Deduction Based Question Answering and its Application

LogAnswer Embedding AD in QAS

Ulrich Furbach
Björn Pelzer

Ingo Glöckner
Hermann Helbig

Deutsche Forschungsgemeinschaft
DFG

Deduction Based Question Answering and its Application

LogAnswer Embedding AD in QAS

Ulrich Furbach
Björn Pelzer

Ingo Glöckner
Hermann Helbig

Loganswer

AR Aspects

(Application)

Deutsche Forschungsgemeinschaft
DFG

LogAnswer - a Question-Answering System

LogAnswer - a Question-Answering System

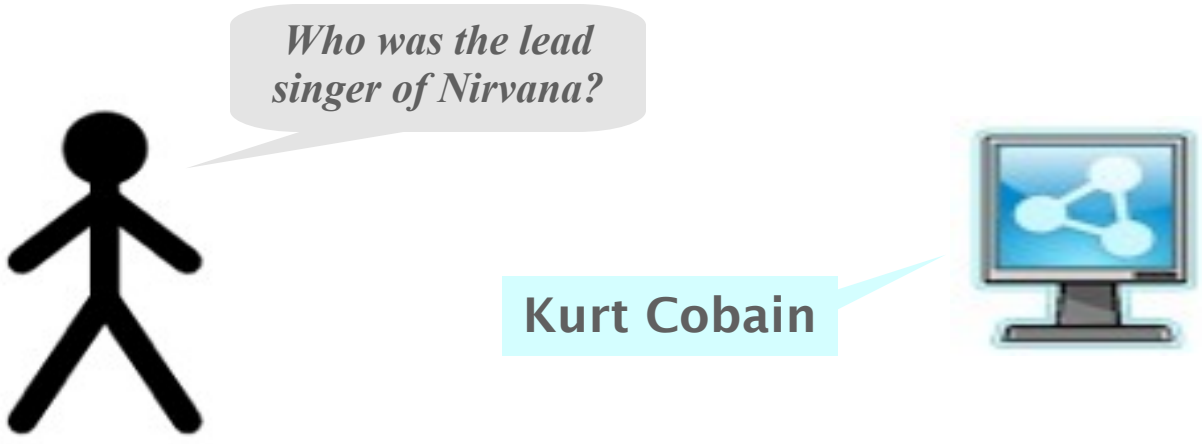
Who was the lead singer of Nirvana?

Kurt Cobain

LogAnswer - a Question-Answering System

An ***open domain question-answering system***

- receives a natural-language question regarding any topic, and
- returns a natural-language answer found in a knowledge-base.



Why use Logic in QA?

Shallow NLP fails when:

Why use Logic in QA?

Shallow NLP fails when:

- words in the knowledge sources do not match exactly:

When did Hannibal cross the Alps?

The Carthaginian general led his army over the Alps in 218 BC.

Why use Logic in QA?

Shallow NLP fails when:

- words in the knowledge sources do not match exactly:

When did Hannibal cross the Alps?

The Carthaginian general led his army over the Alps in 218 BC.

- words match, but they are too far apart:

What is the population of Mongolia?

Mongolia is very sparsely populated for its size. The landlocked country between China and Russia has a population of 2.9 million people.

The LogAnswer project combines

FernUniversität in Hagen **NLP**	UNIVERSITÄT KOBLENZ · LANDAU **Deduction**
• NLP-components • MultiNet knowledge base, based on snapshot of German Wikipedia	• theorem prover E-KRHyper

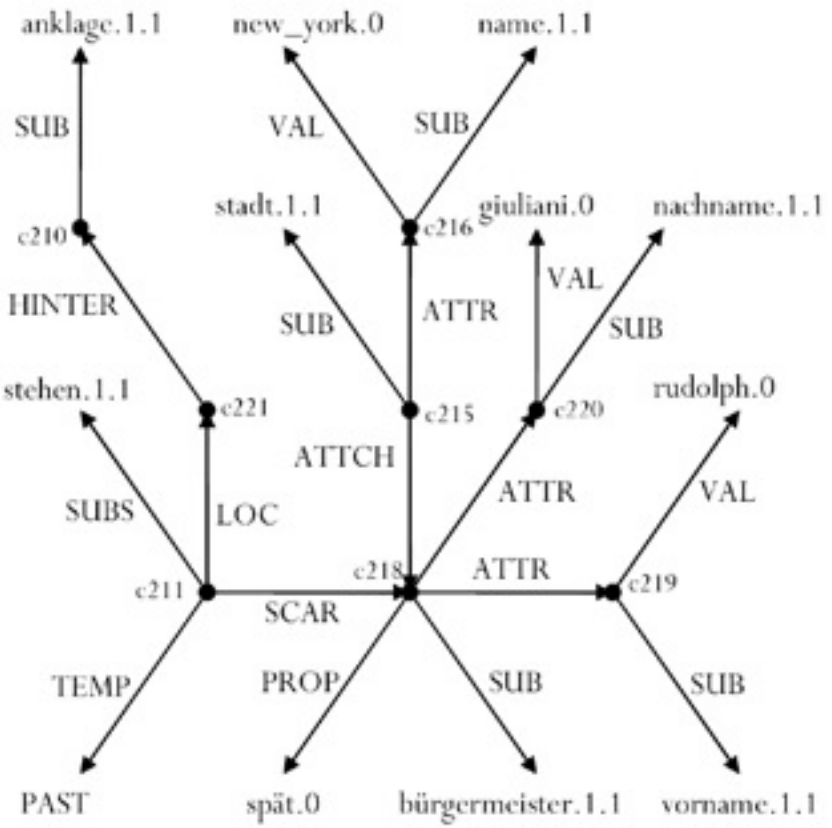
MultiNet Knowledge Base

MultiNet:

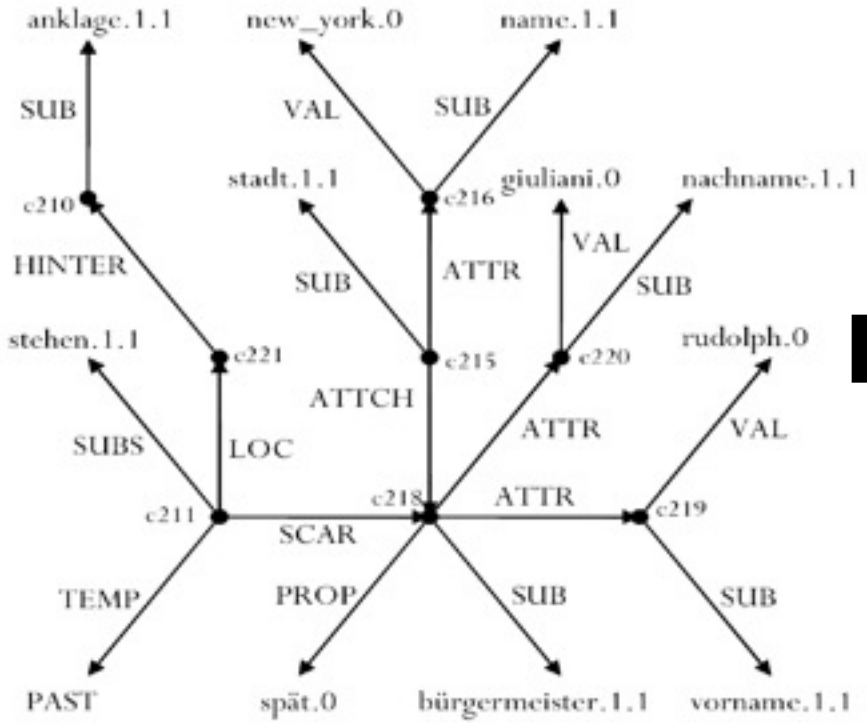
- **Multi**layered Extended Semantic **Net**works
- language independent, but tools mostly German

MultiNet Knowledge Base

"Hinter der Anklage stand der spätere Bürgermeister von New York, Rudolph Giuliani."



MultiNet to First-Order Logic



MultiNet

$$
\begin{aligned}
&\text{hinter}(c221,\ c210)\\
&\wedge\ \text{sub}(c220,\ nachname.1.1)\\
&\wedge\ \text{val}(c220,\ giuliani.0)\\
&\wedge\ \text{sub}(c219,\ vorname.1.1)\\
&\wedge\ \text{val}(c219,\ rudolph.0)\\
&\wedge\ \text{prop}(c218,\ spät.1.1)\\
&\wedge\ \text{attr}(c218,\ c220)\\
&\wedge\ \text{attr}(c218,\ c219)\\
&\wedge\ \text{sub}(c218,\ bürgermeister.1.1)\\
&\wedge\ \text{val}(c216,\ new\_york.0)\\
&\wedge\ \text{sub}(c216,\ name.1.1)\\
&\wedge\ \text{sub}(c215,\ stadt.1.1)\\
&\wedge\ \text{attch}(c215,\ c218)\\
&\wedge\ \text{attr}(c215,\ c216)\\
&\wedge\ \text{subs}(c211,\ stehen.1.1)\\
&\wedge\ \text{loc}(c211,\ c221)\\
&\wedge\ \text{scar}(c211,\ c218)\\
&\wedge\ \text{temp}(c211,\ past)\\
&\wedge\ \text{sub}(c210,\ anklage.1.1)
\end{aligned}
$$

First-Order Logic

MultiNet Knowledge Base

- snapshot of German Wikipedia
- formal representations of ~12 million sentences, generated semi-automatically
- plus ~12,000 background knowledge axioms, manually adapted from MultiNet inference rules and from WordNet

Hypertableau

- splitting with purification solves the problem of variables shared between tableau branches:
- only one branch needs to be worked on at any time — Jelia 96
- emphasis on unit operations
- proof confluent

E-hyper tableau calculus:

- clause tree instead of literal tree — Cade 07
- four extension rules instead of one
- adds term ordering (reduction ordering)
- adds redundancy handling

The LogAnswer question answering system:

"Who was Ian Fleming?"

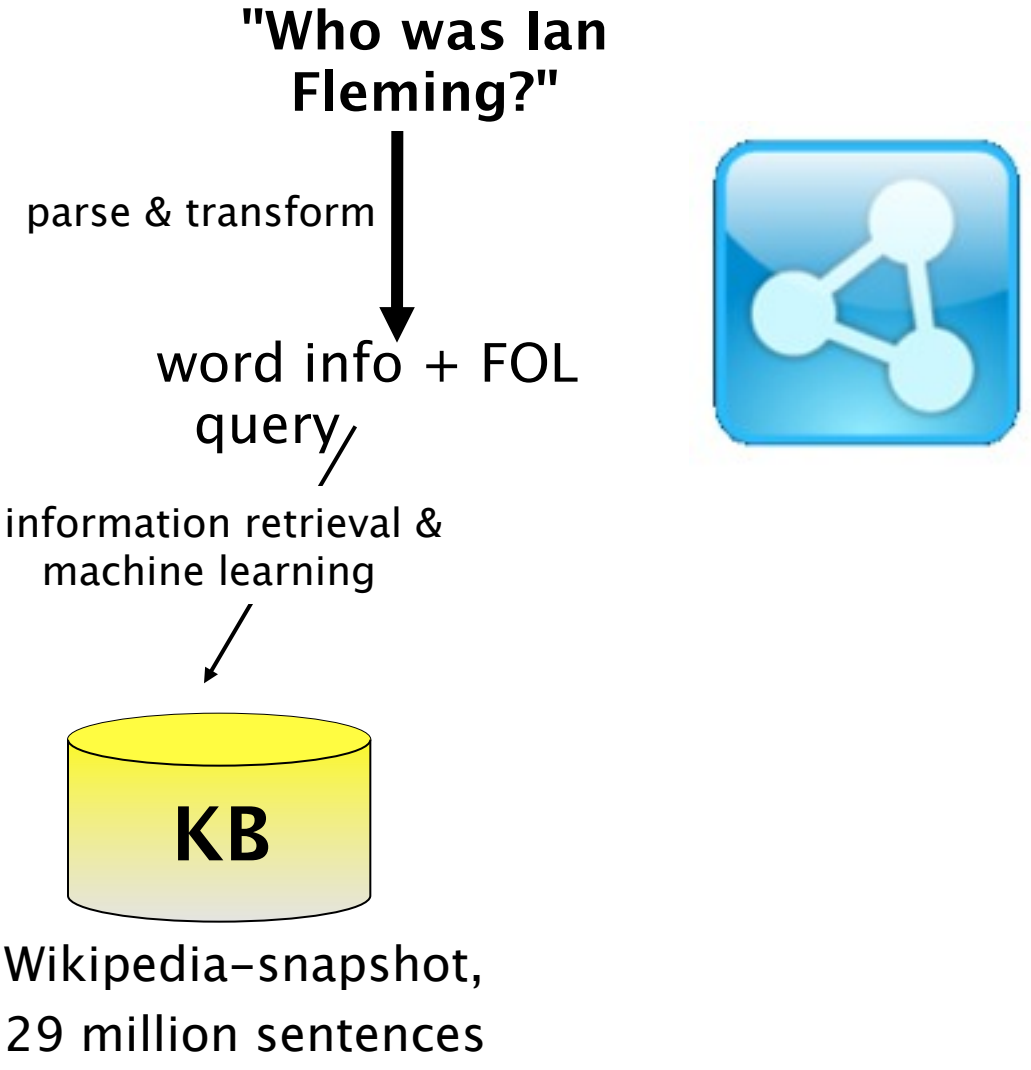
The LogAnswer question answering system:

"Who was Ian Fleming?"

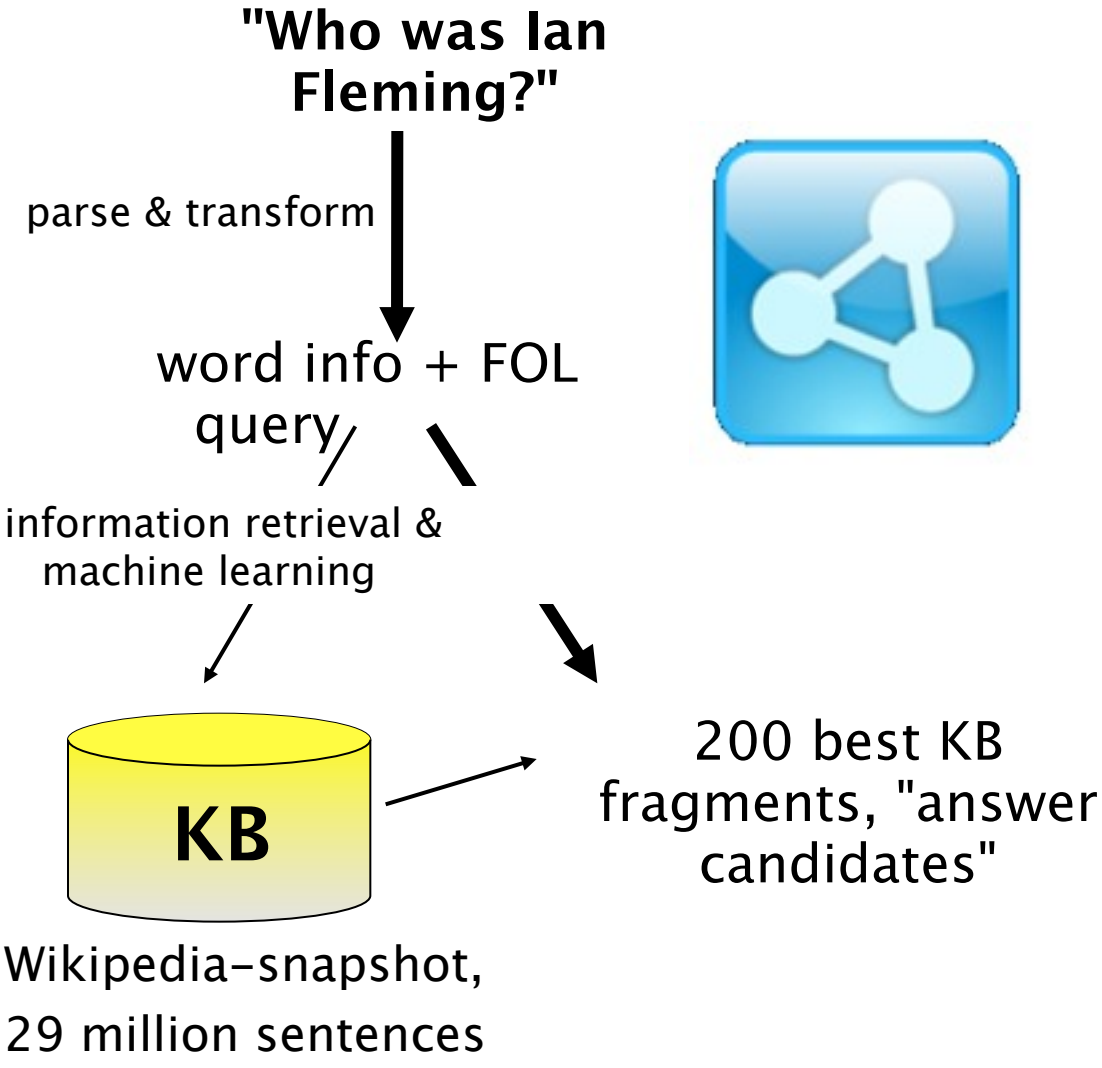
parse & transform

word info + FOL query

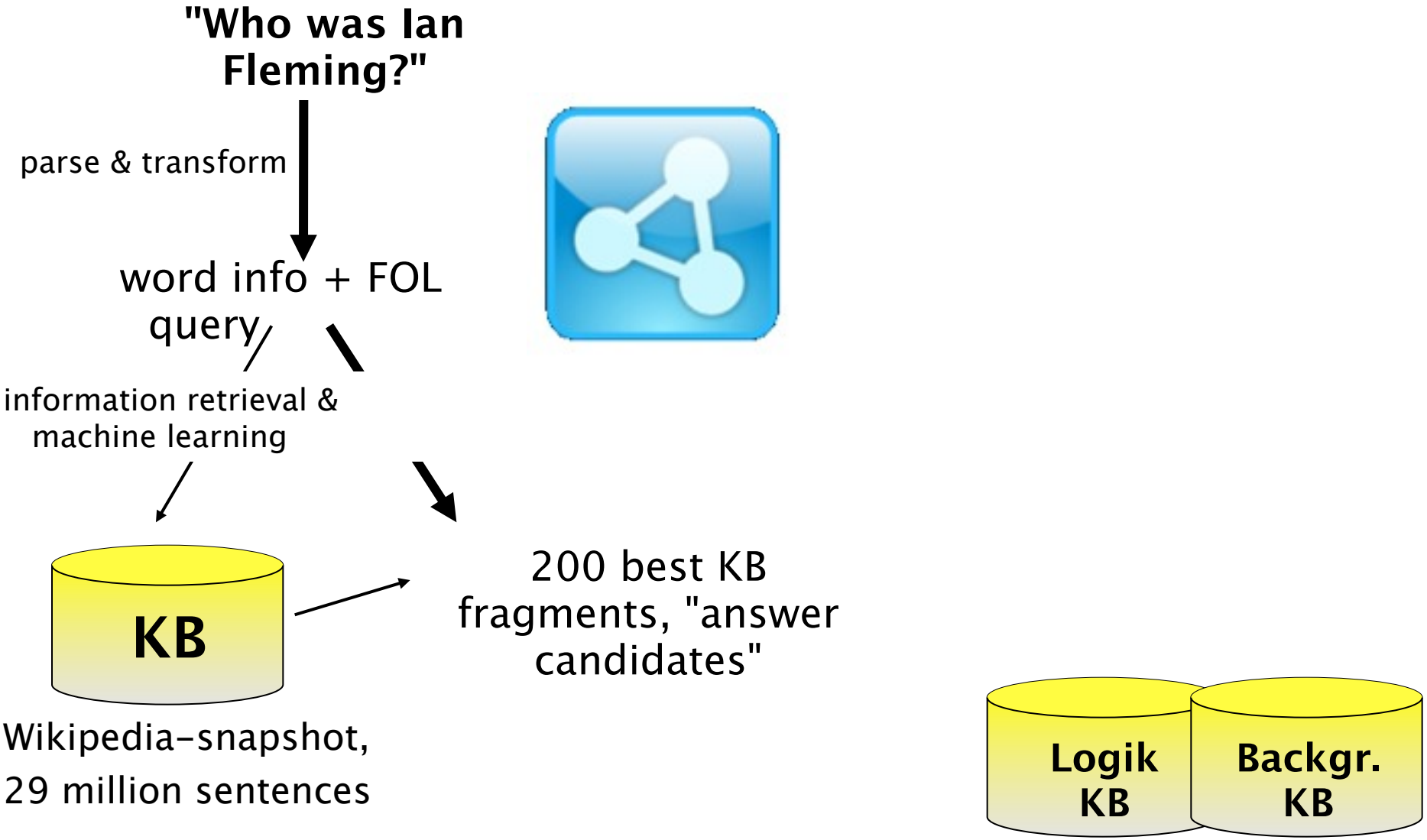
The LogAnswer question answering system:



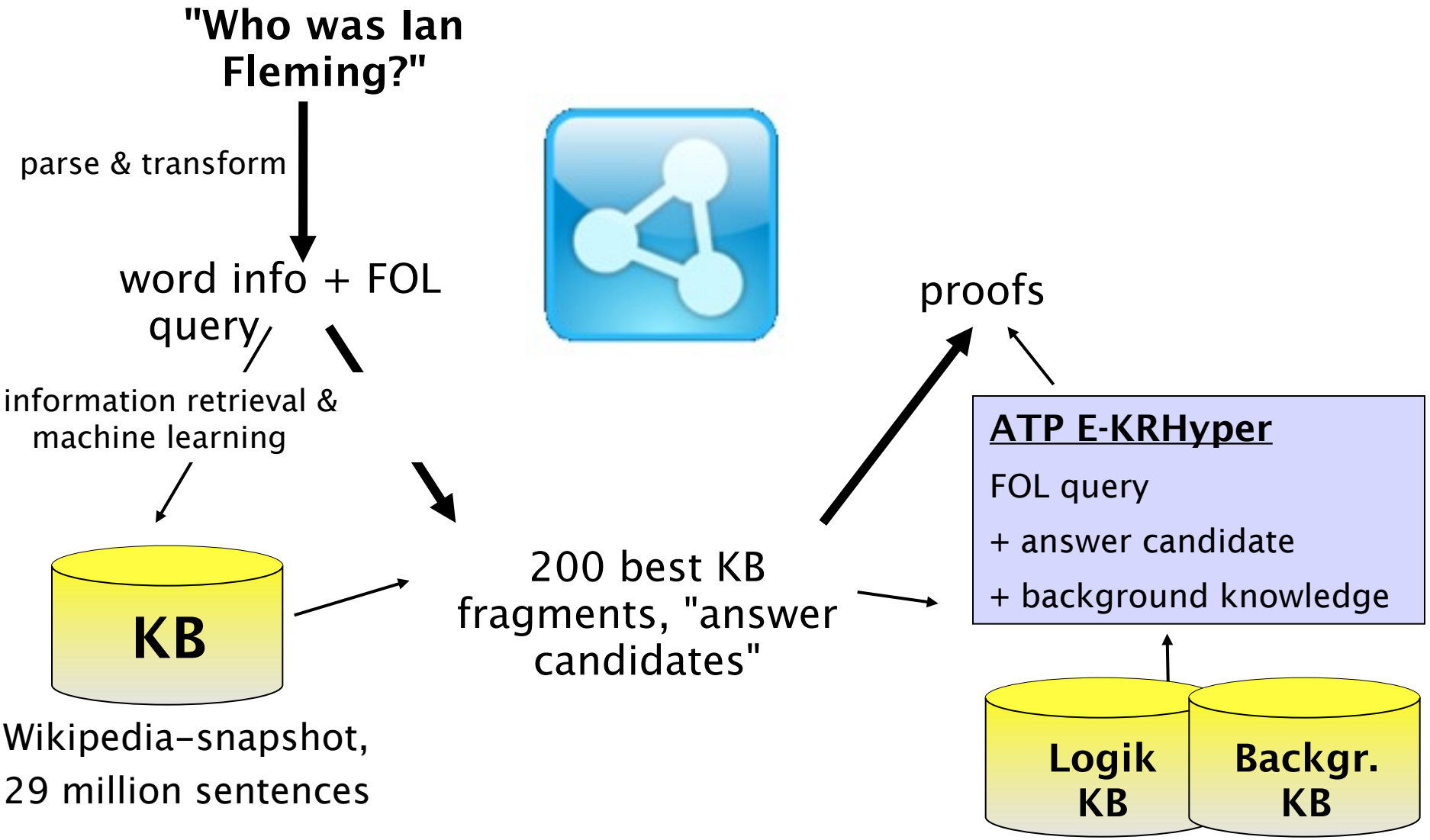
The LogAnswer question answering system:



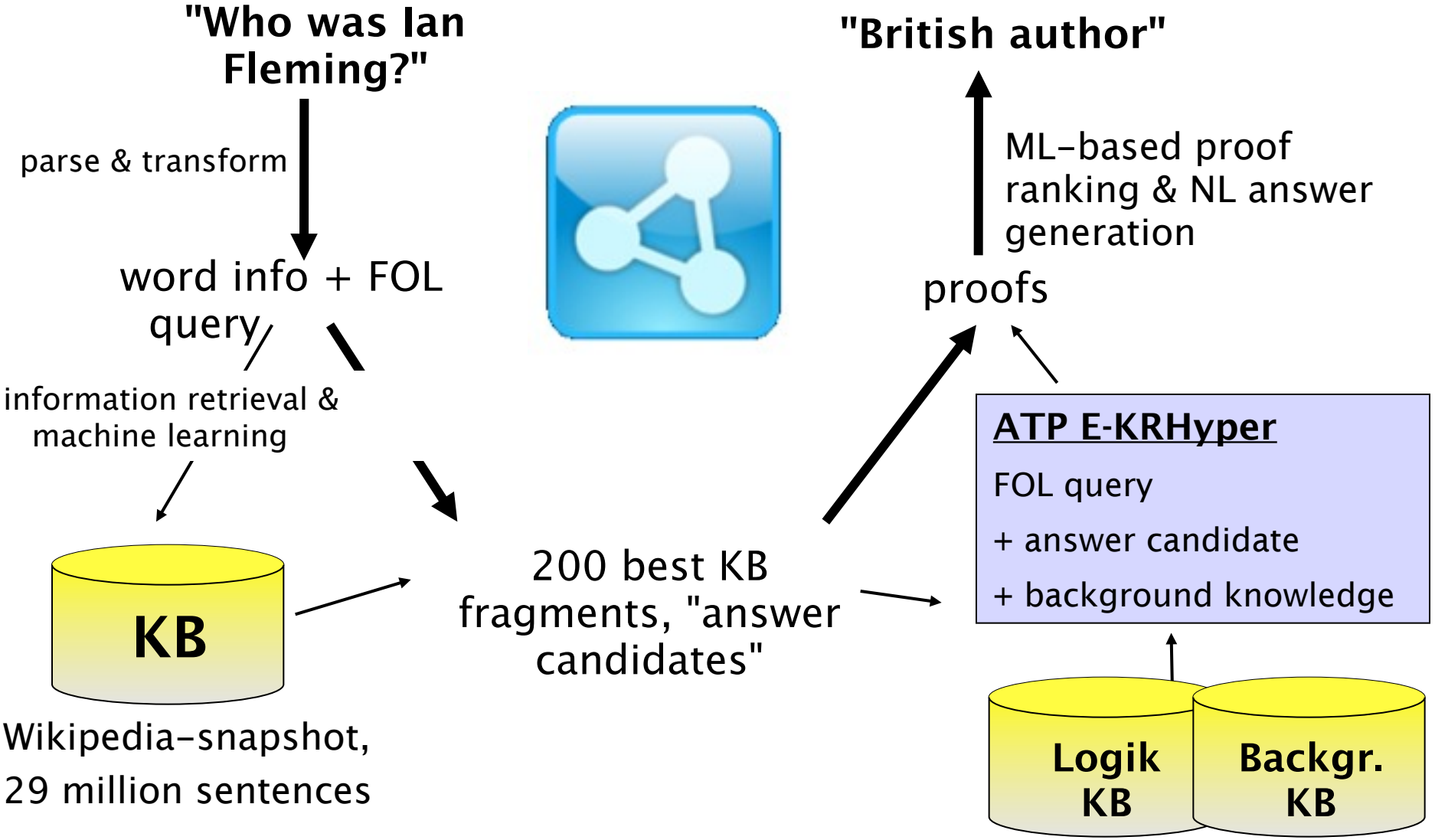
The LogAnswer question answering system:



The LogAnswer question answering system:



The LogAnswer question answering system:



Retrieval of Answer Candidates

Pre-analysed and indexed text passages allow quick computation of syntactic filtering criteria:

Retrieval of Answer Candidates

Pre-analysed and indexed text passages allow quick computation of syntactic filtering criteria:

- *matchRatio*: relative proportion of lexical concepts and numerals in the question which find a match in the text passage,
- *failedMatch*: number of lexical concepts and numerals in the question which find no match in the text passage,
- *failedNames*: proper names which find no match in the text passage,
- *containsBrackets*: indicates whether passage contains parentheses. *"...Sydney (Australia) ..."*

Retrieval of Answer Candidates

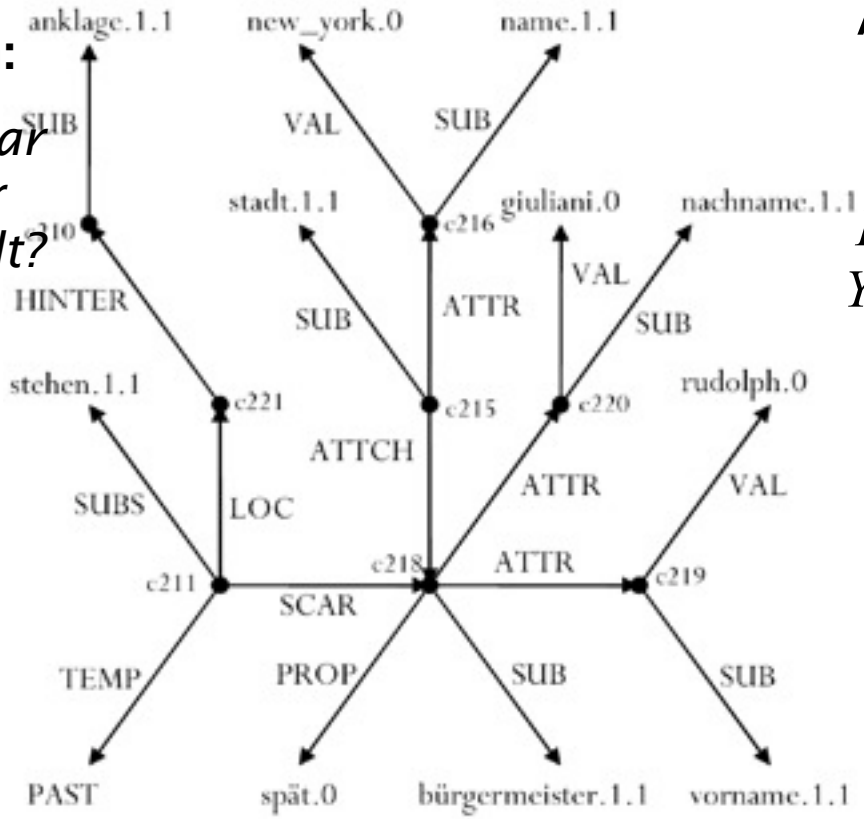
Pre-analysed and indexed text passages allow quick computation of syntactic filtering criteria:

- *matchRatio*: relative proportion of lexical concepts and numerals in the question which find a match in the text passage,
- *failedMatch*: number of lexical concepts and numerals in the question which find no match in the text passage,
- *failedNames*: proper names which find no match in the text passage,
- *containsBrackets*: indicates whether passage contains parentheses. *"...Sydney (Australia) ..."*

Scores are aggregated into a quality estimate using decision trees. The 'best' 200 logical text passage representations are answer candidates and will be evaluated deductively.

User question:

Rudy Giuliani war Bürgermeister welcher US-Stadt?



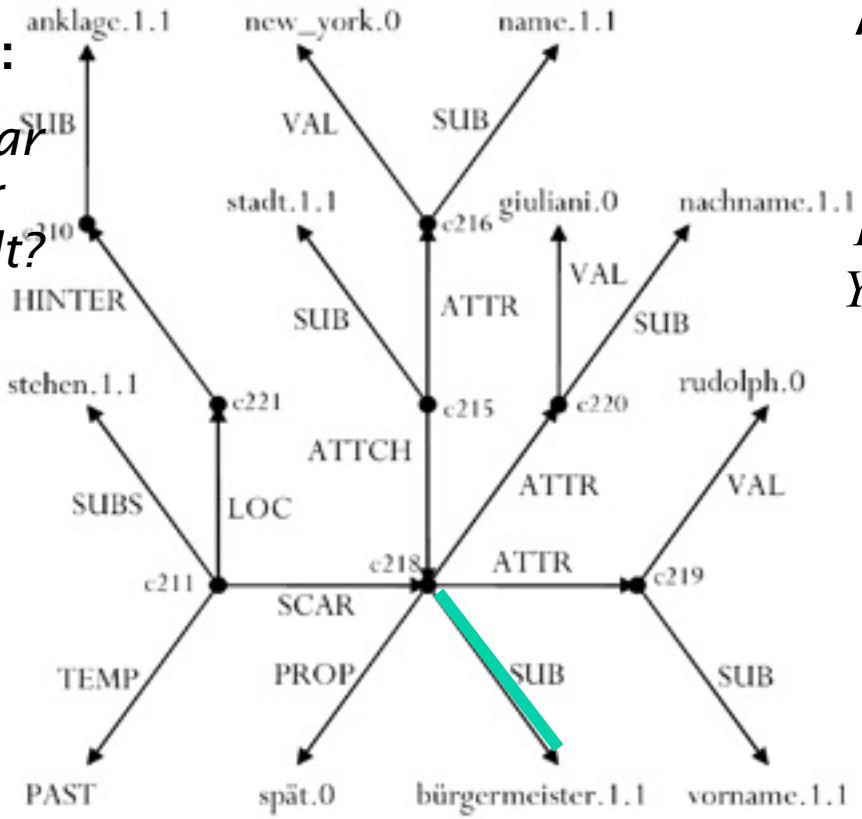
Answer candidate:

"Hinter der Anklage stand der spätere Bürgermeister von New York, Rudolph Giuliani."

¬attch(FOCUS, X1) ∨ ¬sub(FOCUS, us_stadt.1.1)
∨ ¬attr(X1,X2) ∨ ¬val(X2, rudy.0) ∨ ¬sub(X2, vorname.1.1)
∨ ¬attr(X1,X3) ∨ ¬val(X3, giuliani.0) ∨ ¬sub(X3, nachname.1.1)
∨ ¬sub(X1, bürgermeister.1.1)

User question:

Rudy Giuliani war Bürgermeister welcher US-Stadt?



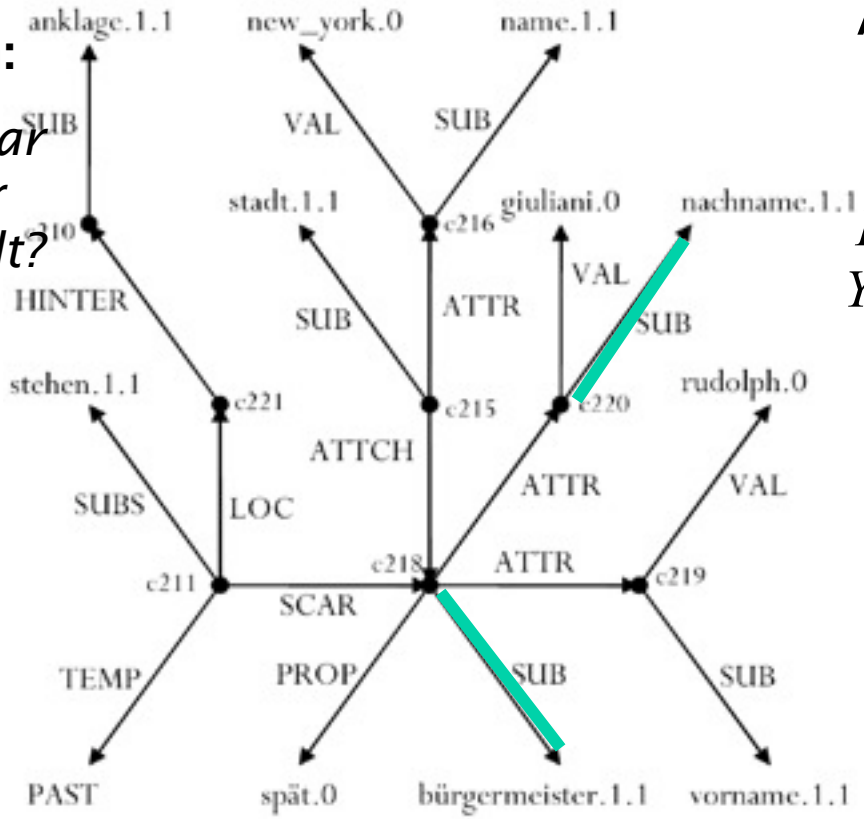
Answer candidate:

"Hinter der Anklage stand der spätere Bürgermeister von New York, Rudolph Giuliani."

¬attch(FOCUS, X1) ∨ ¬sub(FOCUS, us_stadt.1.1)
∨ ¬attr(X1,X2) ∨ ¬val(X2, rudy.0) ∨ ¬sub(X2, vorname.1.1)
∨ ¬attr(X1,X3) ∨ ¬val(X3, giuliani.0) ∨ ¬sub(X3, nachname.1.1)
∨ ¬sub(X1, bürgermeister.1.1)

User question:

Rudy Giuliani war Bürgermeister welcher US-Stadt?



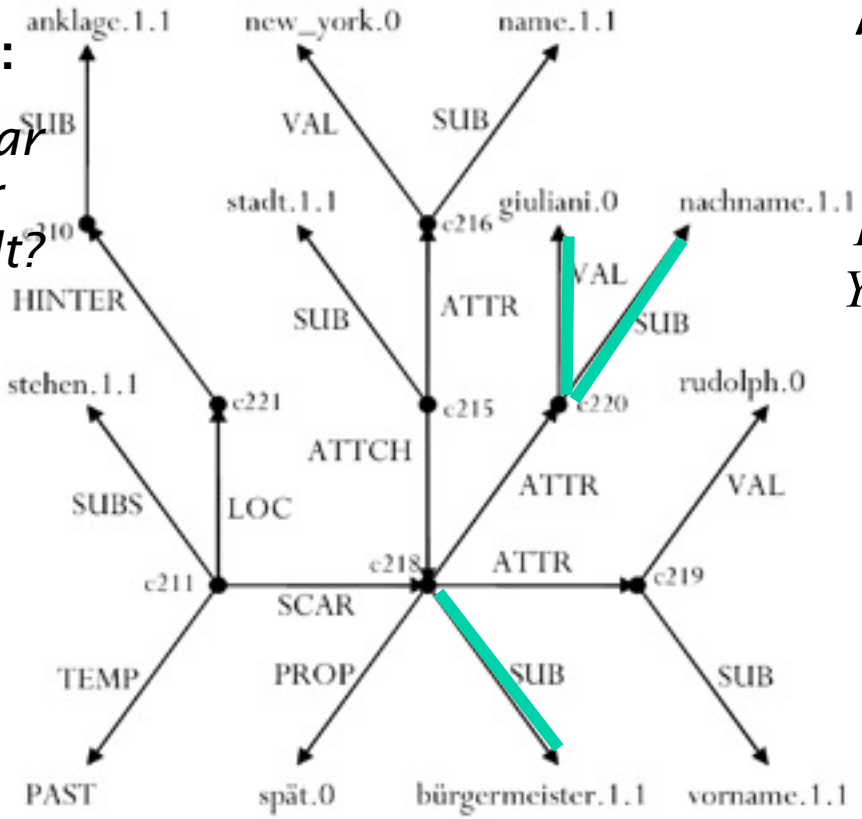
Answer candidate:

"Hinter der Anklage stand der spätere Bürgermeister von New York, Rudolph Giuliani."

¬attch(FOCUS, X1) ∨ ¬sub(FOCUS, us_stadt.1.1)
∨ ¬attr(X1,X2) ∨ ¬val(X2, rudy.0) ∨ ¬sub(X2, vorname.1.1)
∨ ¬attr(X1,X3) ∨ ¬val(X3, giuliani.0) ∨ ¬sub(X3, nachname.1.1)
∨ ¬sub(X1, bürgermeister.1.1)

User question:

Rudy Giuliani war Bürgermeister welcher US-Stadt?



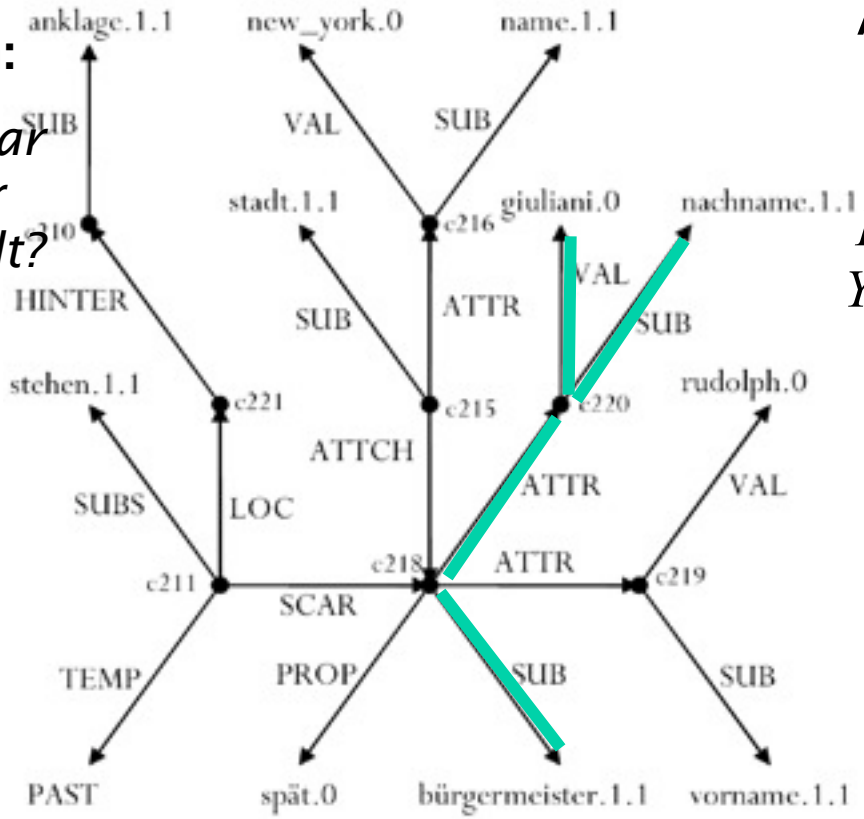
Answer candidate:

"Hinter der Anklage stand der spätere Bürgermeister von New York, Rudolph Giuliani."

¬attch(FOCUS, X1) ∨ ¬sub(FOCUS, us_stadt.1.1)
∨ ¬attr(X1,X2) ∨ ¬val(X2, rudy.0) ∨ ¬sub(X2, vorname.1.1)
∨ ¬attr(X1,X3) ∨ ~~¬val(X3, giuliani.0)~~ ∨ ~~¬sub(X3, nachname.1.1)~~
∨ ~~¬sub(X1, bürgermeister.1.1)~~

User question:

Rudy Giuliani war Bürgermeister welcher US-Stadt?



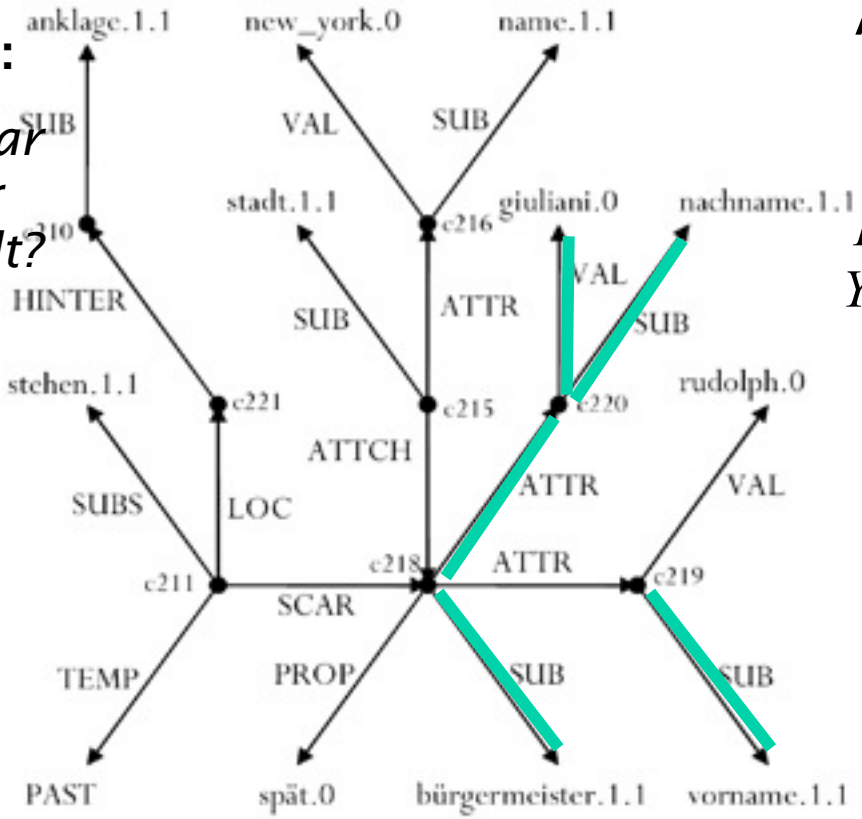
Answer candidate:

"Hinter der Anklage stand der spätere Bürgermeister von New York, Rudolph Giuliani."

¬attch(FOCUS, X1) ∨ ¬sub(FOCUS, us_stadt.1.1)
∨ ¬attr(X1,X2) ∨ ¬val(X2, rudy.0) ∨ ¬sub(X2, vorname.1.1)
∨ ¬attr(X1,X3) ∨ ¬val(X3, giuliani.0) ∨ ¬sub(X3, nachname.1.1)
∨ ¬sub(X1, bürgermeister.1.1)

User question:

Rudy Giuliani war Bürgermeister welcher US-Stadt?



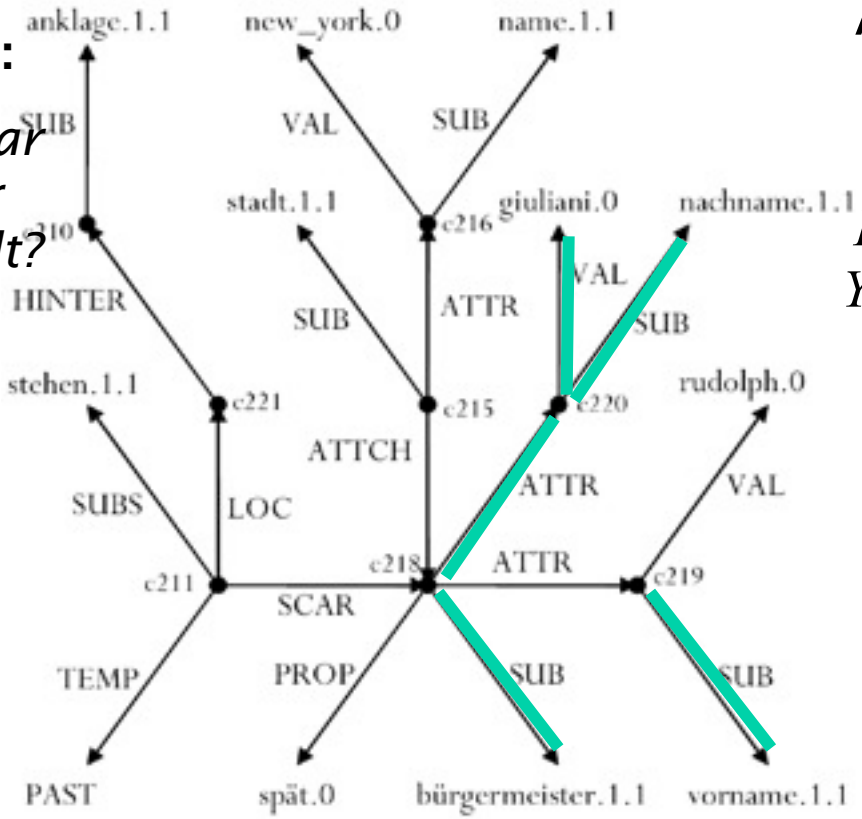
Answer candidate:

"Hinter der Anklage stand der spätere Bürgermeister von New York, Rudolph Giuliani."

¬attch(FOCUS, X1) ∨ ¬sub(FOCUS, us_stadt.1.1)
∨ ¬attr(X1,X2) ∨ ¬val(X2, rudy.0) ∨ ~~¬sub(X2, vorname.1.1)~~
∨ ~~¬attr(X1,X3)~~ ∨ ~~¬val(X3, giuliani.0)~~ ∨ ~~¬sub(X3, nachname.1.1)~~
∨ ~~¬sub(X1, bürgermeister.1.1)~~

User question:

Rudy Giuliani war Bürgermeister welcher US-Stadt?



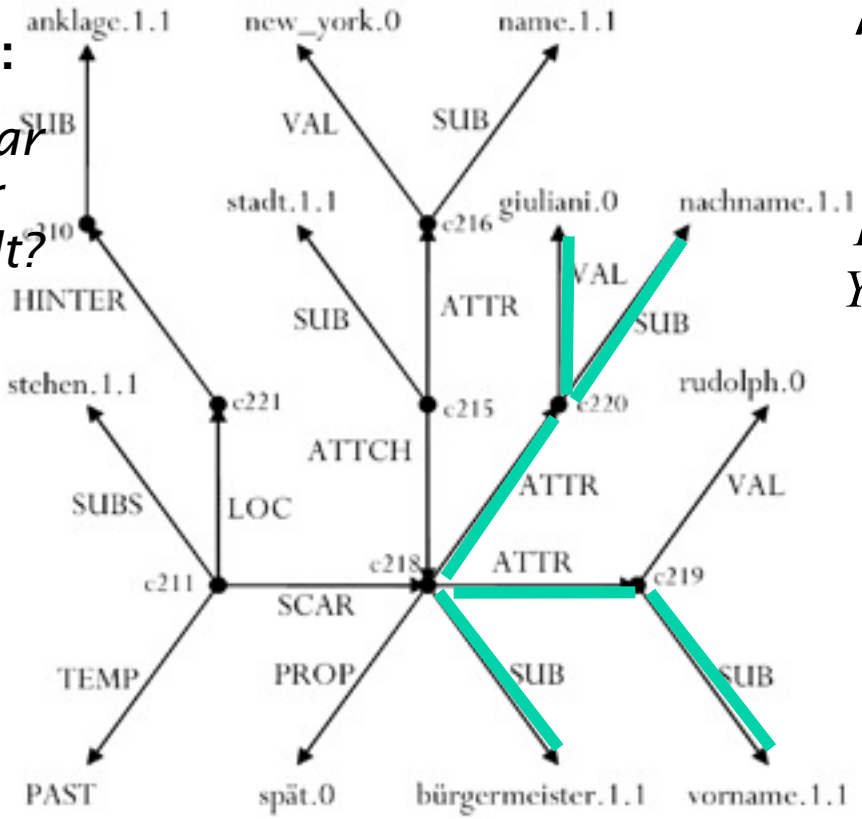
Answer candidate:

"Hinter der Anklage stand der spätere Bürgermeister von New York, Rudolph Giuliani."

¬attch(FOCUS, X1) ∨ ¬sub(FOCUS, us_stadt.1.1)
∨ ¬attr(X1,X2) ∨ ~~¬val(X2, rudy.0)~~ ∨ ~~¬sub(X2, vorname.1.1)~~
∨ ~~¬attr(X1,X3)~~ ∨ ~~¬val(X3, giuliani.0)~~ ∨ ~~¬sub(X3, nachname.1.1)~~
∨ ~~¬sub(X1, bürgermeister.1.1)~~

User question:

Rudy Giuliani war Bürgermeister welcher US-Stadt?



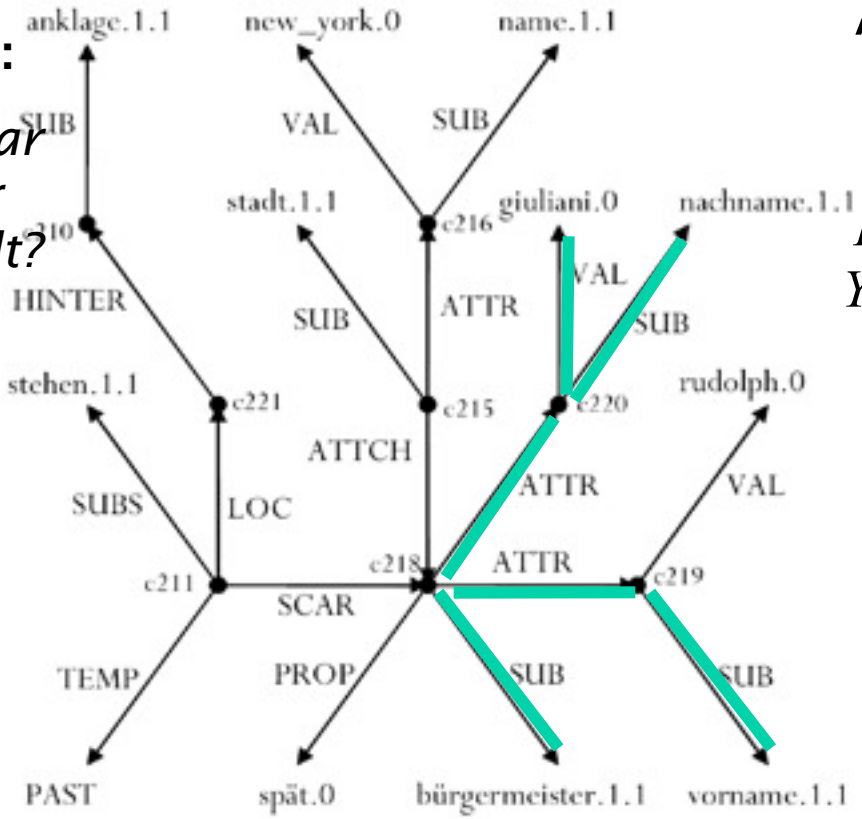
Answer candidate:

"Hinter der Anklage stand der spätere Bürgermeister von New York, Rudolph Giuliani."

¬attch(FOCUS, X1) ∨ ¬sub(FOCUS, us_stadt.1.1)
∨ ¬attr(X1,X2) ∨ ¬val(X2, rudy.0) ∨ ¬sub(X2, vorname.1.1)
∨ ¬attr(X1,X3) ∨ ¬val(X3, giuliani.0) ∨ ¬sub(X3, nachname.1.1)
∨ ¬sub(X1, bürgermeister.1.1)

User question:

Rudy Giuliani war Bürgermeister welcher US-Stadt?



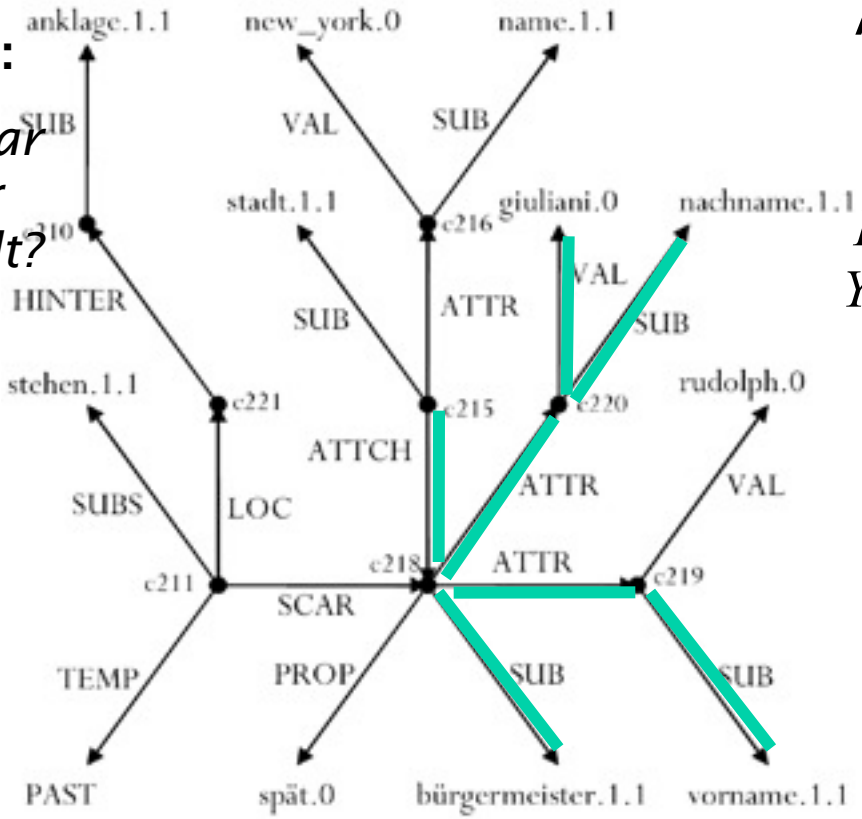
Answer candidate:

"Hinter der Anklage stand der spätere Bürgermeister von New York, Rudolph Giuliani."

¬attch(FOCUS, X1) v ¬sub(FOCUS, us_stadt.1.1)
v ¬attr(X1,X2) v ¬val(X2, rudy.0) v ¬sub(X2, vorname.1.1)
v ¬attr(X1,X3) v ¬val(X3, giuliani.0) v ¬sub(X3, nachname.1.1)
v ¬sub(X1, bürgermeister.1.1)

User question:

Rudy Giuliani war Bürgermeister welcher US-Stadt?



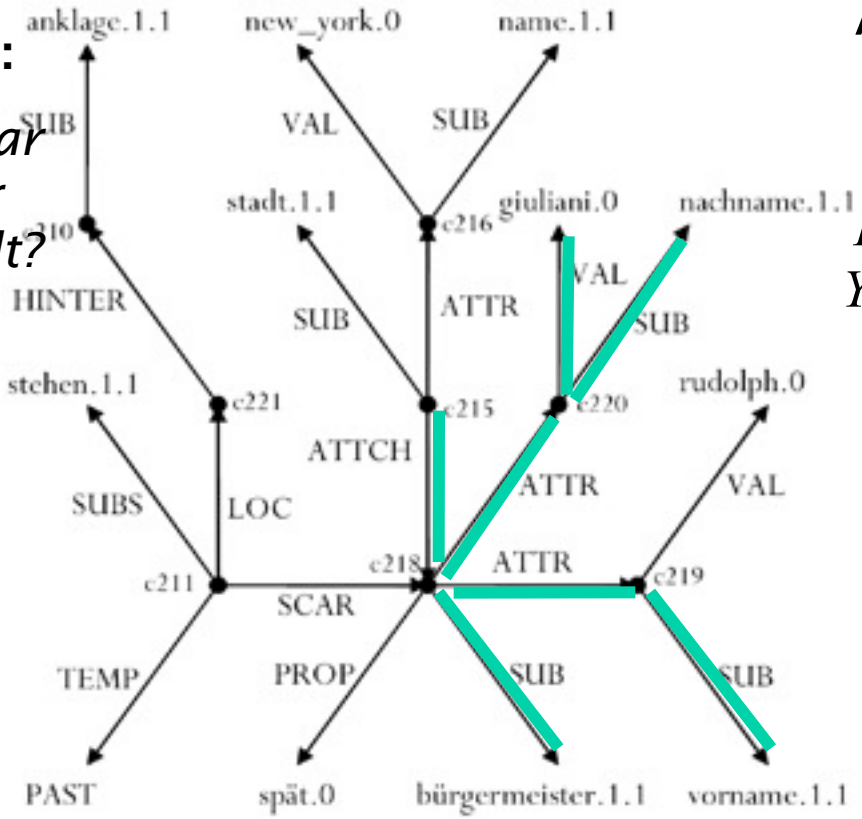
Answer candidate:

"Hinter der Anklage stand der spätere Bürgermeister von New York, Rudolph Giuliani."

¬attch(FOCUS, X1) v ¬sub(FOCUS, us_stadt.1.1)
v ¬attr(X1,X2) v ¬val(X2, rudy.0) v ¬sub(X2, vorname.1.1)
v ¬attr(X1,X3) v ¬val(X3, giuliani.0) v ¬sub(X3, nachname.1.1)
v ¬sub(X1, bürgermeister.1.1)

User question:

Rudy Giuliani war Bürgermeister welcher US-Stadt?



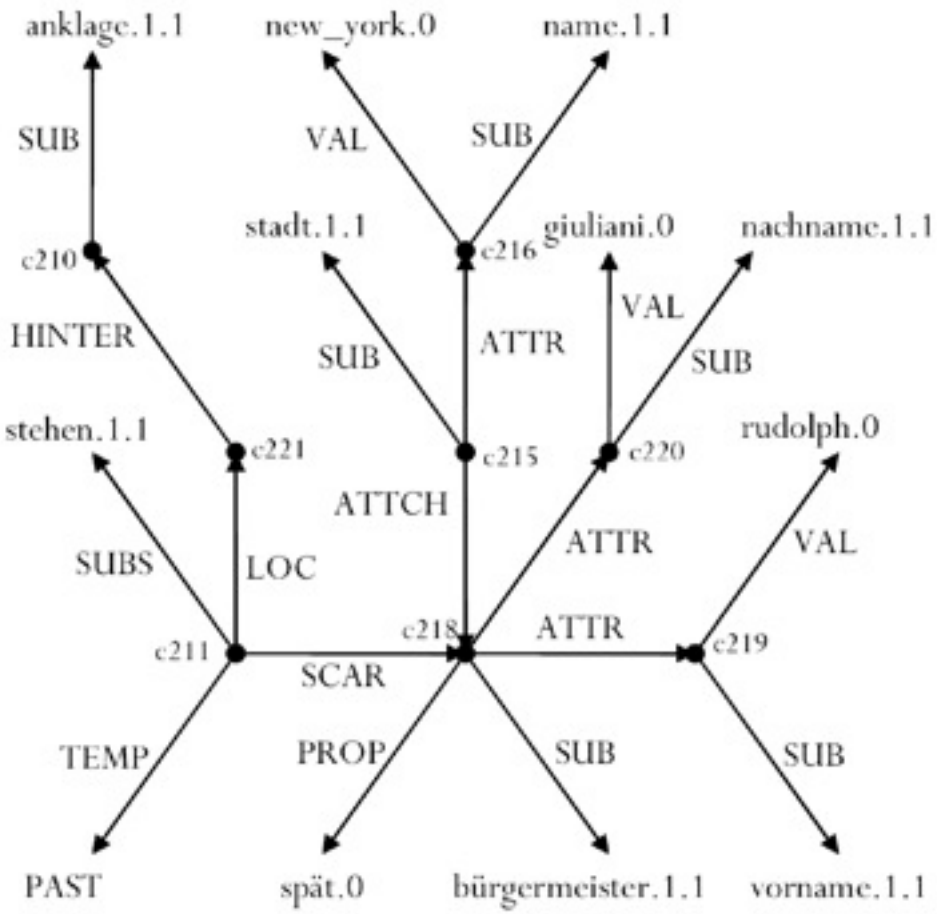
Answer candidate:

"Hinter der Anklage stand der spätere Bürgermeister von New York, Rudolph Giuliani."

¬attch(FOCUS, X1) v ¬sub(FOCUS, us_stadt.1.1)
v ¬attr(X1,X2) v ¬val(X2, rudy.0) v ¬sub(X2, vorname.1.1)
v ¬attr(X1,X3) v ¬val(X3, giuliani.0) v ¬sub(X3, nachname.1.1)
v ¬sub(X1, bürgermeister.1.1)

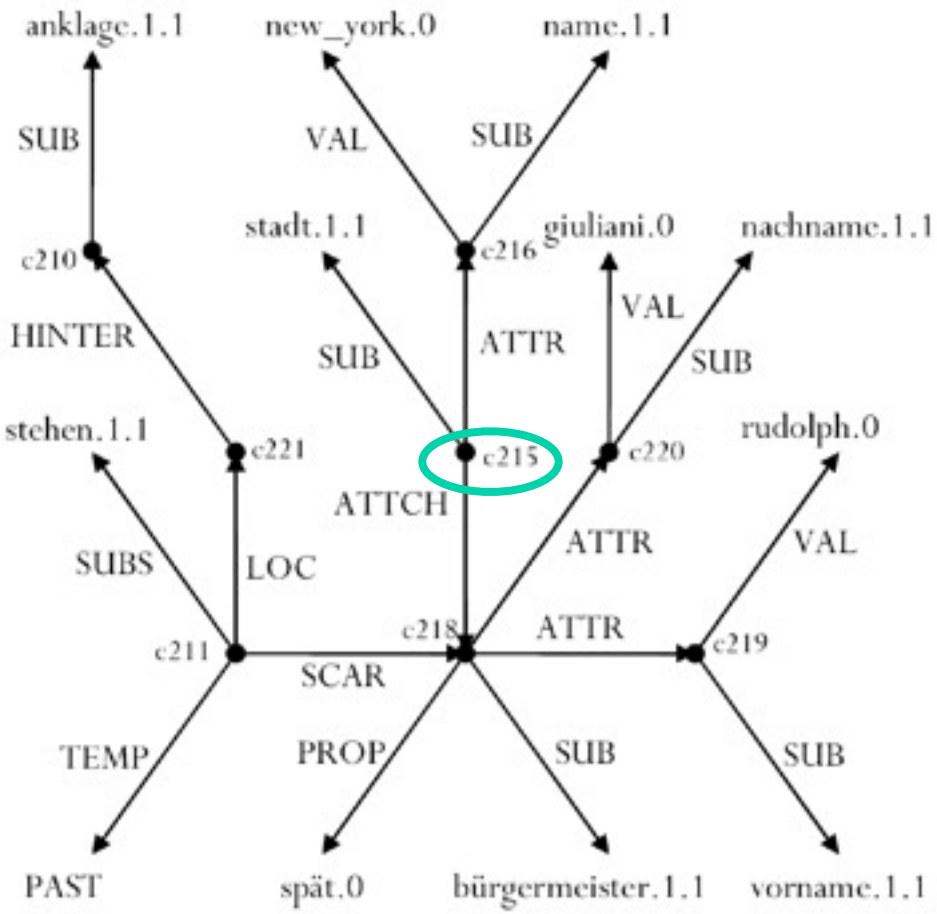
⇒ FOCUS = c215 (2 relaxations)

Answer Generation



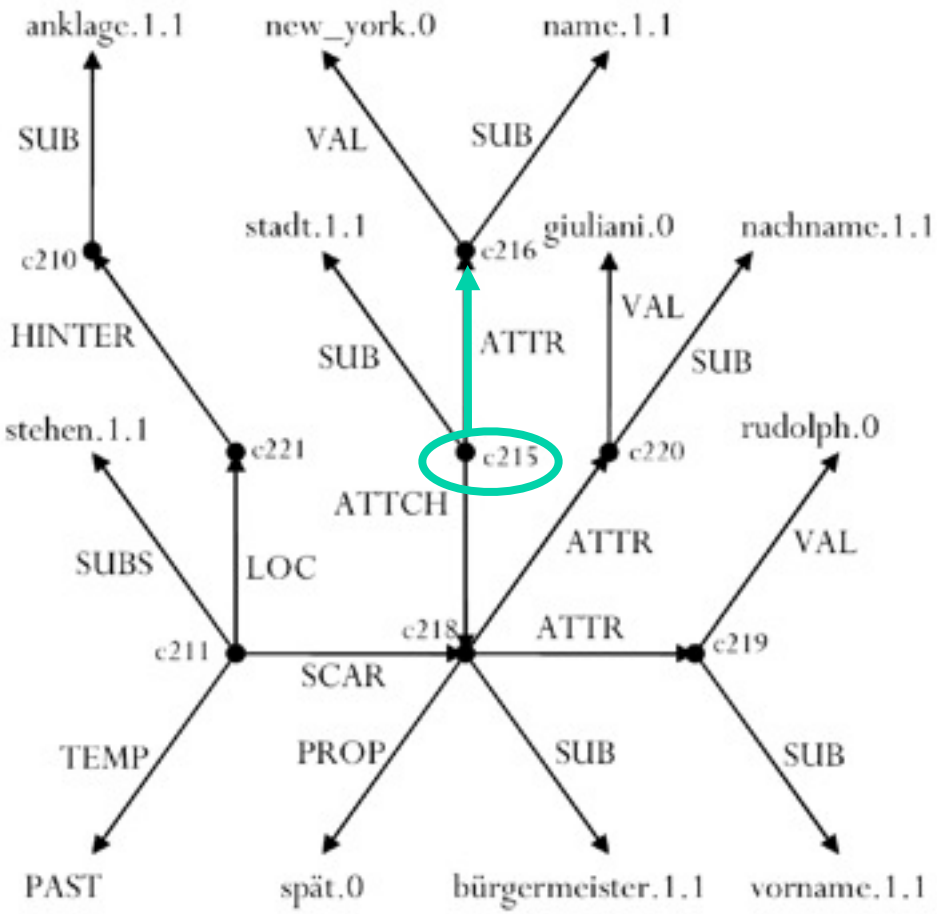
The FOCUS-variable represents the core object of the question.

Answer Generation



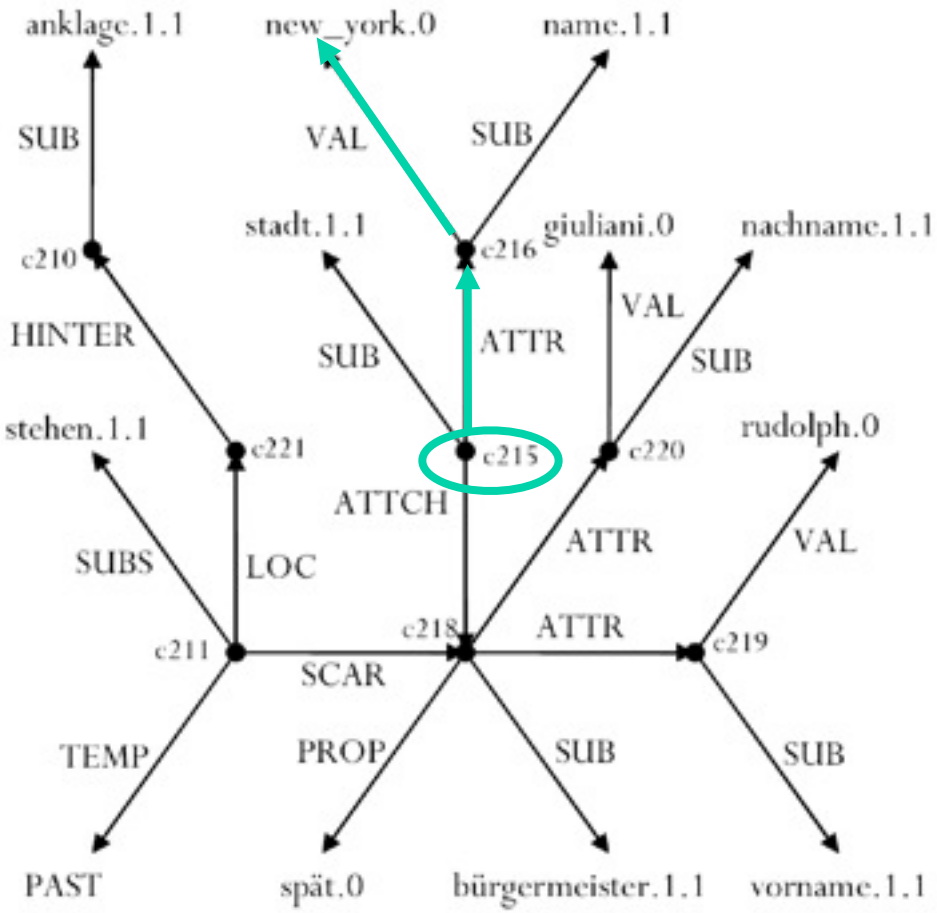
The FOCUS-variable represents the core object of the question.

Answer Generation



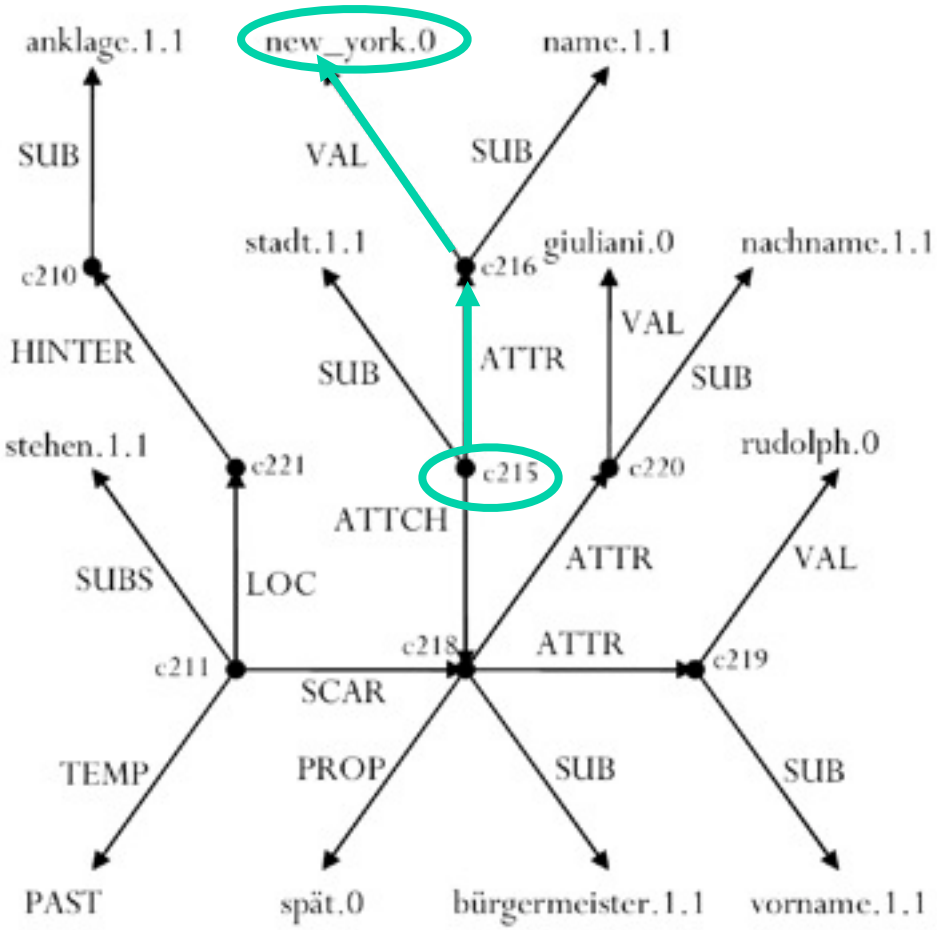
The FOCUS-variable represents the core object of the question.

Answer Generation



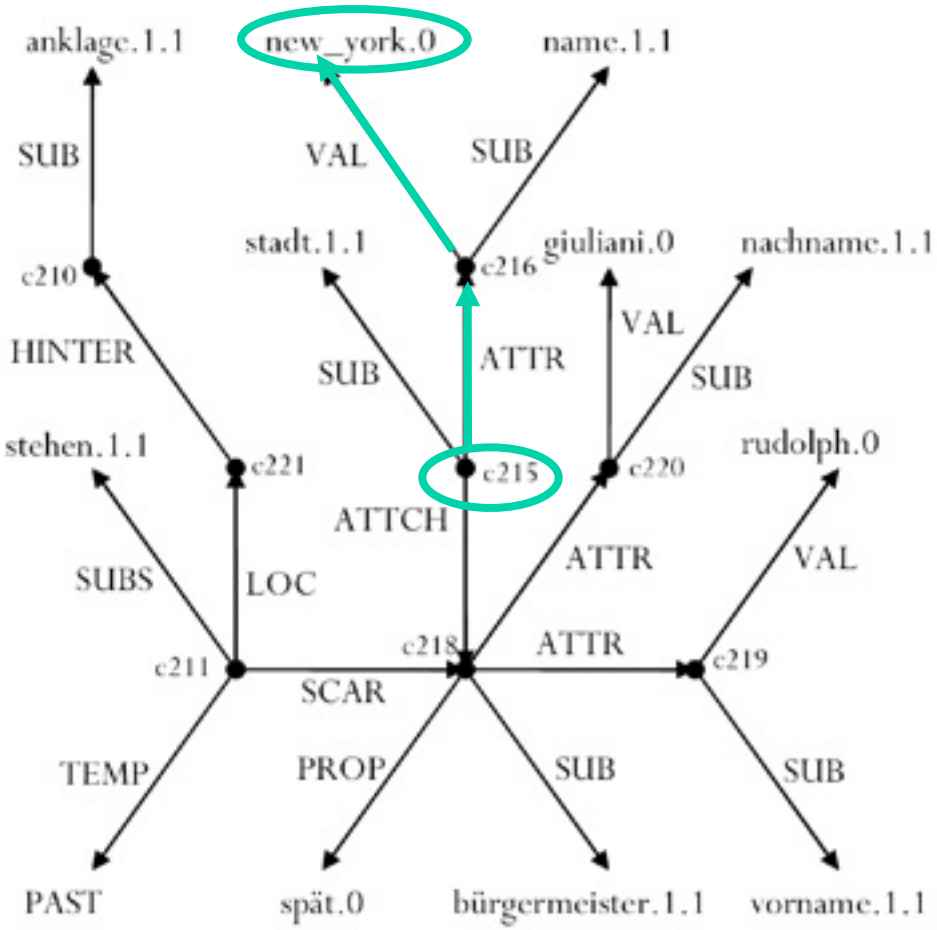
The FOCUS-variable represents the core object of the question.

Answer Generation



The FOCUS-variable represents the core object of the question.

Answer Generation



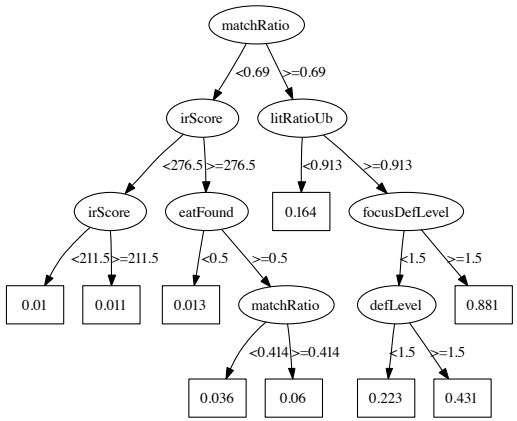
The FOCUS-variable represents the core object of the question.

"New York"

Answer Presentation

Decision tree computes a quality score for each answer, using criteria like:

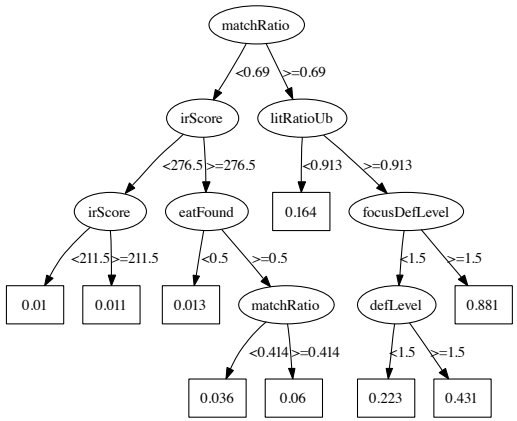
- *skippedLits*: number of query literals skipped by relaxation,
- *npFocus*: FOCUS-variable was bound to a nominal phrase constant,
- *focusEatMatch*: answer type matches expected answer type,
- *irScore*: the original quality estimate for the text passage



Answer Presentation

Decision tree computes a quality score for each answer, using criteria like:

- *skippedLits*: number of query literals skipped by relaxation,
- *npFocus*: FOCUS-variable was bound to a nominal phrase constant,
- *focusEatMatch*: answer type matches expected answer type,
- *irScore*: the original quality estimate for the text passage

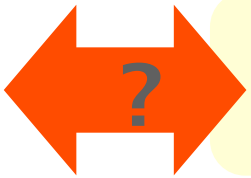


The top 5 answers are presented to the user, together with the text passages to provide context and links to the documents.

Performance Issues

Many prover runs for a single question:

- 200 candidates with up to 5 relaxations each
- >12,000 clauses input for each run



short response time (~5 sec)

Performance Issues

Many prover runs for a single question:

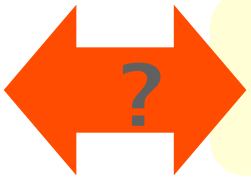
- 2(
  re
- >12,000 clauses input for each run

creating index structures for input can exceed time slot

Performance Issues

Many prover runs for a single question:

- 200 candidates with up to 5 relaxations each
- >12,000 clauses input for each run



short response time (~5 sec)

Plenty of overlap between prover runs:

- all runs use the same background knowledge base (~97% of the clauses in a run)
- all relaxation runs for a candidate use the same candidate

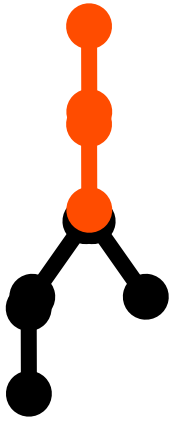
Performance Issues - Incremental Reasoning

Use overlap between relaxation runs for one candidate:

- input sets for runs differ only in one (dropped) query literal
- relaxed query clause subsumes the previous query clause

$$\neg Q_1 \vee \neg Q_2 \vee \neg Q_4 \vee \neg Q_5 \quad \subseteq \neg Q_1 \vee \neg Q_2 \vee \neg Q_3 \vee \neg Q_4 \vee$$

- all clauses derived before first derivation *Split* can be reused in next run



Performance Issues - Incremental Reasoning

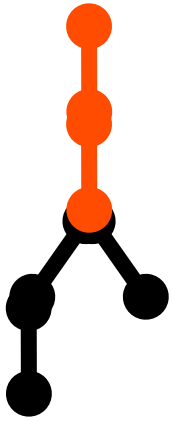
Use overlap between relaxation runs for one candidate:

- input sets for runs differ only in one (dropped) query literal
- relaxed query clause subsumes the previous query clause

$$\neg Q_1 \vee \neg Q_2 \vee \neg Q_4 \vee \neg Q_5 \quad \subseteq \neg Q_1 \vee \neg Q_2 \vee \neg Q_3 \vee \neg Q_4 \vee$$

- all clauses derived before first derivation *Split* can be reused in next run

no rigid variables !



Extensions of Reasoning

- Webservices
- Background knowledge
- Partitioning and Heuristics

Webservices in E-KRHyper

Webservices in E-KRHyper

- **webservice query/answer pair: *kb(q, a)***

  where *q* and *a* may be arbitrarily complex terms

  kb(conv(eur, usd, 299.95), 392.87)

  kb(q, a) is true iff *a* is a webservice-reply to *q*.

- **webservices representation: KB^{ext} of ground unit clauses *kb(q, a)***

- use non-ground kb-literals in clauses to access webservices:

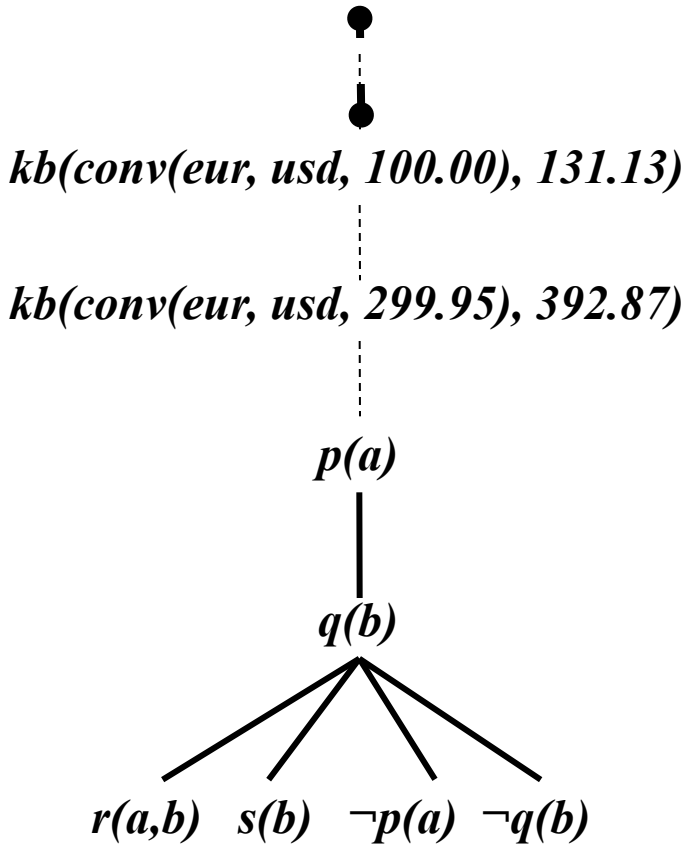
  $dollarprice(x,z) \leftarrow europrice(x,y) \wedge kb(conv(eur, usd, y), z).$

Webservices in E-KRHyper

In theory:

Webservices in E-KRHyper

In theory:



Webservices in E-KRHyper

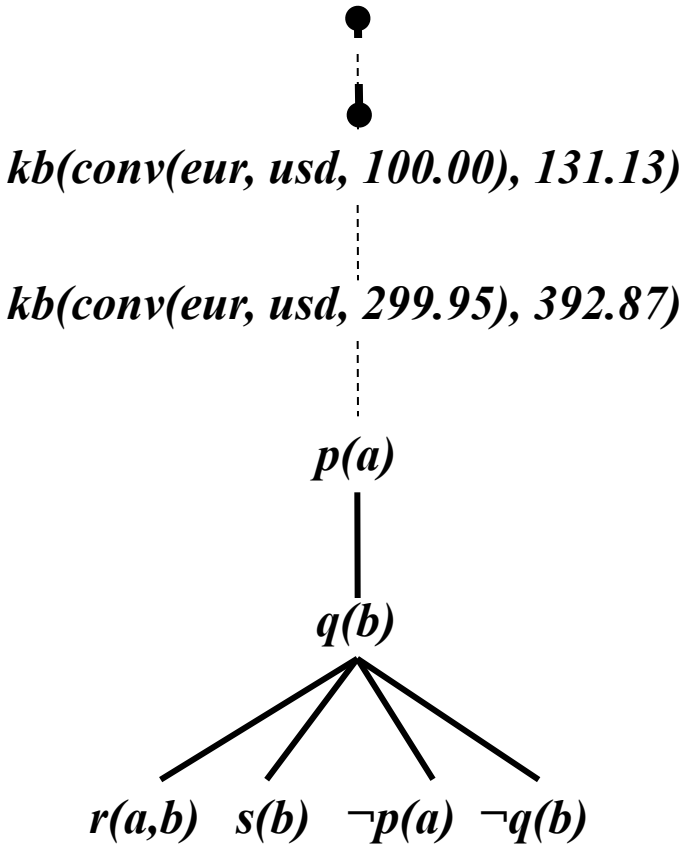
In theory:

Infeasible in practice:

KB^{ext} may be infinite,

retrieving a *kb(q,a)* takes time.

⇒ minimize accesses to KB^{ext}.



Example:

...

dollarprice(x,z) ← europrice(x,y) ∧ kb(conv(eur, usd, y), z).

europrice(coke, 1.20)

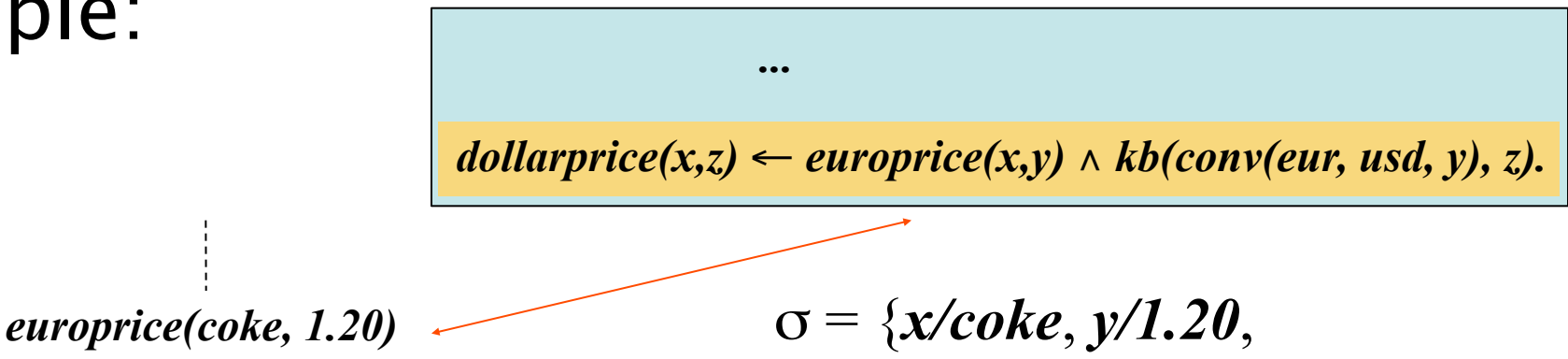
Example:

...

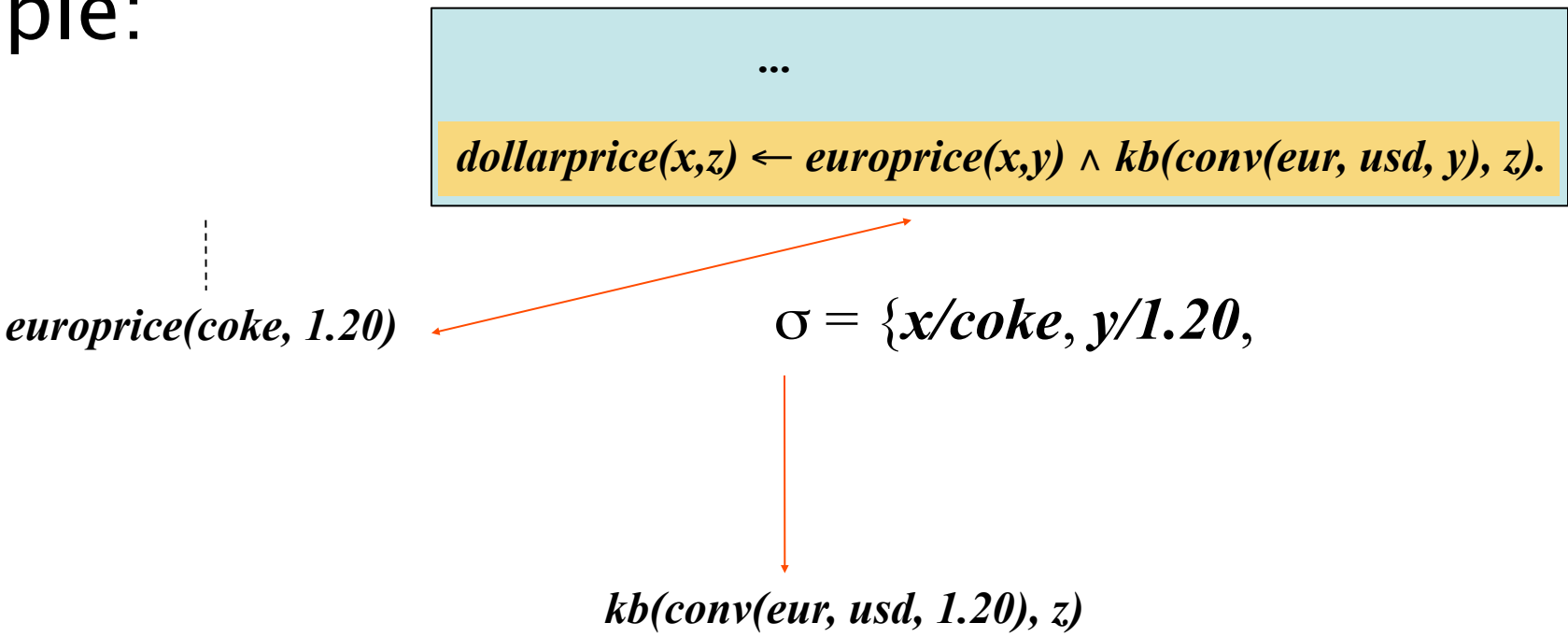
dollarprice(x,z) ← europrice(x,y) ∧ kb(conv(eur, usd, y), z).

europrice(coke, 1.20)

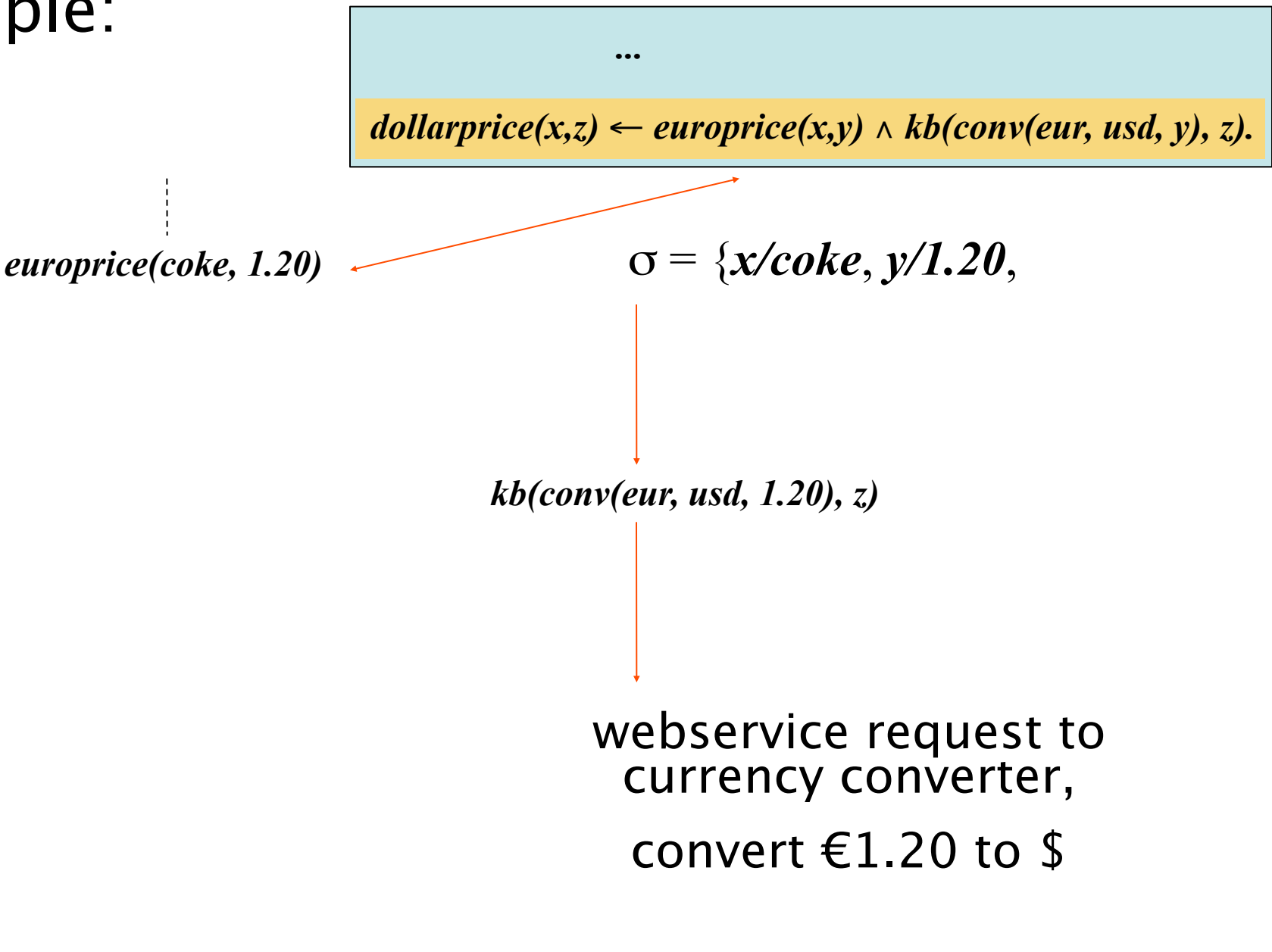
Example:


...
dollarprice(x,z) ← europrice(x,y) ∧ kb(conv(eur, usd, y), z).
europrice(coke, 1.20)
σ = {x/coke, y/1.20,

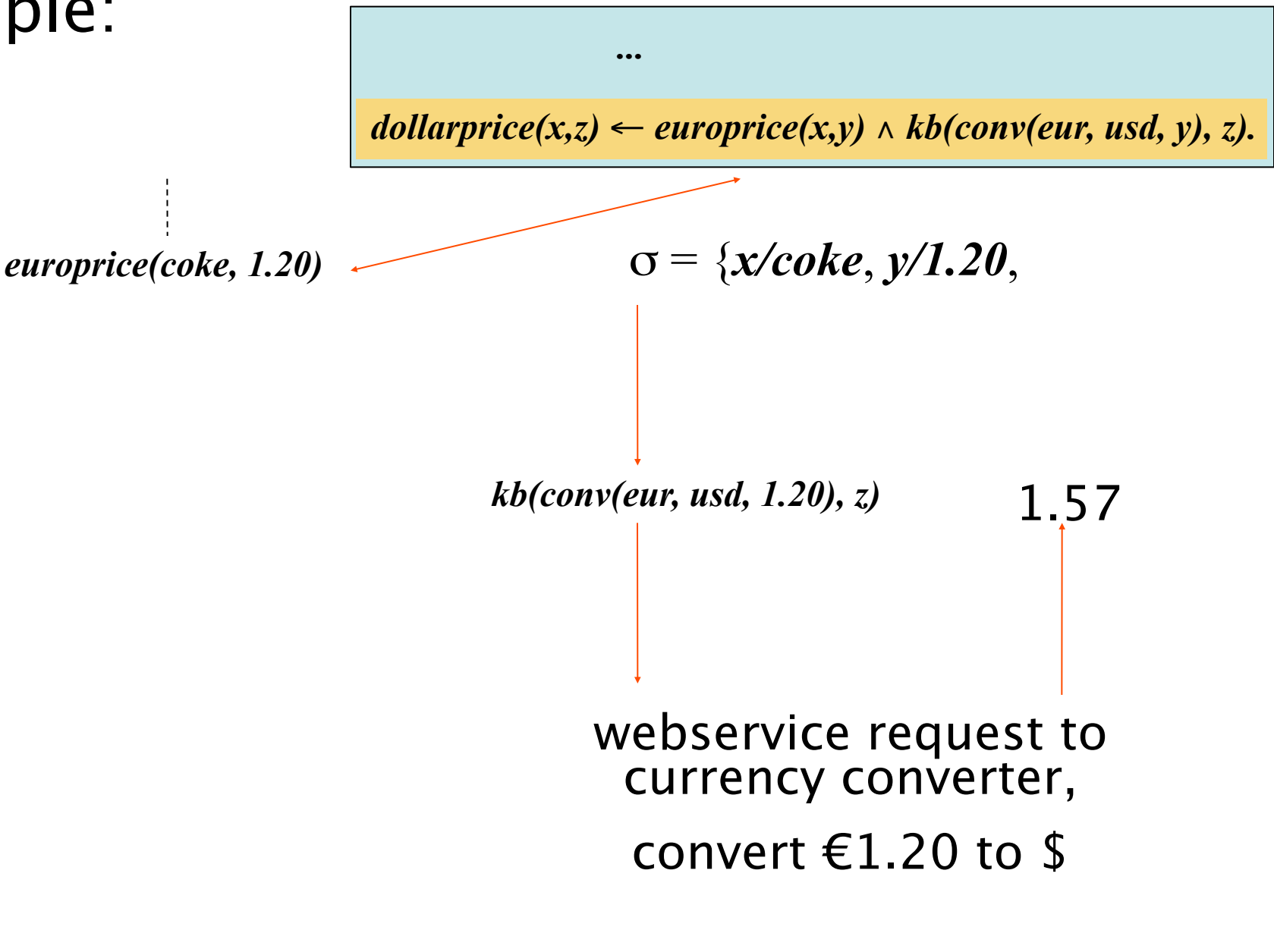
Example:


...
dollarprice(x,z) ← europrice(x,y) ∧ kb(conv(eur, usd, y), z).
europrice(coke, 1.20)
σ = {x/coke, y/1.20,
kb(conv(eur, usd, 1.20), z)

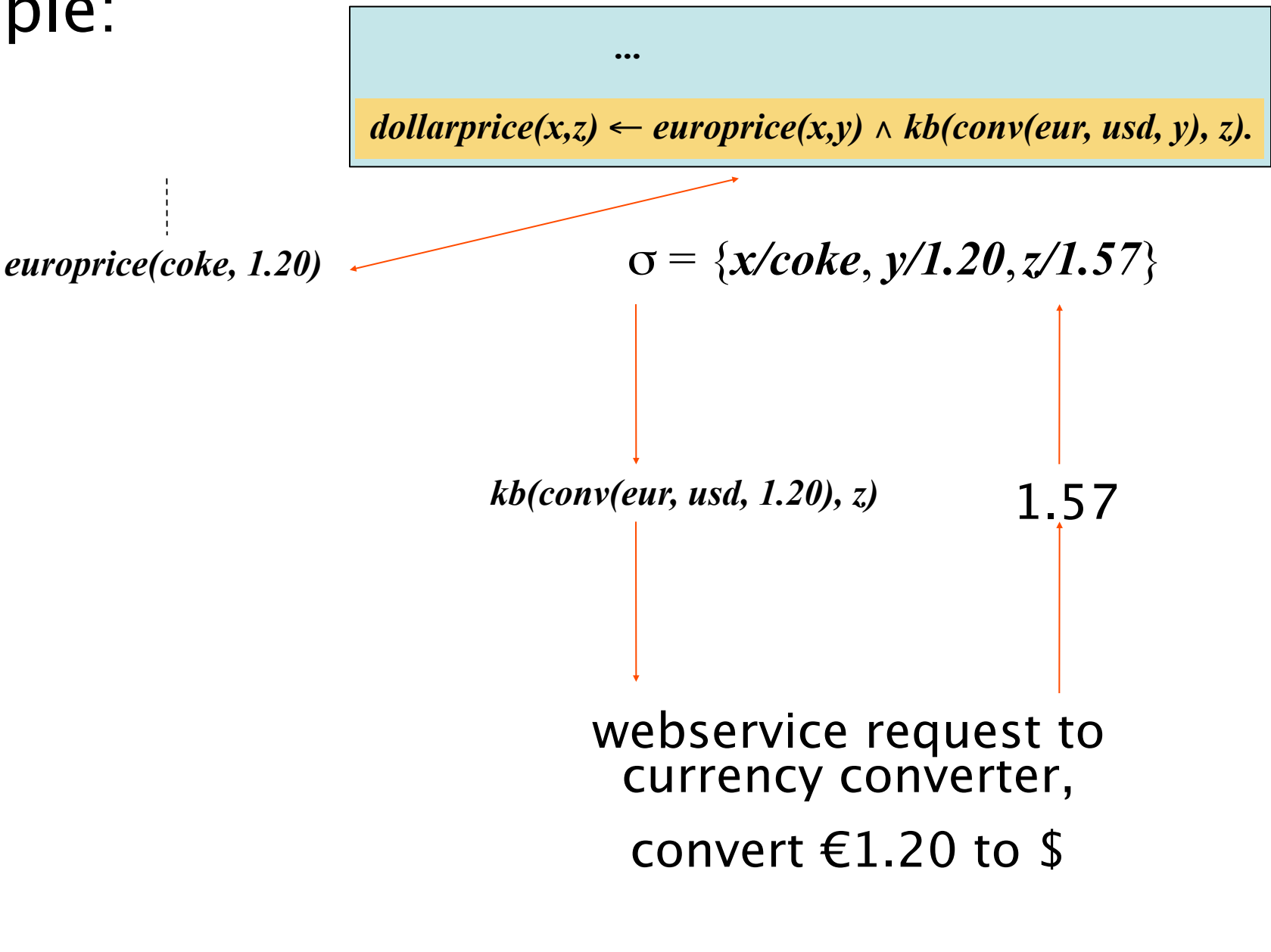
Example:


...
dollarprice(x,z) ← europrice(x,y) ∧ kb(conv(eur, usd, y), z).
europrice(coke, 1.20)
σ = {x/coke, y/1.20,
kb(conv(eur, usd, 1.20), z)
webservice request to currency converter,
convert €1.20 to $

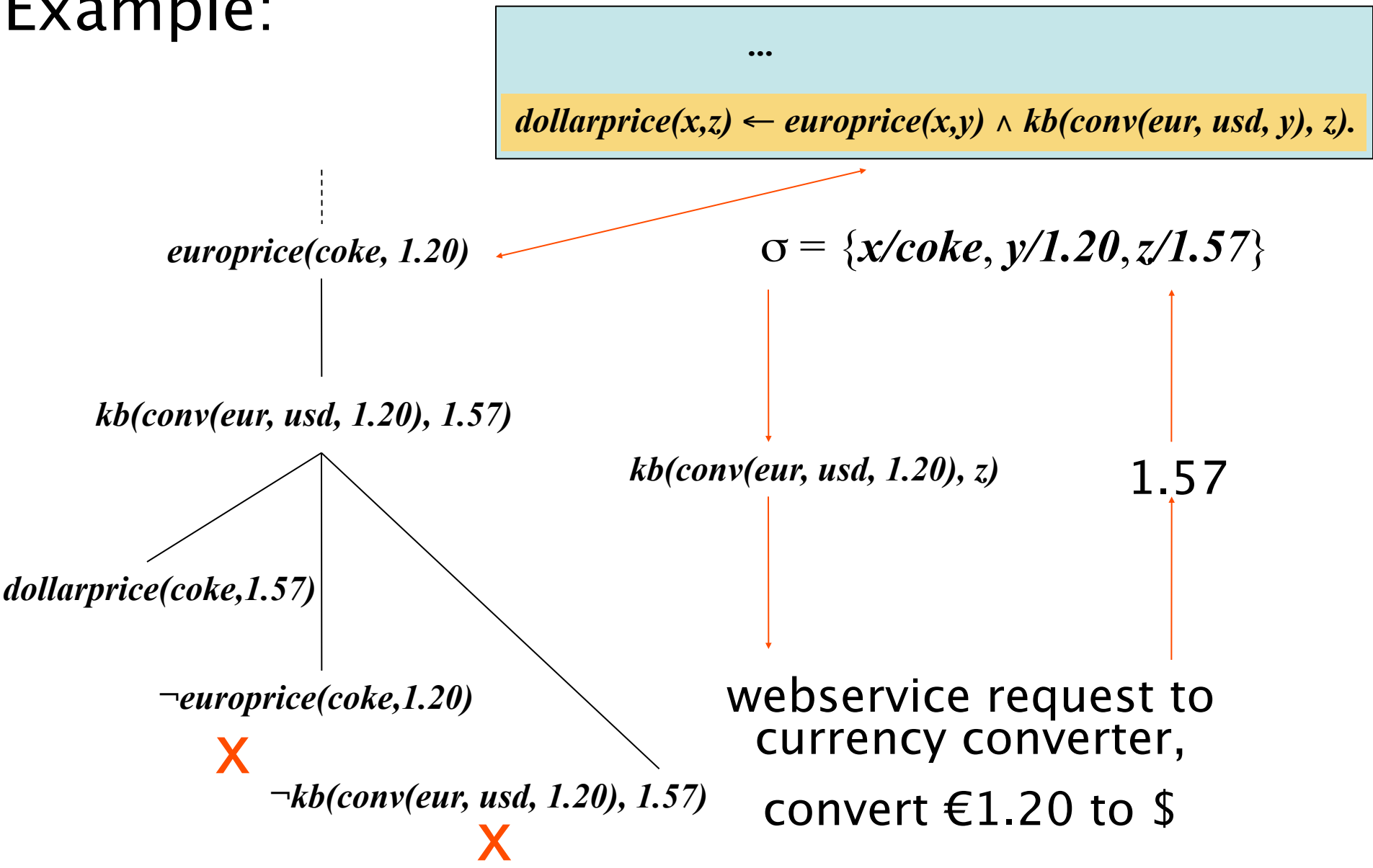
Example:


...
dollarprice(x,z) ← europrice(x,y) ∧ kb(conv(eur, usd, y), z).
europrice(coke, 1.20)
σ = {x/coke, y/1.20,
kb(conv(eur, usd, 1.20), z)
1.57
webservice request to currency converter,
convert €1.20 to $

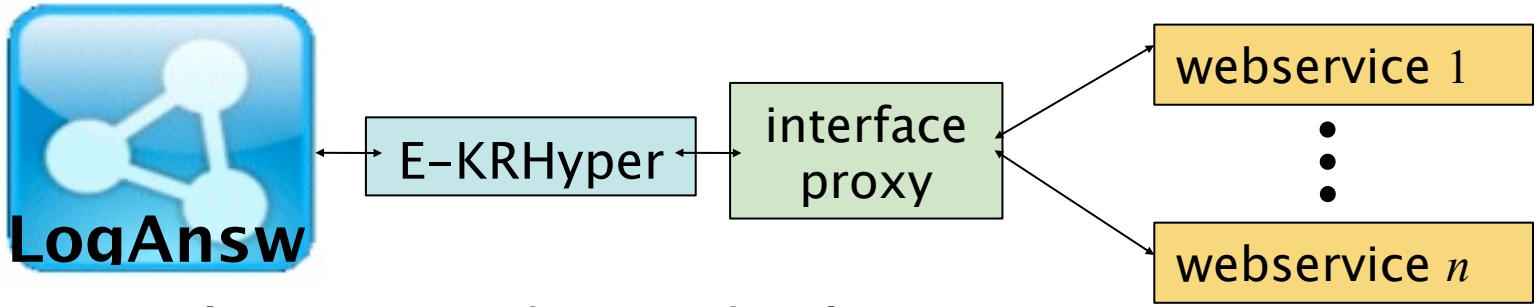
Example:


...
dollarprice(x,z) ← europrice(x,y) ∧ kb(conv(eur, usd, y), z).
europrice(coke, 1.20)
σ = {x/coke, y/1.20, z/1.57}
kb(conv(eur, usd, 1.20), z)
1.57
webservice request to currency converter,
convert €1.20 to $

Example:

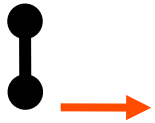

...
dollarprice(x,z) ← europrice(x,y) ∧ kb(conv(eur, usd, y), z).
europrice(coke, 1.20)
σ = {x/coke, y/1.20, z/1.57}
kb(conv(eur, usd, 1.20), 1.57)
kb(conv(eur, usd, 1.20), z)
1.57
dollarprice(coke,1.57)
¬europrice(coke,1.20)
X
¬kb(conv(eur, usd, 1.20), 1.57)
X
webservice request to currency converter,
convert €1.20 to $

Webservice via proxy

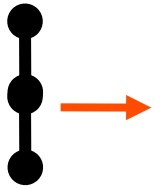


proxy always responds immediately:

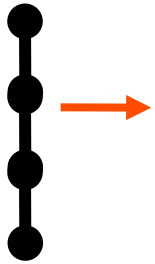
- "wait" - proxy asks webservice; prover does other inferences and will ask again later
- <result> - webservice has been asked and result is in proxy cache
- "no result" - webservice has been asked but cannot provide a useful result



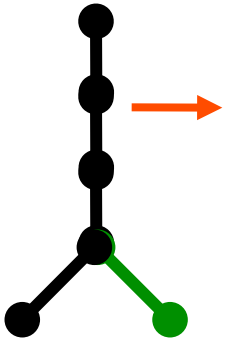
ws-query 1: "wait" Continue in current branch with other inferences.



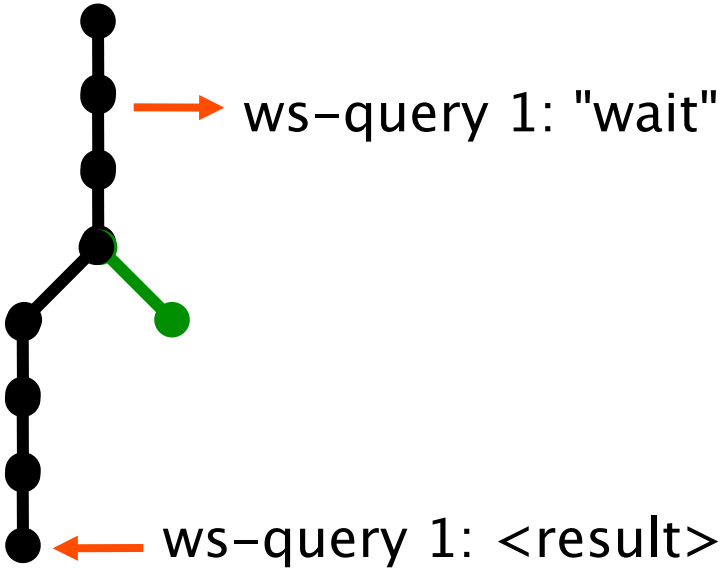
ws-query 1: "wait" Continue in current branch with other inferences.

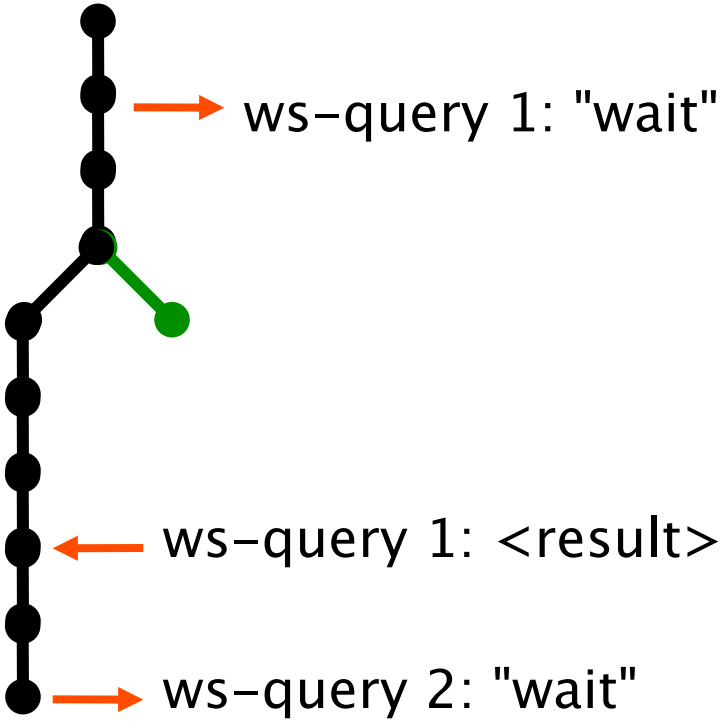


ws-query 1: "wait" Continue in current branch with other inferences.

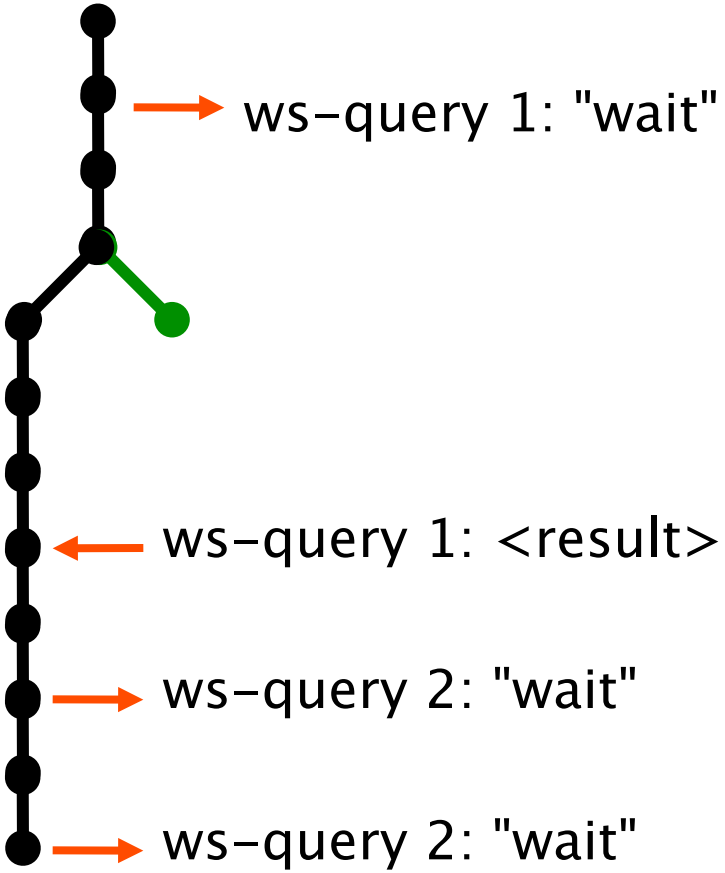


ws-query 1: "wait" Continue in current branch with other inferences.

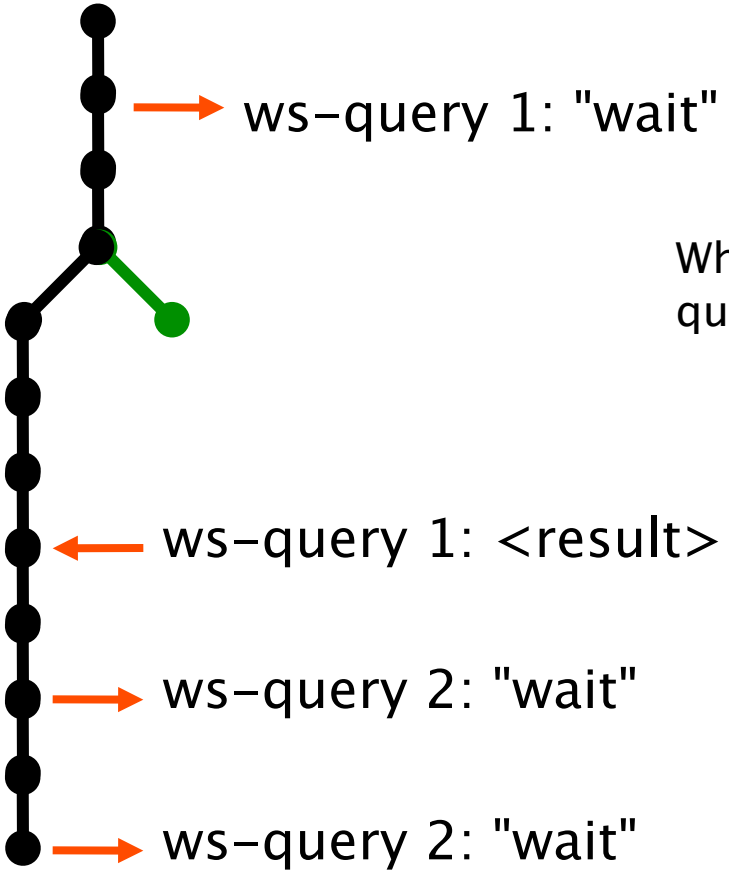

ws-query 1: "wait"
Continue in current branch with other inferences.
ws-query 1: <result>



Continue in current branch with other inferences.



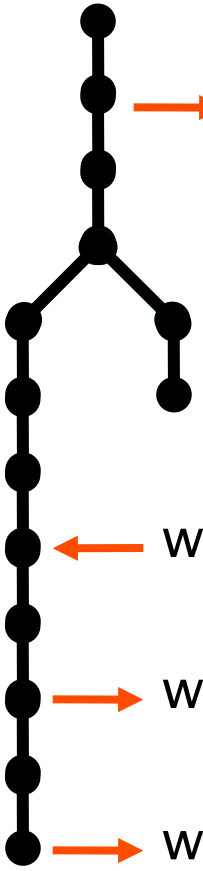
Continue in current branch with other inferences.



Continue in current branch with other inferences.

When the current branch is exhausted and a query is still waiting, postpone the branch.

postponed



ws-query 1: "wait"

Continue in current branch with other inferences.

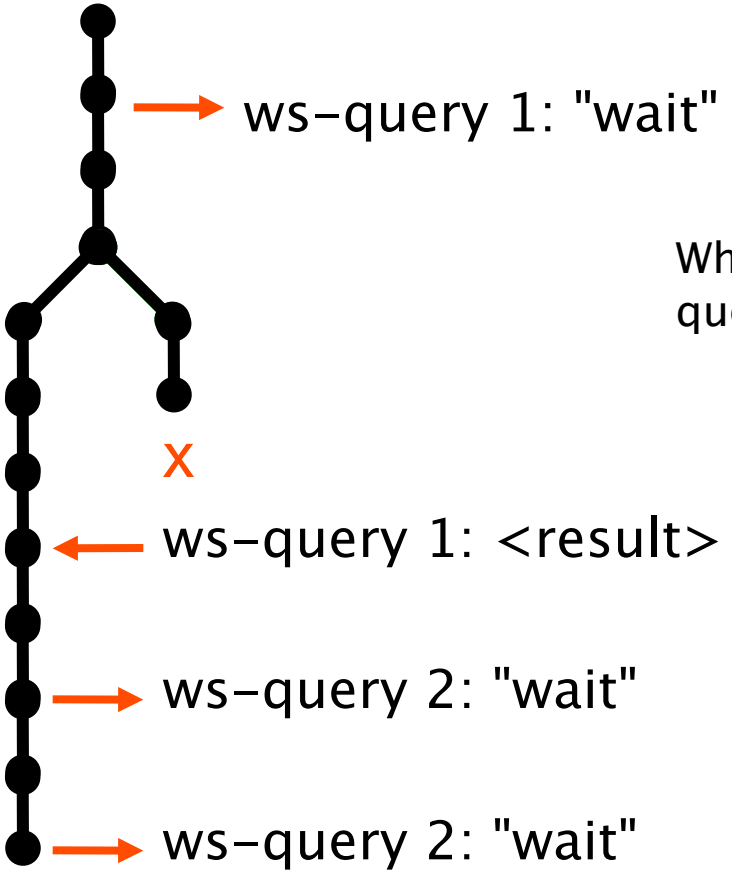
When the current branch is exhausted and a query is still waiting, postpone the branch.

ws-query 1: <result>

ws-query 2: "wait"

ws-query 2: "wait"

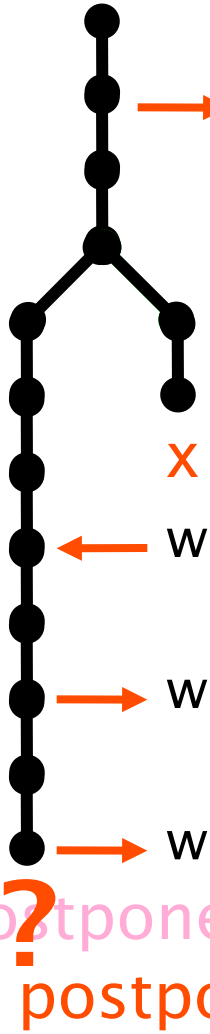
postponed



Continue in current branch with other inferences.

When the current branch is exhausted and a query is still waiting, postpone the branch.

postponed



Continue in current branch with other inferences.

When the current branch is exhausted and a query is still waiting, postpone the branch.

Return to postponed branch when all other branches have been closed or postponed.

Waiting query returns...

- <result>: continue reasoning
- "no result": done, branch is model
- "wait": depends on configuration:
  either treat as "no result",
  or wait indefinitely until some waiting query returns <result> or "no result"

Webservices in Use

Webservices in Use

- Yahoo GeoPlanet
- Weather Service
- Currency Converter

- OpenCyc
- DBPedia
- SpassYago -- coming soon?

Webservices for Abductive Relaxation

logical query representation:

$false \leftarrow Q_1 \wedge \ldots \wedge Q_n$

Webservices for Abductive Relaxation

logical query representation:

$false \leftarrow Q_1 \wedge ... \wedge Q_n$

1. add abductive relaxation clause:

$relAns(rel(c_1, x_1), ..., rel(c_m, x_m)) \leftarrow$
$Q_1' \wedge ... \wedge Q_n' \wedge kb(abduce(c_1), x_1) \wedge ... \wedge kb(abduce(c_m), x_m)$

where Q_1',..., Q_n' result from Q_1,..., Q_n by replacing each occurrence c_i of a constant with a fresh variable x_i.

Webservices for Abductive Relaxation

logical query representation:

$false \leftarrow Q_1 \wedge ... \wedge Q_n$

1. add abductive relaxation clause:

$relAns(rel(c_1, x_1), ..., rel(c_m, x_m)) \leftarrow$
$Q_1' \wedge ... \wedge Q_n' \wedge kb(abduce(c_1), x_1) \wedge ... \wedge kb(abduce(c_m), x_m)$

where Q_1',..., Q_n' result from Q_1,..., Q_n by replacing each occurrence c_i of a constant with a fresh variable x_i.

2. add trivial abduction unit clause:

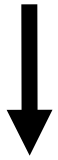
$kb(abduce(x), x) \leftarrow .$

"Who invented Coca-Cola?"

$$\mathit{false} \leftarrow \mathit{is}(x, \mathit{person}) \wedge \mathit{invent}(x, \mathit{cocacola}).$$

"Who invented Coca-Cola?"

false ← ***is*(*x*, *person*)** ∧ ***invent*(*x*, *cocacola*).**



E-KRHyper adds:

***relAns*(*rel*(*person*,*y*), *rel*(*cocacola*, *z*))** ←

***is*(*x*, *y*)** ∧ ***invent*(*x*, *z*)**

∧ ***kb*(*abduce*(*person*), *y*)** ∧ ***kb*(*abduce*(*cocacola*), *z*).**

***kb*(*abduce*(*x*), *x*)** ← .

"Who invented Coca-Cola?"

$$\mathit{false} \leftarrow \mathit{is}(x, \mathit{person}) \wedge \mathit{invent}(x, \mathit{cocacola}).$$

E-KRHyper adds:

$$\mathit{relAns}(\mathit{rel}(\mathit{person}, y), \mathit{rel}(\mathit{cocacola}, z)) \leftarrow$$
$$\mathit{is}(x, y) \wedge \mathit{invent}(x, z)$$
$$\wedge\ \mathit{kb}(\mathit{abduce}(\mathit{person}), y) \wedge \mathit{kb}(\mathit{abduce}(\mathit{cocacola}), z).$$

$$\mathit{kb}(\mathit{abduce}(x), x) \leftarrow .$$

- prover will run until proof is found or timeout
- if timeout, return any derived relAns-facts instead
- let user decide if abductive relaxations were acceptable(?)

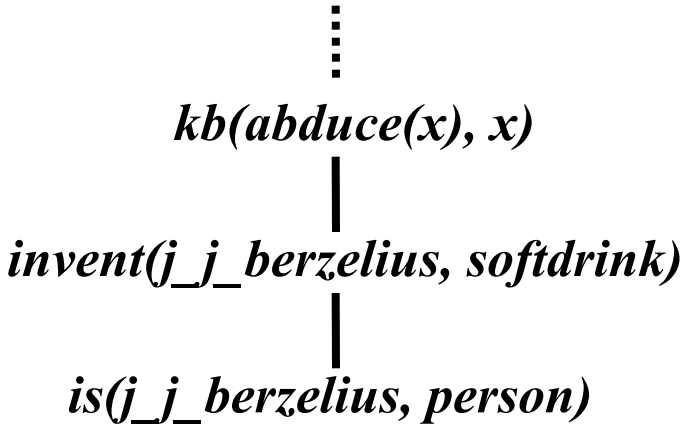
false ⇐ is(x, person) ∧ invent(x, cocacola).

relAns(rel(person,y), rel(cocacola,z)) ⇐

is(x, y) ∧ invent(x, z) ∧ kb(abduce(person), y) ∧ kb(abduce(cocacola), z).

kb(abduce(x), x) ⇐ .

false ⇐ is(x, person) ∧ invent(x, cocacola).
relAns(rel(person,y), rel(cocacola,z)) ⇐
is(x, y) ∧ invent(x, z) ∧ kb(abduce(person), y) ∧ kb(abduce(cocacola), z).
kb(abduce(x), x) ⇐ .

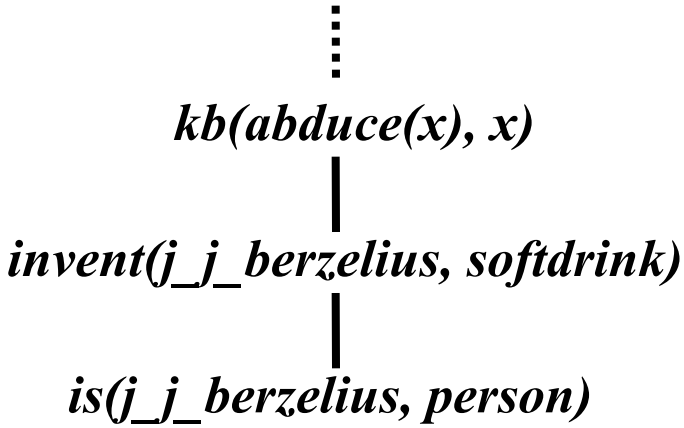


false ⇐ is(x, person) ∧ invent(x, cocacola).

relAns(rel(person,y), rel(cocacola,z)) ⇐

is(x, y) ∧ invent(x, z) ∧ kb(abduce(person), y) ∧ kb(abduce(cocacola), z).

kb(abduce(x), x) ⇐ .

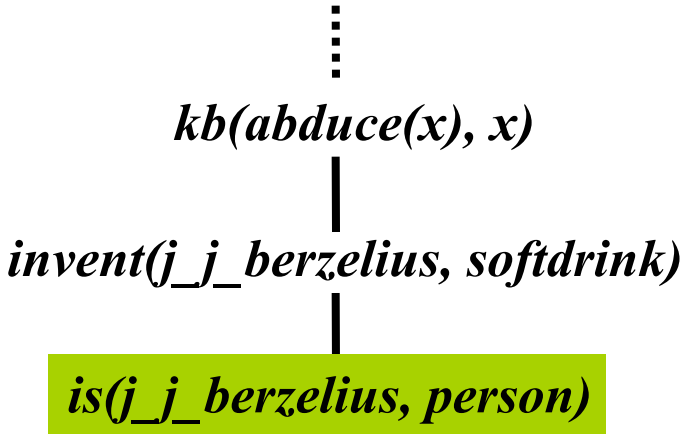


false ← *is(x, person)* ∧ *invent(x, cocacola).*

relAns(rel(person,y), rel(cocacola,z)) ←

is(x, y) ∧ *invent(x, z)* ∧ *kb(abduce(person), y)* ∧ *kb(abduce(cocacola), z).*

kb(abduce(x), x) ← .



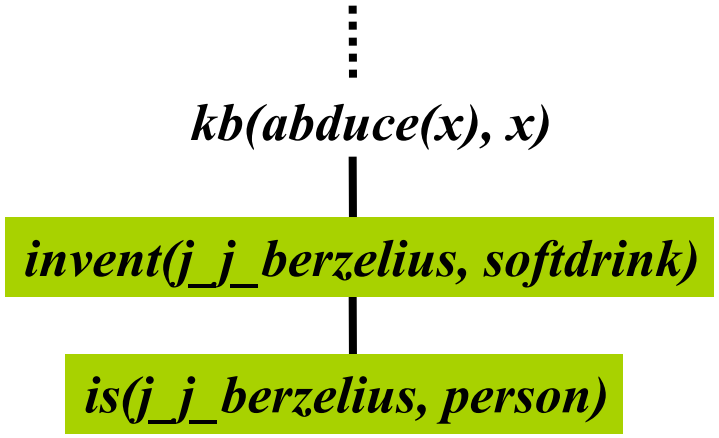
σ = {*x/j_j_berzelius*,
y/person,

false ← *is(x, person)* ∧ *invent(x, cocacola).*

relAns(rel(person,y), rel(cocacola,z)) ←

is(x, y) ∧ *invent(x,z)* ∧ *kb(abduce(person), y)* ∧ *kb(abduce(cocacola), z).*

kb(abduce(x), x) ← .

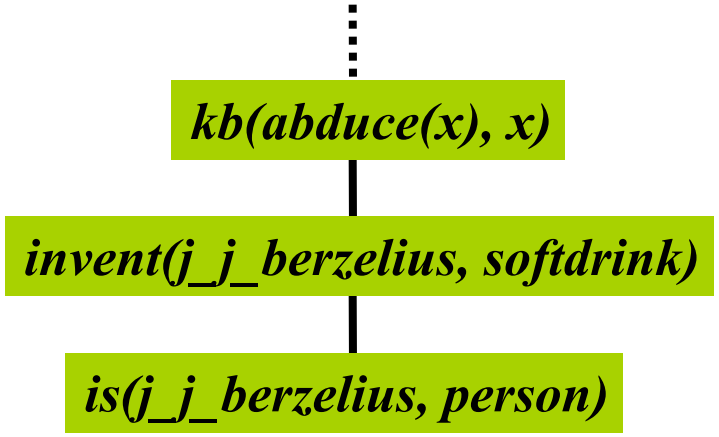


σ = {*x/j_j_berzelius*,
y/person,
z/softdrink

false ← *is(x, person)* ∧ *invent(x, cocacola).*

relAns(rel(person,y), rel(cocacola,z)) ←
is(x, y) ∧ *invent(x,z)* ∧ *kb(abduce(person),y)* ∧ *kb(abduce(cocacola), z).*

kb(abduce(x), x) ← .



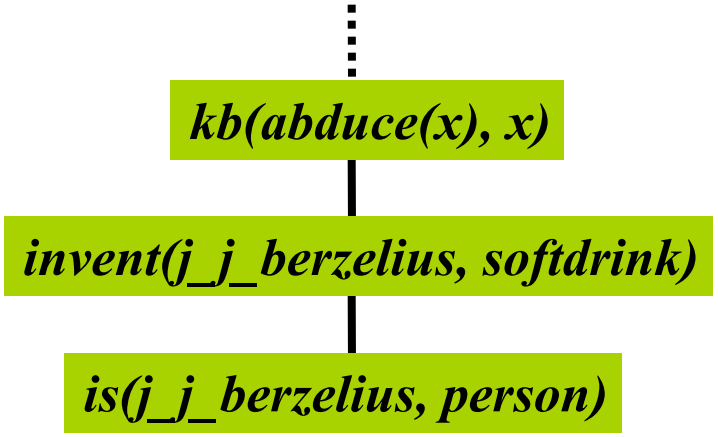
σ = {*x/j_j_berzelius*,
y/person,
z/softdrink

false ← *is(x, person)* ∧ *invent(x, cocacola).*

relAns(rel(person,y), rel(cocacola,z)) ←

is(x, y) ∧ *invent(x,z)* ∧ *kb(abduce(person),y)* ∧ *kb(abduce(cocacola), z)*

kb(abduce(x), x) ← .



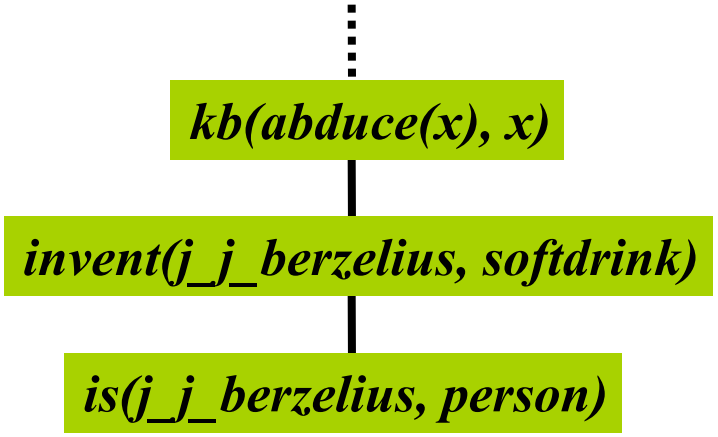
σ = {*x/j_j_berzelius*,
y/person,
z/softdrink

false ← *is(x, person)* ∧ *invent(x, cocacola).*

relAns(rel(person,y), rel(cocacola,z)) ←

is(x, y) ∧ *invent(x,z)* ∧ *kb(abduce(person),y)* ∧ *kb(abduce(cocacola), z)*

kb(abduce(x), x) ← .



webservice request to ontology browser,

superclasses of *cocacola*?

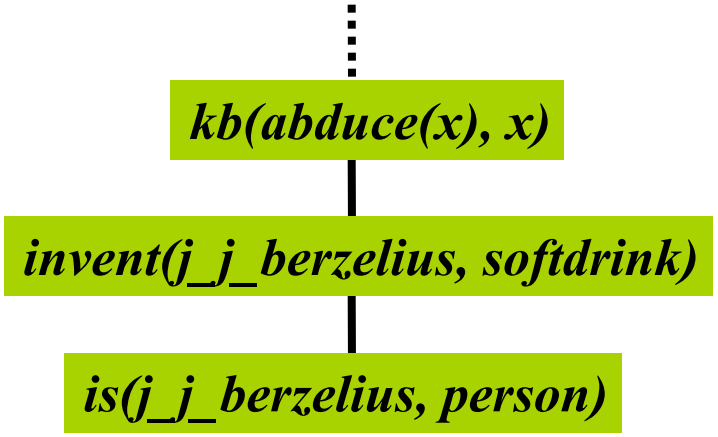
σ = {*x/j_j_berzelius*,
y/person,
z/softdrink

false ← *is(x, person)* ∧ *invent(x, cocacola).*

relAns(rel(person,y), rel(cocacola,z)) ←

is(x, y) ∧ *invent(x,z)* ∧ *kb(abduce(person),y)* ∧ *kb(abduce(cocacola), z)*

kb(abduce(x), x) ← .



webservice request to ontology browser,

superclasses of *cocacola*?
response: *kb(abduce(cocacola),softdrink)*

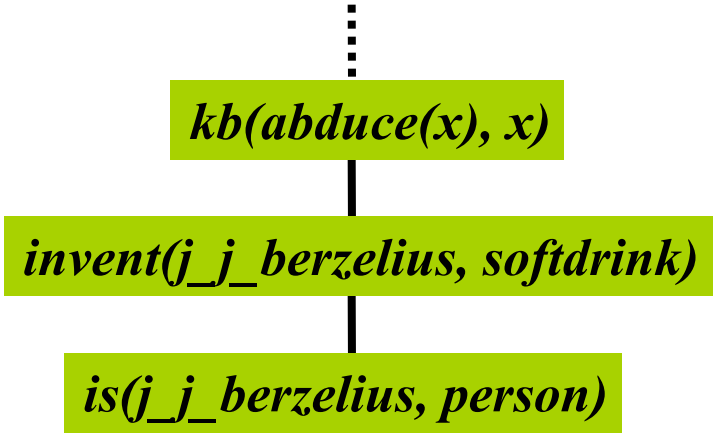
σ = {*x/j_j_berzelius*,
y/person,
z/softdrink

false ← *is(x, person)* ∧ *invent(x, cocacola).*

relAns(rel(person,y), rel(cocacola,z)) ←

is(x, y) ∧ *invent(x,z)* ∧ *kb(abduce(person),y)* ∧ *kb(abduce(cocacola), z)*

kb(abduce(x), x) ← .



webservice request to ontology browser,

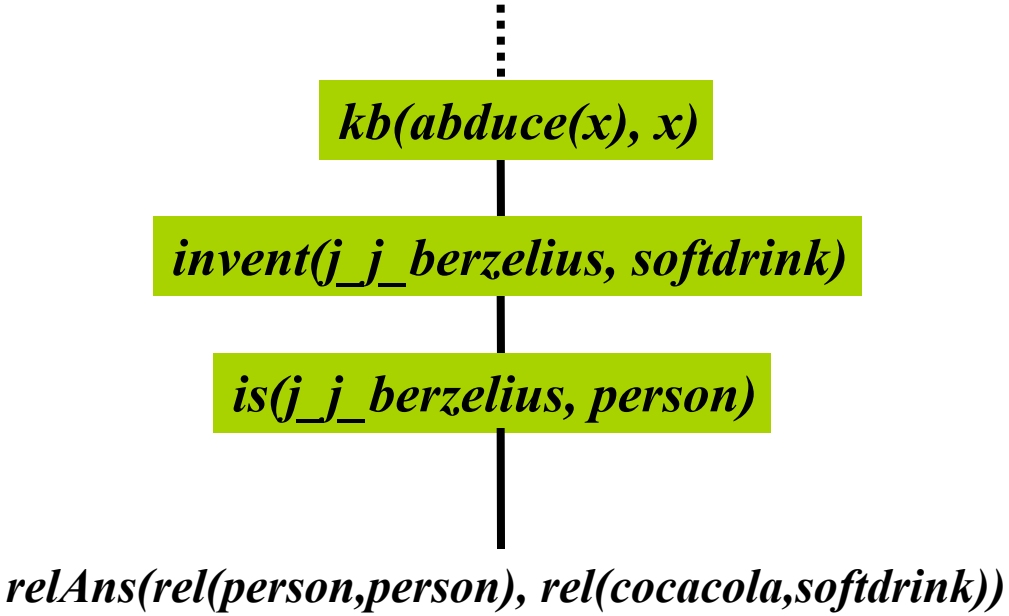
superclasses of *cocacola*?
response: *kb(abduce(cocacola),softdrink)*

σ = {*x/j_j_berzelius*,
y/person,
z/softdrink

false ← *is(x, person)* ∧ *invent(x, cocacola).*

relAns(rel(person,y), rel(cocacola,z)) ←
is(x, y) ∧ *invent(x,z)* ∧ *kb(abduce(person),y)* ∧ *kb(abduce(cocacola), z)*

kb(abduce(x), x) ← .



webservice request to ontology browser,

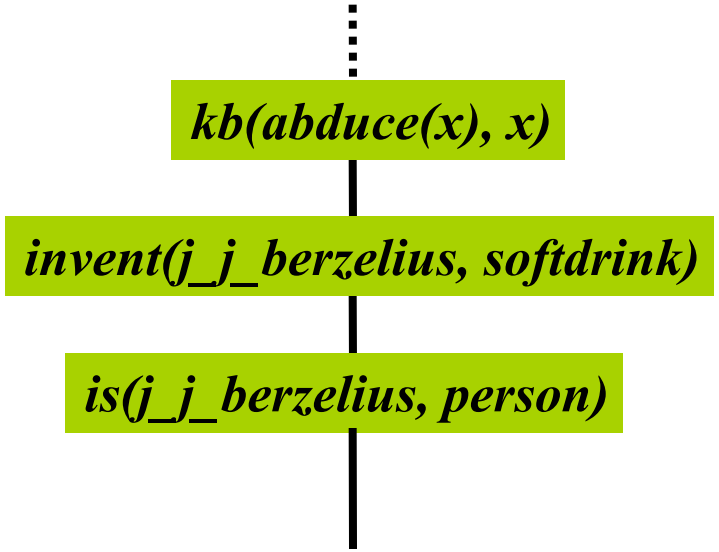
superclasses of *cocacola*?
response: *kb(abduce(cocacola),softdrink)*

σ = {*x/j_j_berzelius*,
y/person,
z/softdrink}

false ← *is(x, person)* ∧ *invent(x, cocacola).*

relAns(rel(person,y), rel(cocacola,z)) ←
is(x, y) ∧ *invent(x,z)* ∧ *kb(abduce(person),y)* ∧ *kb(abduce(cocacola), z)*

kb(abduce(x), x) ← .



relAns(rel(person,person), rel(cocacola,softdrink))

webservice request to ontology browser,

superclasses of *cocacola*?
response: *kb(abduce(cocacola),softdrink)*

σ = {*x/j_j_berzelius*,
y/person,
z/softdrink}

"Who invented Coca-Cola?"
- "J.J. Berzelius, if 'soft drink' can replace 'Coca-Cola'." (No!)

Partitioning of Clause set

Compute dependency graphs in pre-processing

Test:

309 multi-literal clauses

10 061 units

Reduction of 20% of clauses and 10% of time

Issues for AD

Time-constraints

Relaxing queries

Abduction

Webservices

Confidence Measures

QA Forums

Forum users ask and answer questions.

Users can grade answers or answerers.

QA-systems like LogAnser usually give the best answer they can find - even if it is poor.

QA-systems like LogAnser usually give the best answer they can find - even if it is poor.

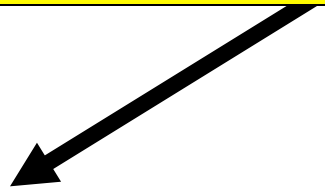
If 18% of questions are answered correctly,
then almost 82% get wrong answers.

High risk that forum users perceive LogAnswer as a nuisance!

QA-systems like LogAnser usually give the best answer they can find – even if it is poor.

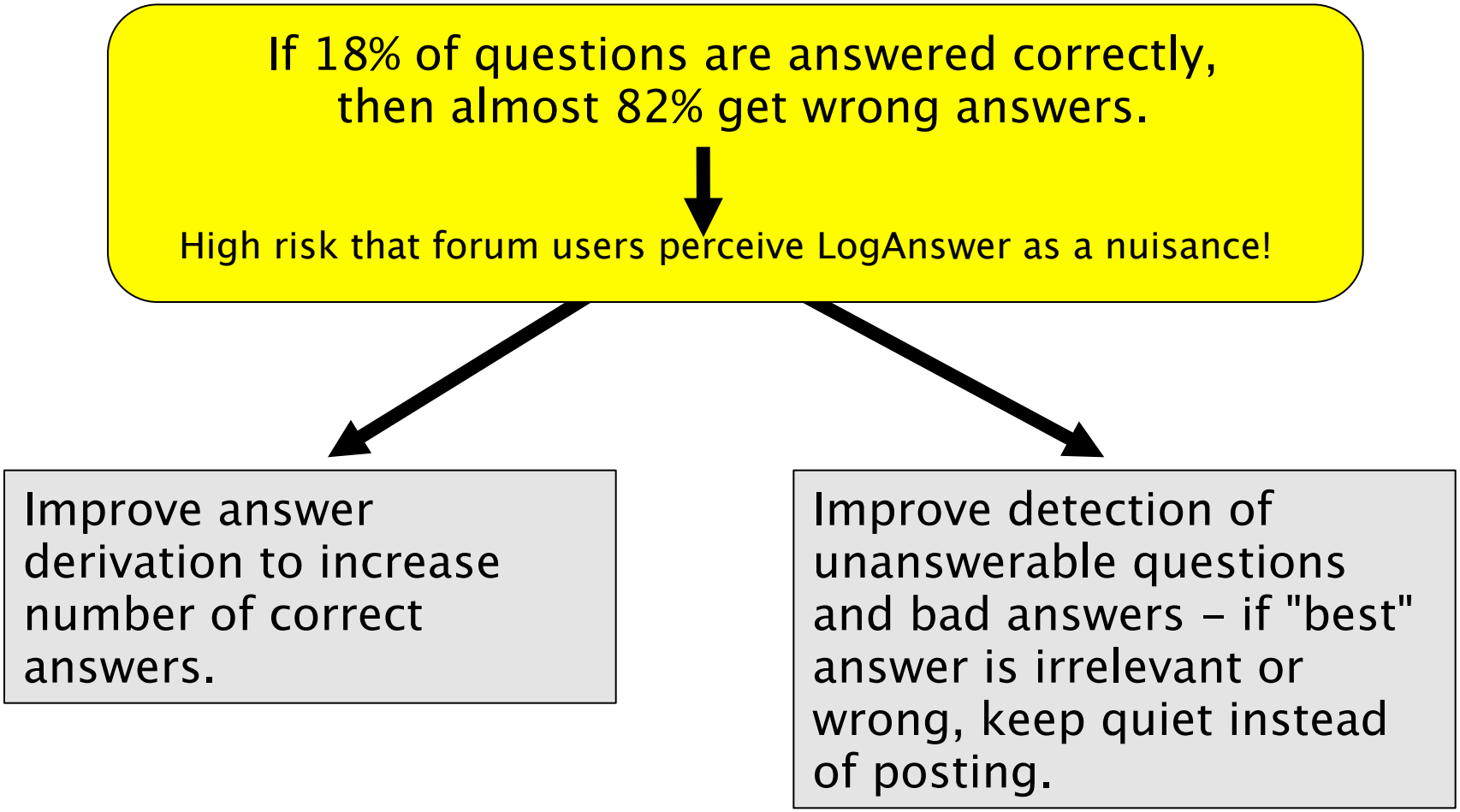
If 18% of questions are answered correctly, then almost 82% get wrong answers.

High risk that forum users perceive LogAnswer as a nuisance!

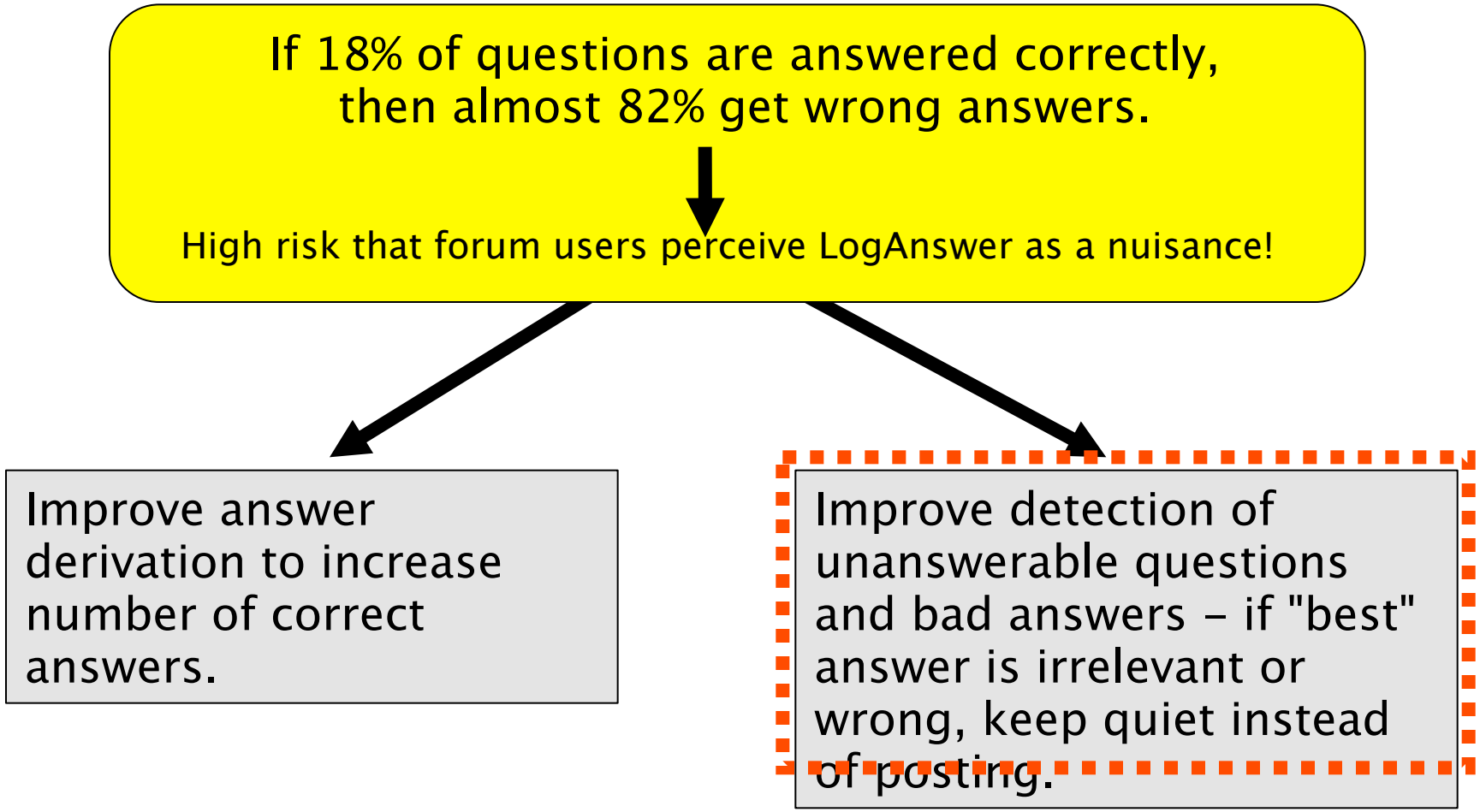


Improve answer derivation to increase number of correct answers.

QA-systems like LogAnser usually give the best answer they can find – even if it is poor.



QA-systems like LogAnser usually give the best answer they can find – even if it is poor.



Block question categories:

Some categories contain mostly unanswerable questions.

- celebrities: **"What is the phone number of Justin Bieber?"**
- video games: **"How do you unlock the Bowser Level in Super Mario Sunshine?"**
- computer help: **"When I create a file in Corel (Windows), how do I..."**
- ...

Block question categories:

Some categories contain mostly unanswerable questions.

- celebrities: **"What is the phone number of Justin Bieber?"**
- video games: **"How do you unlock the Bowser Level in Super Mario Sunshine?"**
- computer help: **"When I create a file in Corel (Windows), how do I..."**
- ...

Blocking selected categories is quite safe (2.2% false positives), but 66% of questions have no category at all.

Block question categories:

Some categories contain mostly unanswerable questions.

- celebrities: **"What is the phone number of Justin Bieber?"**
- video games: **"How do you unlock the Bowser Level in Super Mario Sunshine?"**
- computer help: **"When I create a file in Corel (Windows), how do I..."**
- ...

Blocking selected categories is quite safe (2.2% false positives), but 66% of questions have no category at all.

Block question types:

Some questions ask for opinions, recognizable by keywords.

- **"What do you think of...?"**
- **"What is the best/worst/most beautiful...?"**

External Ontologies

- Cyc is too large for ATP:

  3.3 million formulas in the TPTP-version of OpenCyc, requires 15 minutes to load and 7GB RAM in E-KRHyper.

  - Use query-based axiom selection methods.
  - No equational reasoning for Cyc-axioms.
  - Only use some parts of Cyc (i.e. subclass-assertions).