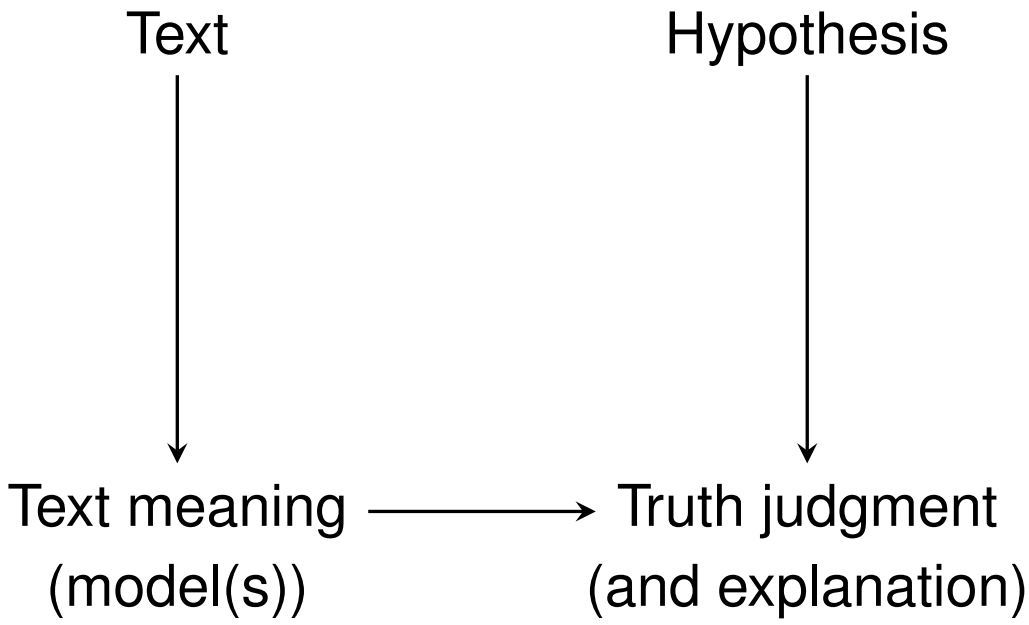
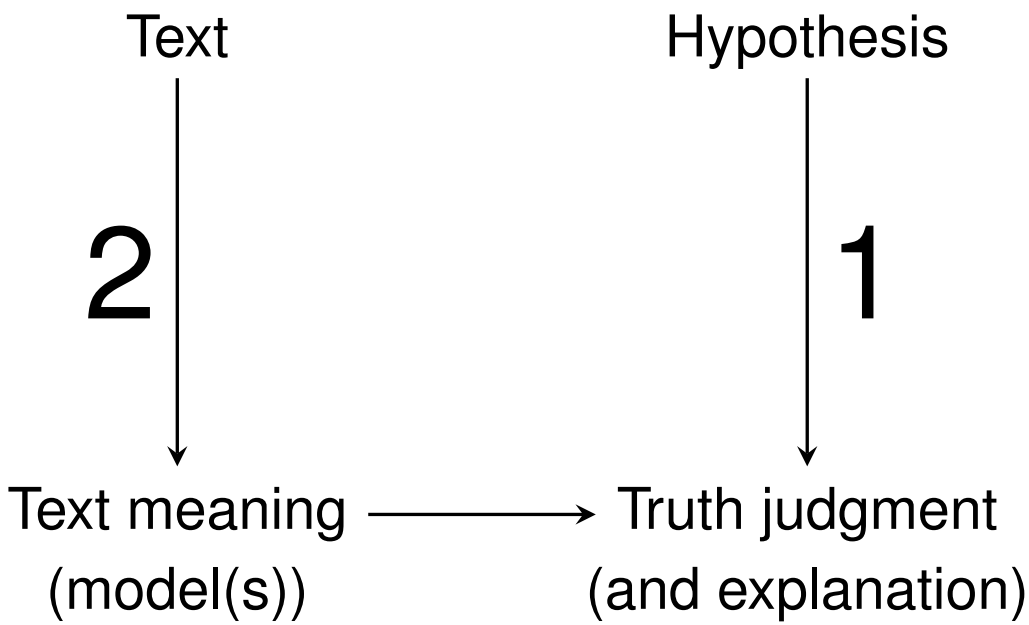
Back to the model

Jason Perry and Chung-chieh Shan
Rutgers University

July 10, 2011


Text
Hypothesis
Text meaning
(model(s))
Truth judgment
(and explanation)



This is a workshop talk.

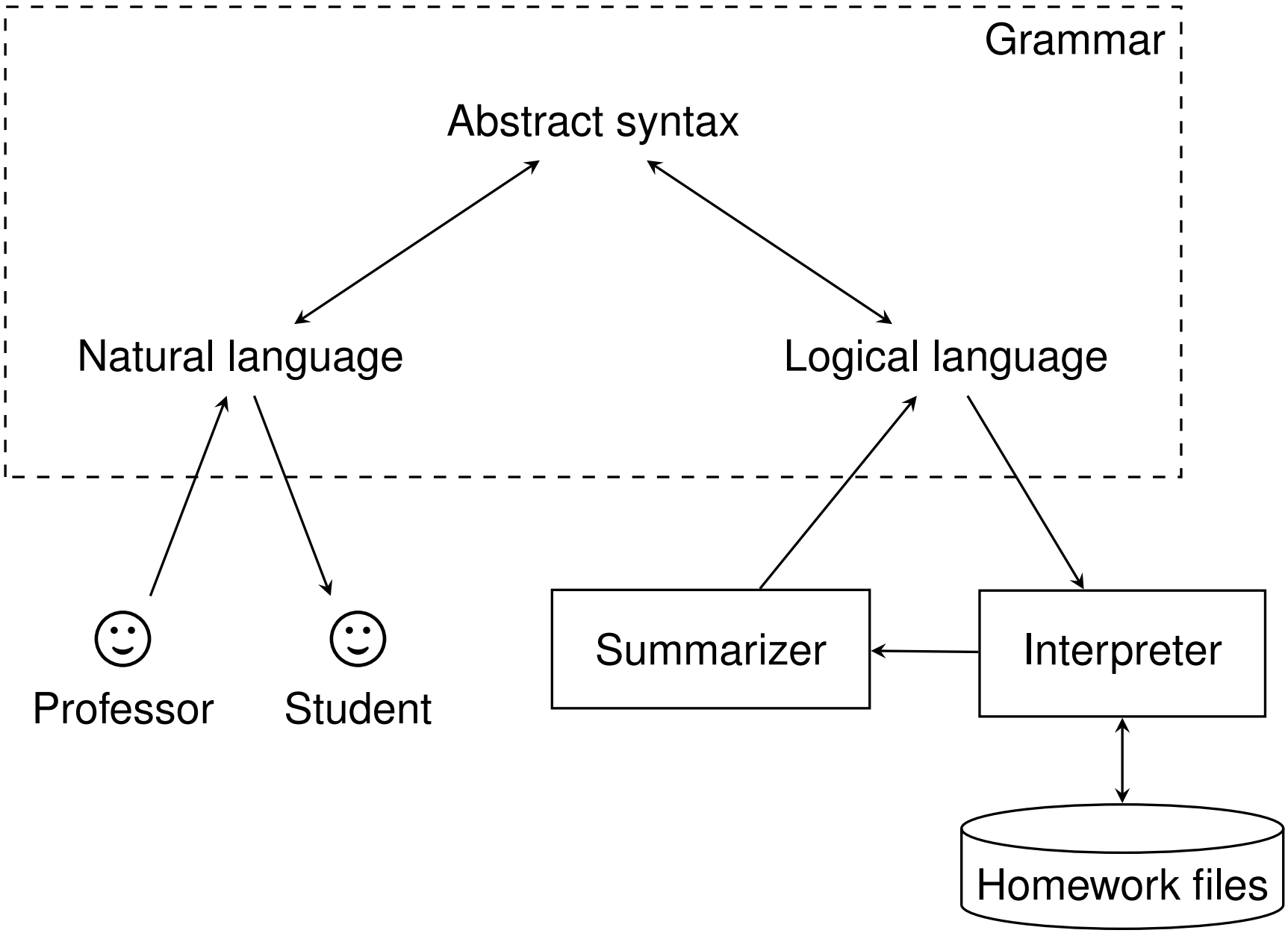
1. Checking programming homework

Every source file must compile
and you must include a README file
that references all header files.

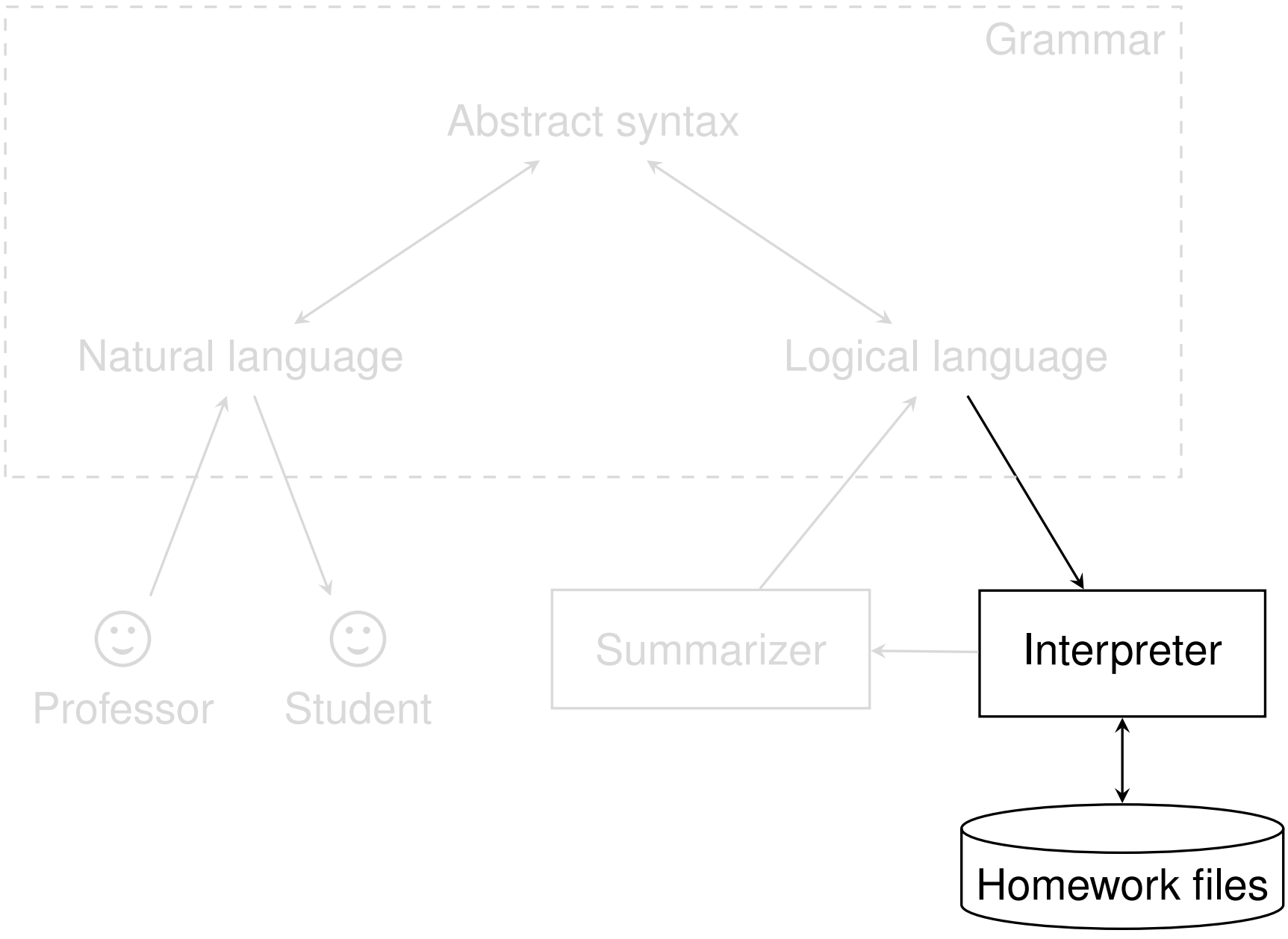
‘foo.c’ doesn’t compile
and ‘readme.txt’ doesn’t mention ‘bar.c’.

```
$ ./ProGrader
Enter sentence to test:
every source file compiles and "readme.txt" mentions every source fi
#-----------Translation--------------
and every [ sourcefile ] ( \x -> ( compiles x ) )
    ( \x -> every [ sourcefile ] ( \y -> ( ( mentions y ) x ) ) "rea
...checking...
#-----------Translated Explanation Tree---------------
a source file doesn't compile or "readme.txt" doesn't mention every
    a source file doesn't compile, because
        "nowork2.c" or "nowork.c" or "hello2.c" or "hello.c" don't c
            "nowork2.c" doesn't compile and
            "nowork.c" doesn't compile and
            "hello2.c" compiles and
            "hello.c" compiles and
    "readme.txt" doesn't mention every source file, because
        "readme.txt" doesn't mention "nowork2.c" and "nowork.c" and
...snip!...
#----Translated Summary-----
    a source file doesn't compile, because
            "nowork2.c" and "nowork.c" don't compile, and
    "readme.txt" doesn't mention every source file, because
            "readme.txt" doesn't mention "nowork2.c" or "hello2.c"
```

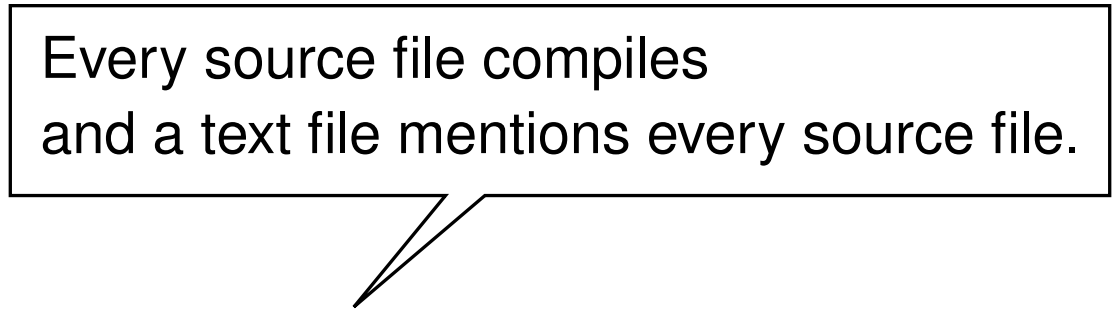
System architecture



Logical interpreter



Generalized quantifiers (Barwise & Cooper, Woods)

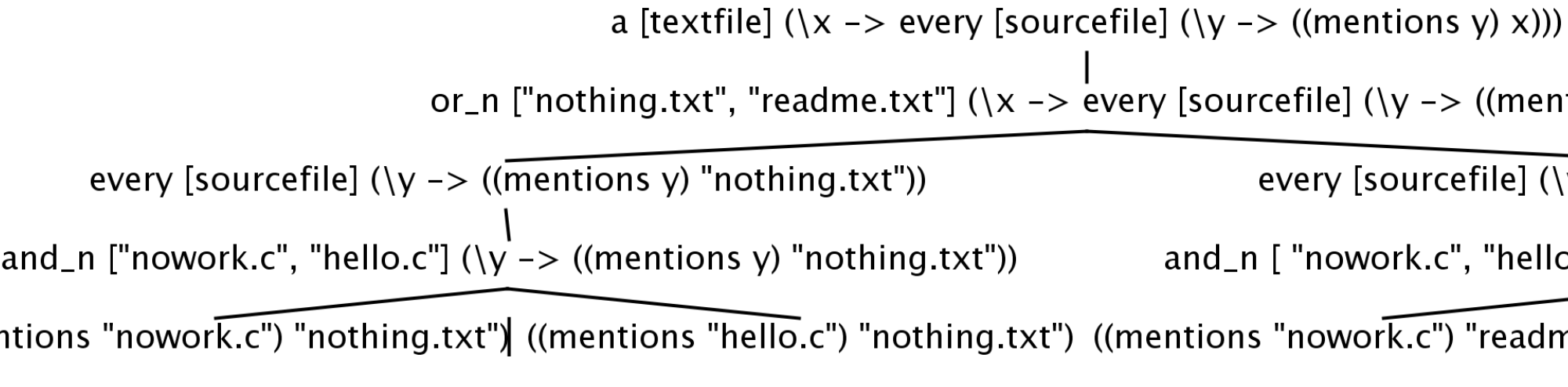



```
and every [ sourcefile ]
          \x -> (compiles x)
    a [ textfile ]
      \x -> every [ sourcefile ]
                  \y -> ((mentions y) x)
```

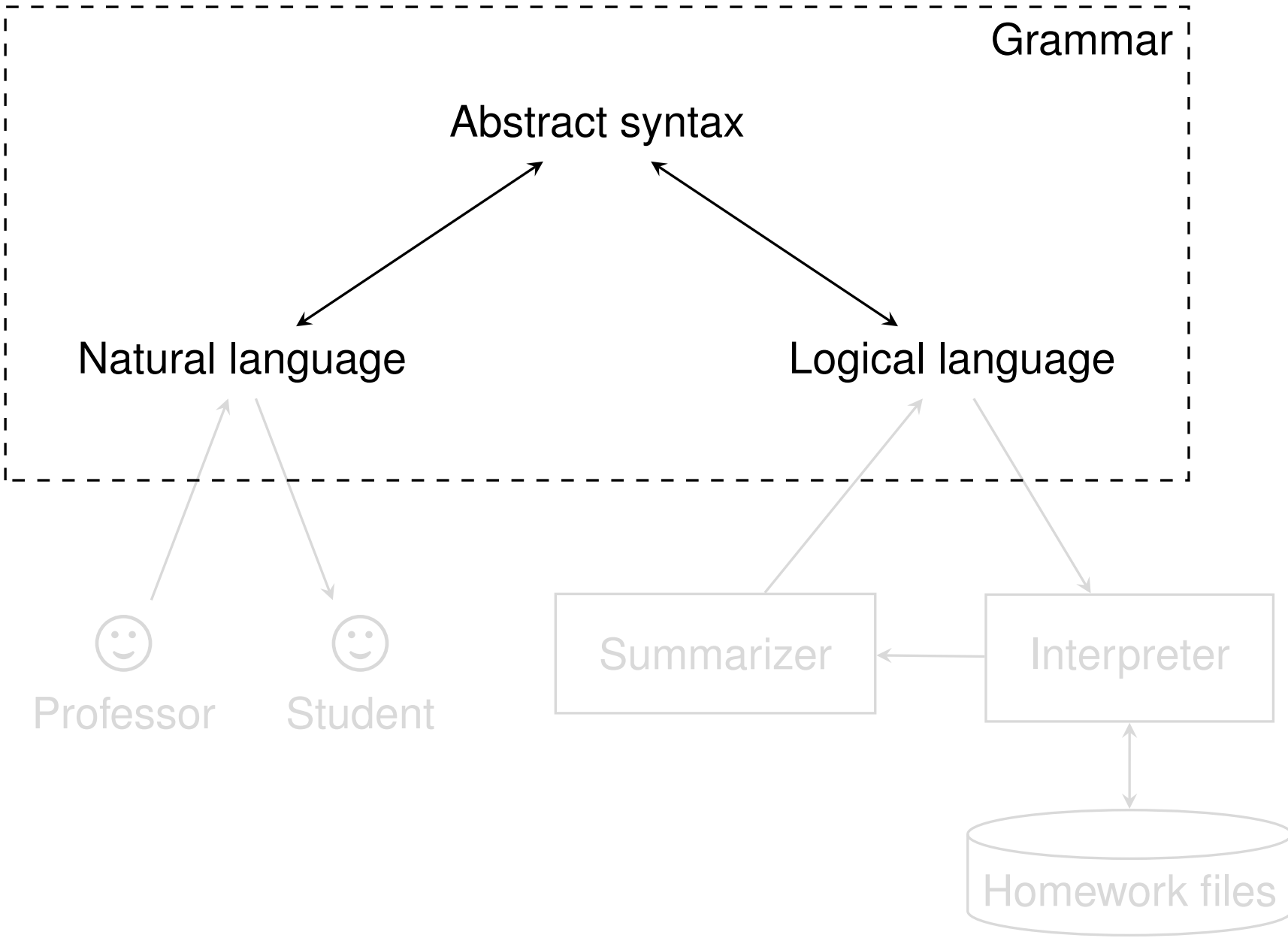
- Easy interpreter (model checker)
- Domain-specific vocabulary
  - ▷ checks predicates
  - ▷ enumerates quantificational domains

Accumulate evidence

Label each node with a formula and its value.



Bidirectional grammar



Abstract syntax vs concrete syntax

$$\texttt{S} \rightarrow \texttt{NP VP}$$

```
"foo.c" compiles
```

Abstract syntax vs concrete syntax

$$\mathtt{S} \rightarrow \mathtt{NP}, \mathtt{VP}$$

```
ApplyS "foo.c" Compiles
```

$$\mathtt{S}_1 = \mathtt{NP}_1\ \mathtt{VP}_1$$

```
"foo.c" compiles
```

Abstract syntax vs concrete syntax

$$\mathtt{S} \rightarrow \mathtt{NP}, \mathtt{VP}$$

```
ApplyS "foo.c" Compiles
```

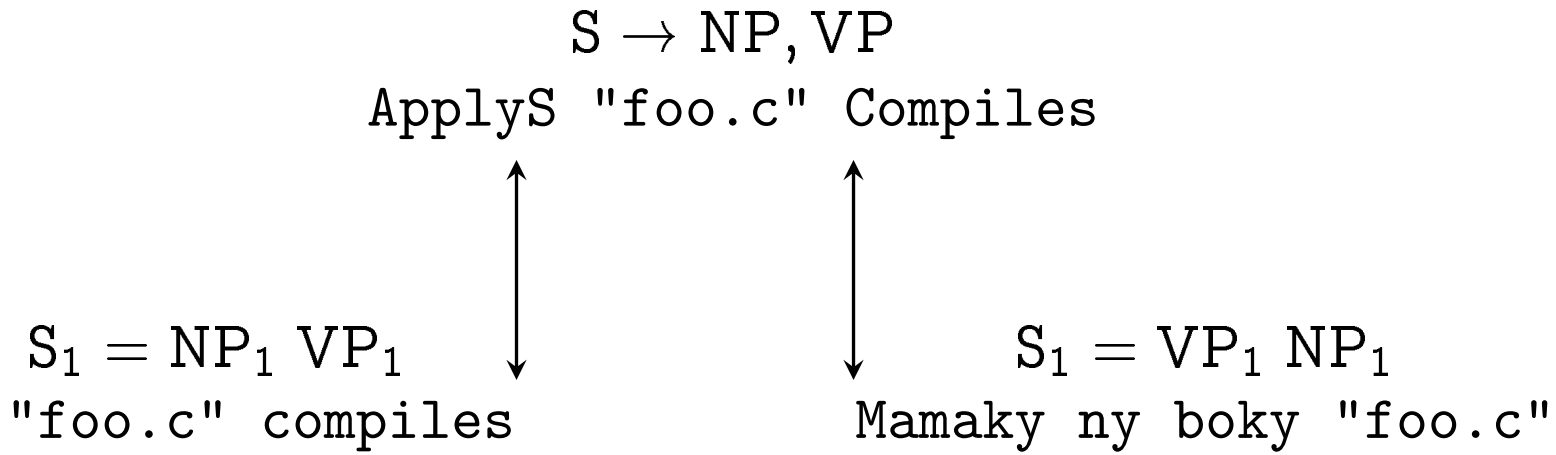
$$\mathtt{S}_1 = \mathtt{NP}_1\ \mathtt{VP}_1$$

```
"foo.c" compiles
```

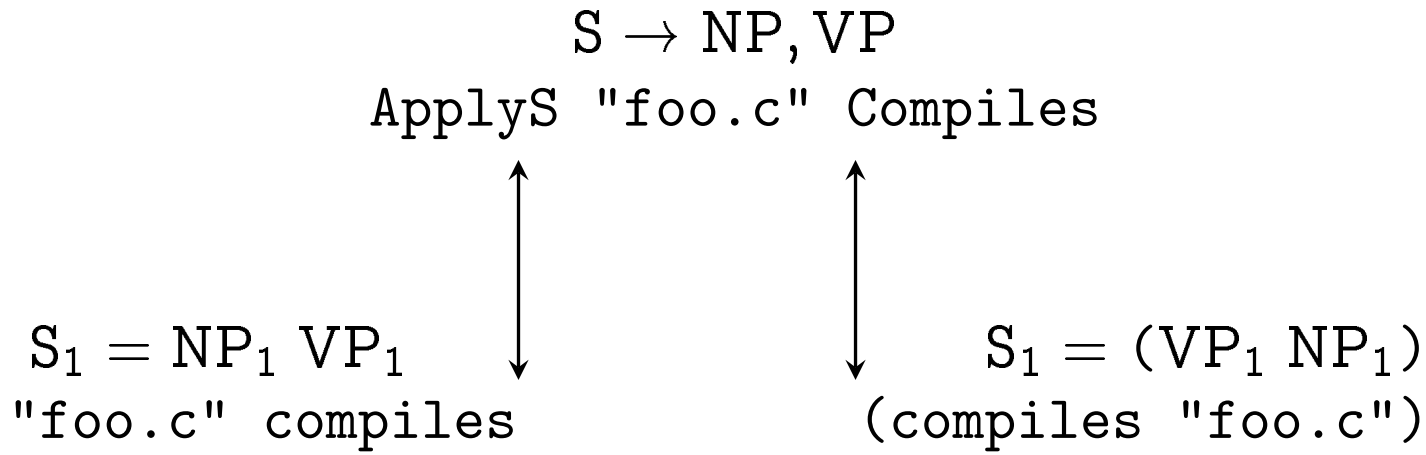
$$\mathtt{S}_1 = \mathtt{VP}_1\ \mathtt{NP}_1$$

```
Mamaky ny boky "foo.c"
```

Abstract syntax vs concrete syntax

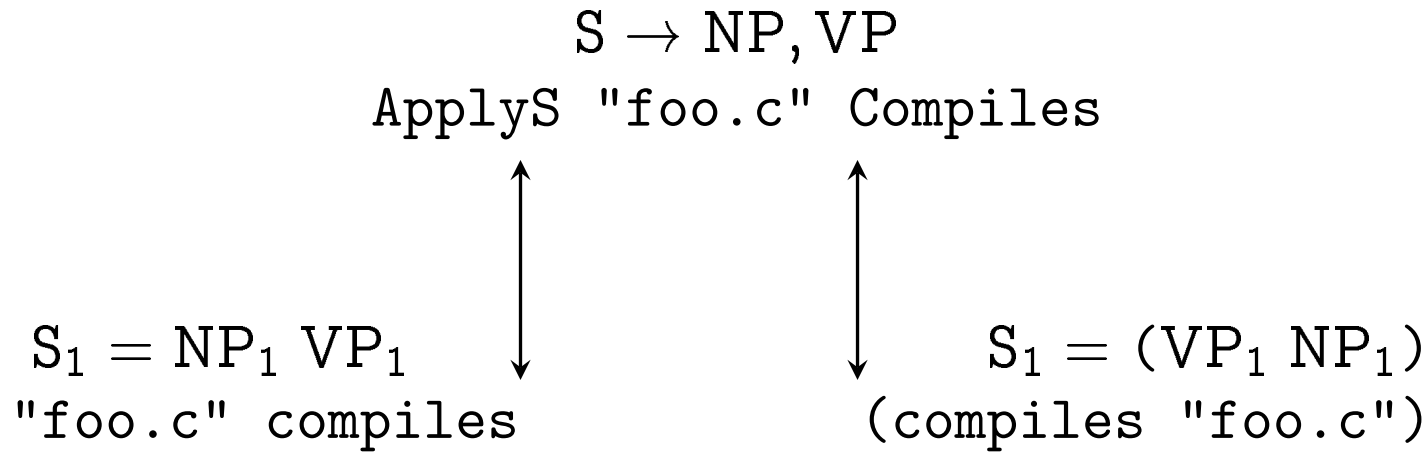


Abstract syntax vs concrete syntax



Grammatical Framework (Ranta) for English↔logic translation

Lambda tricks for quantifiers

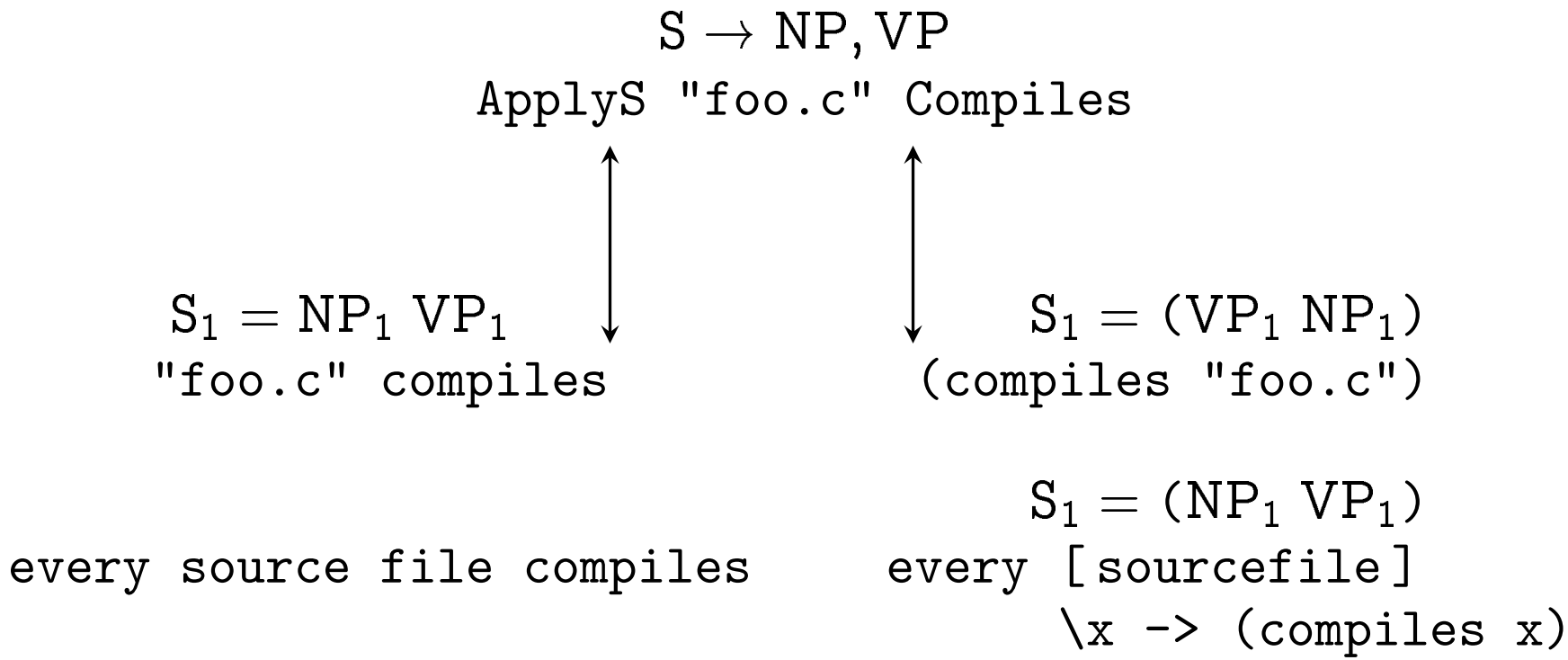



```
every source file compiles
```

```
every [sourcefile]
      \x -> (compiles x)
```

Grammatical Framework (Ranta) for English↔logic translation

Lambda tricks for quantifiers

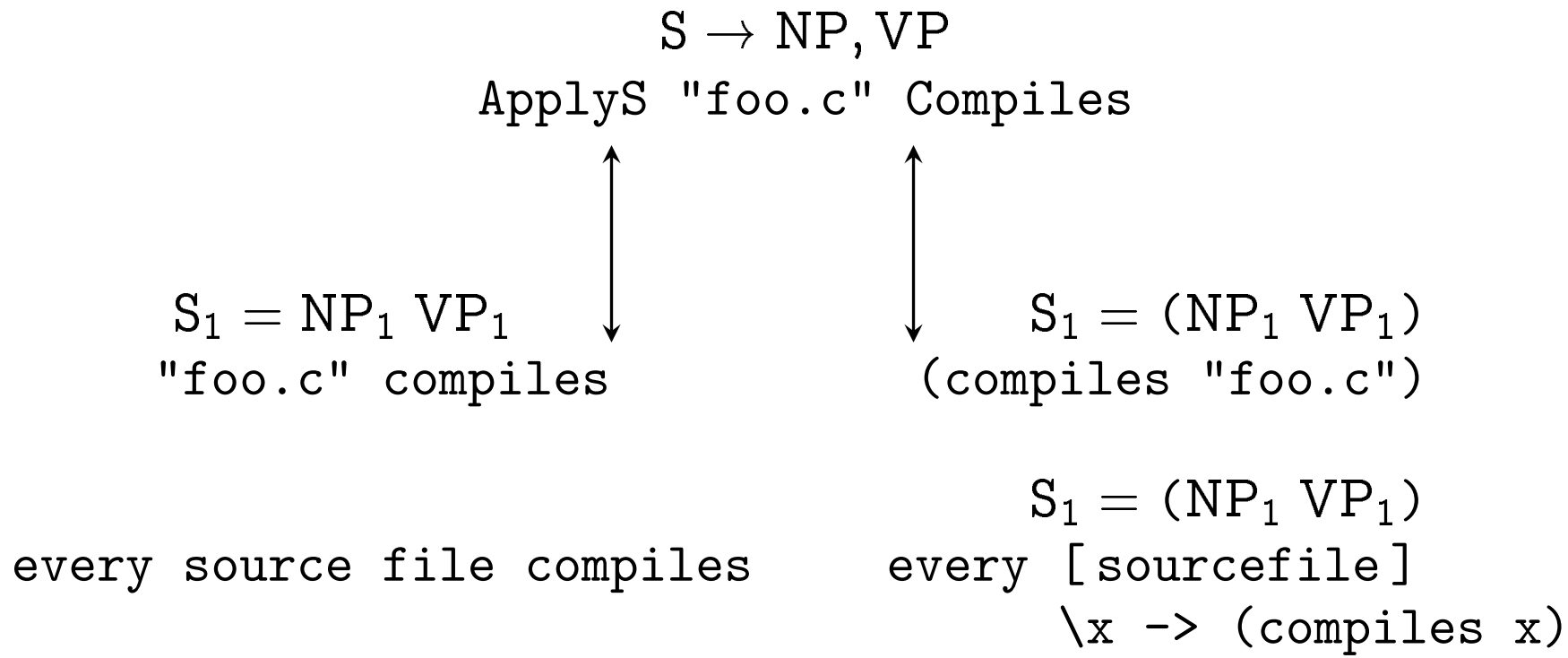


Grammatical Framework (Ranta) for English↔logic translation

▷ Which rule to use? ▷ How to normalize?

```
(\c -> every [sourcefile] \x -> (c x)          compiles)
```

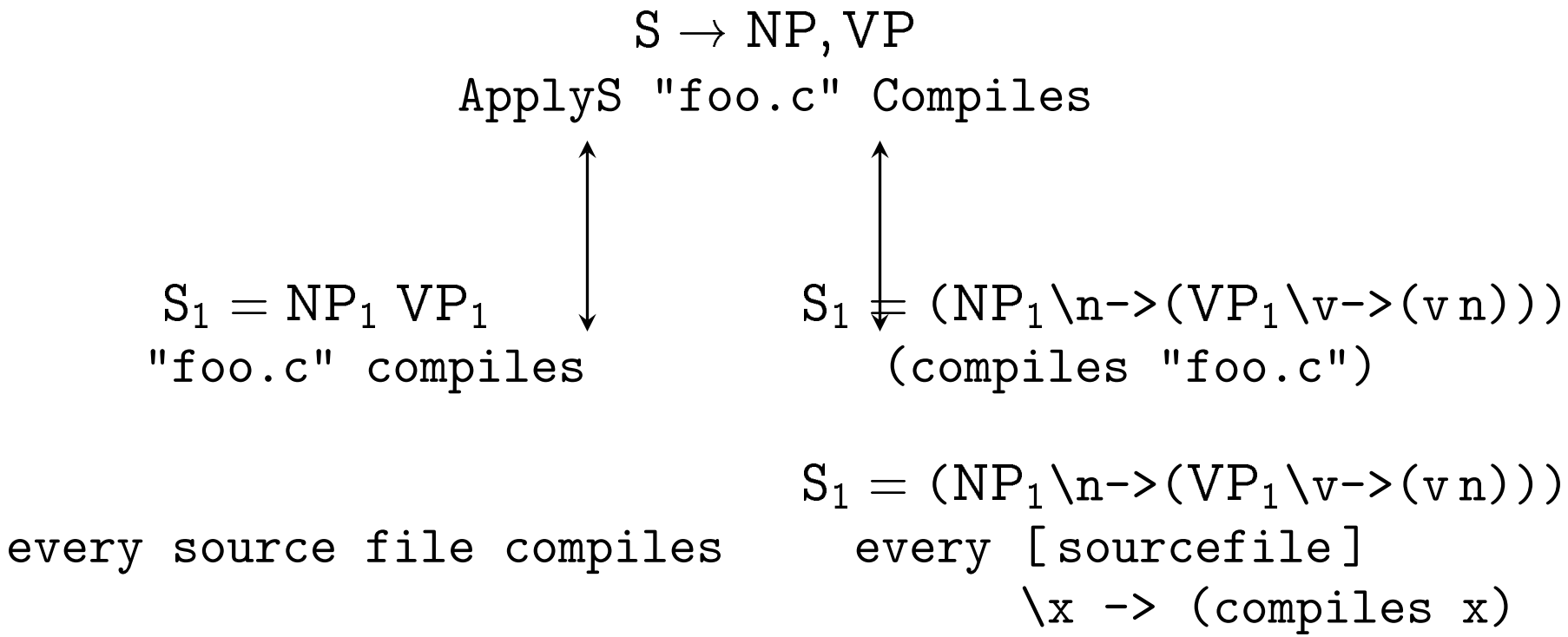
Lambda tricks for quantifiers



Grammatical Framework (Ranta) for English↔logic translation

```
(\c -> (c "foo.c")                                  compiles)
(\c -> every [sourcefile] \x -> (c x)               compiles)
```

Lambda tricks for quantifiers



Grammatical Framework (Ranta) for English↔logic translation

```
(\c -> (c "foo.c")                                  \c -> (c compiles))

(\c -> every [sourcefile] \x -> (c x)   \c -> (c compiles))
```

Continuations from programming languages systematize tricks.

Continuations without lambdas

Multiple Context-Free Grammar encodes `\c -> 1 (c 2) 3`

$NP_1 =$

$NP_2 =$ `"foo.c"`

$NP_3 =$

`\c -> (c "foo.c")`

$NP_1 =$ `every [sourcefile] \x ->`

$NP_2 =$ `x`

$NP_3 =$

```
\c -> every [sourcefile]
                \x -> (c x)
```

$S_1 = NP_1\ VP_1\ (VP_2\ NP_2)\ VP_3\ NP_3$ $S_1 =$ `(`NP_1`\n->(`VP_1`\v->(v n)))`

Continuations without lambdas

Multiple Context-Free Grammar encodes `\c -> 1 (c 2) 3`

$\mathrm{VP}_1 =$

$\mathrm{VP}_2 =$ `compiles`

$\mathrm{VP}_3 =$

`\c -> (c compiles)`

$\mathrm{NP}_1 =$ `every [sourcefile] \x ->`

$\mathrm{NP}_2 =$ `x`

$\mathrm{NP}_3 =$

```
\c -> every [sourcefile]
                \x -> (c x)
```

$\mathrm{S}_1 = \mathrm{NP}_1\ \mathrm{VP}_1\ (\mathrm{VP}_2\ \mathrm{NP}_2)\ \mathrm{VP}_3\ \mathrm{NP}_3$

$\mathrm{S}_1 = (\mathrm{NP}_1$`\n->(`$\mathrm{VP}_1$`\v->(v n)))`

Continuations without lambdas

Multiple Context-Free Grammar encodes `\c -> 1 (c 2) 3`

$\mathrm{VP}_1 =$

$\mathrm{VP}_2 =$ `compiles`

$\mathrm{VP}_3 =$

`\c -> (c compiles)`

$\mathrm{NP}_1 =$ `every [sourcefile] \x ->`

$\mathrm{NP}_2 =$ `x`

$\mathrm{NP}_3 =$

```
\c -> every [sourcefile]
                \x -> (c x)
```

$\mathrm{S}_1 = \mathrm{NP}_1\ \mathrm{VP}_1\ (\mathrm{VP}_2\ \mathrm{NP}_2)\ \mathrm{VP}_3\ \mathrm{NP}_3$ $\mathrm{S}_1 =$ `(`NP_1`\n->(`VP_1`\v->(v n)))`

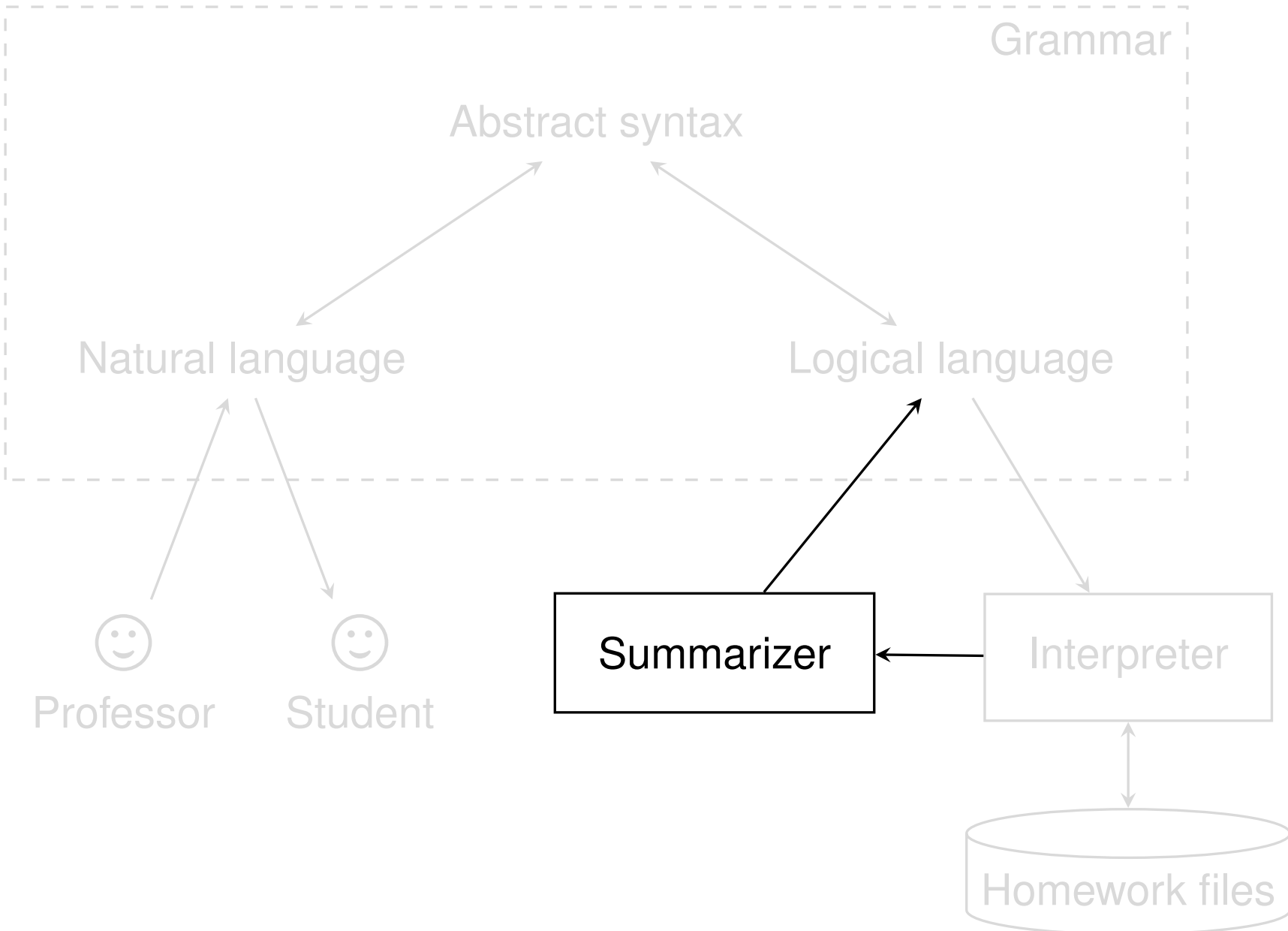
`every [sourcefile] \x -> (compiles x)`

Negation

- Every text file mentions a source file.
- Not every text file mentions a source file.
  A text file doesn't mention a source file.
  A text file mentions no source file.

- Lambda: negation using continuations
- MCFG: normal form is innermost 'not' in logic.
  Keep primal and dual versions of each formula.

Explanation summarization



Too verbose

Translate every node of accumulated evidence:

```
a source file doesn't compile
or every text file doesn't mention every source file, because
    a source file doesn't compile, because
            "nowork2.c" doesn't compile and
            "nowork.c" doesn't compile and
            "hello2.c" compiles and
            "hello.c" compiles and
    every text file doesn't mention every source file, because
            "nothing.txt" doesn't mention every source file, because
                    "nothing.txt" doesn't mention "nowork2.c" and
                    "nothing.txt" doesn't mention "nowork.c" and
                    "nothing.txt" doesn't mention "hello2.c" and
                    "nothing.txt" doesn't mention "hello.c" and
            "readme.txt" doesn't mention every source file, because
                    "readme.txt" doesn't mention "nowork2.c" and
                    "readme.txt" mentions "nowork.c" and
                    "readme.txt" doesn't mention "hello2.c" and
                    "readme.txt" mentions "hello.c"
```

From evidence trees to explanations

- Present evidence from models
  Don't describe inferences in proofs
- Present only *supporting* evidence

```
not every source file compiles, because
        "foo.c" compiles and ...          ??
```

  Select sentences with same truth value as parent
- Reintroduce generalized quantifiers (conjunctions) to group

```
"readme.txt" mentions "hello.c" and
 "readme.txt" mentions "nowork.c"

"readme.txt" mentions "hello.c" and "nowork.c"
```

From evidence trees to explanations

- Present evidence from models
  Don't describe inferences in proofs
- Present only *supporting* evidence

```
not every source file compiles, because
        "foo.c" compiles and ...          ??
```

  Select sentences with same truth value as parent
- Reintroduce generalized quantifiers (conjunctions) to group

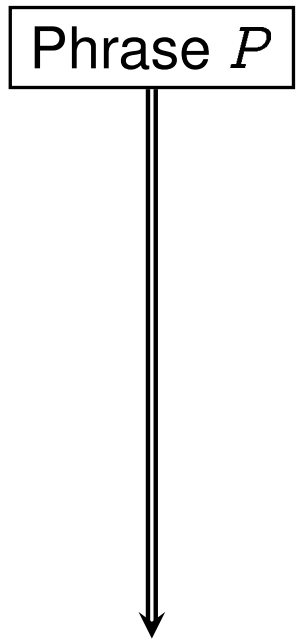
```
"readme.txt" mentions "hello.c" and
 "readme.txt" mentions "nowork.c"

"readme.txt" mentions "hello.c" and "nowork.c"
```

```
a source file doesn't compile, because
    "nowork2.c" and "nowork.c" don't compile, and
every text file doesn't mention every source file, because
    "nothing.txt" and "readme.txt" don't mention
    every source file
```

1. Summary

- Adaptable interpreter
- Quantification and negation in a bidirectional grammar
- Concise explanations

2. Modular meanings



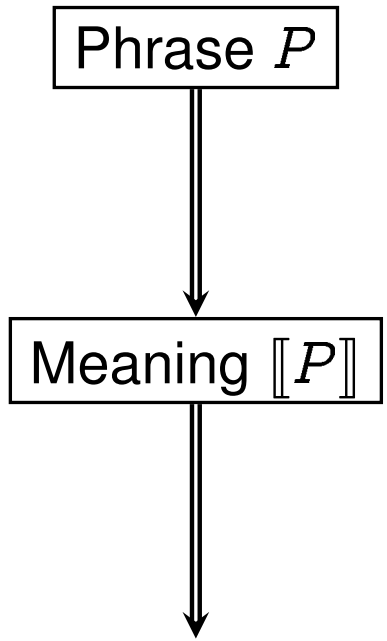
Is P consistent?

Is P positive or negative?

Does P entail that it is raining?

2. Modular meanings

Meaning: all your semantic needs in one place.



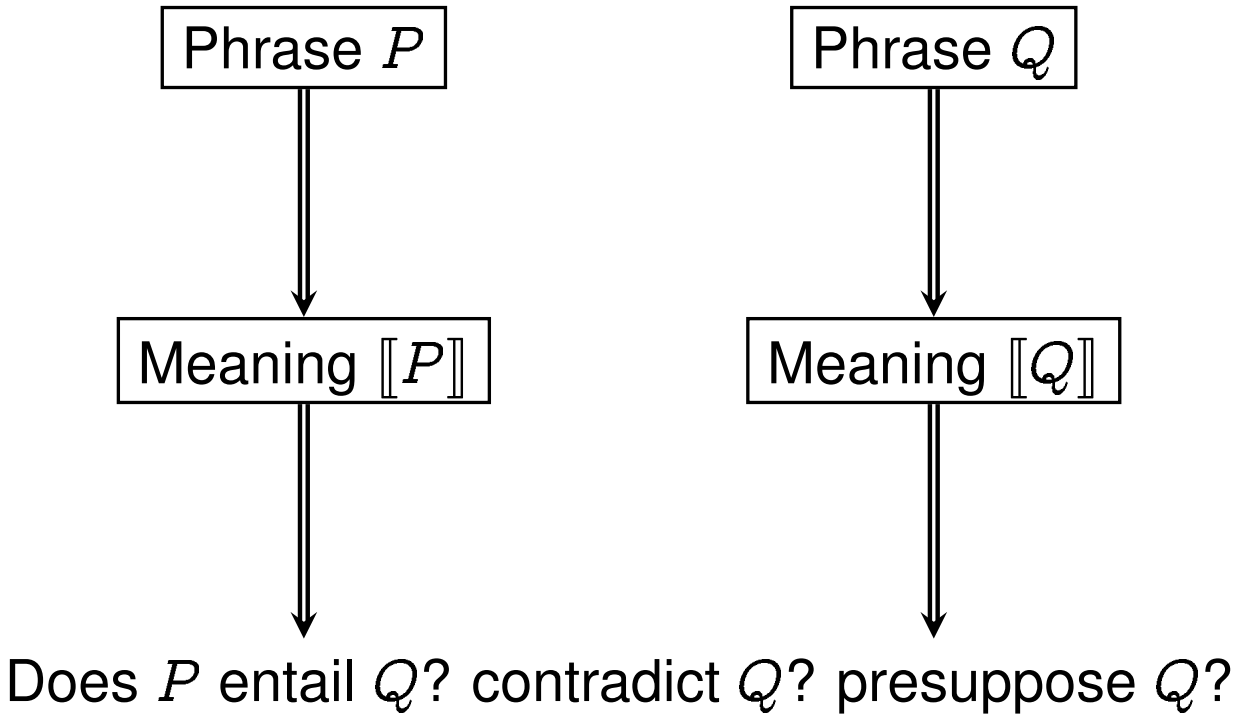
Is P consistent?

Is P positive or negative?

Does P entail that it is raining?

2. Modular meanings

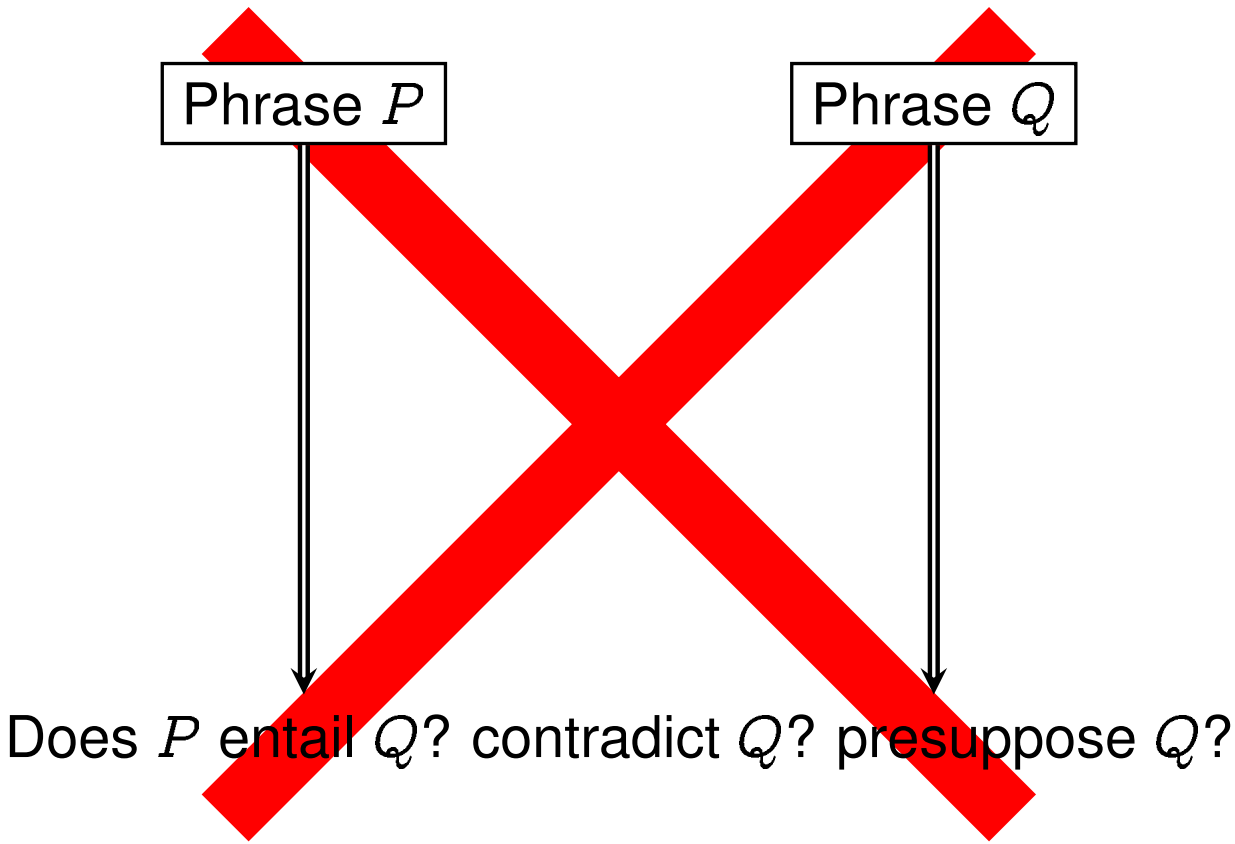
Meaning: all your semantic needs in one place.



Want: open, modular, irredundant semantic vocabulary.
Need: something latent; something factored. Sparsity looms.
Examples: Montague grammar, proof theory, topics and relations.

2. Modular meanings

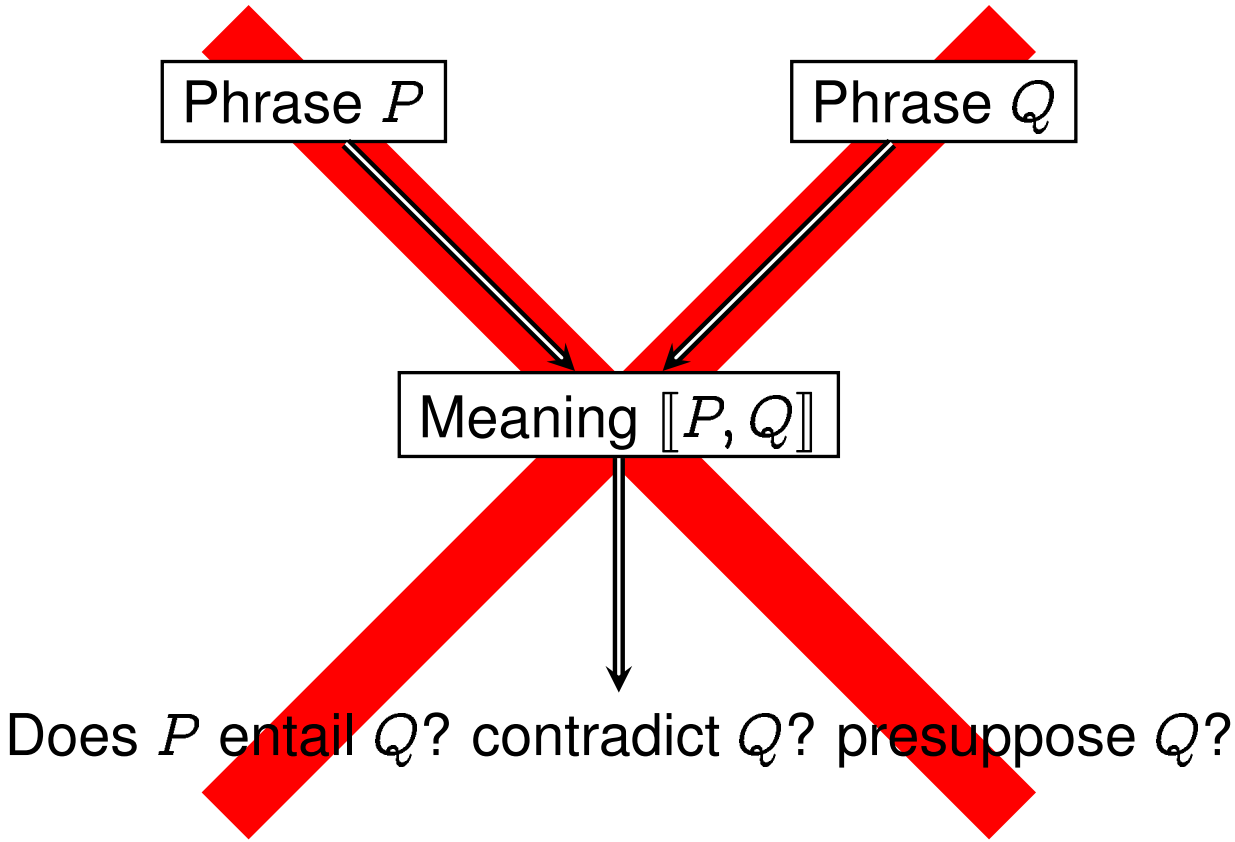
Meaning: all your semantic needs in one place.



Want: open, modular, irredundant semantic vocabulary.
Need: something latent; something factored. Sparsity looms.
Examples: Montague grammar, proof theory, topics and relations.

2. Modular meanings

Meaning: all your semantic needs in one place.



Want: open, modular, irredundant semantic vocabulary.
Need: something latent; something factored. Sparsity looms.
Examples: Montague grammar, proof theory, topics and relations.

Topics as possible worlds projected?

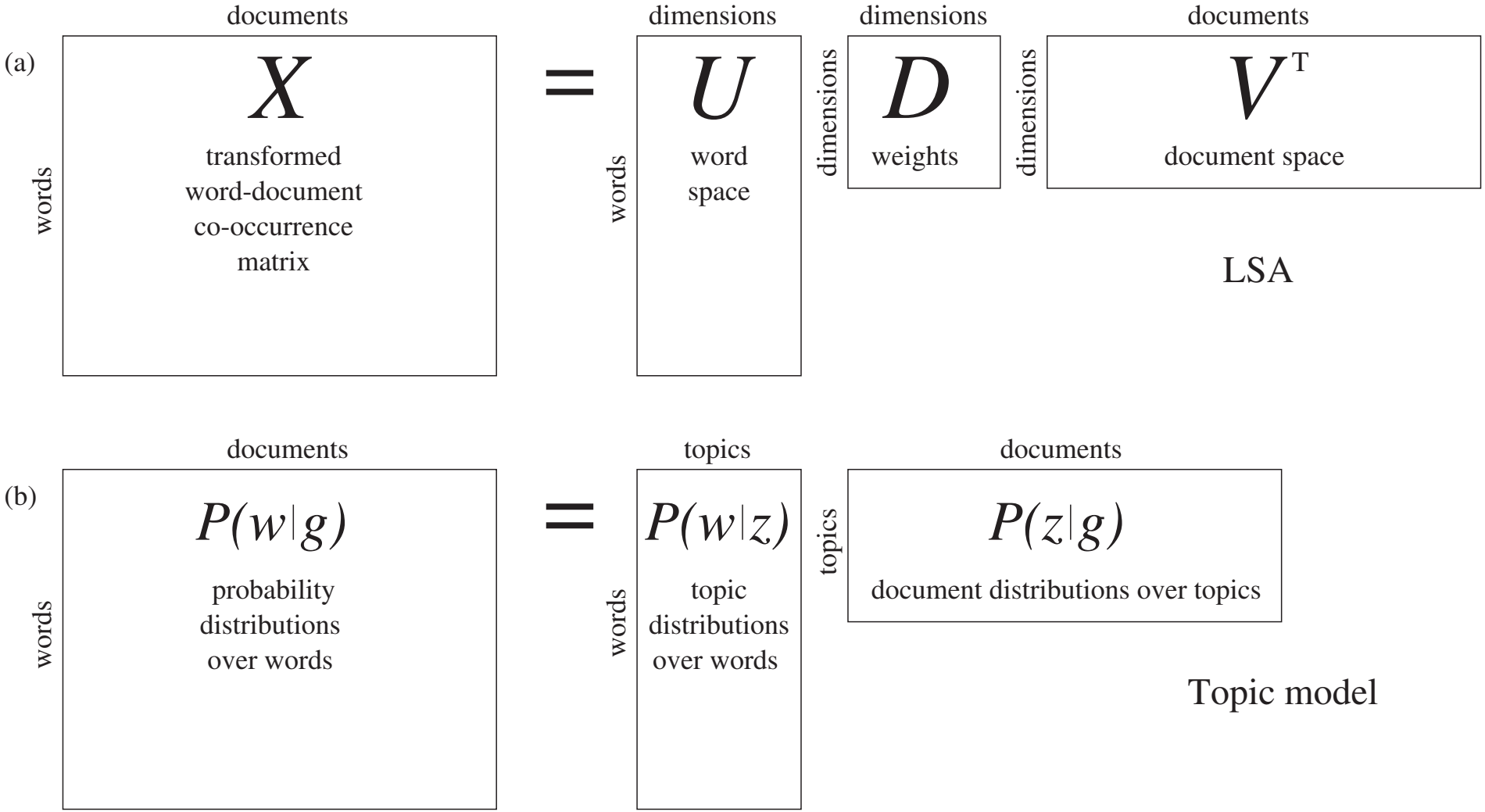


Figure 3. (a) Latent semantic analysis (LSA) performs dimensionality reduction using the singular value decomposition. The transformed word–document co-occurrence matrix, X, is factorized into three smaller

Distributional entailment

Bernardi, Baroni, Ngoc, Shan (work in progress)

- Adjectives: $A\ N < N$
- Quantifiers: $Q1\ N < Q2\ N$
  $Q\ N1 < Q\ N2$

(Avoid consulting corpus for each pair.)

Adjectives

regular-j_train-n	train-n
several-j_bus-n	bus-n
Italian-j_motorcycle-n	motorcycle-n
Scottish-j_hospital-n	hospital-n
black-j_robe-n	robe-n
modern-j_hotel-n	hotel-n
first-j_violin-n	violin-n
international-j_phone-n	phone-n
good-j_dress-n	dress-n
internal-j_phone-n	phone-n
public-j_radio-n	radio-n
own-j_horse-n	horse-n
young-j_gun-n	gun-n
rural-j_bus-n	bus-n
friendly-j_pub-n	pub-n
fine-j_car-n	car-n
several-j_coat-n	coat-n

Adjectives results

	Train / Test		SVM (polynomial)	balAPinc (Kotlerman&al.)	Frequency baseline	Constant baseline
+	1246 restrictive	$A\ N < N$				
−	1244 checked random	$A\ N1 \not< N2$				
+	1385 WordNet	$N1 < N2$	71%	70%	53%	50%
−	461 WordNet	$N1 > N2$				
−	924 checked random	$N1 \not< N2$				

- Almost free training data from restrictive adjectives and quantifier entailments
- Distributional meanings of composite phrases
- Want to derive meaning representations compositionally

Quantifiers

many-j	several-j	YES	many-j	most-j	NO
many-j	some-d	YES	many-j	no-d	NO
each-d	some-d	YES	both-d	many-j	NO
most-j	many-j	YES	both-d	most-j	NO
much-j	some-d	YES	both-d	several-j	NO
every-d	many-j	YES	either-d	both-d	NO
all-d	many-j	YES	many-j	all-d	NO
all-d	most-j	YES	many-j	every-d	NO
all-d	several-j	YES	few-j	many-j	NO
all-d	some-d	YES	few-j	all-d	NO
both-d	either-d	YES	several-j	all-d	NO
both-d	some-d	YES	some-d	many-j	NO
several-j	some-d	YES	some-d	all-d	NO
			some-d	each-d	NO
			some-d	every-d	NO
			several-j	every-d	NO
			several-j	few-j	NO

Quantifiers results

Train / Test		SVM (polynomial)	balAPinc (Kotlerman&al.)	Frequency baseline	Constant baseline
+ 5806 manual non-'several'	$Q1\ N < Q2\ N$				
− 6946 manual non-'several'	$Q1\ N \not< Q2\ N$				
−12752 random non-'several'	$Q1\ N1 \not< Q2\ N2$				
+ 1731 manual 'several'	$Q1\ N < Q2\ N$	86%	73%	45%	73%
− 1509 manual 'several'	$Q1\ N \not< Q2\ N$				
− 3240 random 'several'	$Q1\ N1 \not< Q2\ N2$				
+ 7537 manual	$Q1\ N < Q2\ N$	90%	76%	50%	76%
− 8455 manual	$Q1\ N \not< Q2\ N$				
−15992 random	$Q1\ N1 \not< Q2\ N2$				

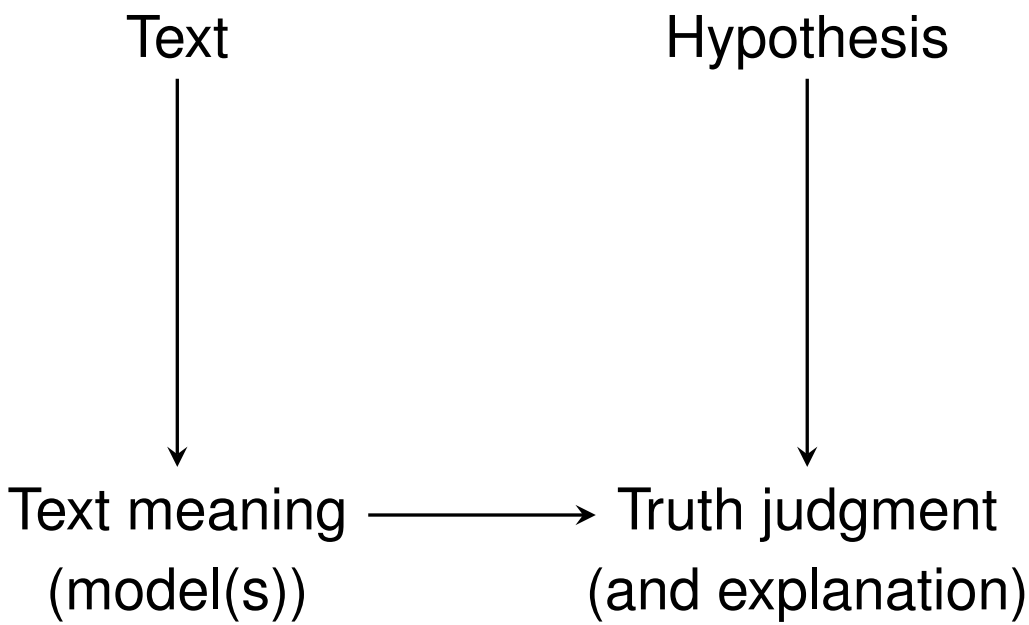
(same $25504/6480$ split)

Quantifier monotonicity

- Upward: `some-d several-j these-d those-d`
- Downward: `few-j all-d no-d every-d`

	Train / Test		SVM (polynomial)	Constant baseline
$+$	6076 manual + WordNet	$Q\ N1 < Q\ N2$	67%	50%
$-$	6076 random	$Q\ N1 \not< Q\ N2$		

(66%/34% split)


Text
Hypothesis
Text meaning
(model(s))
Truth judgment
(and explanation)