— Master's Thesis Presentation —

Prototyping NLU Pipelines

A Type-Theoretical Framework

Jan Frederik Schaefer

FAU Erlangen-Nürnberg

Seminar Wissensrepräsentation und -verarbeitung

presented virtually due to COVID-19

December 16, 2020

Disclaimer

Some of slides have been adapted from previous presentations by me, often about joint research with other people.

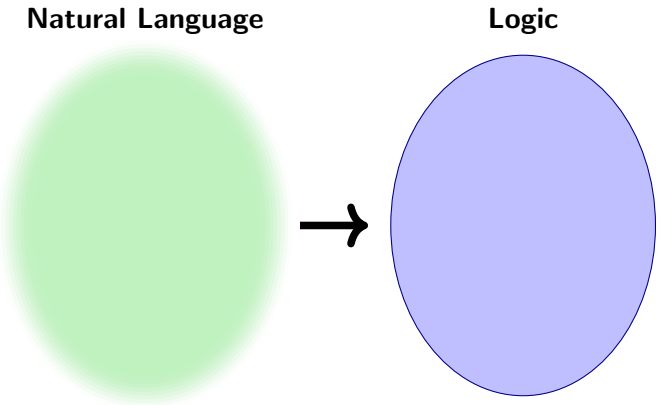
Natural Language Understanding (NLU)

My definition:
NLU means translating natural language into a formal semantic representation.

"How many people live in Slovakia?" $\rightsquigarrow$ *"5.458 million"*

"Do more people live in Slovakia than in Thailand?" $\rightsquigarrow$???

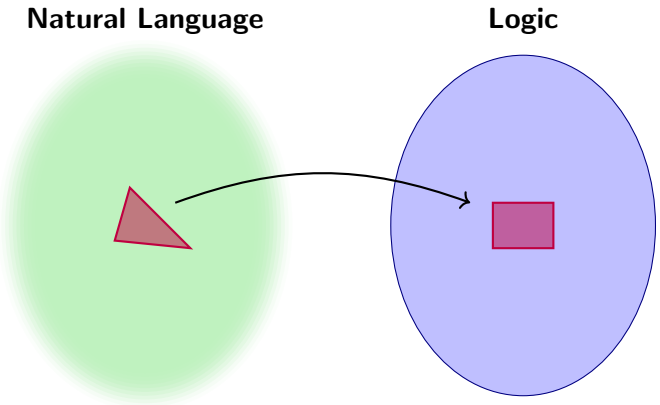
Method of Fragments



How do we get from messy language to formal logic?

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

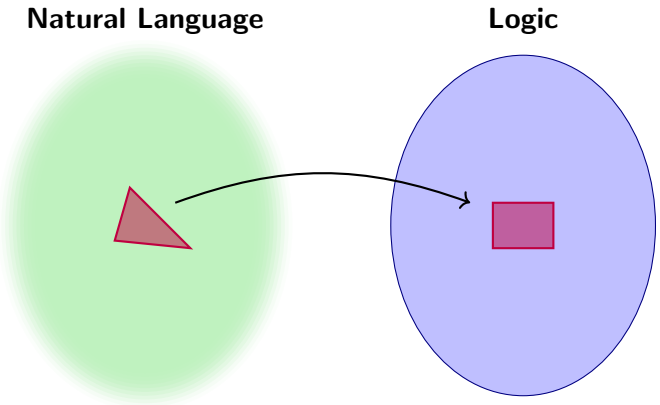
Method of Fragments



How do we get from messy language to formal logic?

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

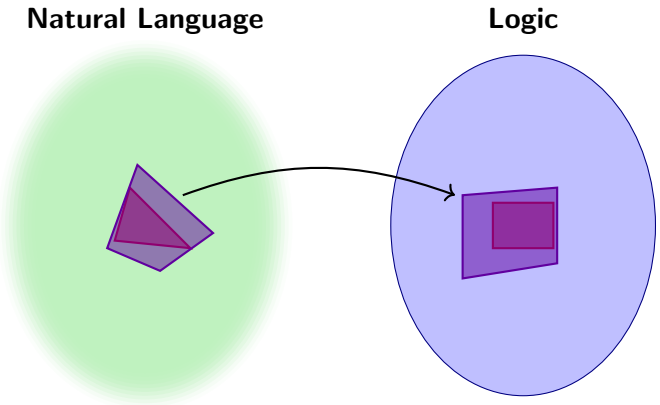
Method of Fragments



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

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

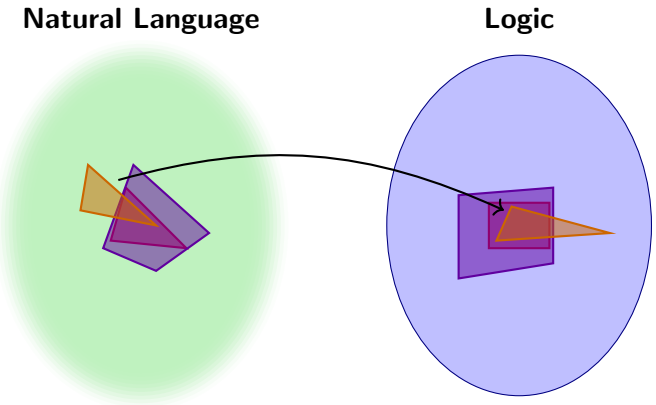
Method of Fragments



"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

Hand-waving is problematic:

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

Montague: Specify

- grammar, *fixes NL subset*
- target logic,
- semantics construction. *maps parse trees to logic*

Example from [Mon74]

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 $\hat{\mathrm{P}}[\alpha'(\mathrm{P}) \vee \beta'(\mathrm{P})]$.

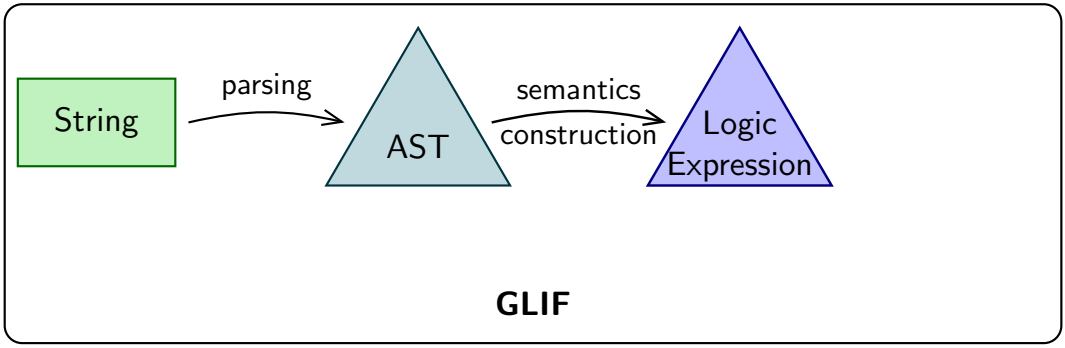
Claim: That doesn't scale well $\rightsquigarrow$ **We need prototyping!**

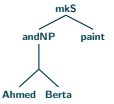
NLU Prototyping

- Traditionally done in Prolog/Haskell
  - $\rightarrow$ requires a lot of work
- A dedicated framework might be better
  - $\rightarrow$ only partial solutions exist
- Can we combine existing partial solutions?
  - $\rightsquigarrow$ GLIF

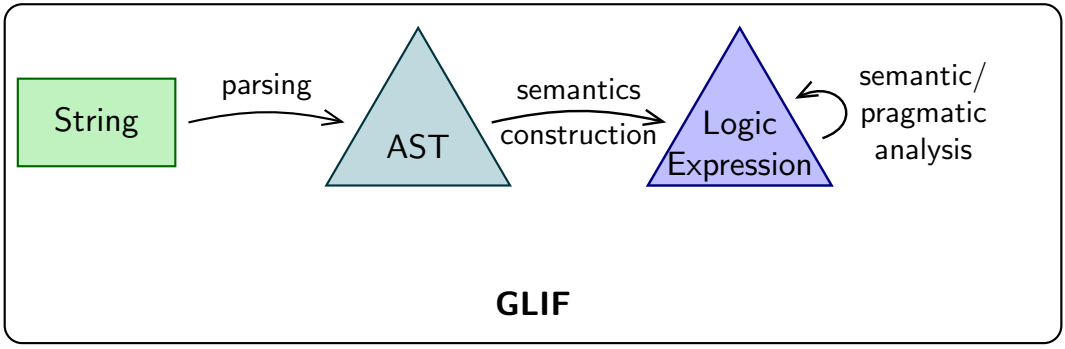
Research Question

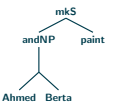
GLIF: Grammatical Logical Inference Framework



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

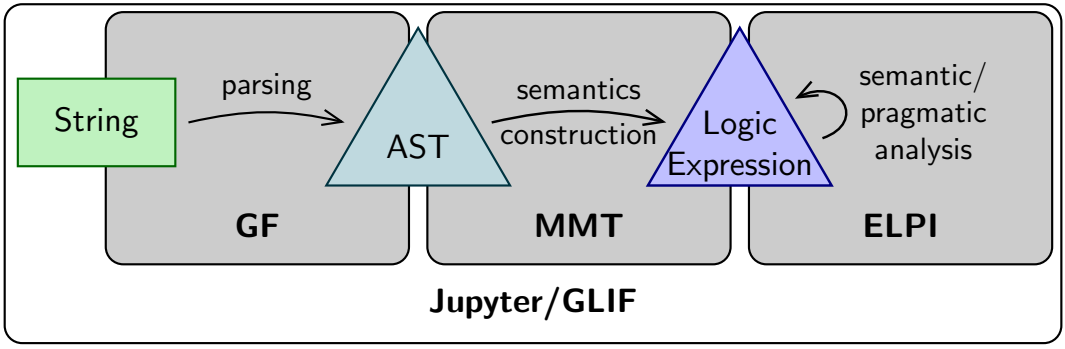
GLIF: Grammatical Logical Inference Framework



"Ahmed and Berta paint." ⟶ 

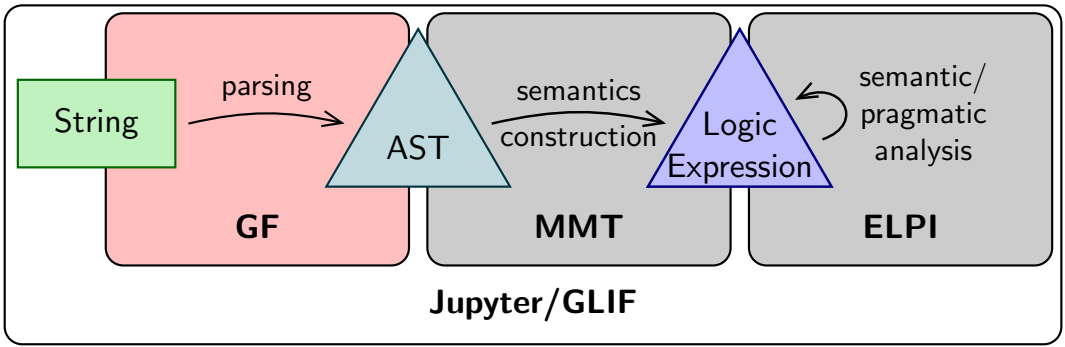
⟶ $p(a) \wedge p(b)$

GLIF: Grammatical Logical Inference Framework



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

Components of GLIF: GF

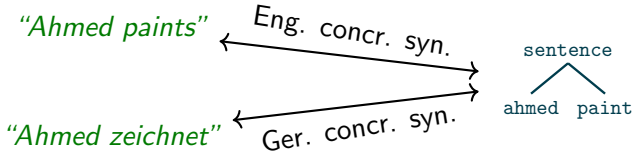


Components of GLIF: Grammatical Framework [GF]

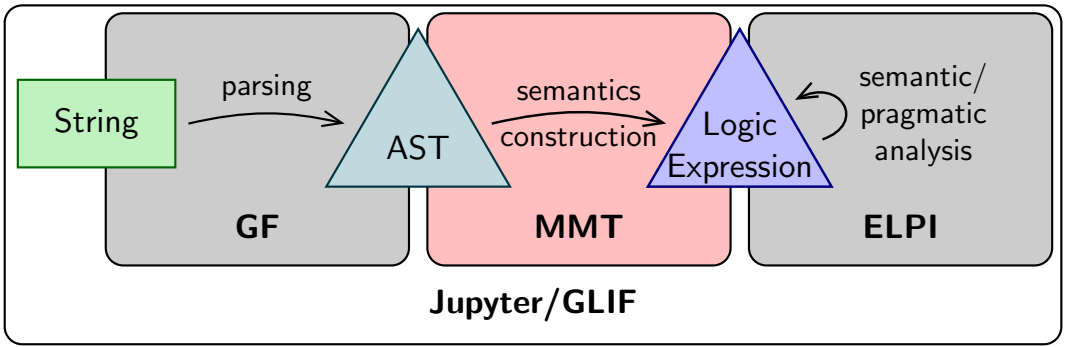
- Specialized for developing natural language grammars
- Separates abstract and concrete syntax

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

- Abstract syntax based on LF
- Comes with large library ≥ 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 discourse 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 discourse 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 discourse 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
```

Discourse 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 discourse 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

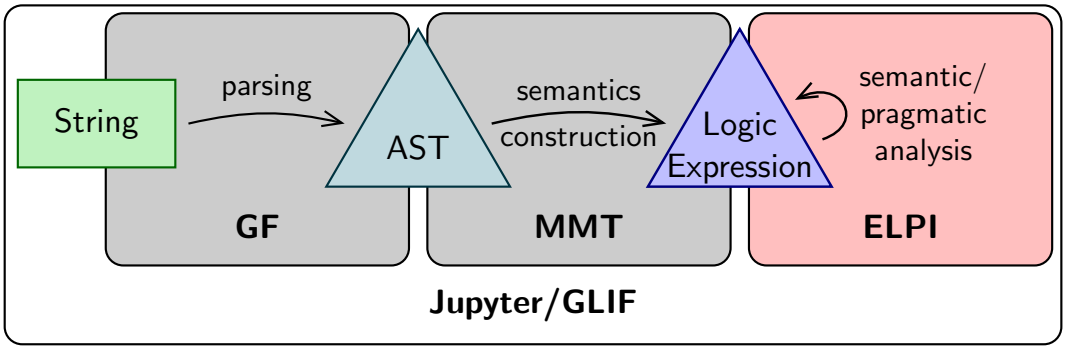
↓semantics construction

`(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
- Generic inference/reasoning step after semantics construction
- Goal: Use it for semantic/pragmatic analysis

MMT

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

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

ELPI

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

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

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)

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

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

Semantic/Pragmatic Analysis

"The trophy doesn't fit in the brown suitcase because it's too big." [LDM12]

Semantic/Pragmatic Analysis

"The trophy doesn't fit in the brown suitcase because it's too big." [LDM12]
"The trophy doesn't fit in the brown suitcase because it's too small." [LDM12]

Semantic/Pragmatic Analysis

"The trophy doesn't fit in the brown suitcase because it's too big." [LDM12]
"The trophy doesn't fit in the brown suitcase because it's too small." [LDM12]

"The ball has a radius of 2m."
"The ball has a mass of 2m."

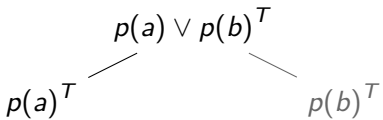
"We saw her duck."

$\rightsquigarrow$ semantics construction creates preliminary semantic representation(s) that gets refined by the semantic/pragmatic analysis

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"

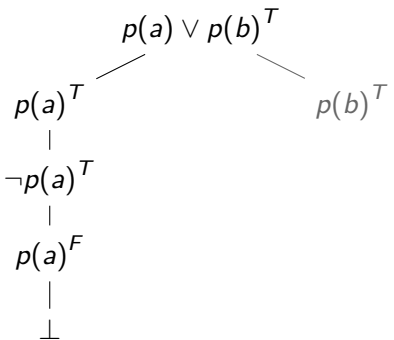


Example: Tableaux Machine [KK03]

- 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"

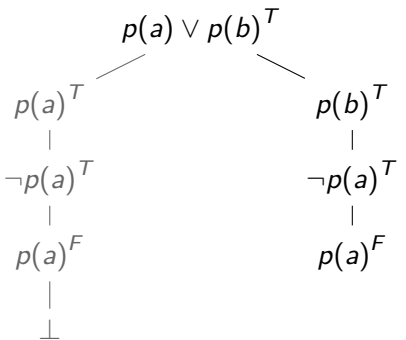


Example: Tableaux Machine [KK03]

- 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

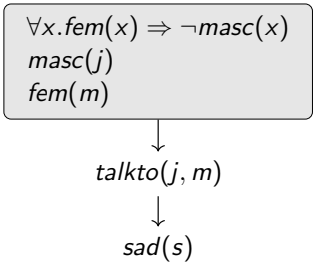
Background Knowledge

"John talks to Mary."

$talkto(j, m)$

"Sasha is sad."

$sad(s)$



Example: Tableaux Machine

Background Knowledge

"John talks to Mary."

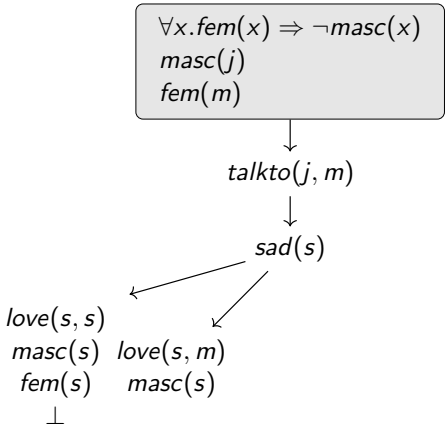
$talkto(j, m)$

"Sasha is sad."

$sad(s)$

"He loves her."

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



Example: Tableaux Machine

Background Knowledge

"John talks to Mary."

$talkto(j, m)$

"Sasha is sad."

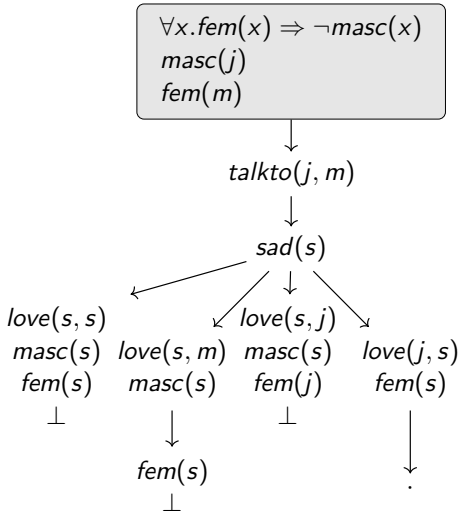
$sad(s)$

"He loves her."

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

"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."

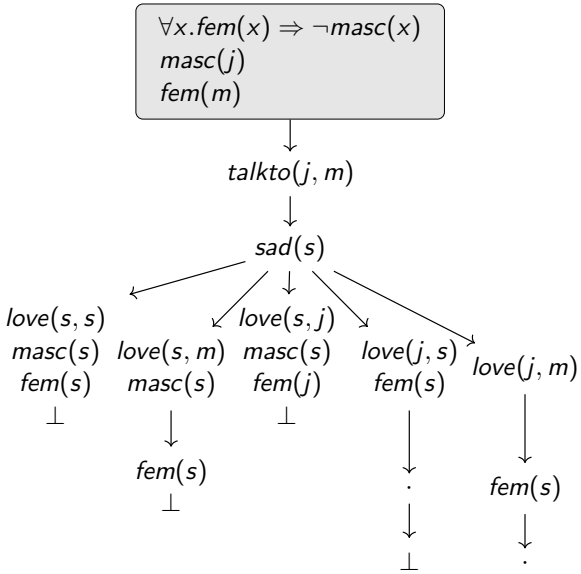
$\exists X.masc(X) \wedge$
$\exists Y.fem(Y) \wedge love(X, Y)$

"Sasha is a woman."

$fem(s)$

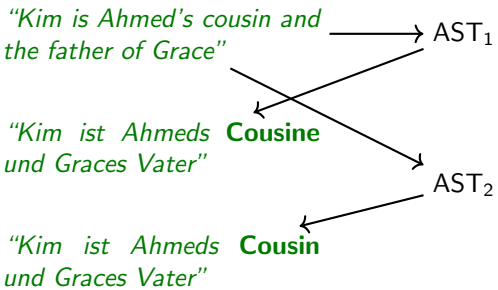
"John doesn't love Sasha."

$\neg love(j, s)$



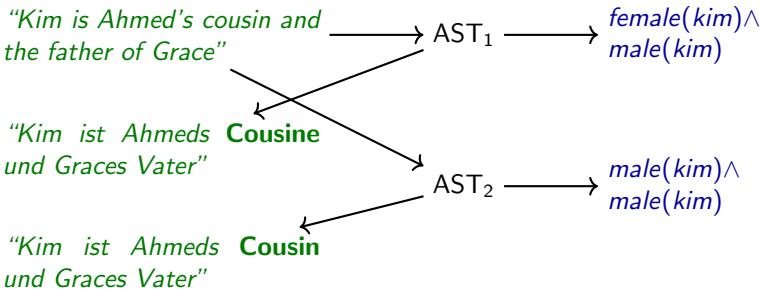
Example: Translation

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



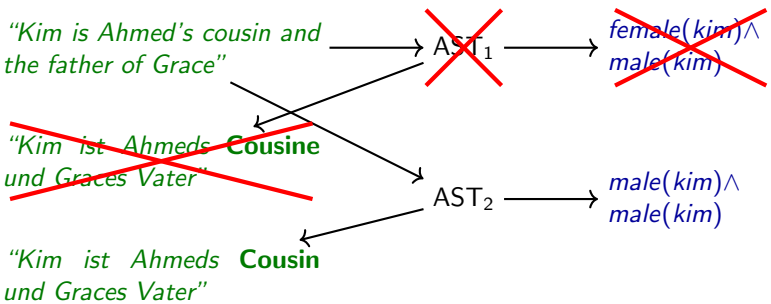
Example: Translation

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

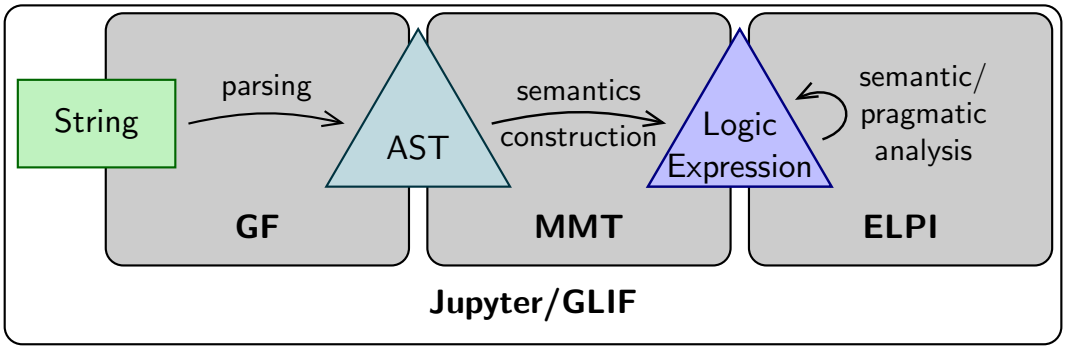


Example: Translation

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



GLIF Summary



Supporting the Semantic/Pragmatic Analysis

Observation:

- Parsing and semantics construction are based on specialized frameworks
- ELPI is a more general programming language

Can we do something more specialized?

- Not really – there is no "standard recipe" for semantic/pragmatic analysis
- But: It usually involves inferential reasoning

Let's generate provers!

Natural Deduction in MMT/LF

$$\frac{A \wedge B}{A}\ \wedge El$$

$$\frac{A \vee B \qquad \begin{array}{c}[A]^1\\ \vdots\\ C\end{array} \qquad \begin{array}{c}[B]^1\\ \vdots\\ C\end{array}}{C}\ \vee E^1$$

```
// ⊢X is type of proofs for X (judgments as types)
⊢ : o → type

∧El : Π_{A:o}Π_{B:o} ⊢A∧B → ⊢A
∨E  : Π_{A:o}Π_{B:o}Π_{C:o} ⊢A∨B → (⊢A → ⊢C) → (⊢B → ⊢C) → ⊢C
```

Generating Provers in ELPI

LF rule $\wedge\mathtt{El} : \Pi_{\mathtt{A}:\mathtt{o}}\Pi_{\mathtt{B}:\mathtt{o}}\ \vdash\mathtt{A}\wedge\mathtt{B} \longrightarrow \vdash\mathtt{A}$

ELPI equivalent

direct: `pi A \ pi B \ ded (and A B) => ded A.`

syn. sugar: `ded A :- ded (and A B).`

Generating Provers in ELPI

LF rule $\wedge\mathtt{El} : \Pi_{\mathtt{A}:\mathtt{o}}\Pi_{\mathtt{B}:\mathtt{o}}\ \vdash\mathtt{A}\wedge\mathtt{B} \to \vdash\mathtt{A}$

ELPI equivalent

direct: `pi A \ pi B \ ded (and A B) => ded A.`

syn. sugar: `ded A :- ded (and A B).`

Example: Or-Elimination

LF: $\vee\mathtt{E} : \Pi_{\mathtt{A}:\mathtt{o}}\Pi_{\mathtt{B}:\mathtt{o}}\Pi_{\mathtt{C}:\mathtt{o}}\ \vdash\mathtt{A}\vee\mathtt{B} \to (\vdash\mathtt{A} \to \vdash\mathtt{C}) \to (\vdash\mathtt{B} \to \vdash\mathtt{C}) \to \vdash\mathtt{C}$

ELPI: `ded C :- ded (or A B), ded A => ded C, ded B => ded C.`

Example: Forall-Introduction

LF: $\forall\mathtt{I} : \Pi_{\mathtt{P}:\iota\to\mathtt{o}}\ (\Pi_{\mathtt{x}:\iota}\ \vdash\mathtt{P}\ \mathtt{x}) \to \vdash\forall\mathtt{P}$

ELPI: `ded (forall P) :- pi x \ ded (P x).`

Controlling the Proof Search

- Problem: Search diverges *searching harder than checking*
- Solution: Control search with helper predicates:

  inspired by ProofCert project by Miller et al.

  - Intuition: Decide whether to apply rule
  - Do not affect correctness
  - Extra argument tracks aspects of proof state

Before: `ded A :-` `ded (and A B).`

Now: `ded X A :- help/andEl X A B X1, ded X1 (and A B).`

Helper Predicates

Name	**Predicate**	**Argument**
Iter. deepening	checks depth	remaining depth
Proof term	generates term	proof term
Product	calls other predicates	arguments for other predicates
Backchaining	Prolog's backchaining ($\approx$ forward reasoning from axioms via $\Rightarrow/\forall$ elimination rules)	pattern of formula to be proven (e.g. a conjunction)

Example helper: Iterative deepening

```
help/andEl (idcert N) _ _ (idcert N1) :- N > 0, N1 is N - 1.
```

Tableau Provers

$$\frac{A \wedge B^F}{A^F \mid B^F}\ \wedge^F$$

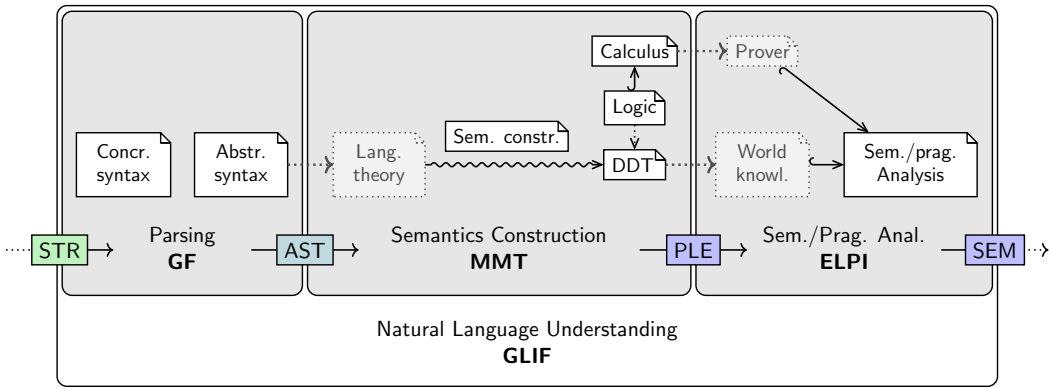
$$\dfrac{\begin{array}{ccc} & [A^F] & [B^F] \\ A \wedge B^F & \vdots & \vdots \\ & \bot & \bot \end{array}}{\bot}\ \wedge^F$$

LF: $\wedge^{\boldsymbol{F}} : \Pi_{\mathtt{A}:\mathtt{o}}\Pi_{\mathtt{B}:\mathtt{o}}\ \mathtt{A}\wedge\mathtt{B}^{\boldsymbol{F}} \to (\mathtt{A}^{\boldsymbol{F}} \to \bot) \to (\mathtt{B}^{\boldsymbol{F}} \to \bot) \to \bot$

ELPI:
```
closed X :- help/andF X A B X1 X2 X3, f X1 (and A B),
            f/hyp A => closed X2, f/hyp B => closed X3.
```

With iterative deepening we get a working prover!
→ Other helpers result in more efficient provers

Conclusion: Prototyping NLU Pipelines



References I

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

[KK03] 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.

[LDM12] Hector Levesque, Ernest Davis, and Leora Morgenstern. "The Winograd Schema Challenge". In: *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*. 2012.

[Mon70] R. Montague. "English as a Formal Language". In: Reprinted in [Tho74], 188–221. Edizioni di Communita, Milan, 1970, pp. 189–224.

References II

[Mon74] 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.

[Tho74] R. Thomason, ed. *Formal Philosophy: selected Papers of Richard Montague*. Yale University Press, New Haven, CT, 1974.