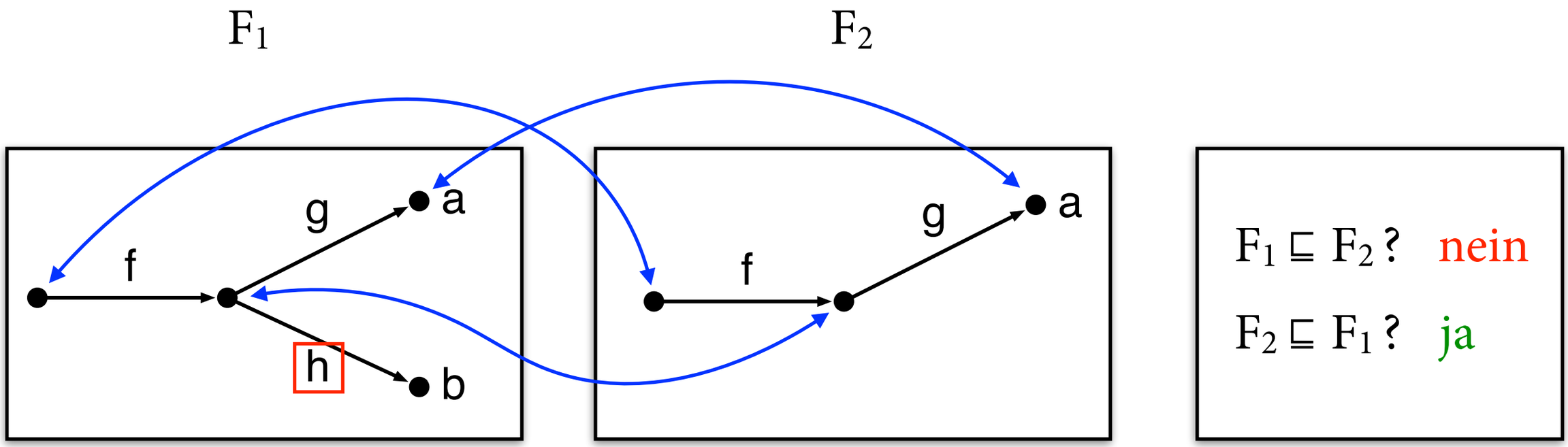
Beispiel

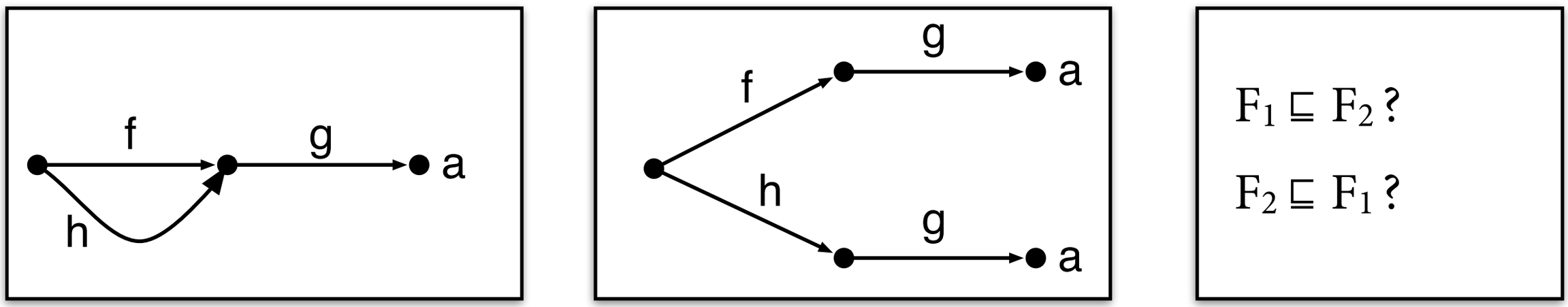


Beispiel

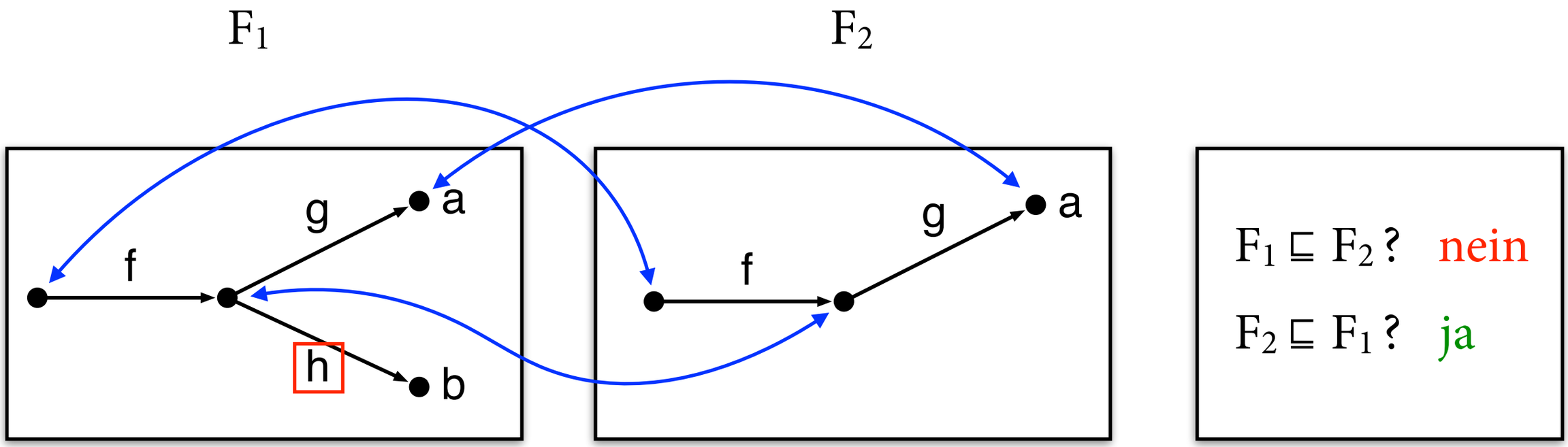


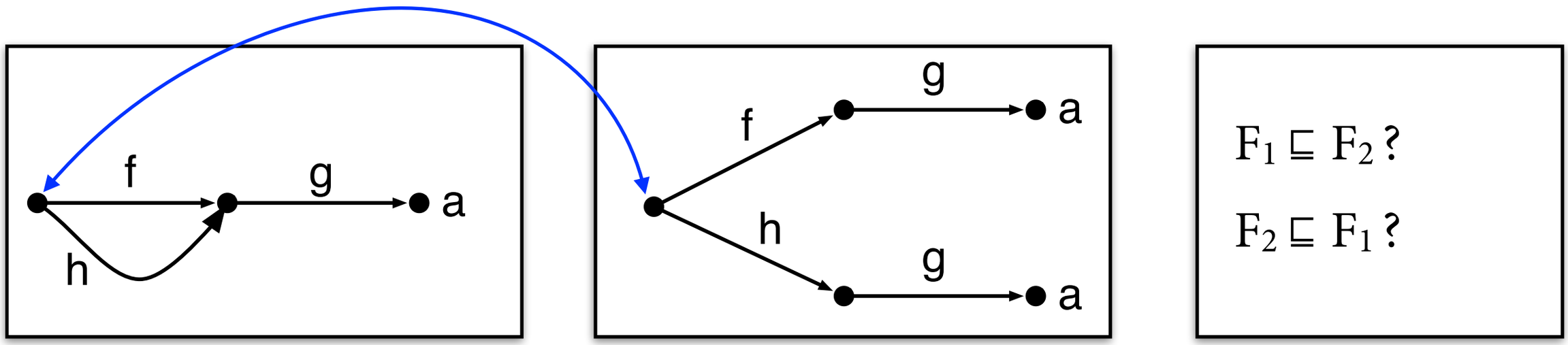
Beispiel



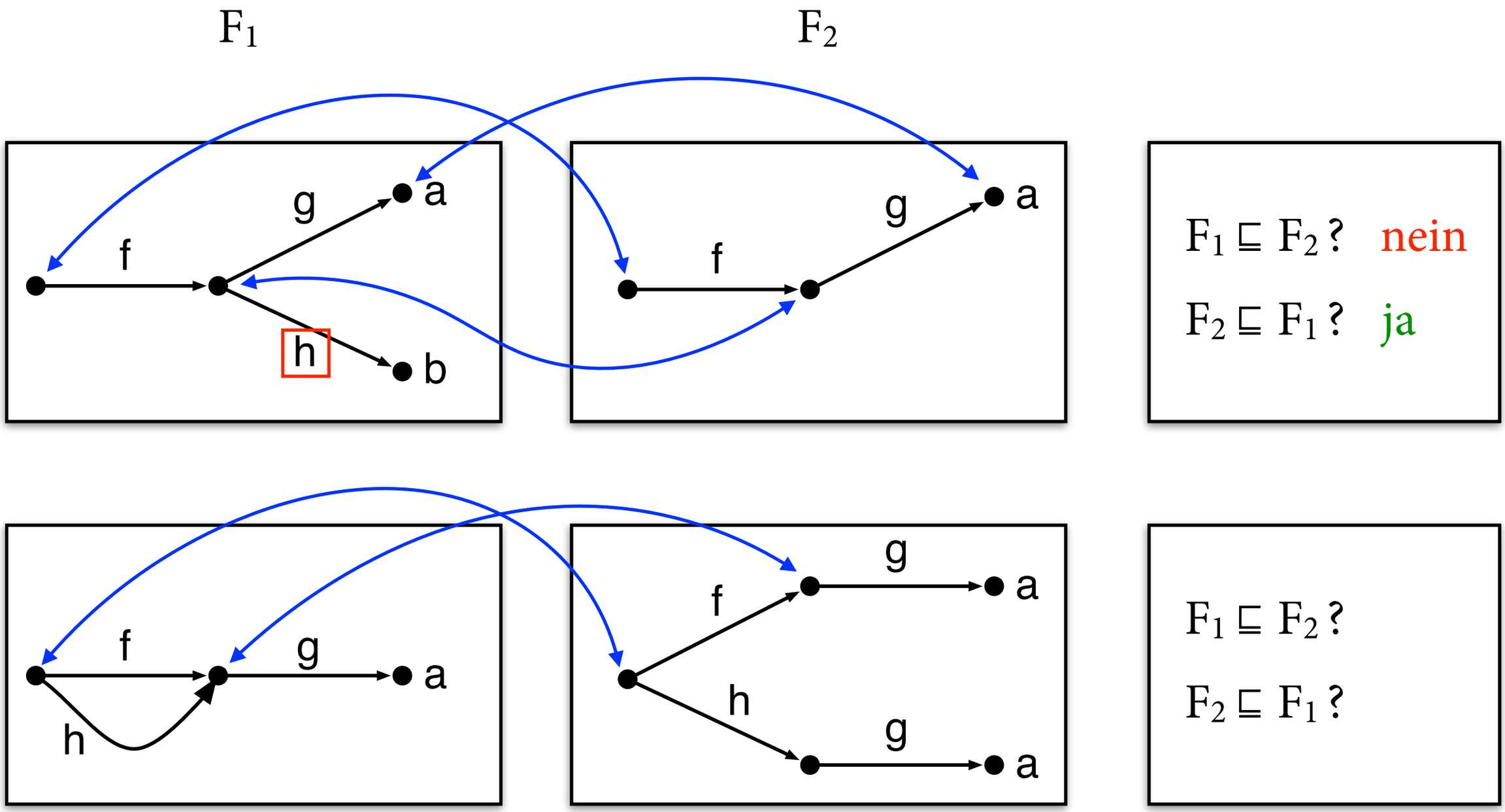


Beispiel

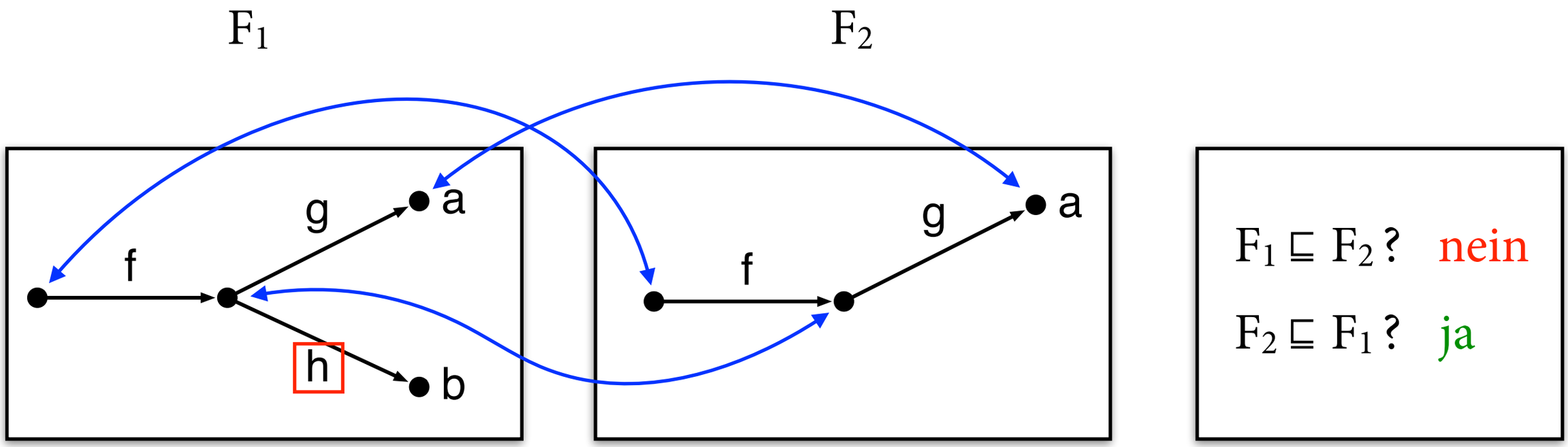




Beispiel

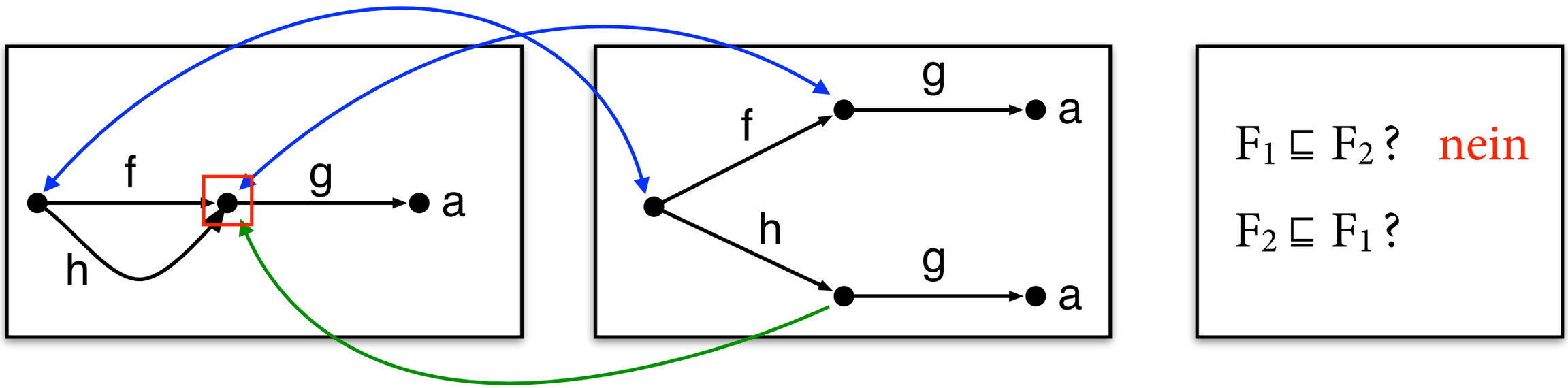


Beispiel



$F_1 \sqsubseteq F_2$? nein

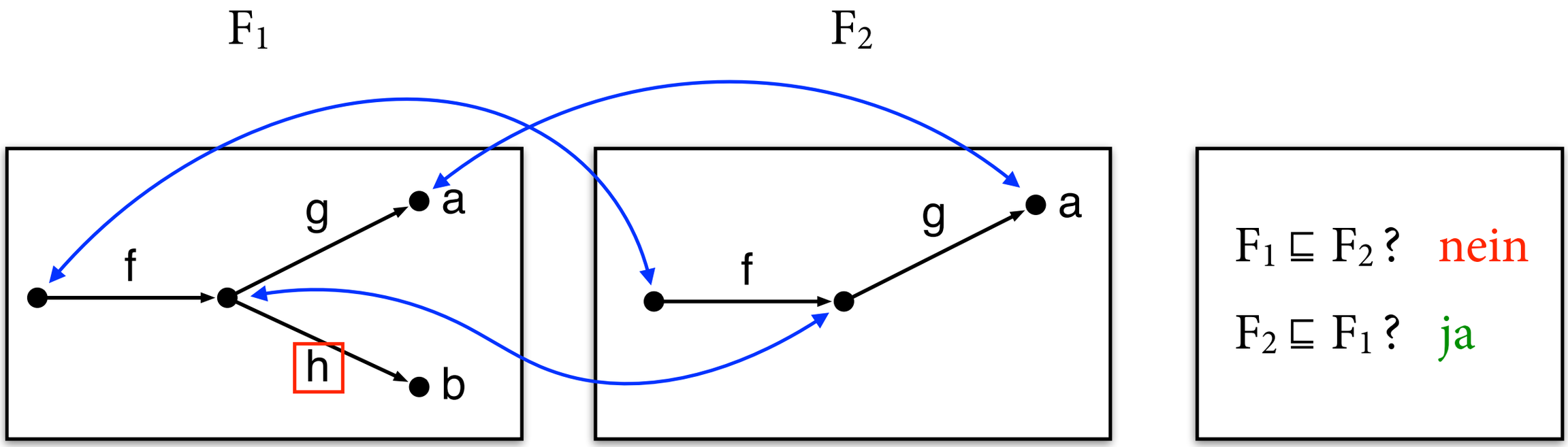
$F_2 \sqsubseteq F_1$? ja



$F_1 \sqsubseteq F_2$? nein

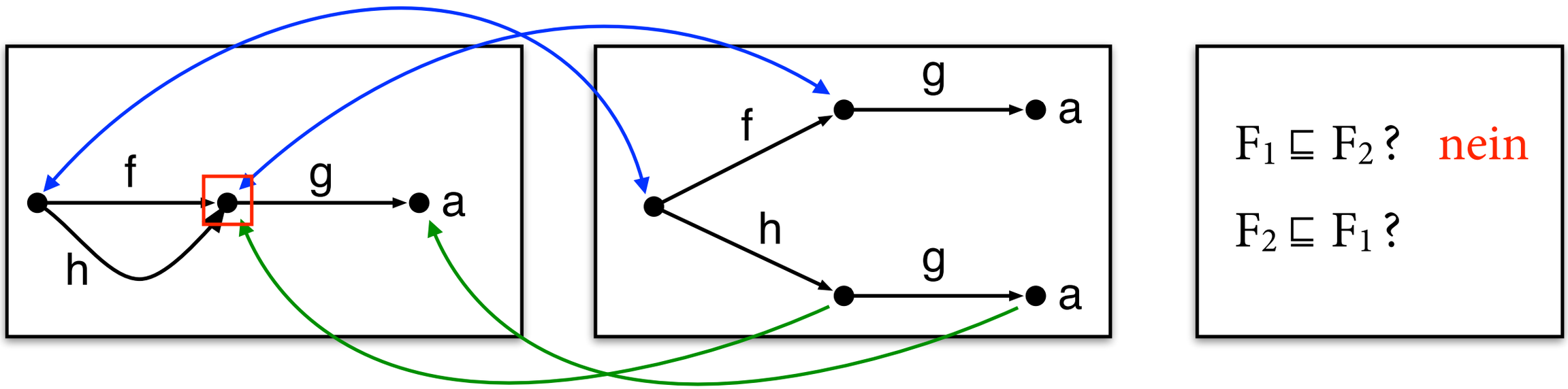
$F_2 \sqsubseteq F_1$?

Beispiel



$F_1 \sqsubseteq F_2$? nein

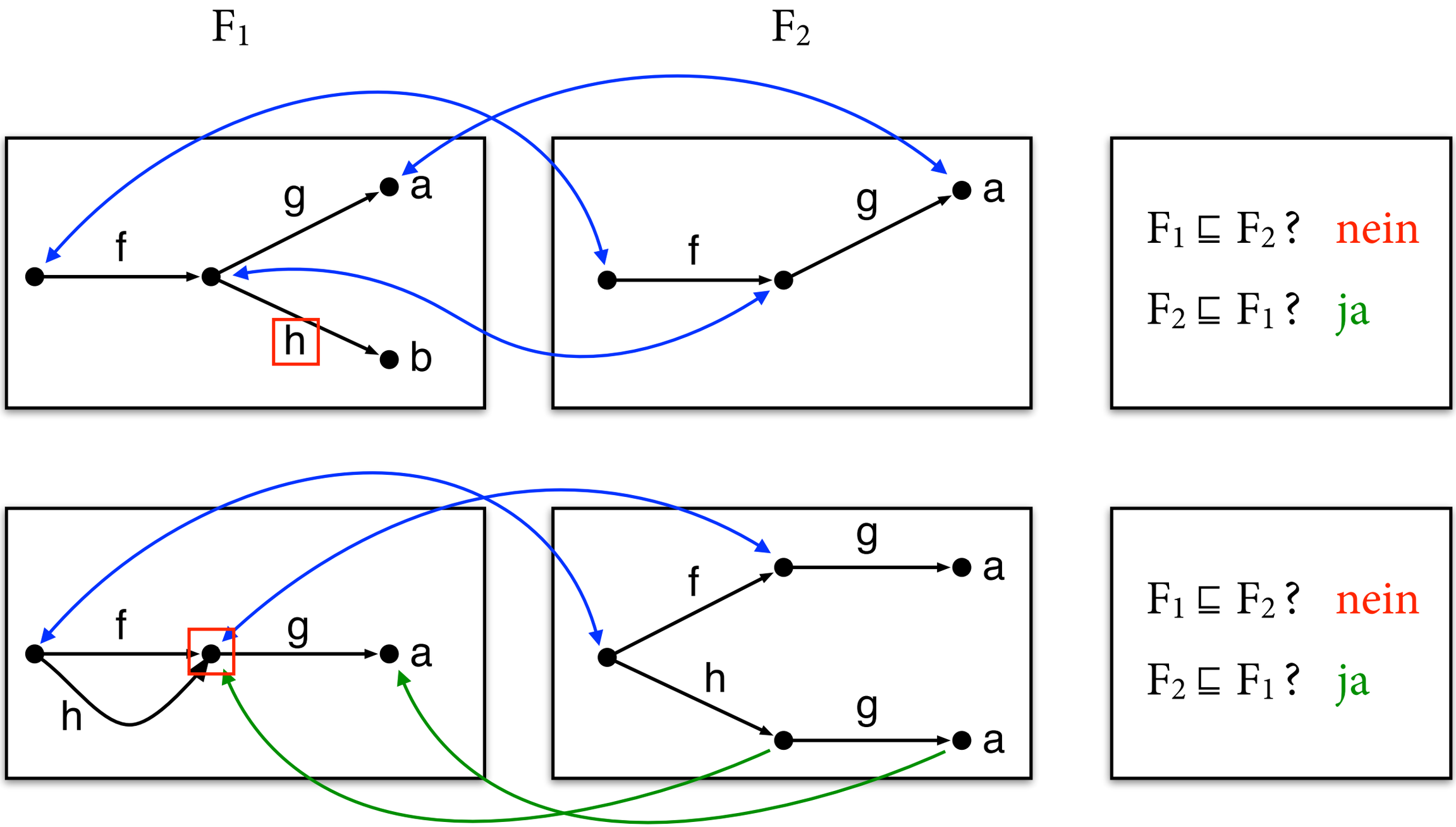
$F_2 \sqsubseteq F_1$? ja



$F_1 \sqsubseteq F_2$? nein

$F_2 \sqsubseteq F_1$?

Beispiel



Unifikation

- Grundidee: Rekursiv durch Featurestrukturen gehen und Pointer auf korrespondierende Knoten setzen.
  - Dabei wird Inhalt von F_2 nach F_1 kopiert.
  - In ähnlicher Weise wie bei Subsumption auf Konsistenz der Information testen.

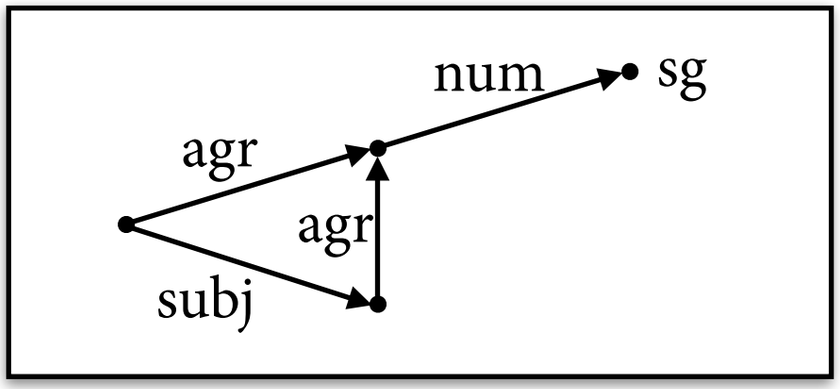
- Rolle des Kopierens:
  - Im worst case kopiert Algorithmus fast die ganze FS F_2, bevor er erkennt, dass die FS nicht unifiziert werden können.
  - Schnellere Algorithmen verzögern und vermeiden Kopieren so gut wie möglich, z.B. Tomabechi (1991).

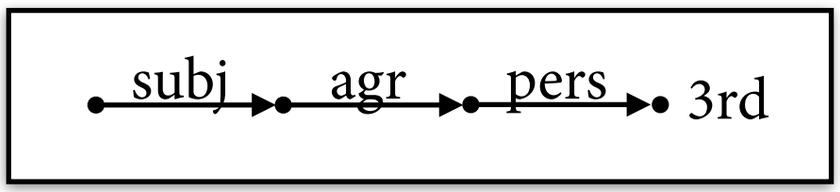
Unifikationsalgorithmus

```
function UNIFY(f1, f2) returns fstructure or failure

  f1-real ← Real contents of f1
  f2-real ← Real contents of f2

  if f1-real is null then
    f1.pointer ← f2
    return f2
  else if f2-real is null then
    f2.pointer ← f1
    return f1
  else if f1-real and f2-real are identical then
    f1.pointer ← f2
    return f2
  else if both f1-real and f2-real are complex feature structures then
    f2.pointer ← f1
    for each feature in f2-real do
      other-feature ← Find or create
                          a feature corresponding to feature in f1-real
      if UNIFY(feature.value, other-feature.value) returns failure then
        return failure
    return f1
  else return failure
```

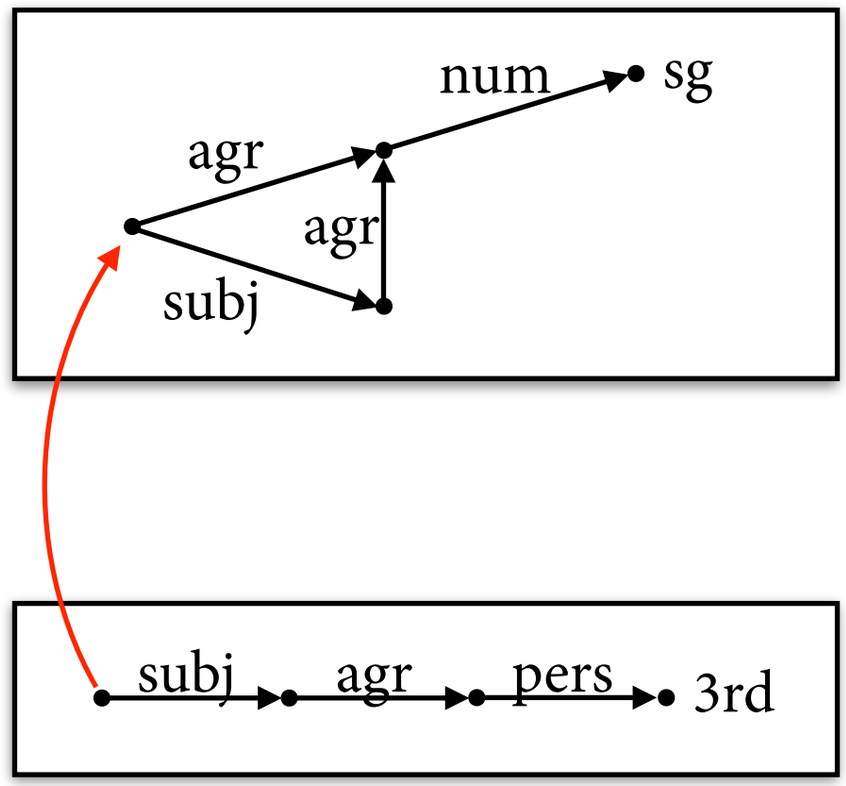
(aus, aber nicht von, Jurafsky & Martin)

Beispiel

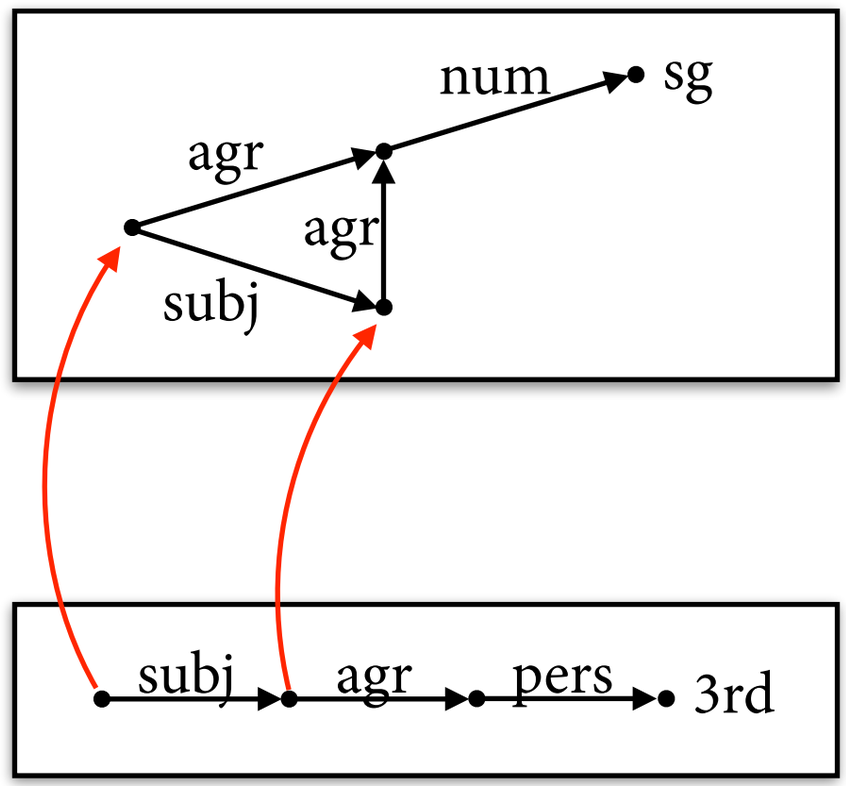




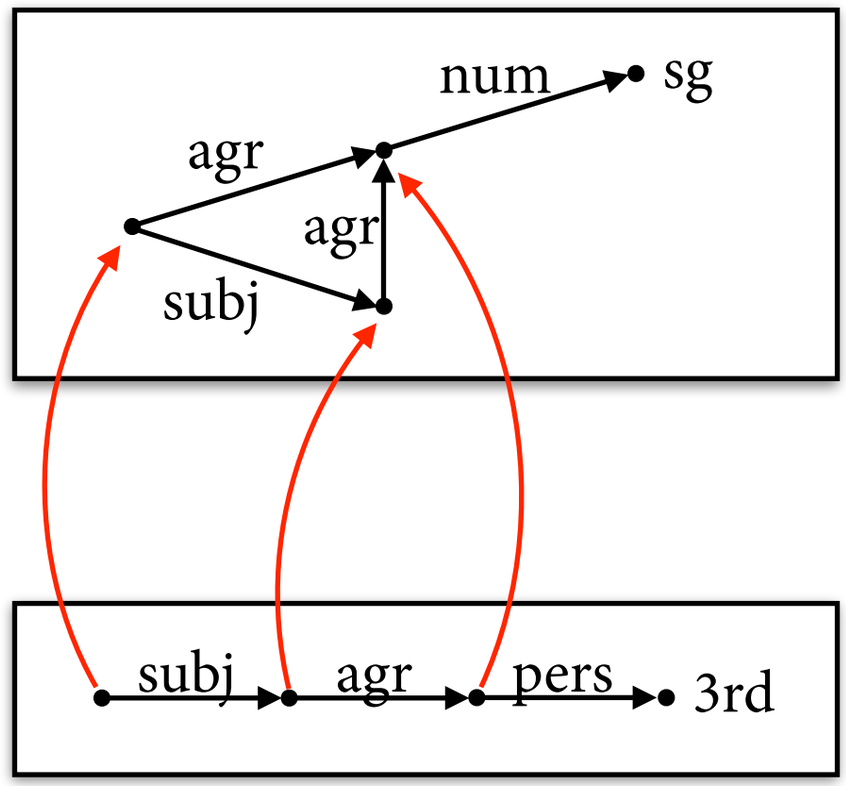
Beispiel



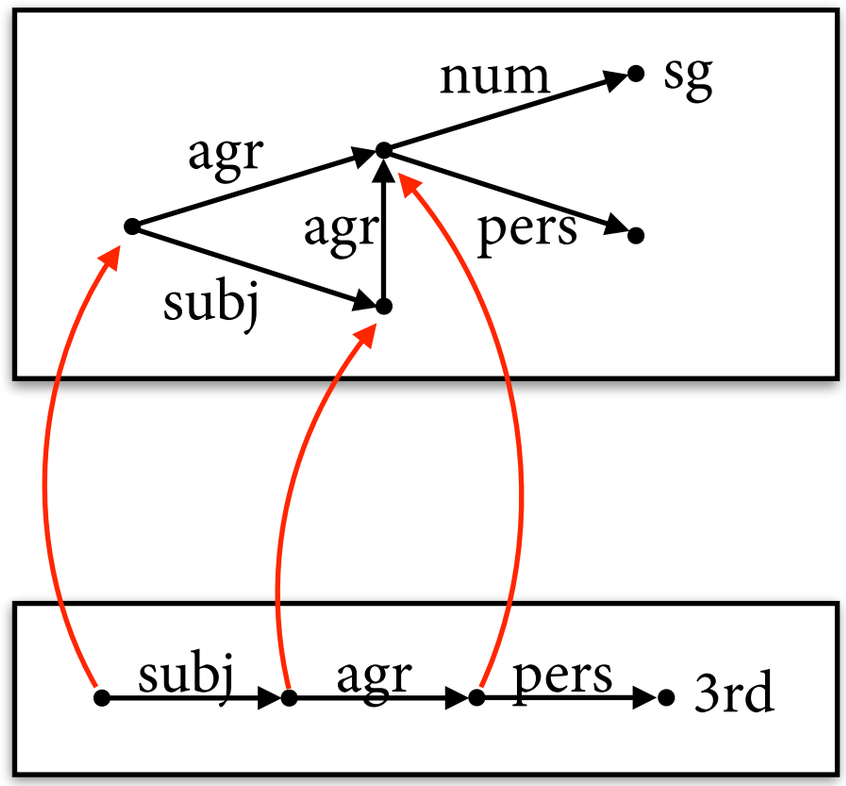
Beispiel



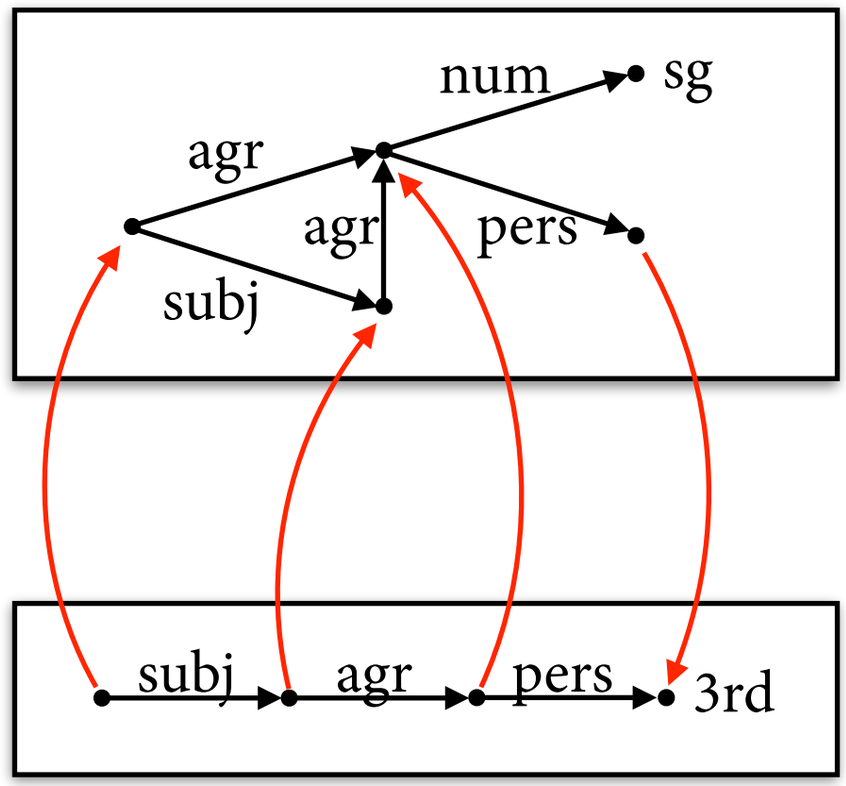
Beispiel



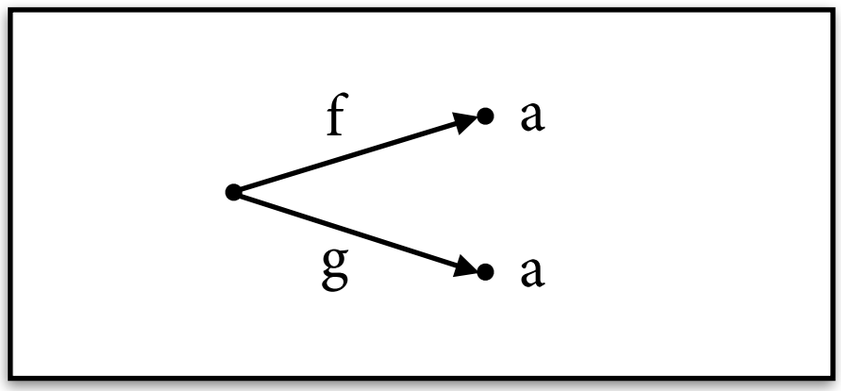
Beispiel

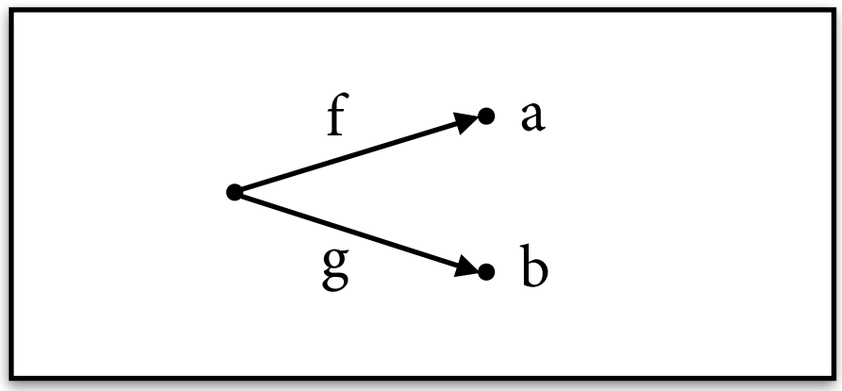


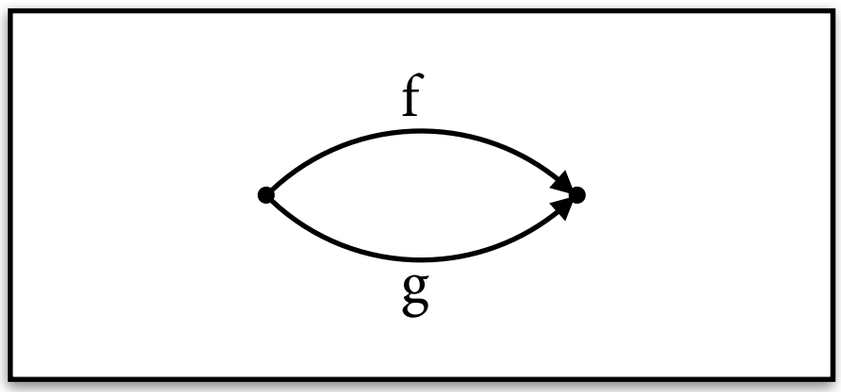
Beispiel

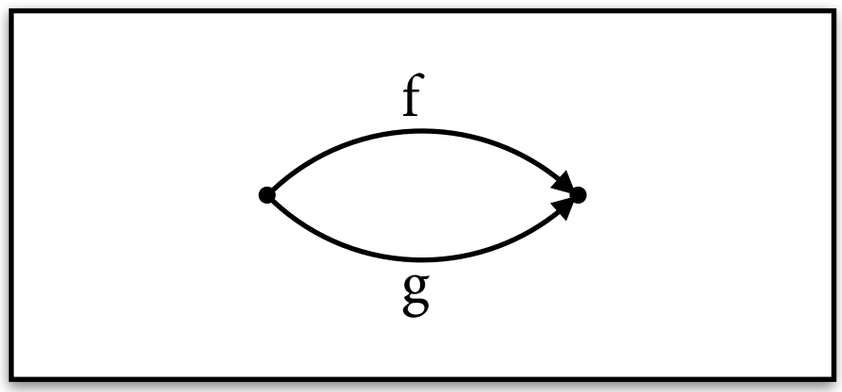


Beispiel

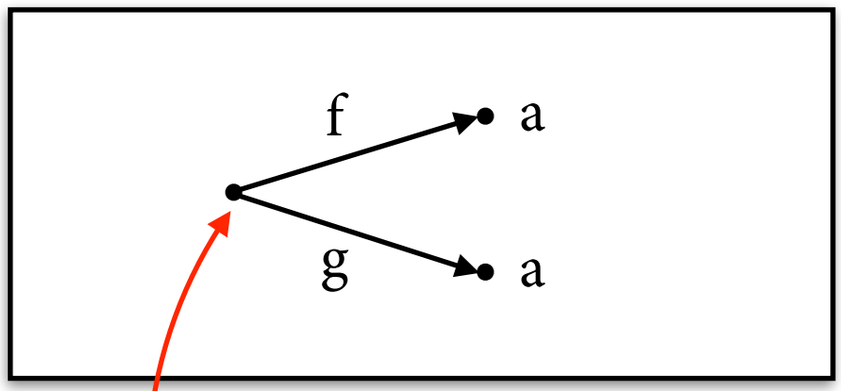


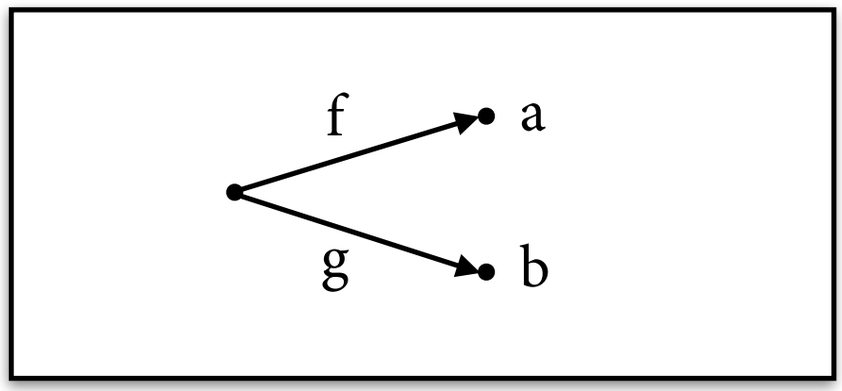


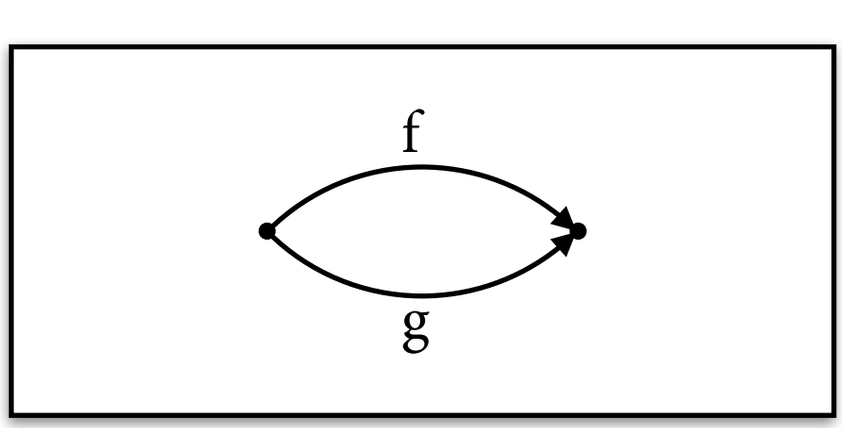




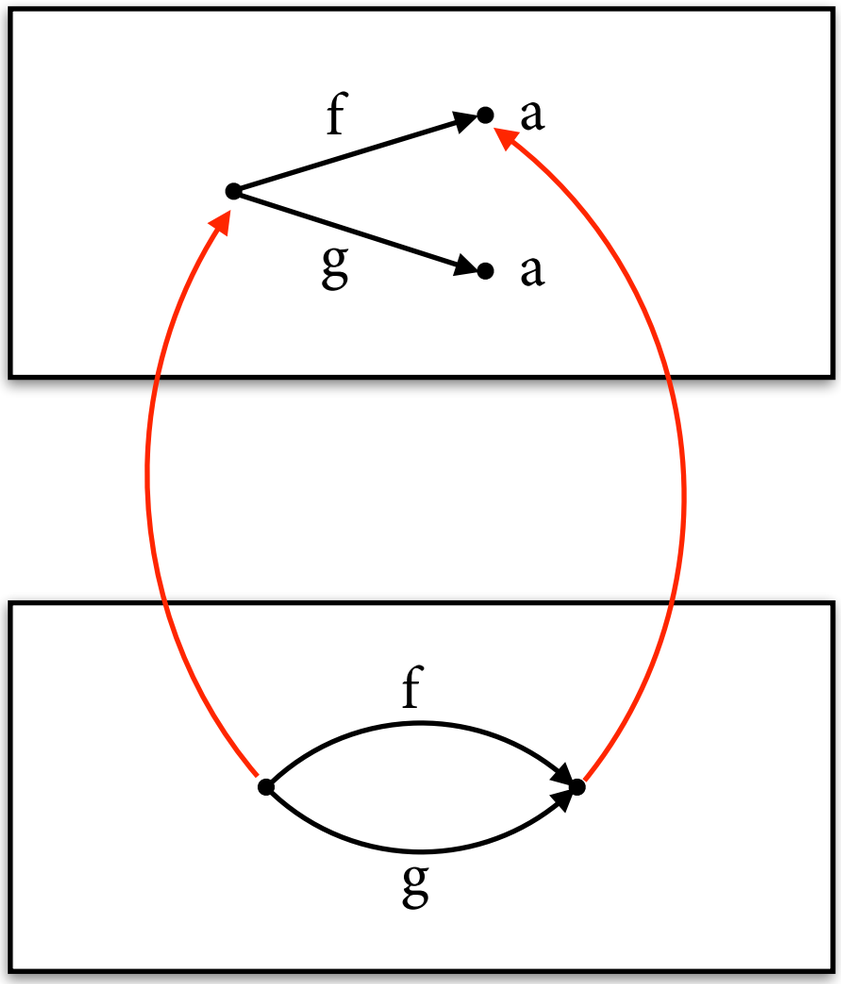
Beispiel

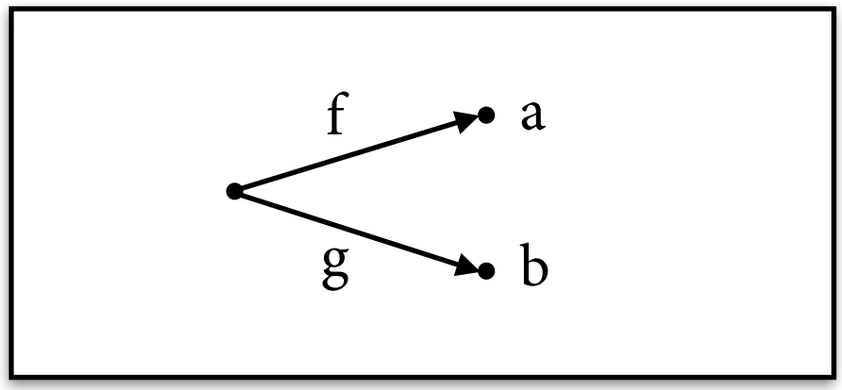


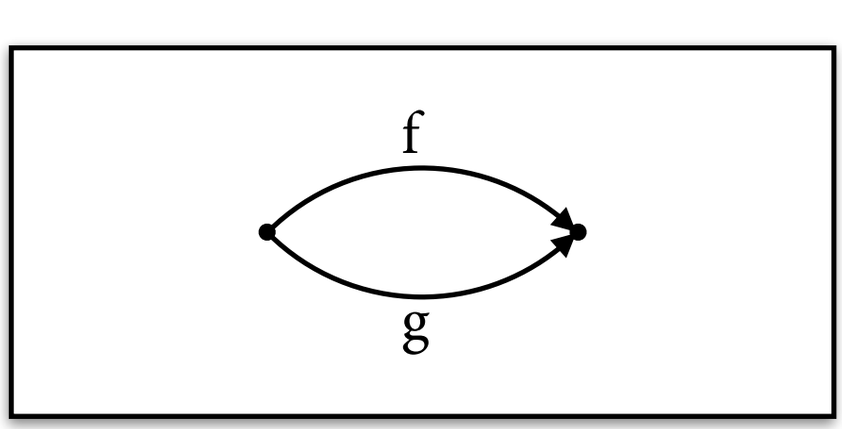




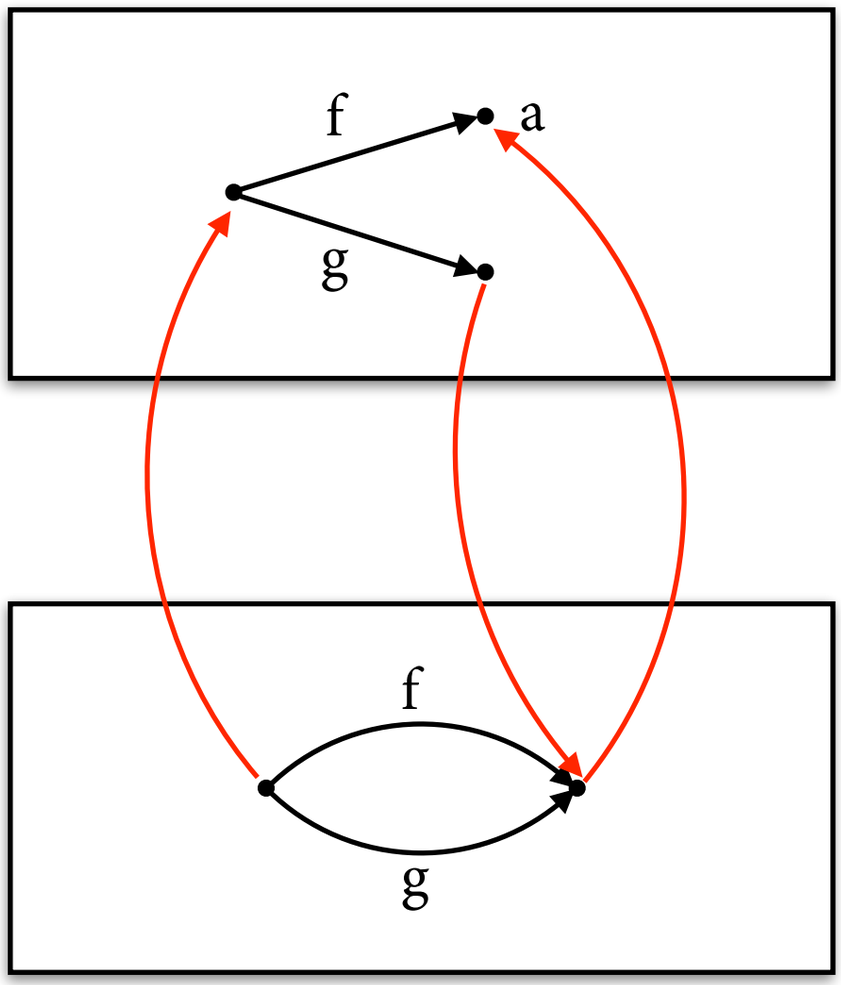
Beispiel

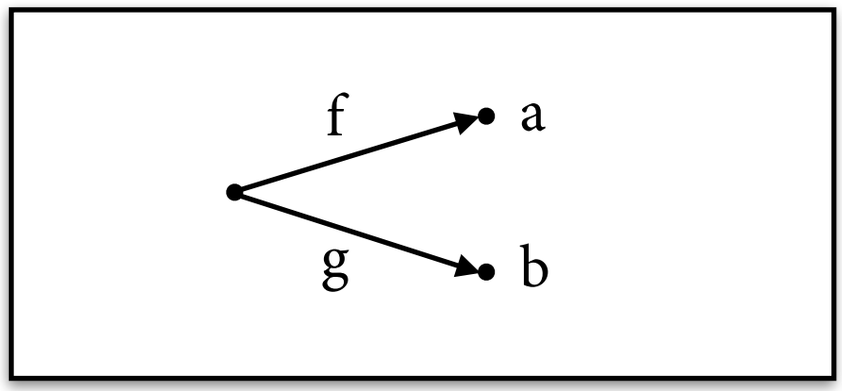


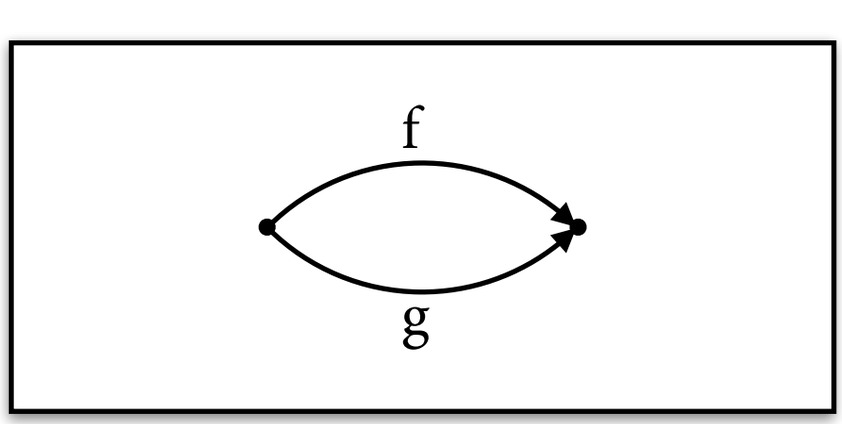




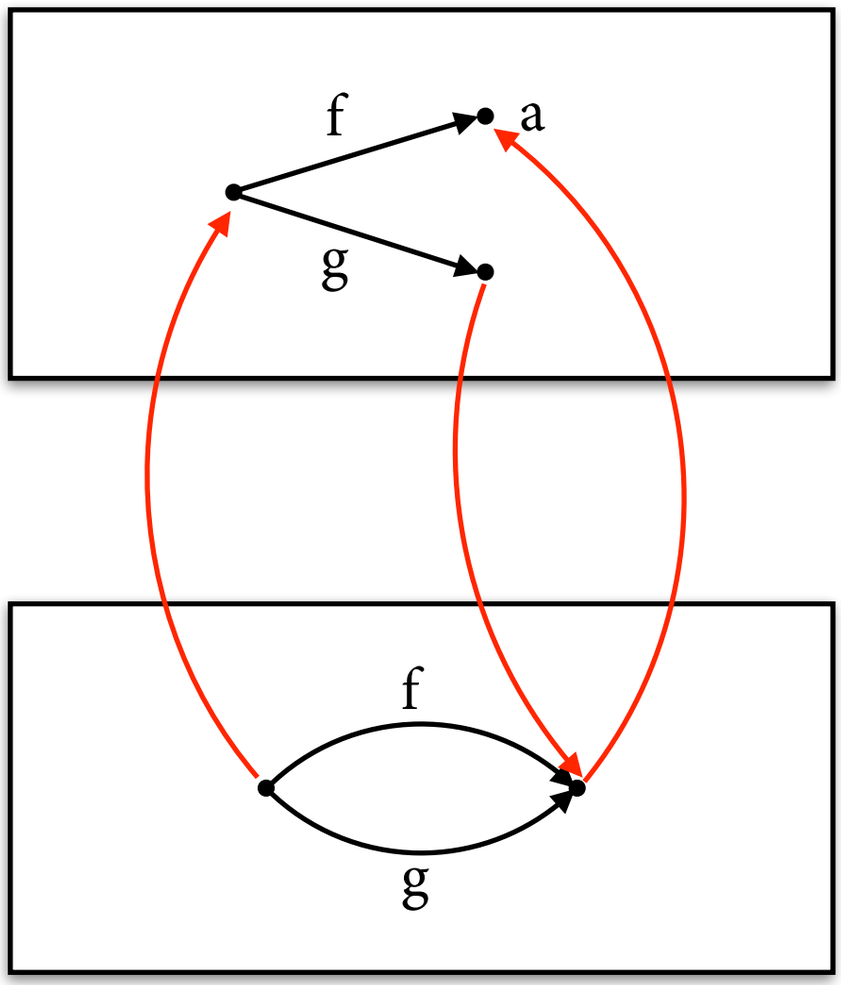
Beispiel

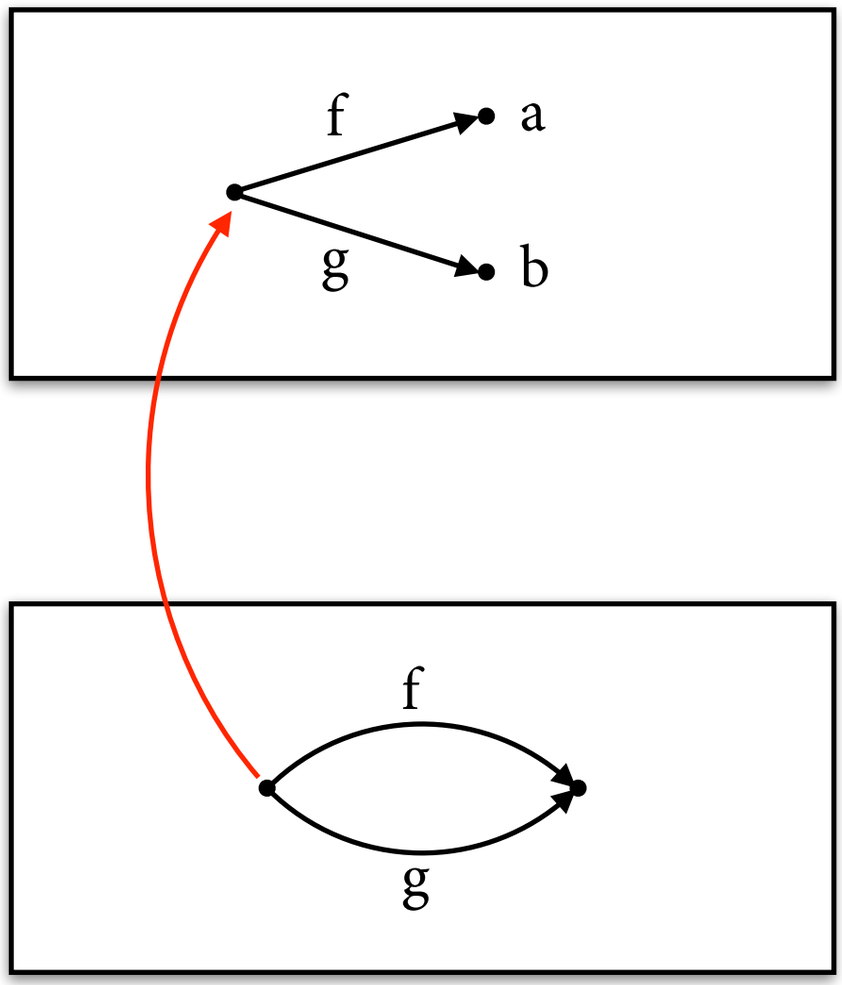




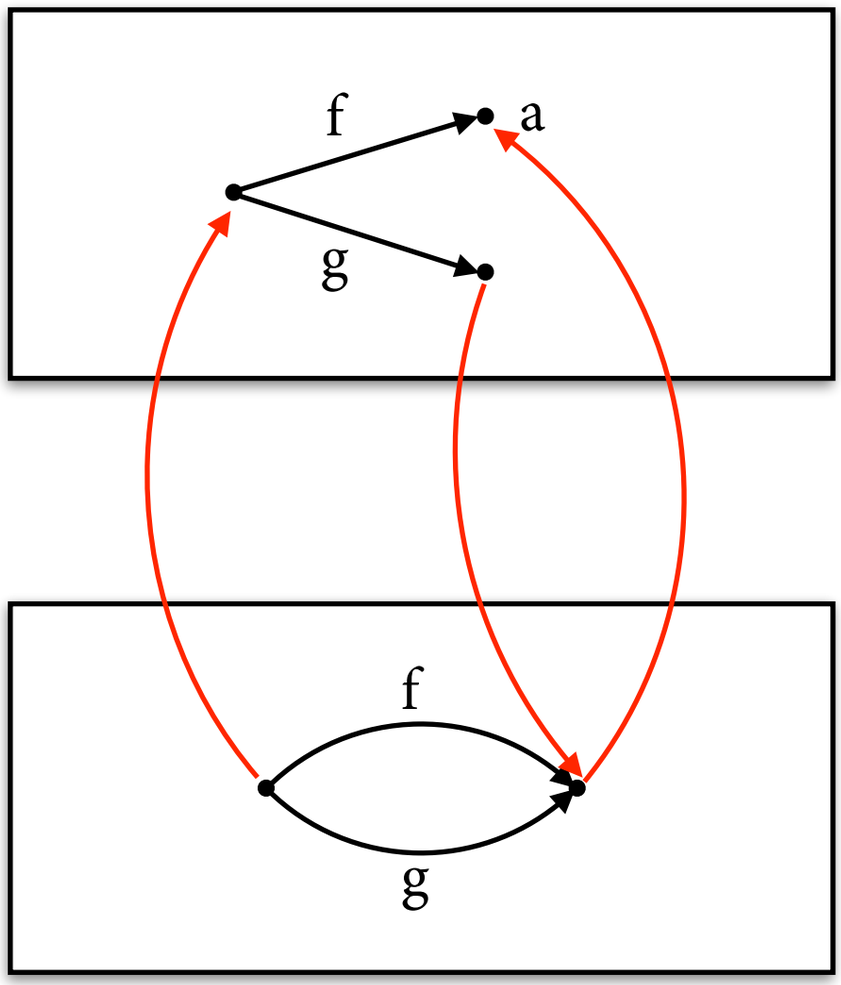


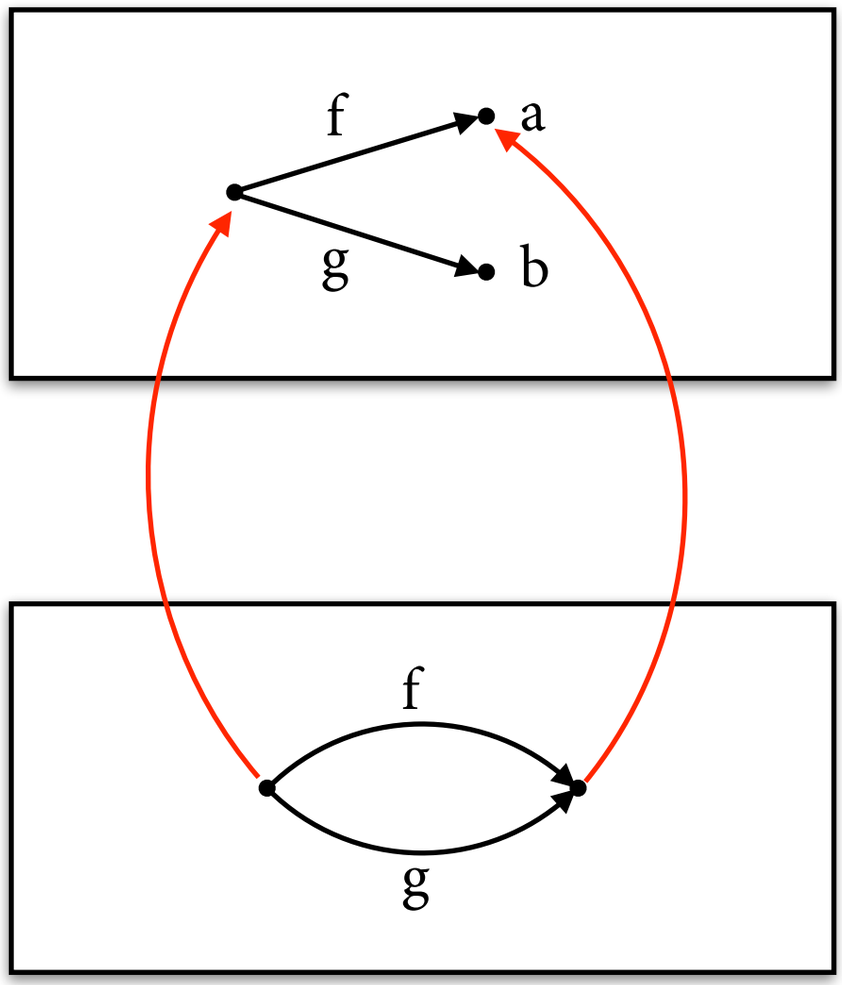
Beispiel



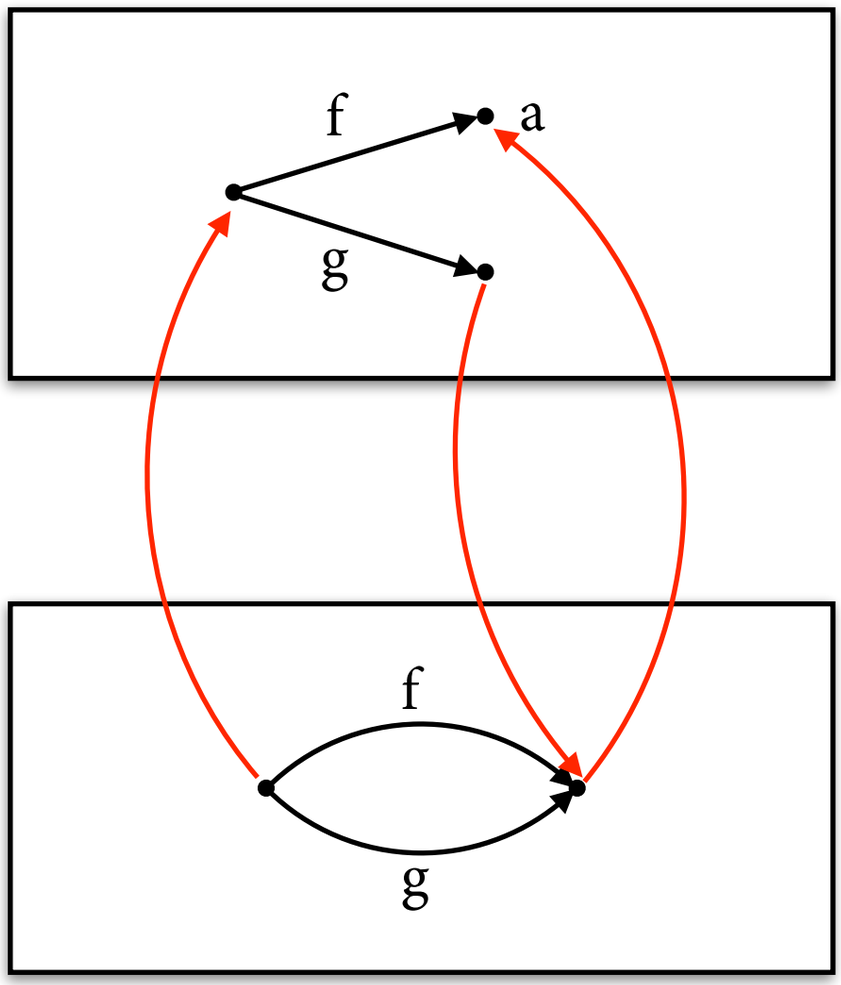


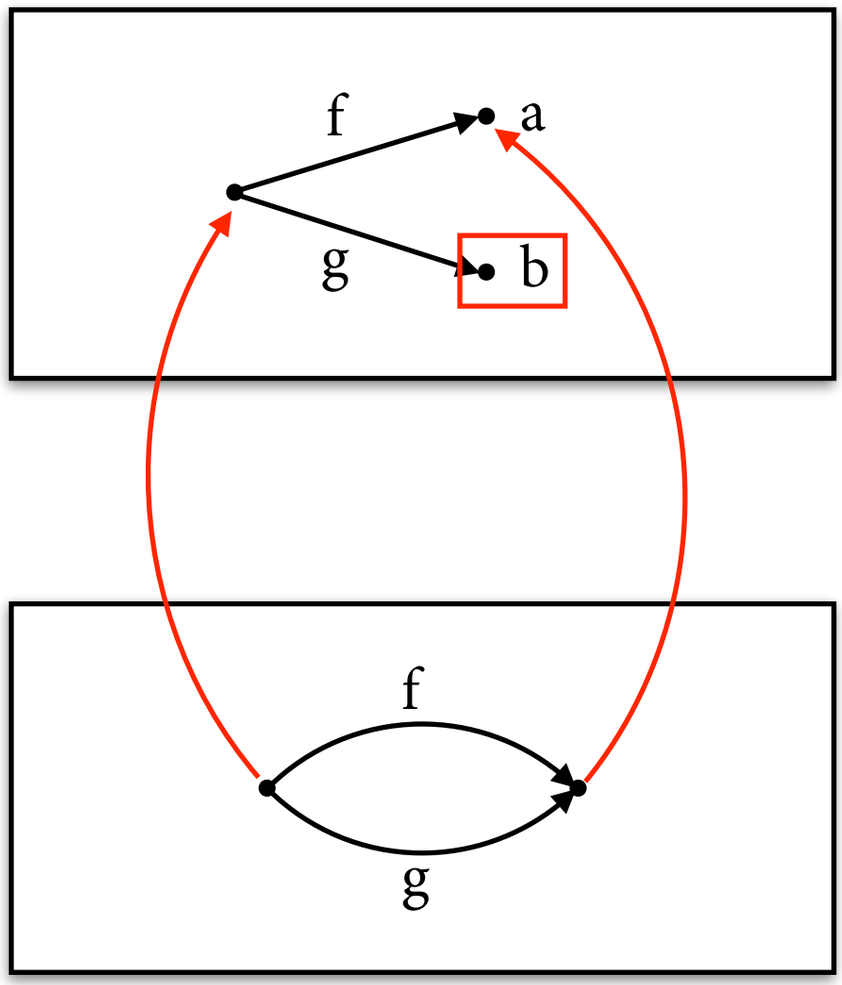
Beispiel





Beispiel





Evaluation

Unifier	**tcpu** (s)	(s)	**space** (kb)	**copying over** (nodes)	**early** (nodes)	(nodes)	**redundant** (nodes)
quick check	*on*	*off*	*on*	*on/off*	*on*	*off*	*on/off*
unify1	0.281	1.151	6,672	39.5	226.9	208.2	156.0
unify2	0.205	0.407	5,614	4.2	78.3	52.6	120.6
unify3	0.232	0.406	5,702	9.7	161.0	137.4	126.2
tomabechi	0.170	0.267	5,214	–	–	–	83.6
tom-smart	0.167	0.258	2,174	–	–	–	–

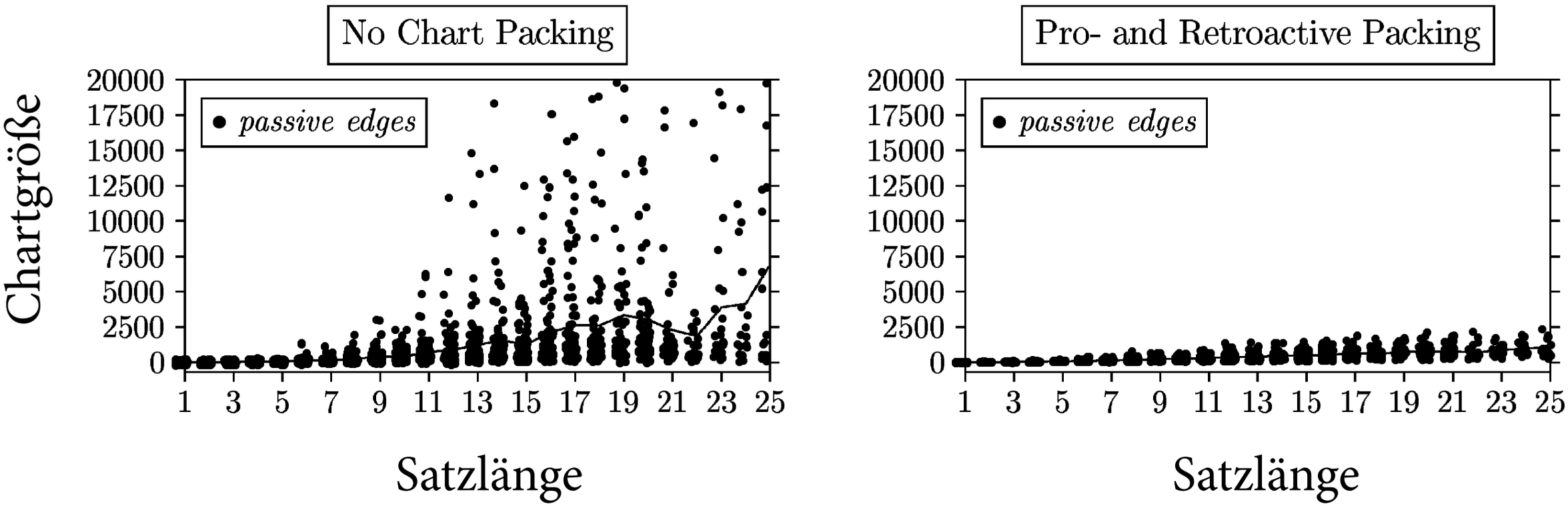
(Callmeier 2000; englisches Verbmobil-Testset mit LinGO-Grammatik; gemessen auf 500 MHz Pentium III)

Ambiguitäten packen

- Unter bestimmten Umständen kann man zwei FSen als äquivalent gelten lassen.
  - wenn Parser (i,k,σ) kannte und (i,k,τ) findet und $\tau \sqsubseteq \sigma$ (d.h. ist allgemeiner), dann darf er (i,k,σ) löschen.
  - Chart bleibt kleiner ⇒ Parsing wird schneller.
- Löschen von Einträgen, die aus (i,k,σ) abgeleitet wurden, ist nicht trivial.

(Oepen & Carroll 00 etc.)

Evaluation



(Oepen & Carroll 2000; englisches Verbmobil-Testset mit LinGO-Grammatik)

Approximation

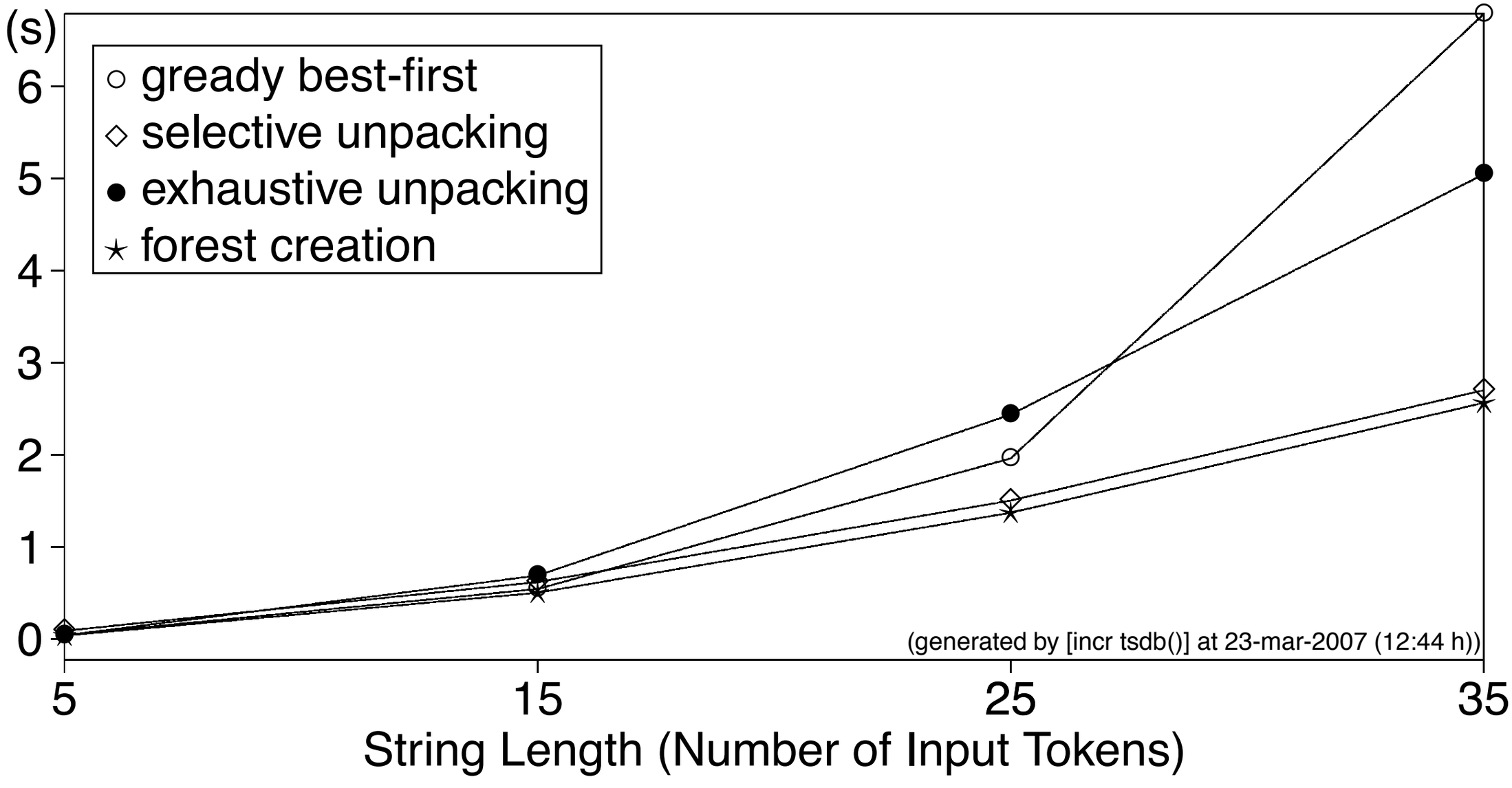
- HPSG-Schemata definieren Phrasenstruktur. Kann man mit kfG approximieren.
  - [cat=V, subcat=<NPnom>] →
    [cat=V, subcat=<NPnom, NPacc>] [cat=N, subcat=<>]

- Mit (statistischer) kfG parsen; dann versuchen, beste Parsebäume in HPSG-Parses zu expandieren. (Kann fehlschlagen!)

- Unifikationen nur für Strukturen ausführen, die überhaupt eine Chance haben.

(Kiefer & Krieger 00; Zhang & Krieger 11; etc.)

Polynomial Time (Practical) Parsing



→ Average Time for 20-Word Sentences around One Second

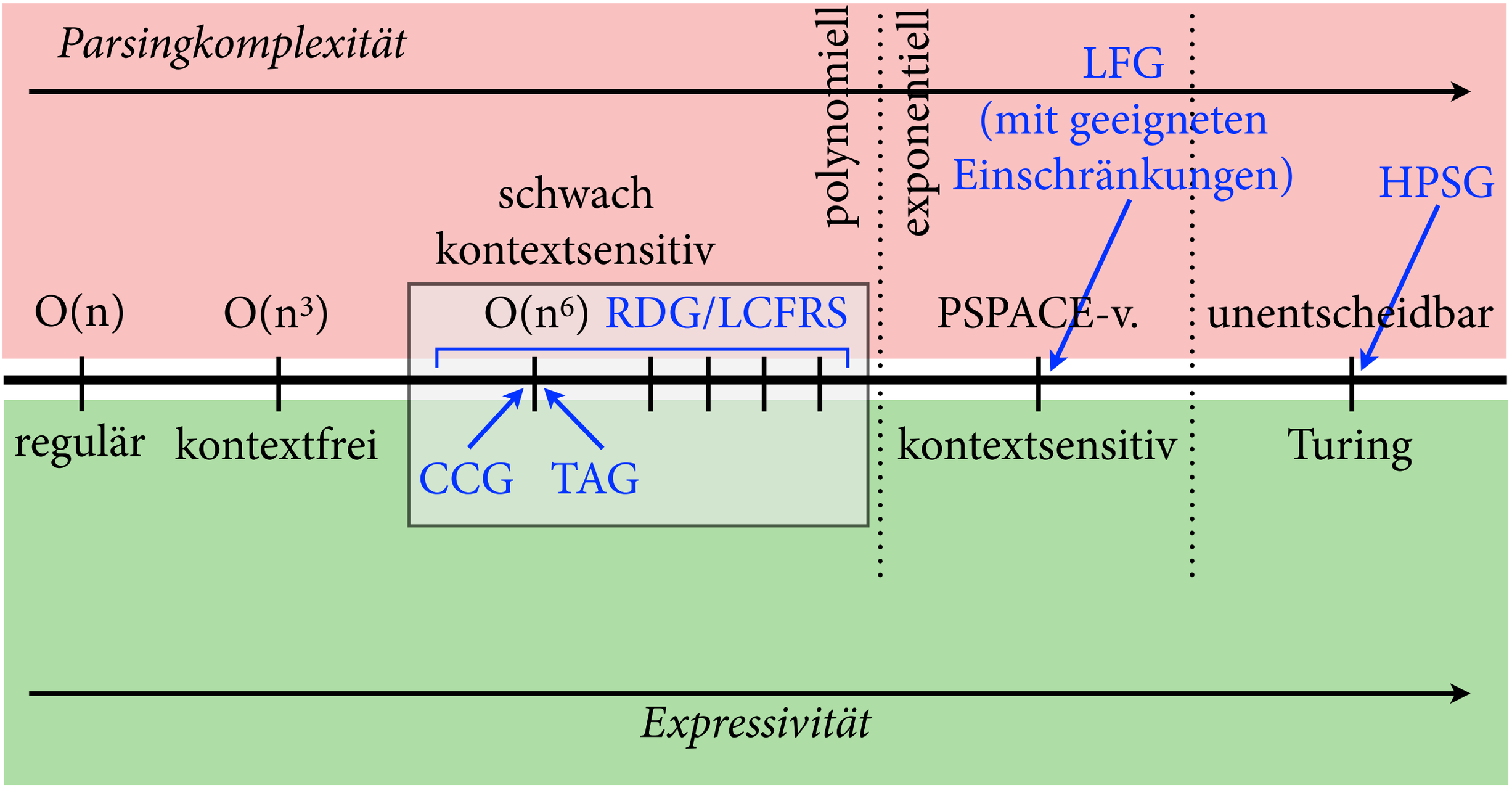
Stand der Kunst

- LKB: Grammatikentwicklungssystem.
  - verwendet Type Description Language (TDL), um Grammatiken aufzuschreiben
- TDL-Grammatiken kann man effizient mit dem PET-Parser verarbeiten.
  - LKB + PET + andere = DELPH-IN
  - DELPH-IN ist internationales Konsortium, das zueinander kompatible Parser, Grammatiken etc. entwickelt: http://www.delph-in.net/

Zusammenfassung

- Expressivität von HPSG: turing-vollständig.
- Parsing von HPSG:
  - Wird in der Praxis dominiert von Zeit für Unifikation und Subsumptionstests.
  - Kann durch geschickte Algorithmen für diese Probleme in der Praxis effizient werden.
  - Algorithmen auch für andere Grammatikformalismen anwendbar, z.B. FTAG, LFG.

Fazit


Parsingkomplexität
polynomiell
exponentiell
LFG
(mit geeigneten
Einschränkungen)
HPSG
schwach
kontextsensitiv
O(n)
O(n³)
O(n⁶)
RDG/LCFRS
PSPACE-v.
unentscheidbar
regulär
kontextfrei
CCG
TAG
kontextsensitiv
Turing
Expressivität