GLIF: A Declarative Framework for Symbolic Natural Language Understanding

Jan Frederik Schaefer Michael Kohlhase

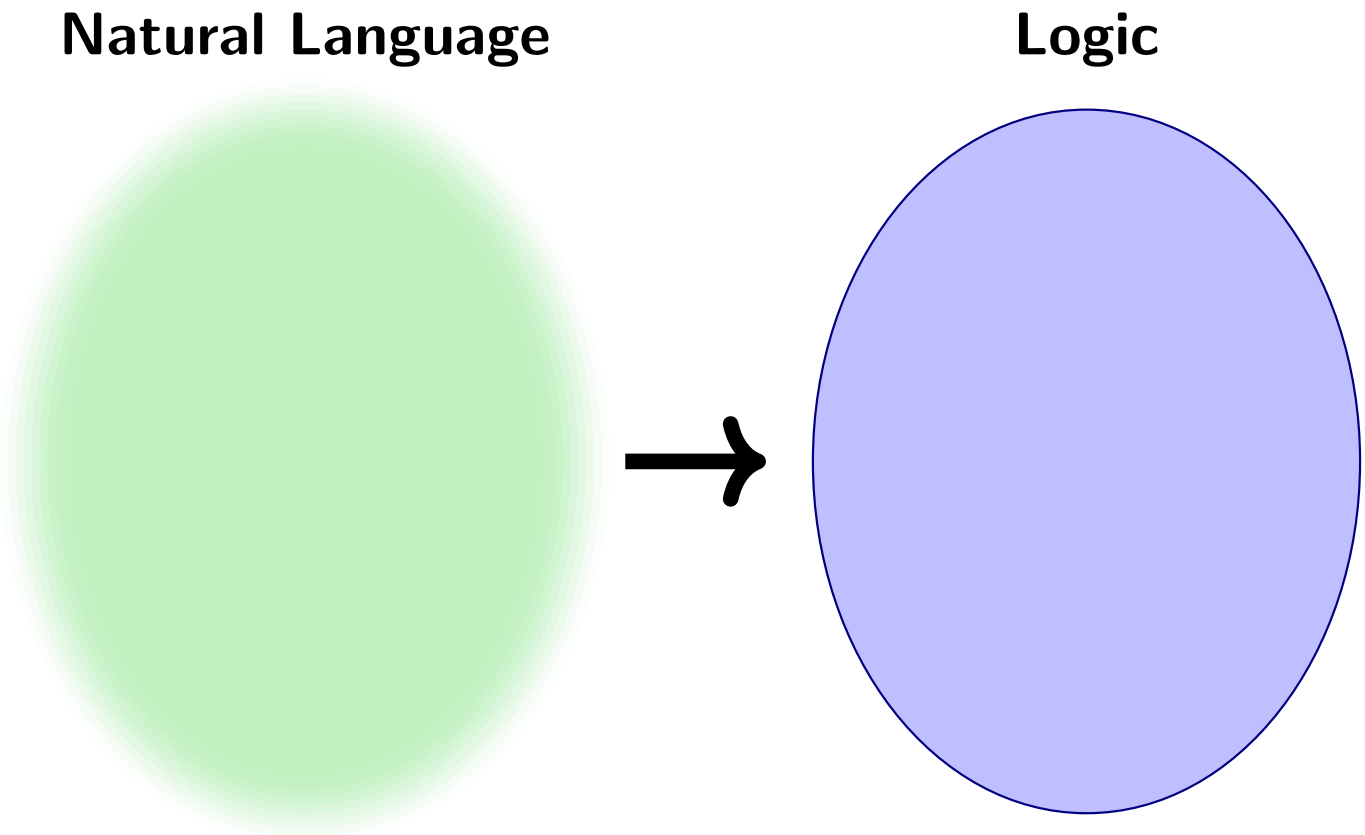
FAU Erlangen-Nürnberg

FCR 2020

virtual event due to COVID-19

September 22, 2020

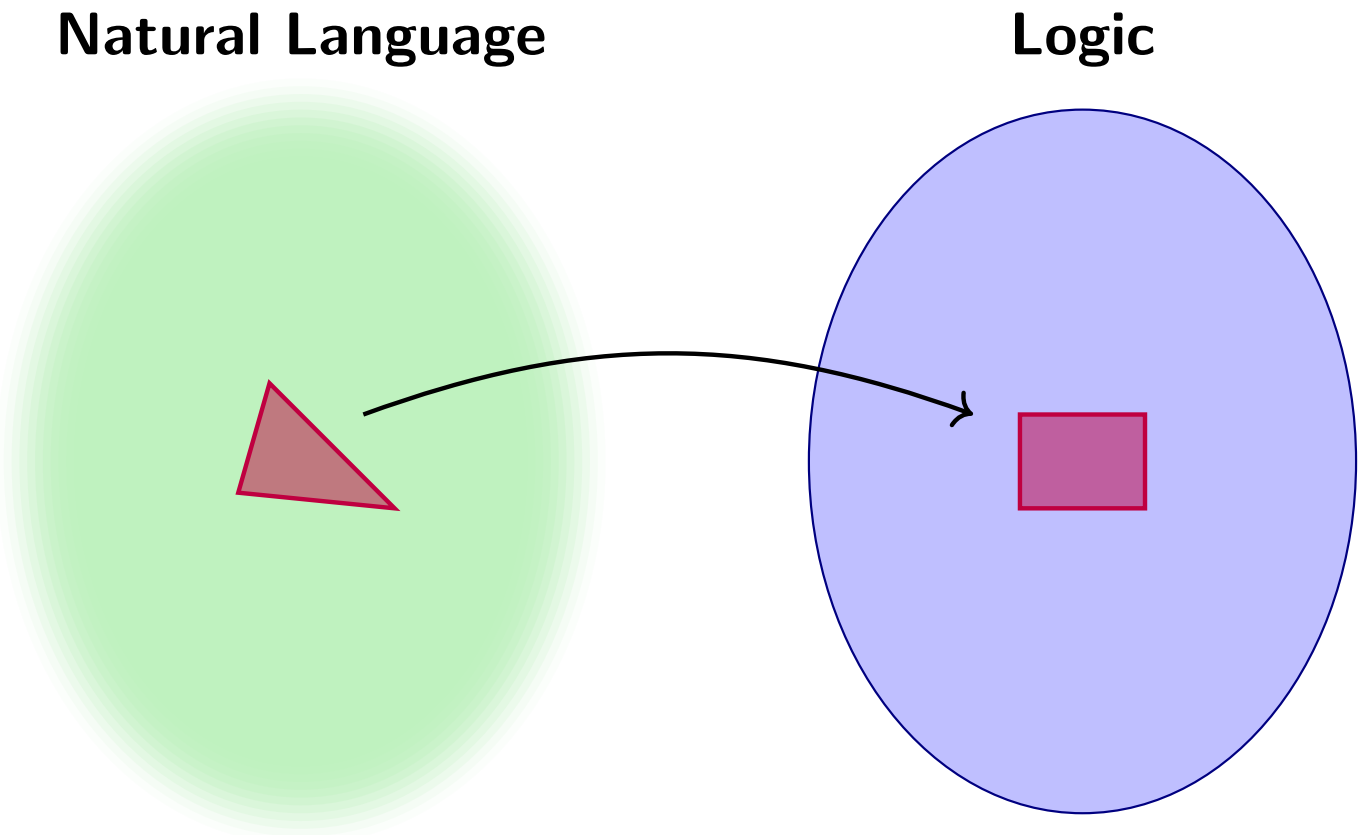
Method of Fragments



How do we get from messy language to formal logic?

Montague [Mon70]: Look at a "nice" subset and map into logic.

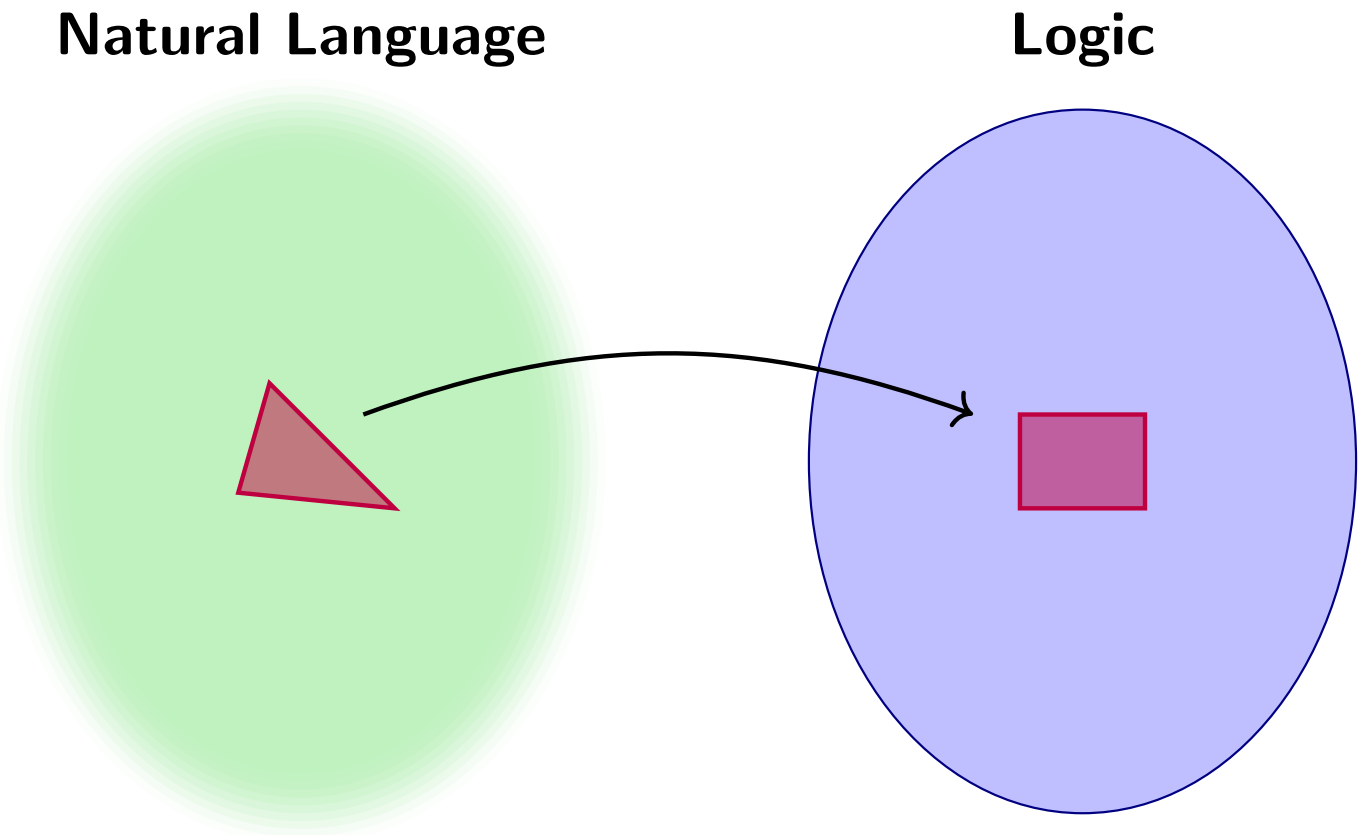
Method of Fragments



How do we get from messy language to formal logic?

Montague [Mon70]: Look at a "nice" subset and map into logic.

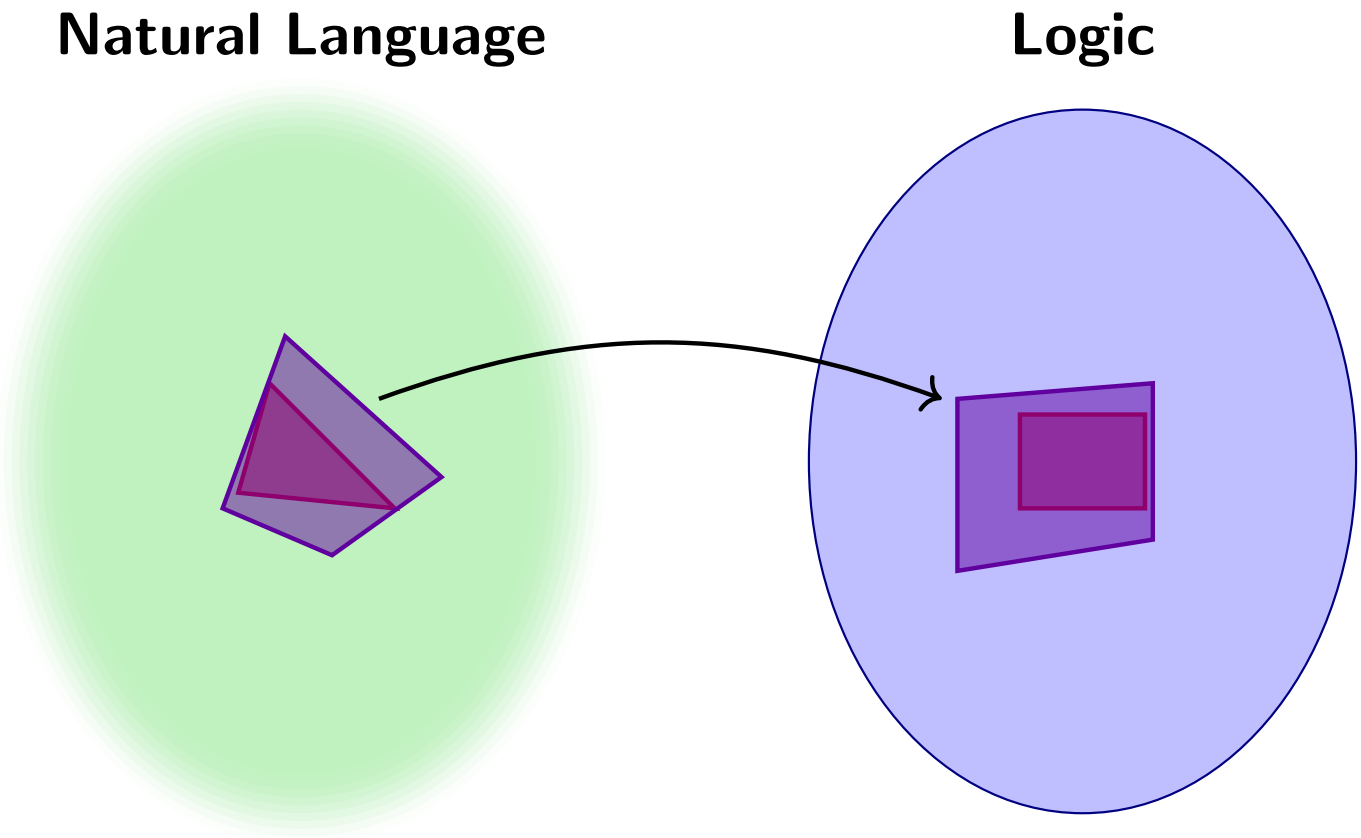
Method of Fragments



"Ahmed paints and Berta is quiet." $p(a) \wedge q(b)$

"Ahmed doesn't paint." $\neg p(a)$

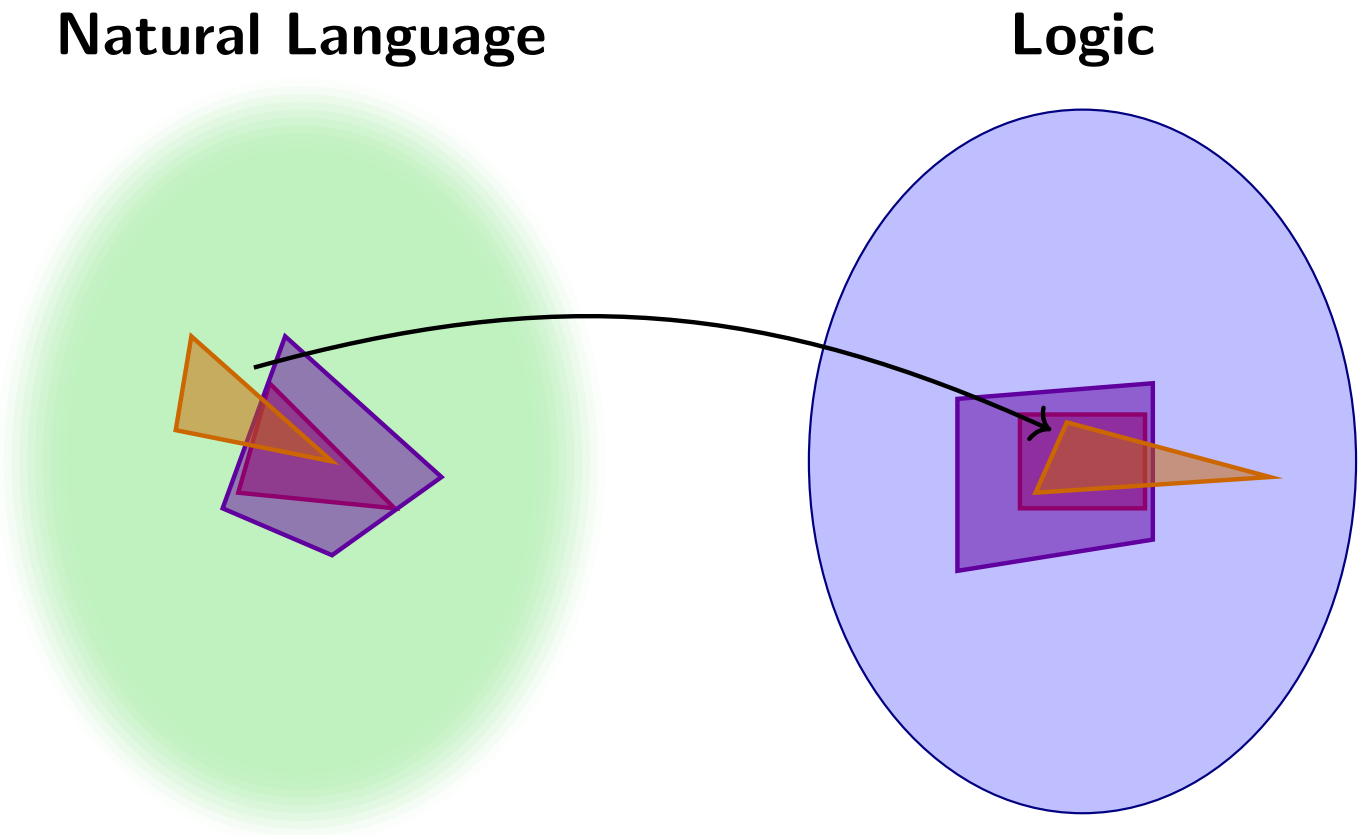
Method of Fragments



"Every student paints and is quiet." $\forall x.s(x) \Rightarrow (p(x) \wedge q(x))$

"Nobody paints." $\neg\exists x.p(x)$

Method of Fragments



"Ahmed isn't allowed to paint." $\neg\Diamond p(a)$

"Ahmed and Berta must paint." $(\Box p(a)) \wedge \Box p(b)$

Method of Fragments

If we only hand-wave, we gloss over problems:

"Ahmed paints. He is quiet." $\overset{?}{\rightsquigarrow}$ $p(a) \wedge q(a)$

Specify:

- Grammar — *fixes NL subset*
- Target logic
- Semantics construction — *maps parse trees to logic*

On paper [Mon74]: — *difficult to scale*

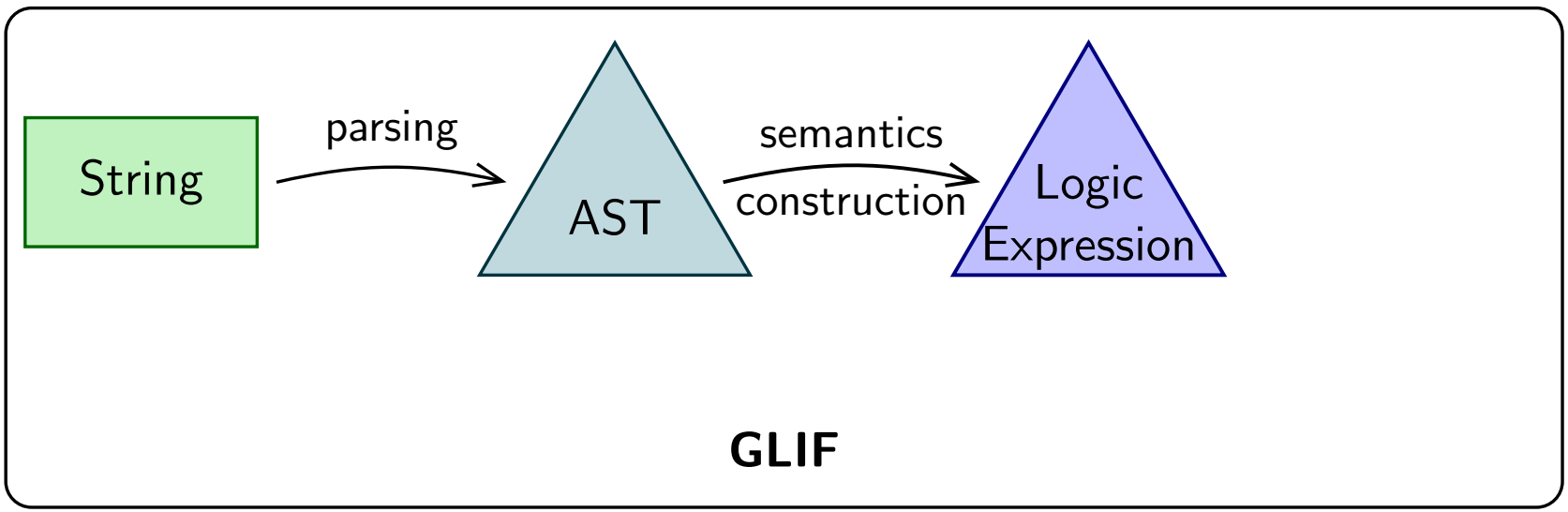
T11. If $\phi, \psi \in \mathrm{P_t}$ and ϕ, ψ translate into ϕ', ψ' respectively, then ϕ **and** ψ translates into $[\phi \wedge \psi]$, ϕ **or** ψ translates into $[\phi \vee \psi]$.

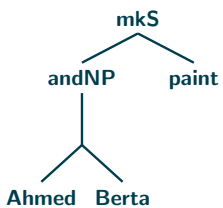
T12. If $\gamma, \delta \in \mathrm{P_{IV}}$ and γ, δ translate into γ', δ' respectively, then γ **and** δ translates into $\hat{x}[\gamma'(x) \wedge \delta'(x)]$, γ **or** δ translates into $\hat{x}[\gamma'(x) \vee \delta'(x)]$.

T13. If $\alpha, \beta \in \mathrm{P_T}$ and α, β translate into α', β' respectively, then α **or** β translates into $\hat{\mathrm{P}}[\alpha'(\mathrm{P}) \vee \beta'(\mathrm{P})]$.

GLIF: Grammatical Logical Inference Framework

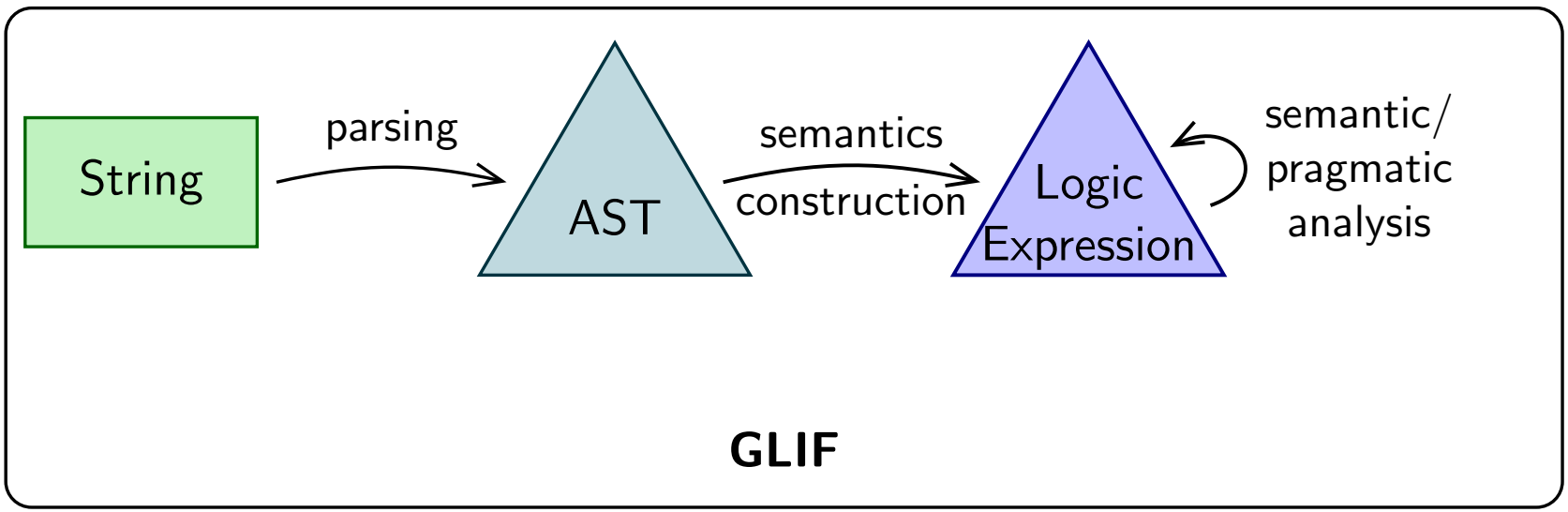
We have a tool for this!

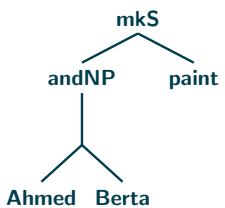


"Ahmed and Berta paint." →  → $p(a) \wedge p(b)$

GLIF: Grammatical Logical Inference Framework

We have a tool for this!

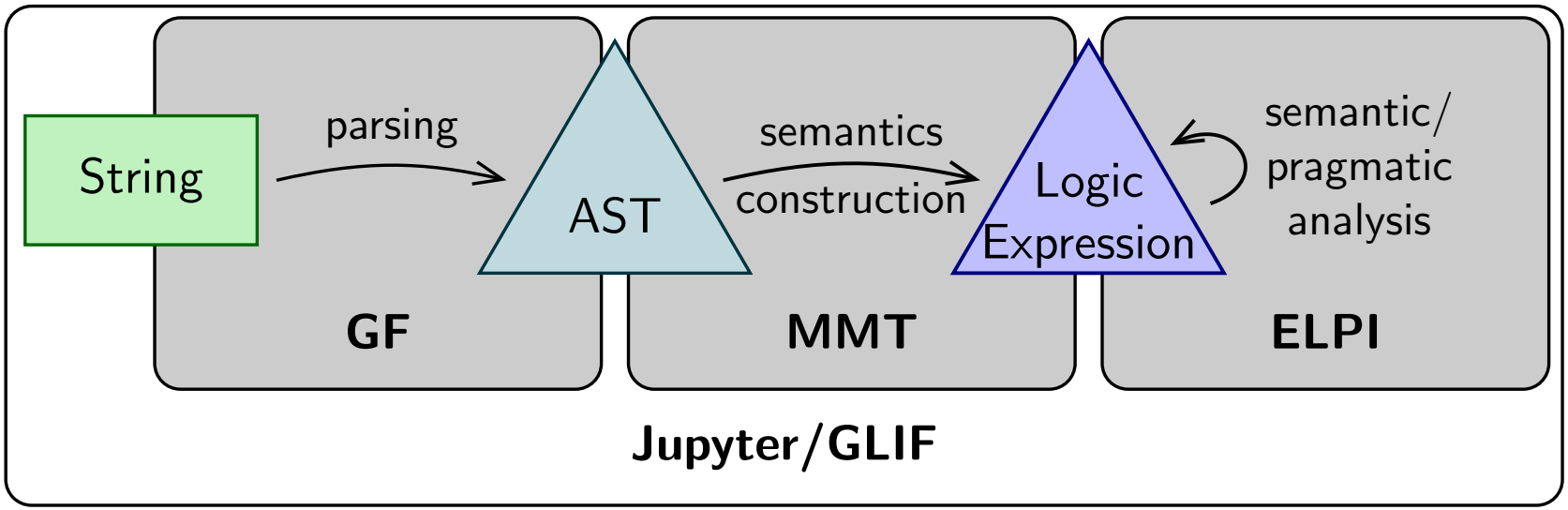


"Ahmed and Berta paint." → 

→ $p(a) \wedge p(b)$

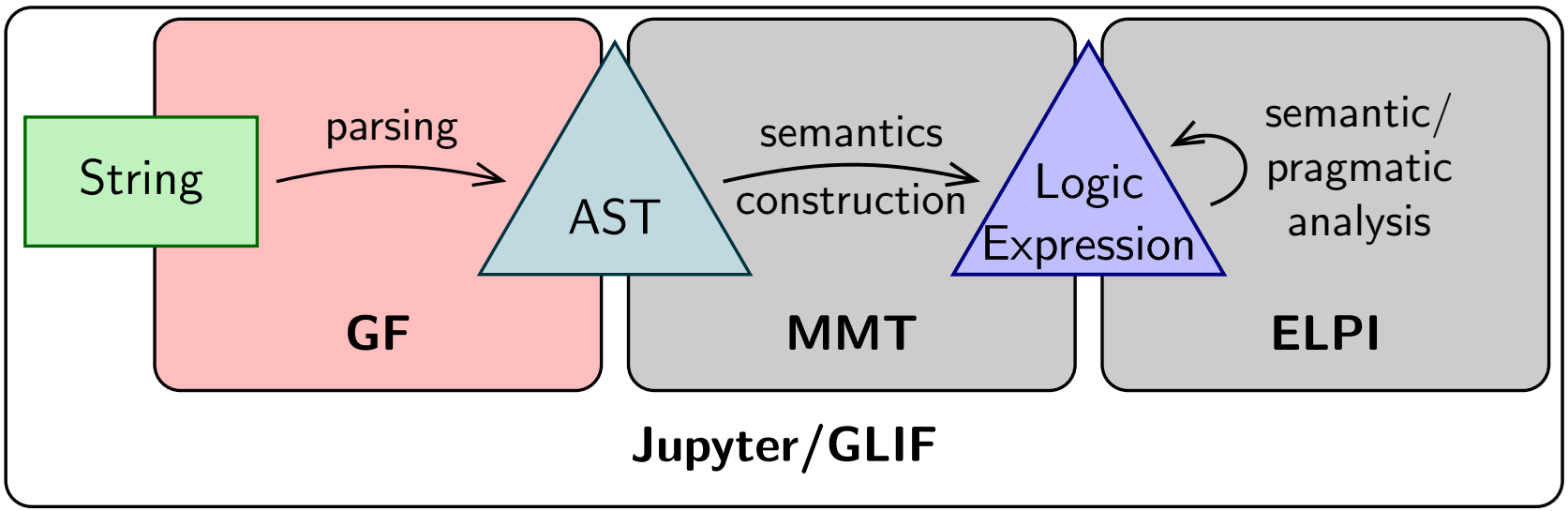
GLIF: Grammatical Logical Inference Framework

It combines existing tools.



GF	(= **grammar** framework)
+ **MMT**	(= **logic** framework)
+ **ELPI**	(= **inference** framework)
= **GLIF**	(= **natural language understanding** framework)

Components of GLIF: GF

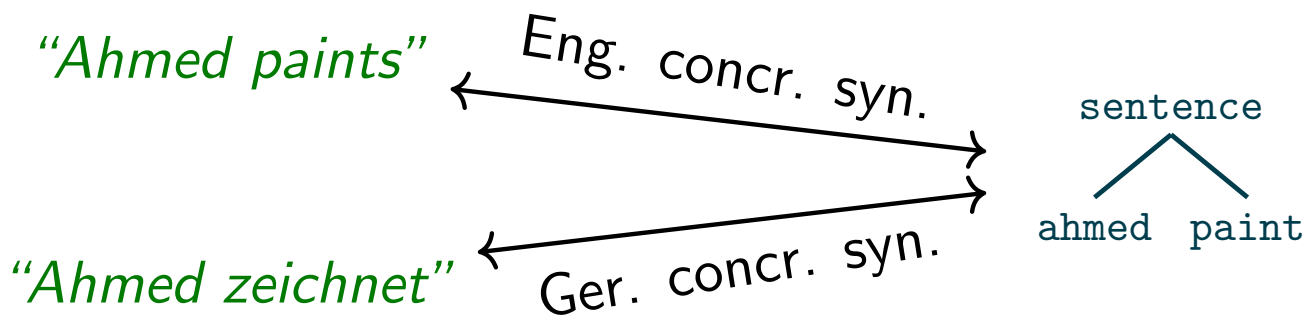


Components of GLIF: Grammatical Framework [GF]

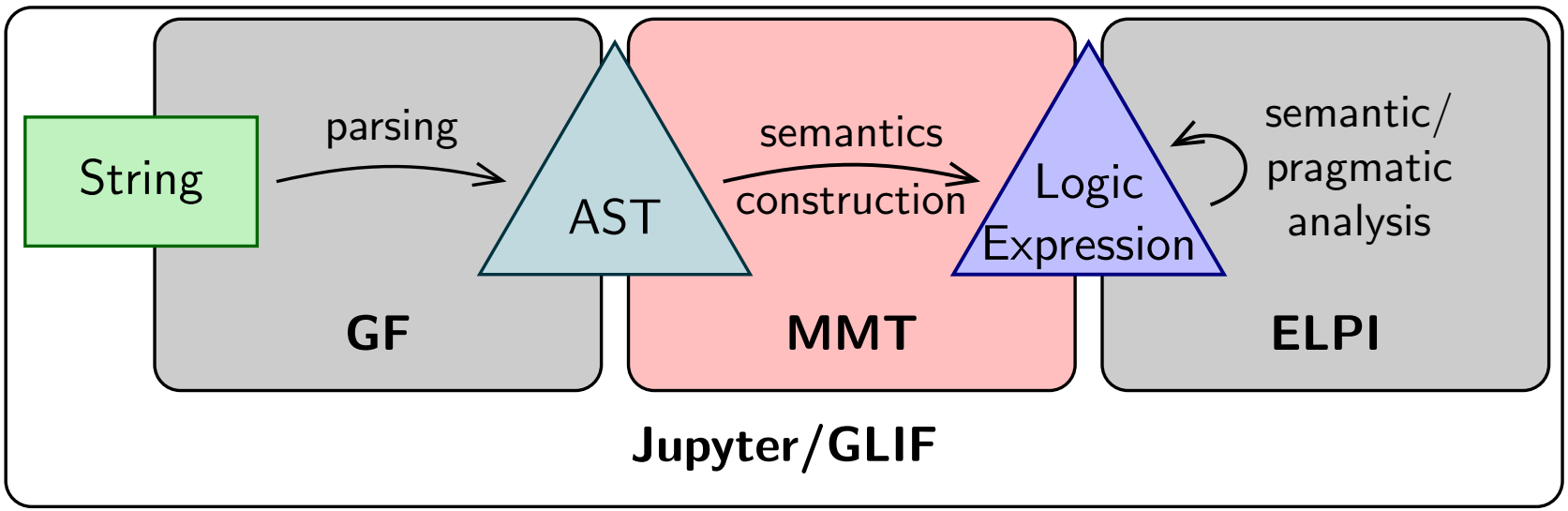
- Specialized for developing natural language grammars
- Separates abstract and concrete syntax

```
sentence : NP -> VP -> S;              --abstract
sentence np vp = np.s ++ vp.s!np.n;    --concrete
```

- Abstract syntax based on type theory
- Comes with large library $\geq$ 36 *languages*



Components of GLIF: MMT



Components of GLIF: MMT

- Modular logic development and knowledge representation
- Not specialized in one logical framework *we use LF*
- We will use MMT to:
  - represent abstract syntax
  - specify target logic and domain theory
  - specify semantics construction

Components of GLIF: MMT

- Modular logic development and knowledge representation
- Not specialized in one logical framework *we use LF*
- We will use MMT to:
  - **represent abstract syntax**
  - specify target logic and domain theory
  - specify semantics construction

GF

```
cat
  NP; VP; S;
fun
  sentence :
    NP -> VP -> S;
```

$\longmapsto$

MMT

```
NP : type
VP : type
S  : type
sentence :
  NP → VP → S
```

Components of GLIF: MMT

- Modular logic development and knowledge representation
- Not specialized in one logical framework *we use LF*
- We will use MMT to:
  - represent abstract syntax
  - **specify target logic and domain theory**
  - specify semantics construction

Logic Syntax

```
o : type //propositions
¬ : o → o
∧ : o → o → o
∨ : o → o → o

ι : type //individuals
∀ : (ι → o) → o
∃ : (ι → o) → o
```

Domain Theory

```
paint : ι → o
quiet : ι → o
ahmed : ι
berta : ι
```

idea: $\forall f$ or $\forall \lambda x.f(x)$
instead of $\forall x.f(x)$

Components of GLIF: MMT

- Modular logic development and knowledge representation
- Not specialized in one logical framework *we use LF*
- We will use MMT to:
  - represent abstract syntax
  - specify target logic and domain theory
  - **specify semantics construction**

Semantics Construction

map symbols in abstract syntax to terms in logic/domain theory

Simple setting

```
S        ↦ o
NP       ↦ ι
VP       ↦ ι → o
sentence ↦ λn.λv.v n
ahmed    ↦ ahmed
```

More advanced

```
NP       ↦ (ι → o) → o
sentence ↦ λn.λv.n v
everyone ↦ λp.∀λx.p x
berta    ↦ λp.p berta
```

Example: Parsing + Semantics Construction

"Ahmed and Berta paint"

↓parsing

sentence (andNP ahmed berta) paint

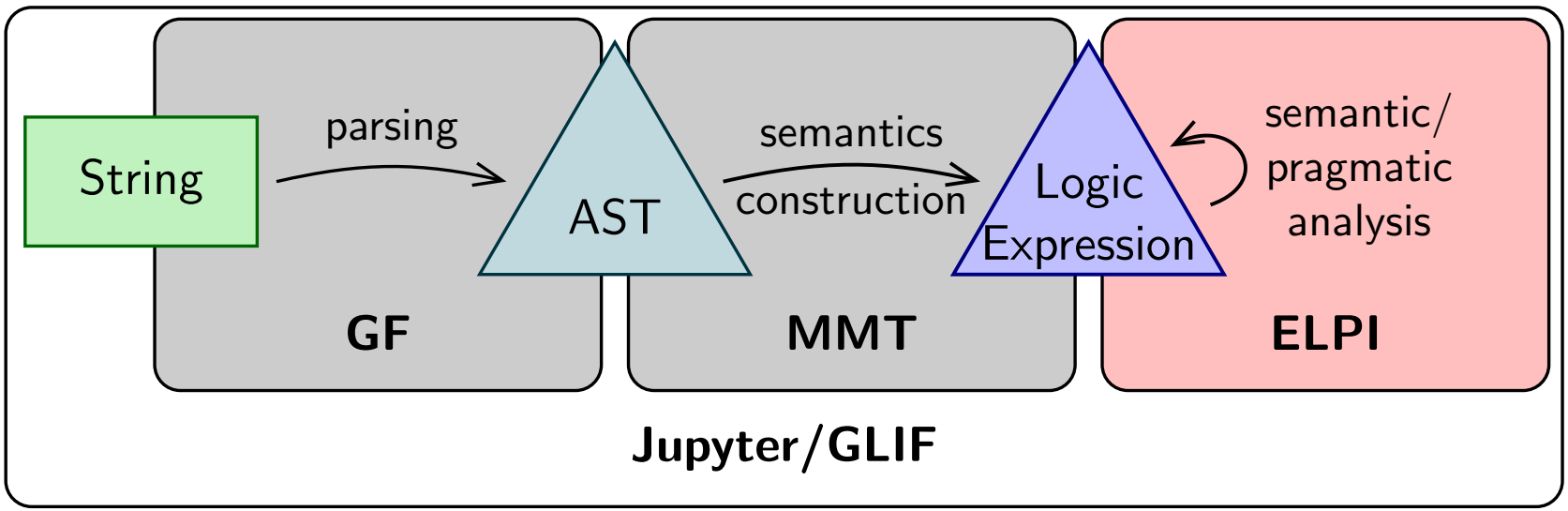
↓semantics construction

`(λn.λv.n v) ((λa.λb.λp.a p ∧ b p) (λp.p ahmed) (λp.p berta)) paint`

↓β-reduction

`paint ahmed ∧ paint berta`

Components of GLIF: ELPI



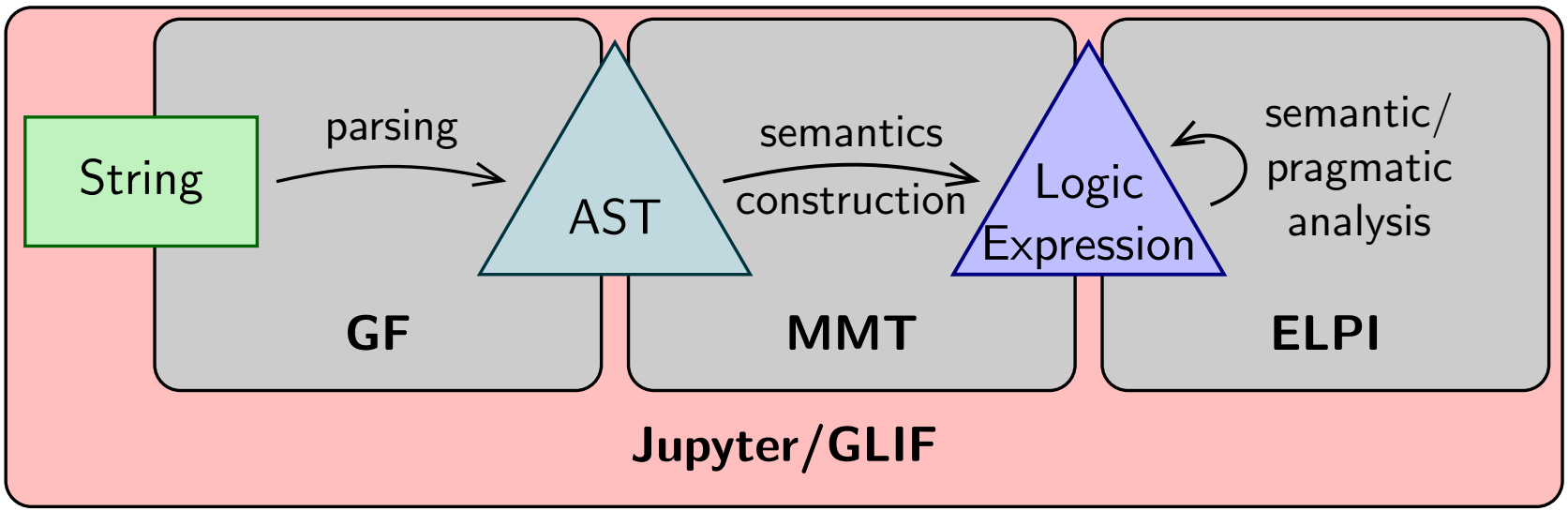
Components of GLIF: ELPI

- Implementation and extension of λProlog ≈ *Prolog + HOAS*
- MMT can generate logic signatures
- First experiments with prover generation
- Generic inference/reasoning step after semantics construction

```
kind o type.
type not o -> o.
type and o -> o -> o.

kind i type.
type forall (i -> o) -> o.
```

Components of GLIF: Jupyter



Components of GLIF: Jupyter

- Unified, notebook-based interface
- Supports implementation and testing
- Useful for prototype, demos, teaching, ...

```
    sentence = [n,v] n v |
    andNP = [a,b] [p] (a p) ∧ (b p) |
    paint = paint |
    ahmed = [p] p ahmed |
    berta = [p] p berta |
|
```

```
Created view GrammarSemantics
```

```
parse "Ahmed and Berta paint" | construct
```

```
(paint ahmed)∧(paint berta)
```

Example: Epistemic Q&A

John knows that Mary or Eve knows that Ping has a dog. (S_1)
Mary doesn't know if Ping has a dog. (S_2)
Does Eve know if Ping has a dog? (Q)

$$S_1 = \Box_{john}(\Box_{mary} hd(ping)) \vee \Box_{eve} hd(ping)$$
$$S_2 = \neg((\Box_{mary} hd(ping)) \vee \Box_{mary} \neg hd(ping))$$
$$Q = (\Box_{eve} hd(ping)) \vee \Box_{eve} \neg hd(ping)$$

$$\begin{array}{lll} S_1, S_2 \vdash_{S5_n} Q & \rightsquigarrow & \text{yes} \\ S_1, S_2 \vdash_{S5_n} \neg Q & \rightsquigarrow & \text{no} \\ \text{else} & \rightsquigarrow & \text{maybe} \end{array}$$

Example: Controlled Natural Languages

- Formal languages
- that are a subset of natural language
- and have fixed semantics *formal verification, . . .*

"S is a subset of every set iff S is empty"

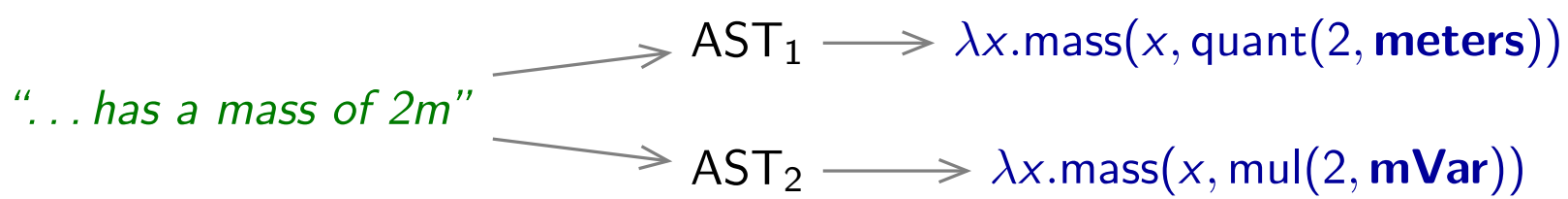
$\rightsquigarrow (\forall V_{new}.set(V_{new}) \Rightarrow subset(V_S, V_{new})) \Leftrightarrow empty(V_S)$

Example: Controlled Natural Languages

- Formal languages
- that are a subset of natural language
- and have fixed semantics *formal verification, . . .*

"S is a subset of every set iff S is empty"
$\rightsquigarrow (\forall V_{new}.set(V_{new}) \Rightarrow subset(V_S, V_{new})) \Leftrightarrow empty(V_S)$

Use inference for disambiguation:

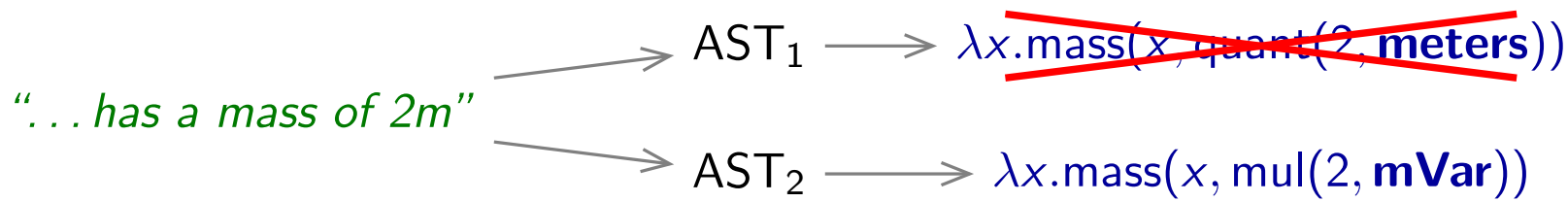


Example: Controlled Natural Languages

- Formal languages
- that are a subset of natural language
- and have fixed semantics *formal verification, . . .*

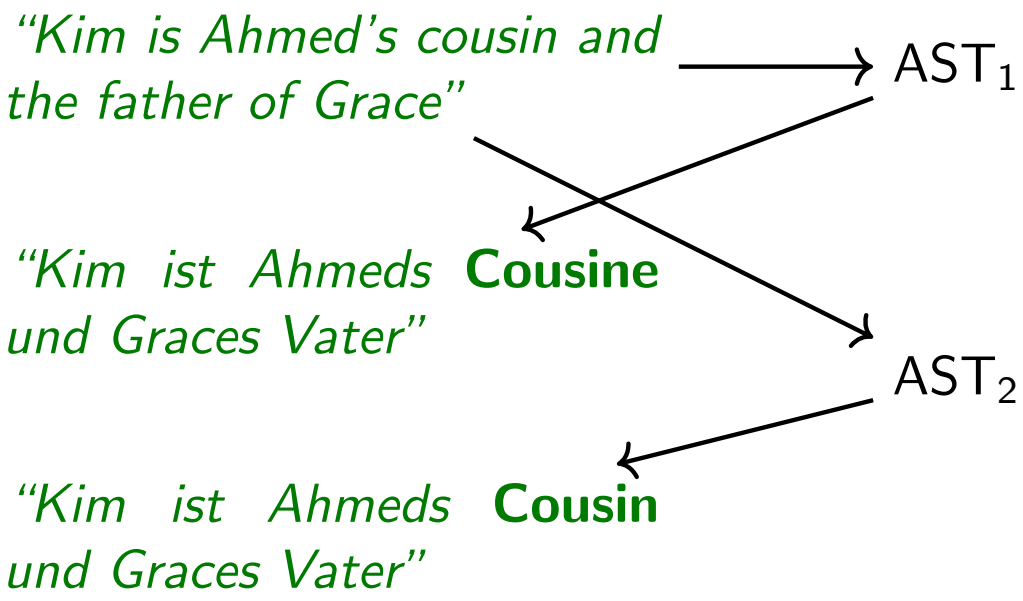
"S is a subset of every set iff S is empty"
$\rightsquigarrow (\forall V_{new}.set(V_{new}) \Rightarrow subset(V_S, V_{new})) \Leftrightarrow empty(V_S)$

Use inference for disambiguation:



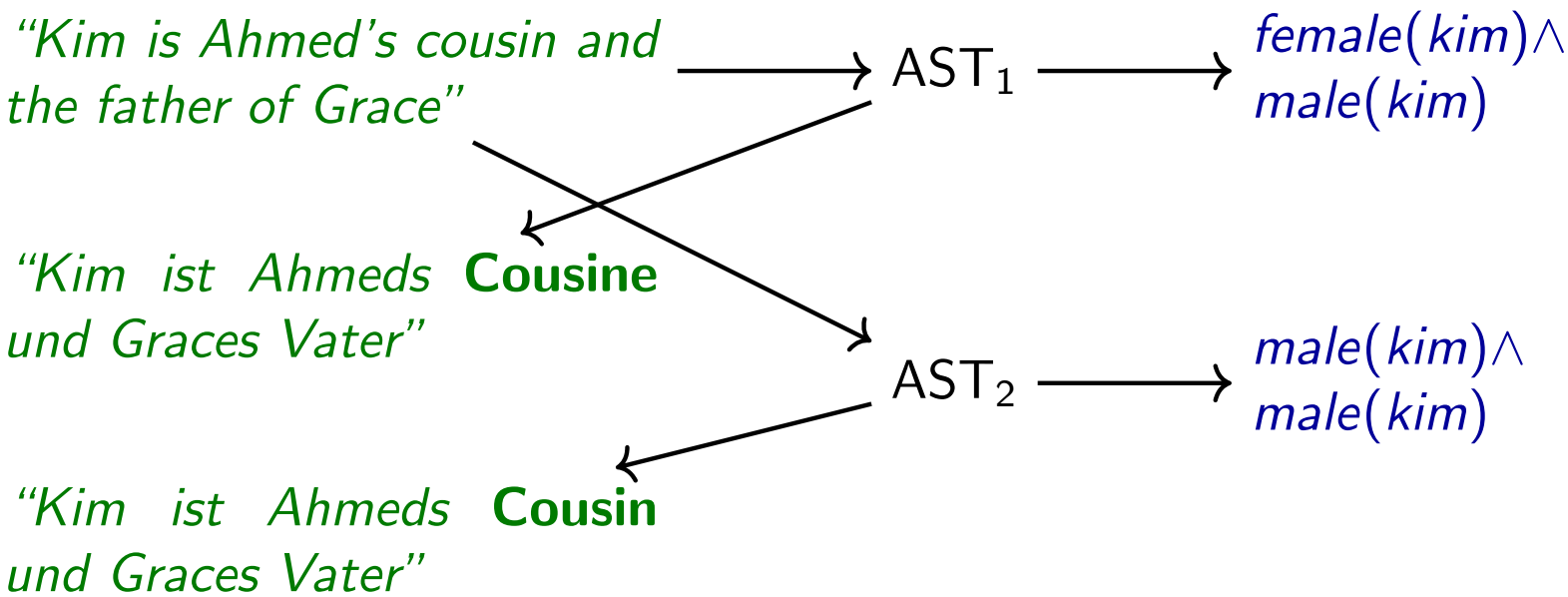
Example: Translation

- Two German words for *"cousin"*, depending on the gender
- Two entries in abstract syntax: `cousin_female` and `cousin_male`
- Use inference to discard ASTs



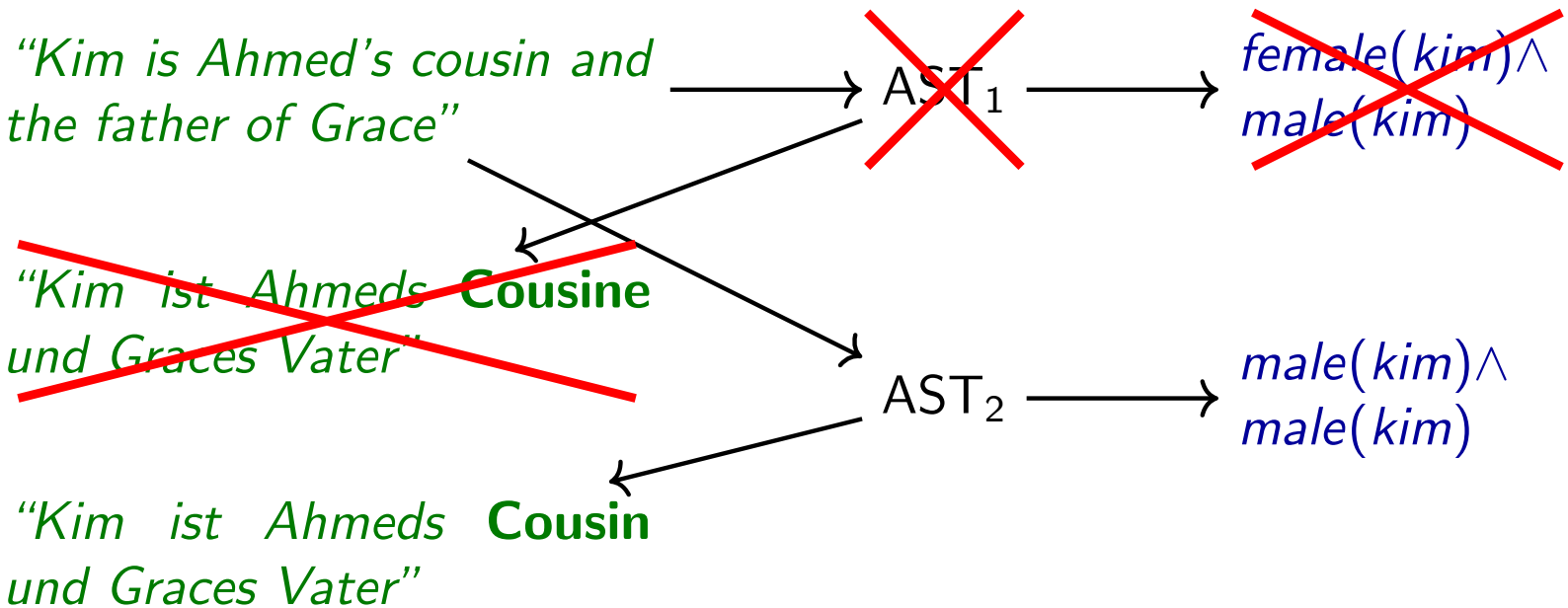
Example: Translation

- Two German words for *"cousin"*, depending on the gender
- Two entries in abstract syntax: `cousin_female` and `cousin_male`
- Use inference to discard ASTs



Example: Translation

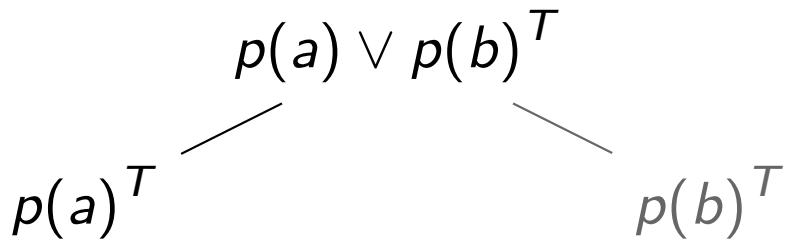
- Two German words for *"cousin"*, depending on the gender
- Two entries in abstract syntax: `cousin_female` and `cousin_male`
- Use inference to discard ASTs



Example: Tableaux Machine [KK03]

- Can use tableaux for model generation
- Tableau machine: pick "best" branch as model and continue there with next sentence *like a human?*

"Ahmed or Berta paints"



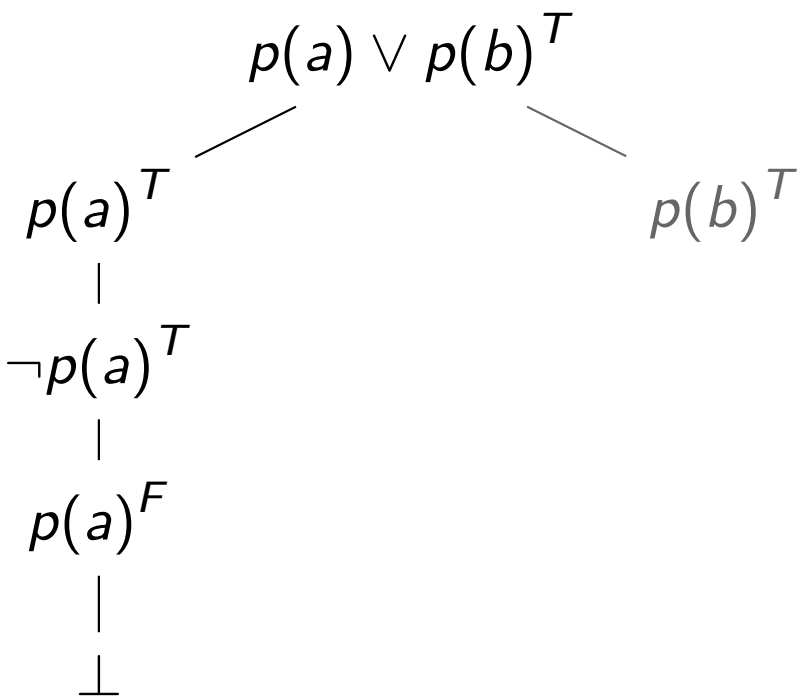
"Ahmed doesn't paint"

Example: Tableaux Machine [KK03]

- Can use tableaux for model generation
- Tableau machine: pick "best" branch as model and continue there with next sentence *like a human?*

"Ahmed or Berta paints"

"Ahmed doesn't paint"

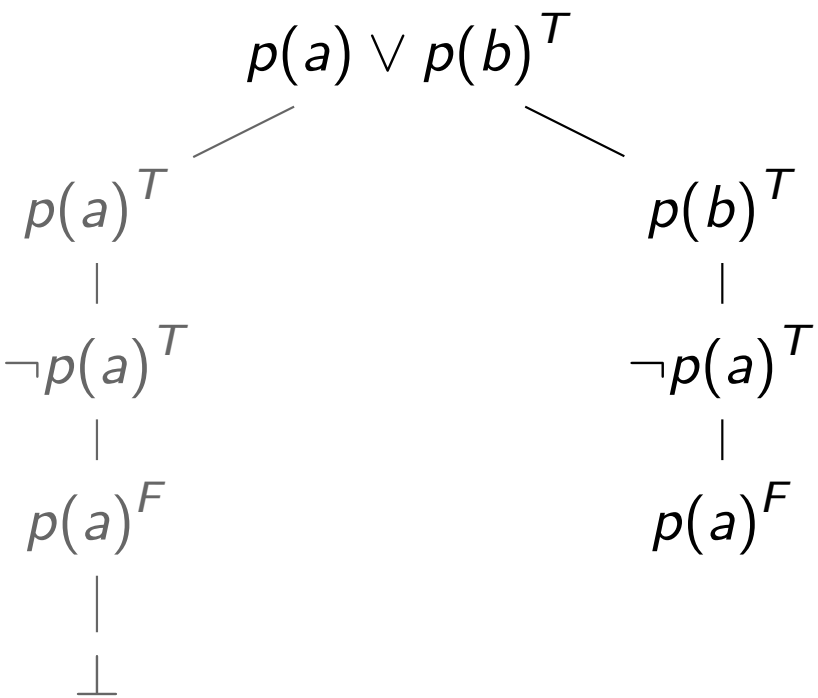


Example: Tableaux Machine [KK03]

- Can use tableaux for model generation
- Tableau machine: pick "best" branch as model and continue there with next sentence *like a human?*

"Ahmed or Berta paints"

"Ahmed doesn't paint"

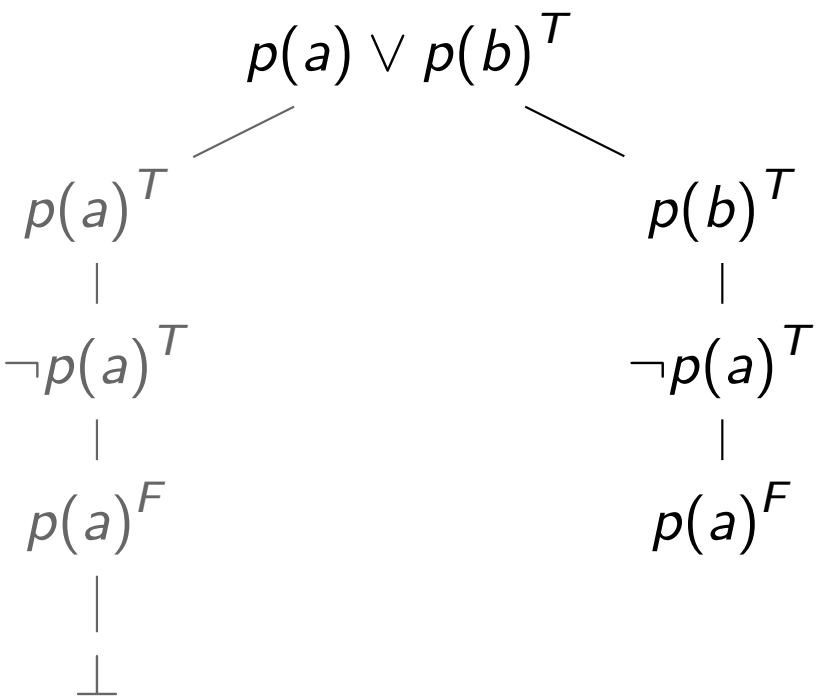


Example: Tableaux Machine [KK03]

- Can use tableaux for model generation
- Tableau machine: pick "best" branch as model and continue there with next sentence *like a human?*

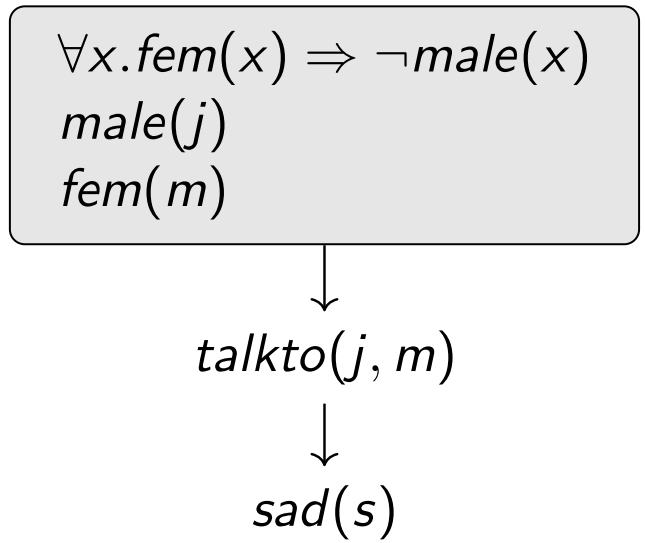
"Ahmed or Berta paints"

"Ahmed doesn't paint"



Example: Tableaux Machine

Background Knowledge



"John talks to Mary."

$talkto(j, m)$

"Sasha is sad."

$sad(s)$

Example: Tableaux Machine

Background Knowledge

"John talks to Mary."

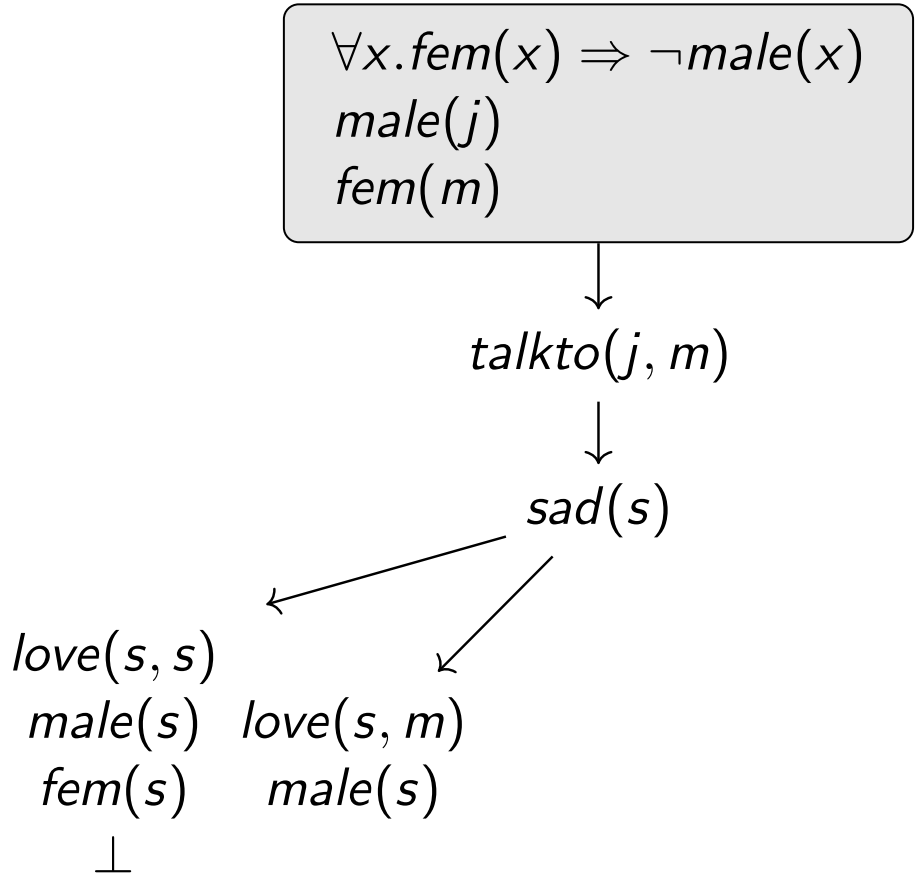
$talkto(j, m)$

"Sasha is sad."

$sad(s)$

"He loves her."

$\exists X.male(X) \wedge$
$\quad \exists Y.fem(Y) \wedge love(X, Y)$



Example: Tableaux Machine

Background Knowledge

"John talks to Mary."

$talkto(j, m)$

"Sasha is sad."

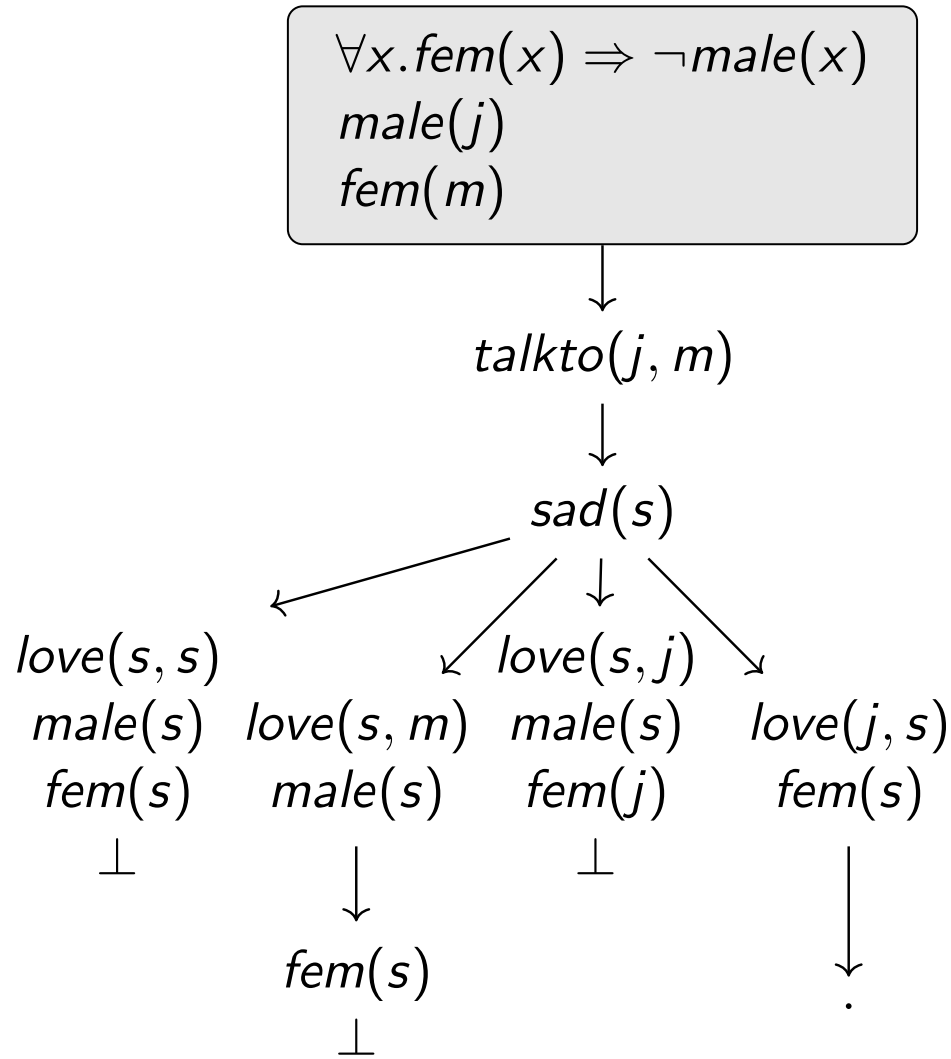
$sad(s)$

"He loves her."

$\exists X.male(X) \wedge$
$\quad \exists Y.fem(Y) \wedge love(X, Y)$

"Sasha is a woman."

$fem(s)$



Example: Tableaux Machine

Background Knowledge

"John talks to Mary."

$talkto(j, m)$

"Sasha is sad."

$sad(s)$

"He loves her."

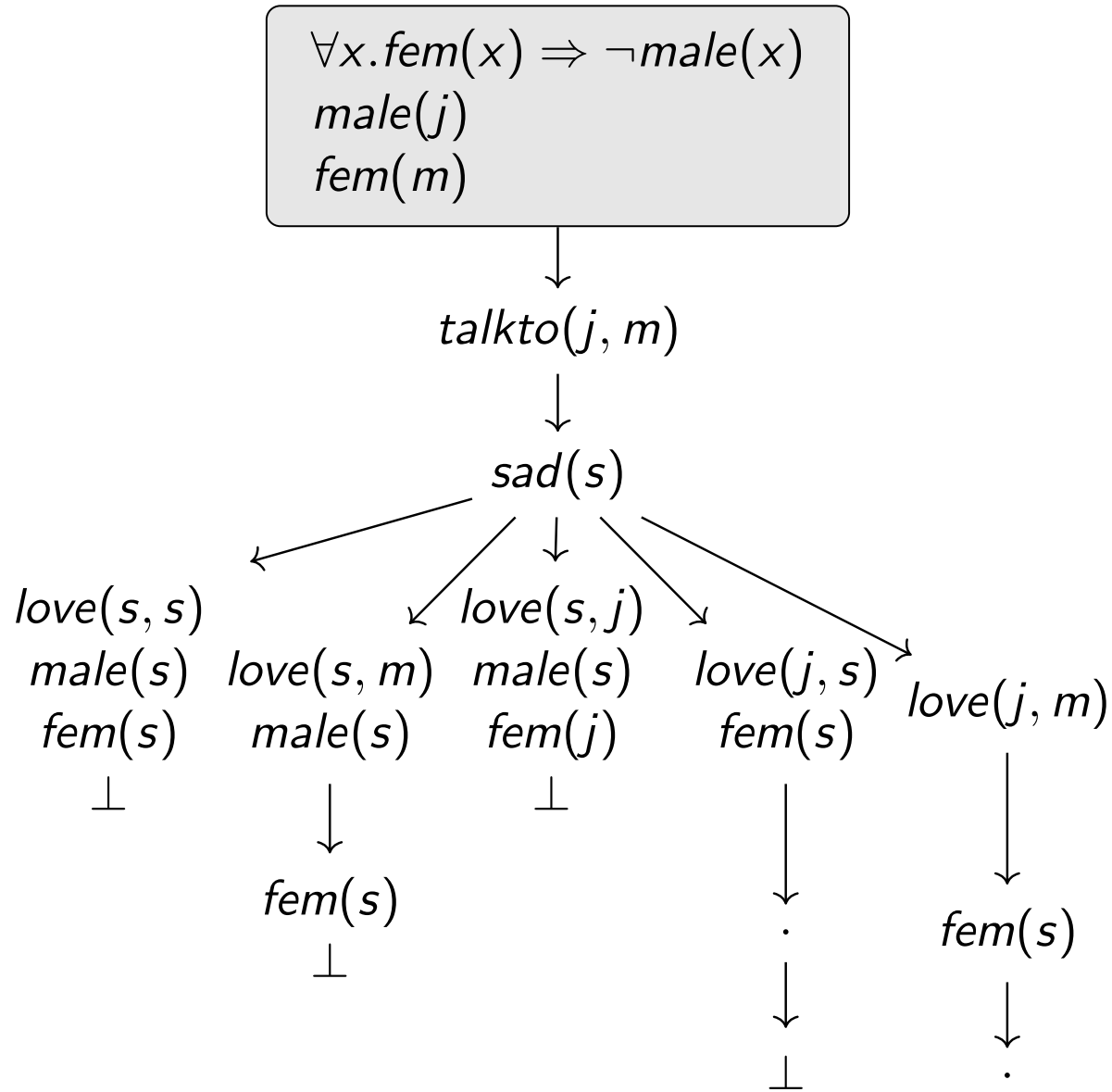
$\exists X.male(X) \wedge$
$\quad \exists Y.fem(Y) \wedge love(X, Y)$

"Sasha is a woman."

$fem(s)$

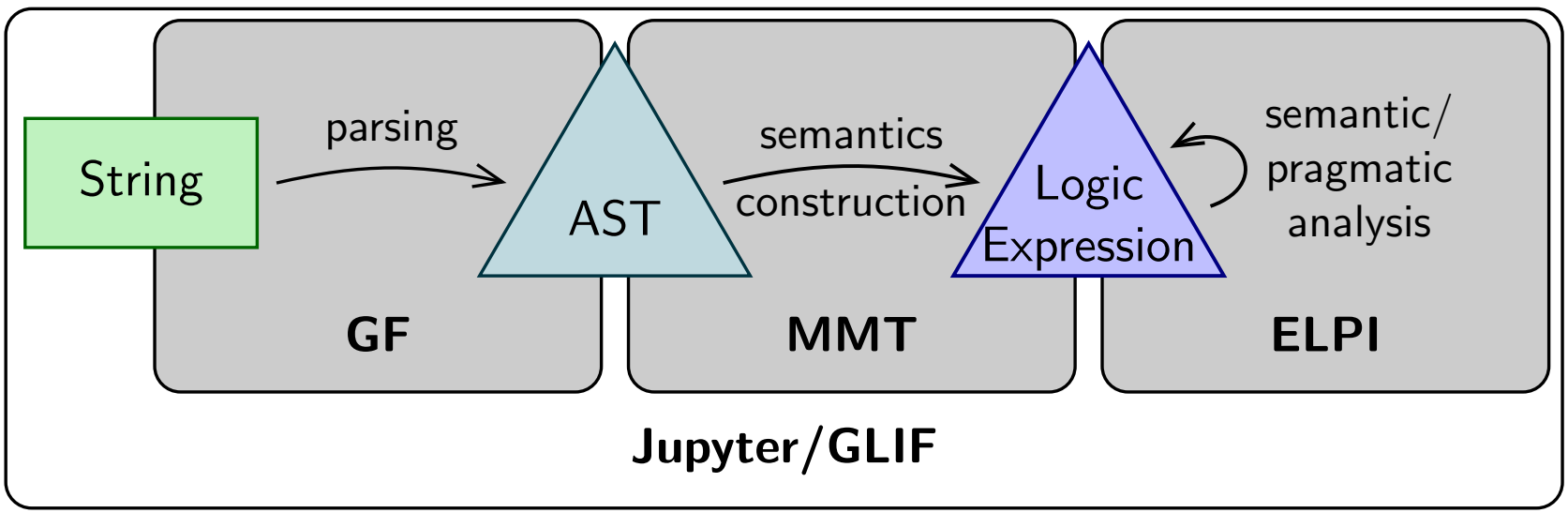
"John doesn't love Sasha."

$\neg love(j, s)$



Conclusion

- GLIF is a tool for prototyping natural language understanding pipelines
- Combines existing, declarative frameworks



References I

GF - Grammatical Framework. URL: http://www.grammaticalframework.org (visited on 09/27/2017).

Michael Kohlhase and Alexander Koller. "Resource-Adaptive Model Generation as a Performance Model". In: *Logic Journal of the IGPL* 11.4 (2003), pp. 435–456. URL: http://jigpal.oxfordjournals.org/cgi/content/abstract/11/4/435.

R. Montague. "English as a Formal Language". In: Reprinted in [Tho74], 188–221. Edizioni di Communita, Milan, 1970, pp. 189–224.

References II

Richard Montague. "The Proper Treatment of Quantification in Ordinary English". In: *Formal Philosophy. Selected Papers*. Ed. by R. Thomason. New Haven: Yale University Press, 1974.

R. Thomason, ed. *Formal Philosophy: selected Papers of Richard Montague*. Yale University Press, New Haven, CT, 1974.