What does it mean? A framework for prototyping Montague-style semantics

One approach to study the meaning of natural language is to translate utterances to logical expressions, which have well-defined semantics and allow for rigorous inference. Richard Montague, a well-known pioneer of this approach, described the translation of a particular subset of English into logic in great detail. Over the years, there have been many similar publications that propose a novel way to translate sentences into a logic (often first-order logic, but more obscure logics are also common). Sometimes, the approach was extended with pragmatic analysis based on logical inference.

We claim that research (and education) in this direction could benefit from more prototyping to test and demonstrate new ideas.

In this talk, I will present GLIF, a declarative framework for prototyping the translation of natural language to logics. GLIF combines existing, specialized frameworks that solve part of the problem: the Grammatical Framework (development of natural language grammars), MMT (logic development) and ELPI (inference). These frameworks can be connected seamlessly because of their compatible underlying logical frameworks. We successfully use GLIF in a lecture on symbolic natural language semantics.

What does it mean?
A framework for prototyping Montague-style semantics

Jan Frederik Schaefer

FAU Erlangen-Nürnberg/KWARC

Seminar: Computing Meaning
Hildesheim
July 21, 2022

whoami

- PhD student in the KWARC group (Erlangen)   *Knowledge representation*
- Supervisor: Michael Kohlhase
- My interest: Mathematical language, precise semantics extraction
- Background for this talk:
    - We teach a lecture in *logic-based natural language semantics*
    - Wanted more hands-on experience
    - $\rightsquigarrow$ new framework: GLIF   *Grammar, Logic, Inference*

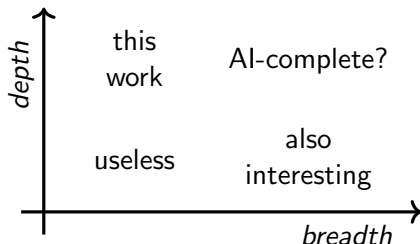# Natural Language Semantics (Symbolic)

For this talk:
 Translating natural language into a formal semantic representation (logic).

Example:
 "Every student paints and is quiet." $\rightsquigarrow$ $\forall x.s(x) \Rightarrow (p(x) \wedge q(x))$
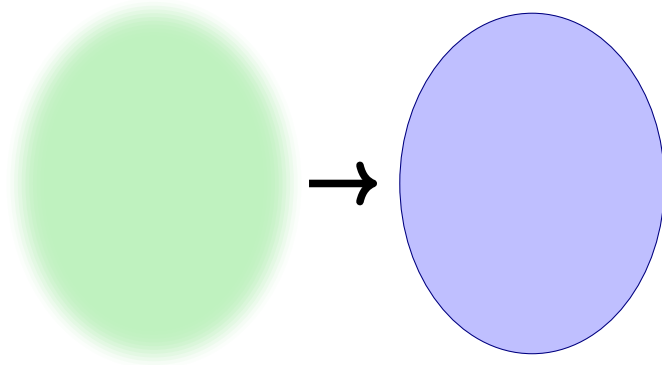
Rule-based (no ML):
 Parsing $\rightsquigarrow$ semantics construction $\rightsquigarrow$ inference.

Method of Fragments

How do we get from messy language to formal logic?

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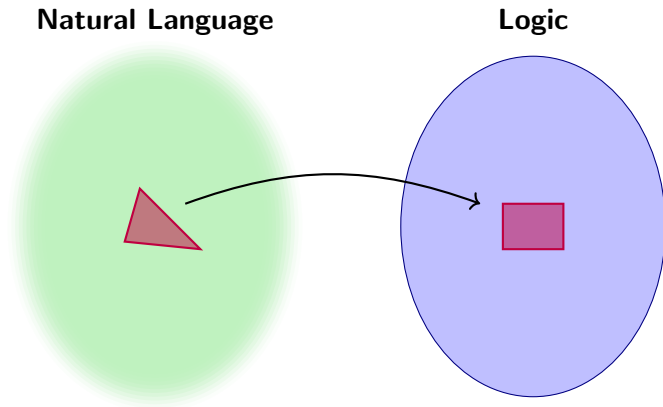
Method of Fragments

Natural Language **Logic**
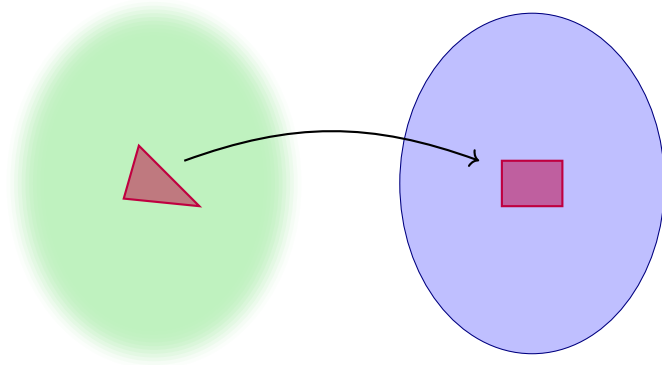
How do we get from messy language to formal logic?

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

Method of Fragments

Natural Language	Logic

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

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

Method of Fragments

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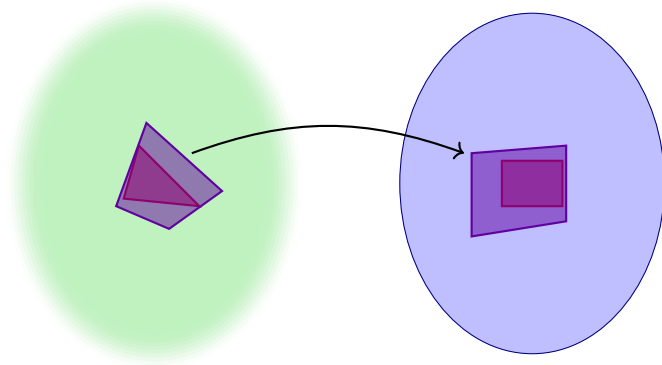
"Every student paints and is quiet." $\forall x.s(x) \Rightarrow (p(x) \wedge q(x))$

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

Method of Fragments

Natural Language
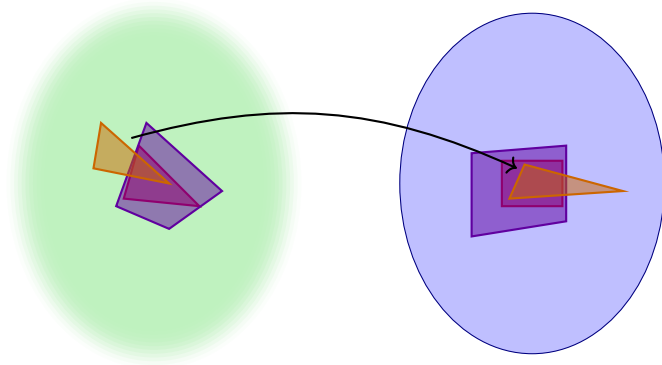
Logic

"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." $\quad\overset{?}{\leadsto}\quad$ $p(a) \wedge q(a)$

Montague: Specify

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

Example from [Mon74]

> T11. If $\phi, \psi \in P_t$ 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 $\gamma, \delta \in P_{IV}$ 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 $\alpha, \beta \in P_T$ and α, β translate into α', β' respectively, then α **or** β translates into $\hat{P}[\alpha'(P) \vee \beta'(P)]$.
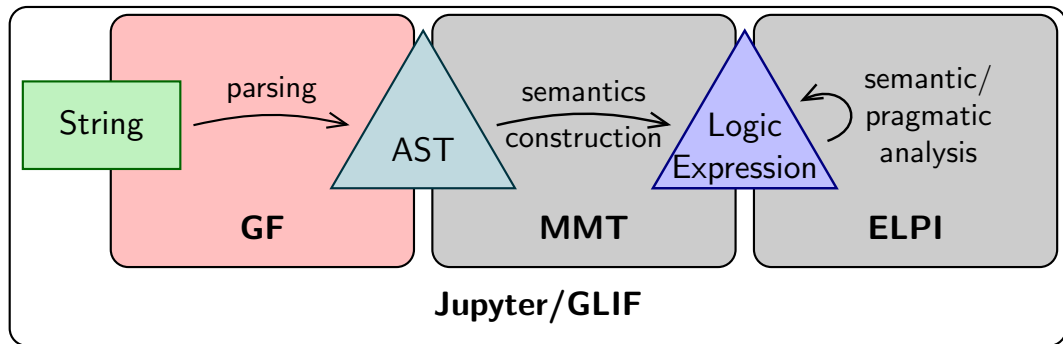
Claim: That doesn't scale well $\leadsto$ **We need prototyping!**

NLU Prototyping
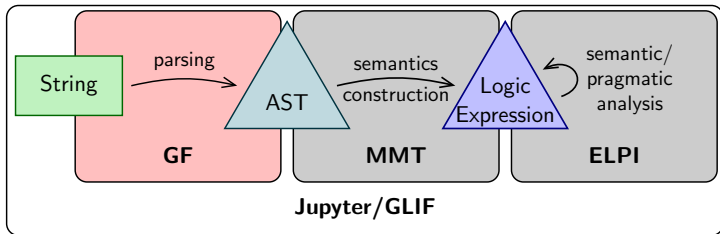
```
> translate "Every student paints and is quiet."
```
$\forall x.s(x) \Rightarrow (p(x) \land q(x))$

- 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

Components of GLIF: GF
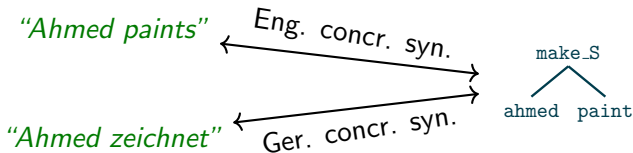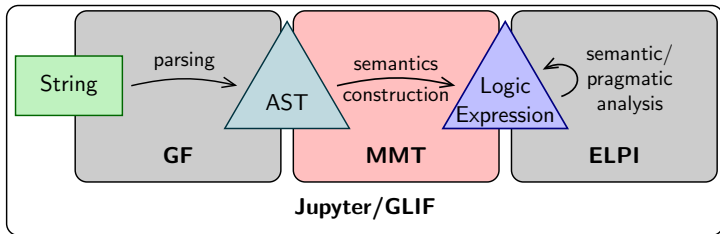
Components of GLIF: Grammatical Framework [GF]

- Specialized for developing natural language grammars
- Separates abstract and concrete syntax
  ```
  make_S : NP -> VP -> S;                     abstract
  make_S 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 repr.
- Not specialized in one logical framework   *we use LF*
- We will use MMT to:
    1. represent abstract syntax
    2. specify target logic and discourse domain theory
    3. specify semantics construction

Components of GLIF: MMT

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

<table>
<tr><th>GF</th><th></th><th>MMT</th></tr>
</table>

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

$\longmapsto$

```
NP : type
VP : type
S  : type
make_S :
  NP → VP → S
```

Components of GLIF: MMT

- Modular logic development and knowledge repr.
- Not specialized in one logical framework   *we use LF*
- We will use MMT to:
  1. represent abstract syntax
  2. **specify target logic and discourse domain theory**
  3. 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

```
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 repr.
- Not specialized in one logical framework   *we use LF*
- We will use MMT to:
  1. represent abstract syntax
  2. specify target logic and discourse domain theory
  3. **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
make_S   ↦  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"

$\downarrow$parsing

make_S (andNP ahmed berta) paint

$\downarrow$semantics construction

$(\lambda n.\lambda v.n\ v)\ ((\lambda a.\lambda b.\lambda p.a\ p \wedge b\ p)\ (\lambda p.p\ \text{ahmed})\ (\lambda p.p\ \text{berta}))$ pain

$\downarrow\beta$-reduction

paint ahmed $\wedge$ paint berta

Example: Student Project [Int]

parse `"John has not always run"` | **construct**

`¬H (run john)`

parse `"John has to have been allowed to always run"` | **construct**

`□P ◊(H (run john)∧G (run john))`

parse `"John probably will never run"` | **construct**

`Prob G ¬(run john)`

parse `"it has to be possible that John runs"` | **construct**

`□◊(run john)`

parse `"Mary saw that John would kill the dog"` | **construct**

`P ◉mary◉F (kill john dog)`

parse `"Mary runs and John sees it"` | **construct**

`(run mary)∧◉john◉(run mary)`

Components of GLIF: ELPI

- Implementation and extension of λProlog $\qquad \approx$ *Prolog + HOAS*
- MMT can generate logic signatures
- First experiments with prover generation
- Generic inference/reasoning step after semantics construction
- Goal: Use it for semantic/pragmatic analysis

<div align="center">

MMT   **ELPI**

</div>

```
o : type //propositions        kind o type.
¬ : o → o                       not : o -> o.
∧ : o → o → o                   and : o -> o -> o.
∨ : o → o → o                   or  : o -> o -> o.

ι : type //individuals          kind i type.
∀ : (ι → o) → o                 type forall (i -> o) -> o.
∃ : (ι → o) → o                 type exists (i -> o) -> o.
```

"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."

⤳ semantics construction creates preliminary semantic representation(s) that get refined by the semantic/pragmatic analysis

Example: Discard wrong readings in controlled natural language

"the ball has a mass of 5kg" $\rightarrow$ AST $\longrightarrow$ mass(theball, quant(5, kilo gram))

Example: Discard wrong readings in controlled natural language

"the ball has a mass of 5kg" $\rightarrow$ AST $\longrightarrow$ $\mathrm{mass}(\mathrm{theball}, \mathrm{quant}(5, \mathrm{kilo\ gram}))$

$\longrightarrow$ AST_1 $\longrightarrow$ $\lambda x. E_{\mathrm{kin}}(x, \mathrm{quant}(2, \textbf{milli Newton}))$

"a kinetic energy of 12mN"

$\longrightarrow$ AST_2 $\longrightarrow$ $\lambda x. E_{\mathrm{kin}}(x, \mathrm{quant}(2, \textbf{meter}{\cdot}\textbf{Newton}))$

Example: Discard wrong readings in controlled natural language

"the ball has a mass of 5kg" $\rightarrow$ AST $\longrightarrow$ mass(theball, quant(5, kilo gram))

"a kinetic energy of 12mN"

$\longrightarrow$ AST$_1$ $\longrightarrow$ $\lambda x.E_{\text{kin}}(x, \text{quant}(2, \textbf{milli Newton}))$

$\longrightarrow$ AST$_2$ $\longrightarrow$ $\lambda x.E_{\text{kin}}(x, \text{quant}(2, \textbf{meter}\cdot\textbf{Newton}))$

```
In [20]:  1  parse "the ball has a mass of 5 k g and a kinetic energy of 12 m N" |
          2      construct
```
(mass theball (quant 5 kilo gram)) ∧ (ekin theball (quant 12 milli Newton))
(mass theball (quant 5 kilo gram)) ∧ (ekin theball (quant 12 meter·Newton))
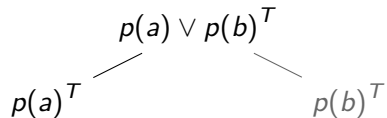
```
In [21]:  1  parse "the ball has a mass of 5 k g and a kinetic energy of 12 m N" |
          2      construct | filter -predicate=filter_pred
```
(mass theball (quant 5 kilo gram)) ∧ (ekin theball (quant 12 meter·Newton))

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"

$$p(a) \vee p(b)^T$$

$$p(a)^T \qquad\qquad p(b)^T$$

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" — $p(a) \lor p(b)^T$

$p(a)^T$ $p(b)^T$

"Ahmed doesn't paint" — $\neg p(a)^T$
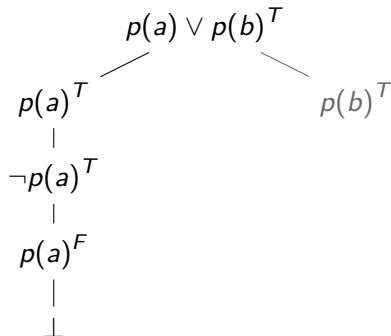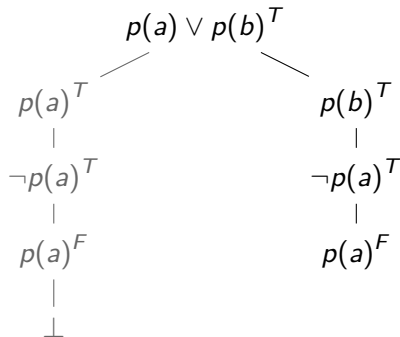
$p(a)^F$

$\bot$

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" $p(a) \lor p(b)^T$

$p(a)^T$ $p(b)^T$

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

$p(a)^F$ $p(a)^F$

$\perp$

Example: Tableaux Machine

Background Knowledge

"John talks to Mary."
talkto(j, m)

"Sasha is sad."
sad(s)

$$\forall x.fem(x) \Rightarrow \neg masc(x)$$
$$masc(j)$$
$$fem(m)$$

$\downarrow$

$$talkto(j, m)$$

$\downarrow$

$$sad(s)$$

Example: Tableaux Machine

Background Knowledge

"John talks to Mary."
$talkto(j, m)$

"Sasha is sad."
$sad(s)$

"He loves her."
$\exists X.masc(X) \wedge$
$\quad \exists Y.fem(Y) \wedge love(X, Y)$

$$\boxed{\begin{array}{l} \forall x.fem(x) \Rightarrow \neg masc(x) \\ masc(j) \\ fem(m) \end{array}}$$

$$\downarrow$$

$$talkto(j, m)$$

$$\downarrow$$

$$sad(s)$$

$love(s, s)$
$masc(s) \quad love(s, m)$
$fem(s) \quad masc(s)$
$\bot$

Example: Tableaux Machine

Background Knowledge

"John talks to Mary."
talkto(j, m)

"Sasha is sad."
sad(s)

"He loves her."
∃X.masc(X)∧
  ∃Y.fem(Y) ∧ love(X, Y)

"Sasha is a woman."
fem(s)

$$\forall x.fem(x) \Rightarrow \neg masc(x)$$
$$masc(j)$$
$$fem(m)$$

↓

talkto(j, m)

↓

sad(s)

↓

love(s, s)   love(s, j)
masc(s) love(s, m) masc(s)   love(j, s)
fem(s)   masc(s)   fem(j)   fem(s)
⊥   ↓   ⊥

          fem(s)   ↓
          ⊥   ·

Example: Tableaux Machine

Background Knowledge

$$\forall x. fem(x) \Rightarrow \neg masc(x)$$
$$masc(j)$$
$$fem(m)$$

"John talks to Mary."
$talkto(j, m)$

"Sasha is sad."
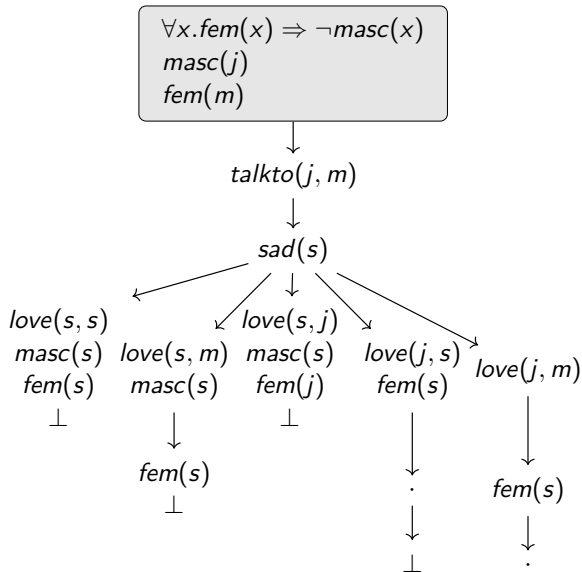$sad(s)$

"He loves her."
$\exists X. masc(X) \wedge$
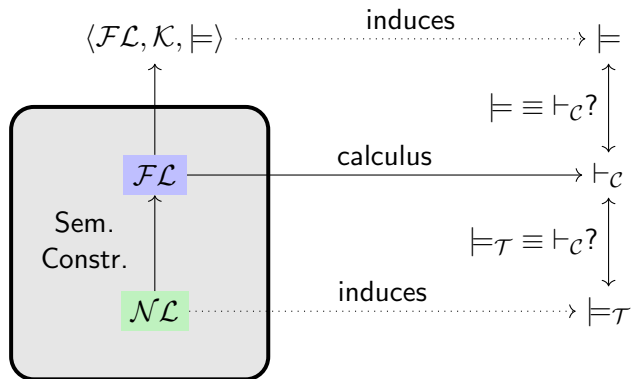$\quad \exists Y. fem(Y) \wedge love(X, Y)$

"Sasha is a woman."
$fem(s)$

"John doesn't love Sasha."
$\neg love(j, s)$

$talkto(j, m)$

$sad(s)$

$love(s, s)$
$masc(s)$
$fem(s)$
$\perp$

$love(s, m)$
$masc(s)$
$\downarrow$
$fem(s)$
$\perp$

$love(s, j)$
$masc(s)$
$fem(j)$
$\perp$

$love(j, s)$
$fem(s)$
$\downarrow$
$\cdot$
$\downarrow$
$\perp$

$love(j, m)$
$\downarrow$
$fem(s)$
$\downarrow$
$\cdot$

Making it a Science

1. Test: Does *"Ahmed and Berta paint."* $\models_\mathcal{T}$ *"Berta paints."*?
2. Model prediction: Yes, because $p(a) \land p(b) \vdash_\mathcal{C} p(b)$.
3. Correct result: Ask people.

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)

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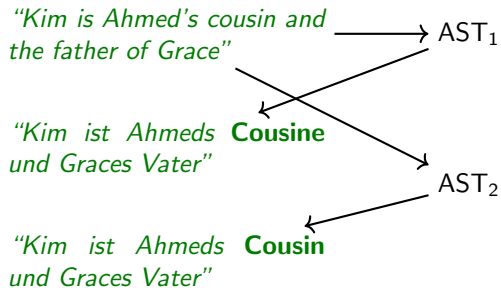$$Q = \Box_{eve} hd(ping) \lor \Box_{eve} \neg hd(ping)$$

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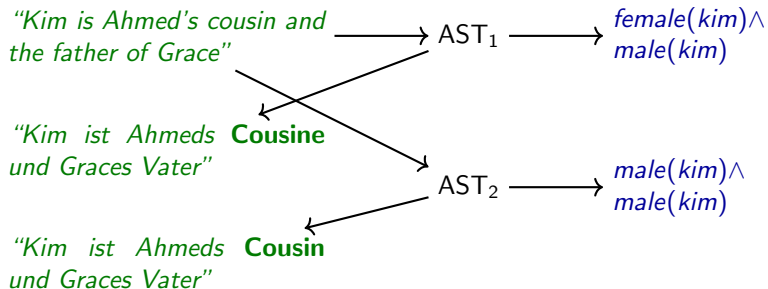\text{else} & \rightsquigarrow & \text{maybe}
\end{array}
$$

Example: 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

"Kim is Ahmed's cousin and the father of Grace" $\longrightarrow$ AST$_1$

*"Kim ist Ahmeds **Cousine** und Graces Vater"*

AST$_2$

*"Kim ist Ahmeds **Cousin** und Graces Vater"*

Example: Translation

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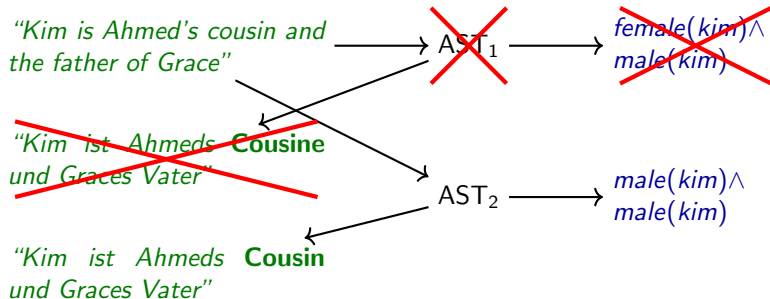- Use inference to discard ASTs

"Kim is Ahmed's cousin and the father of Grace" $\longrightarrow$ AST$_1$ $\longrightarrow$ *female(kim)$\wedge$ male(kim)*

*"Kim ist Ahmeds **Cousine** und Graces Vater"*

AST$_2$ $\longrightarrow$ *male(kim)$\wedge$ male(kim)*

*"Kim ist Ahmeds **Cousin** und Graces Vater"*

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

"Kim is Ahmed's cousin and the father of Grace" → ~~AST₁~~ → ~~female(kim)∧ male(kim)~~

~~"Kim ist Ahmeds **Cousine** und Graces Vater"~~

AST₂ → male(kim)∧ male(kim)

"Kim ist Ahmeds **Cousin** und Graces Vater"

Example: Input Language for SageMath

- Can we make a natural input language for SageMath?   *WolframAlpha-like*

```
sage: g = AlternatingGroup (5)
sage: g.cardinality ()
60
```

"Let G be the alternating group on 5 symbols. What is the cardinality of G?"

Example: Input Language for SageMath
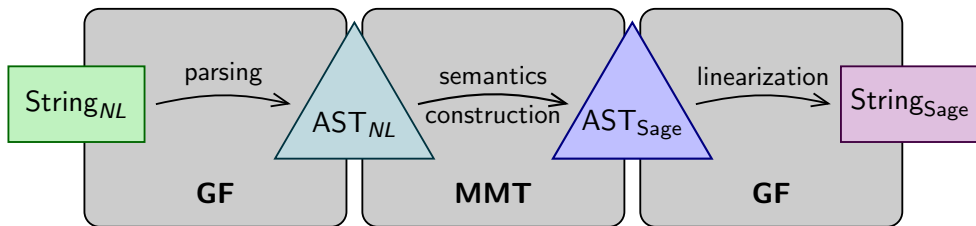
Example: Input Language for SageMath

```
> Let G be the alternating group on 5 symbols.
# G = AlternatingGroup(5)

> Let |H| be a notation for the cardinality of H.
# def bars(H): return H.cardinality()

> What is |G|?
# print(bars(G))
60

> Let A_N be a notation for the alternating group on N symbols.
# def A(N): return AlternatingGroup(N)

> What are the cardinalities of A_4 and A_5?
# print(A(4).cardinality()); print(A(5).cardinality())
12
60
```
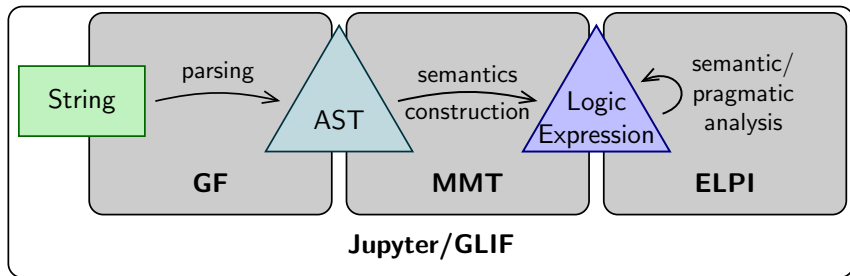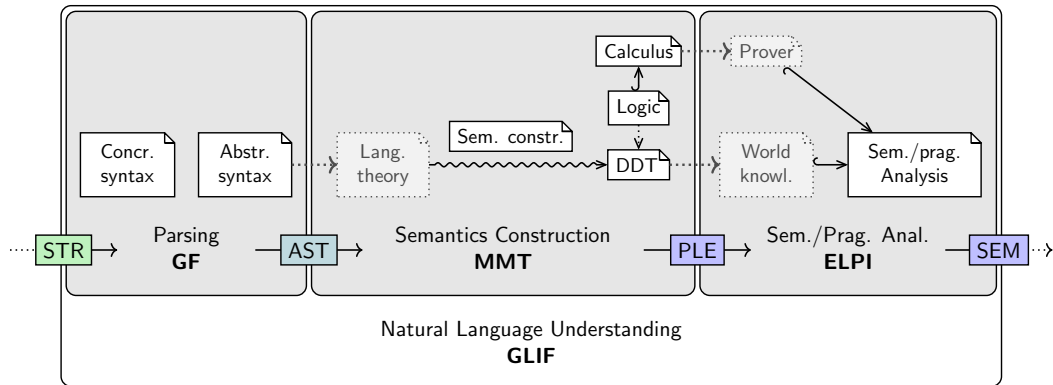
Conclusion

Summary:

- GLIF = GF + MMT + ELPI
- Prototyping natural language semantics
- We use it for teaching

Examples:

1. *"a kinetic energy of 12mN"*
2. *"He loves her"* (tableaux machine)
3. *"John knows that Eve has a dog"*
4. *"Ahmed's cousin"*
5. *"What is the cardinality of G?"*

Pipeline Specification

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

$$\frac{A \vee B \quad \overset{[A]^1}{\underset{C}{\vdots}} \quad \overset{[B]^1}{\underset{C}{\vdots}}}{C} \vee E^1$$

```
// ⊢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 $\wedge El : \Pi_{A:o}\Pi_{B:o} \vdash A\wedge B \rightarrow \vdash A$

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 $\wedge$El : $\Pi_{A:o}\Pi_{B:o}$ $\vdash A \wedge B \rightarrow \vdash A$

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: $\vee$E : $\Pi_{A:o}\Pi_{B:o}\Pi_{C:o}$ $\vdash A \vee B \rightarrow (\vdash A \rightarrow \vdash C) \rightarrow (\vdash B \rightarrow \vdash C) \rightarrow \vdash C$

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

Example: Forall-Introduction

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

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

$$\frac{A \wedge B^{F}}{A^{F} \mid B^{F}} \wedge^{F} \qquad\qquad \frac{A \wedge B^{F} \quad \overset{[A^{F}]}{\underset{\perp}{\vdots}} \quad \overset{[B^{F}]}{\underset{\perp}{\vdots}}}{\perp} \wedge^{F}$$

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

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!
$\to$ Other helpers result in more efficient provers

References I

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

📄 *Case study of implementing intensional logic in GLIF*. URL: https://github.com/us77ipis/glif-intensional-logic (visited on 07/19/2022).

📄 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.

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

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

📄 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.