Language Research in the KWARC group

Jan Frederik Schaefer

FAU Erlangen-Nürnberg/KWARC

Workshop: Approaches to the Logic and Syntax of Mathematical Texts

Erlangen

Dec. 6, 2022

Corpus work with arxiv at KWARC

arxiv.org:

- Open-access pre-print server
- $> 2,000,000$ scientific articles
- Fields: physics, mathematics, computer science, ...
- LaTeX sources
- $\rightsquigarrow$ a great corpus

Problem:

This: *"The average is $\frac{A+B}{2}$."*

Could be written like this:
```
The average is $\frac{A+B}{2}$.
```

Or like this:
```
\def\avg#1#2{\ensuremath{\frac{#1+#2}2}}
% ...
The average is \avg AB.
```

with $Z(\beta, \alpha, \underline{\lambda})$ a normalization constant. Plugging this solution into the expression (C4) of G we get:

$$G(\underline{p}, \underline{\lambda}) = \sum_{i=1}^{L} \sum_{y} \lambda_i(y) p_i(y) + \sum_{i=1}^{L-1} \sum_{y,y'} \lambda_{i,i+1}(y, y') p_{i,i+1}(y, y') - \frac{1}{\beta} \log Z(\beta, \alpha, \underline{\lambda}) \ . \tag{C7}$$

The condition on the Lagrange multipliers is finally obtained by looking at the stationary points of G with respect to the $\lambda_i(y)$'s, $\lambda_{i,i+1}(y, y')$'s. Let $\underline{\lambda}^*(\alpha)$ be a set of Lagrange multipliers achieving the stationary point, we then have

$$G(\underline{p}) = G(\underline{p}, \underline{\lambda}^*(\alpha)) \ ,$$

where we have emphasized the dependence in α of the Lagrange multipliers.

Example from [BP22]

Problem:

This: *"The average is $\frac{A+B}{2}$."*

Could be written like this:
```
The average is $\frac{A+B}{2}$.
```

Or like this:
```
\def\avg#1#2{\ensuremath{\frac{#1+#2}2}}
% ...
The average is \avg AB.
```

Solution: Convert to more managable format: HTML with MathML.

HTML, MathML

"The average is $\frac{A+B}{2}$*."*

```
<p>The average is <math>...</math>.</p>
```

Presentation MathML

```
<mfrac>
    <mrow>
        <mi>A</mi>
        <mo>+</mo>
        <mi>B</mi>
    </mrow>
    <mn>2</mn>
</mfrac>
```

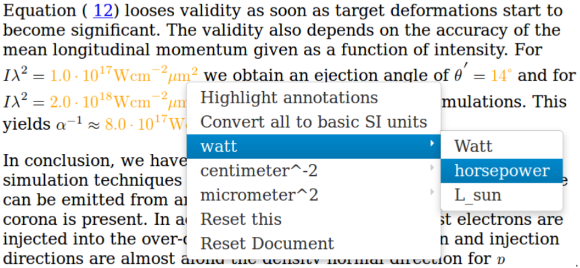
Content MathML

```
<apply>
    <divide/>
    <apply>
        <plus/>
        <ci>A</ci>
        <ci>B</ci>
    </apply>
    <cn type="integer">2</cn>
</apply>
```

ar5iv corpus

- Use LaTeXML to convert arxiv to HTML+MathML *Done by Deyan Ginev*
- $\rightsquigarrow$ ar5iv corpus $\approx 2 \cdot 10^6$ *documents*
- Goal: Extract semantic information and provide services

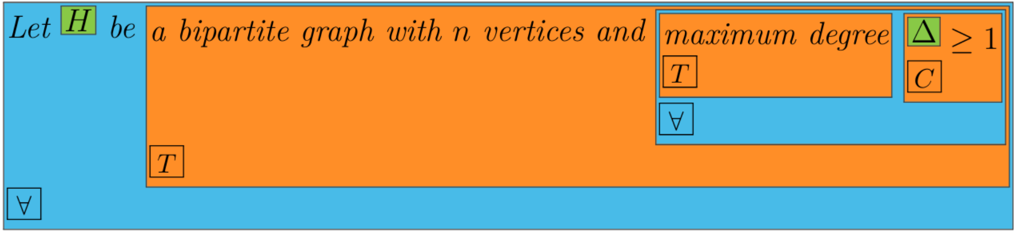
search, interactive documents, ...



Screenshot from [MK]

- Use LaTeXML to convert arxiv to HTML+MathML *Done by Deyan Ginev*
- $\rightsquigarrow$ ar5iv corpus $\approx 2 \cdot 10^6$ *documents*
- Goal: Extract semantic information and provide services

  search, interactive documents, . . .



Screenshot from [Sch16]

—And Now for Something Completely Different—

GLIF: A tool for prototyping natural language semantics

Natural Language Semantics (Symbolic)

For me:

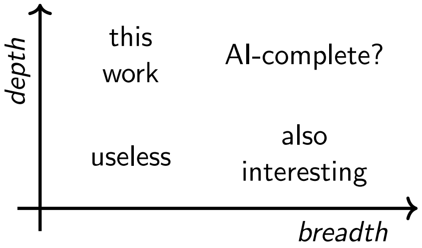
Translating natural language into a formal semantic representation (logic).

Example:

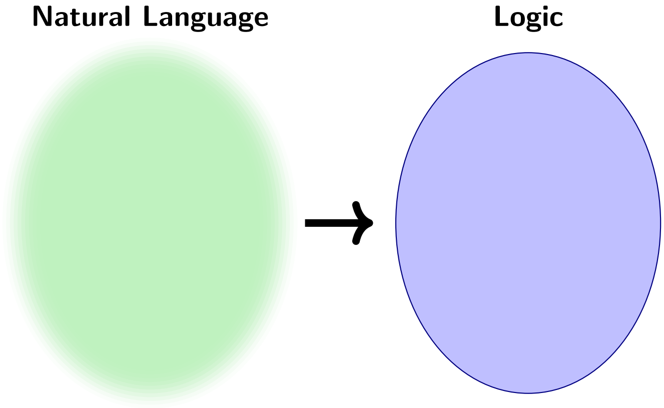
"Every student paints and is quiet." $\rightsquigarrow$ $\forall x.s(x) \Rightarrow (p(x) \wedge q(x))$

Rule-based (no ML):

Parsing $\rightsquigarrow$ semantics construction $\rightsquigarrow$ inference.



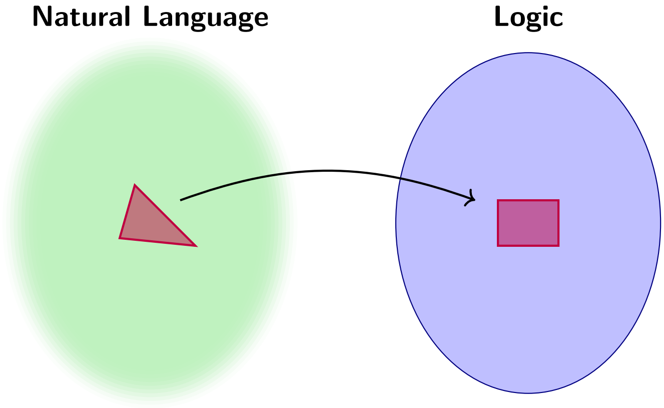
Method of Fragments



How do we get from messy language to formal logic?

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

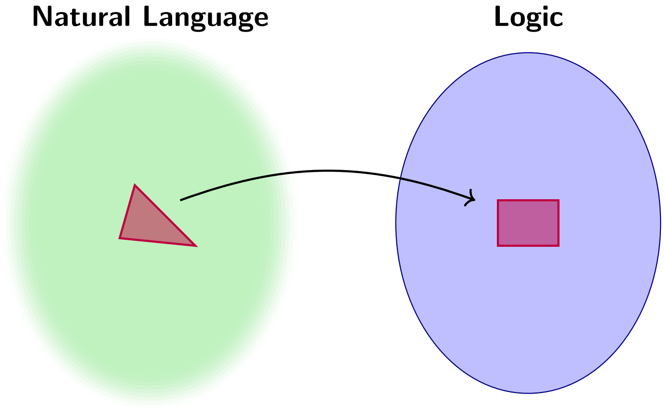
Method of Fragments



How do we get from messy language to formal logic?

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

Method of Fragments

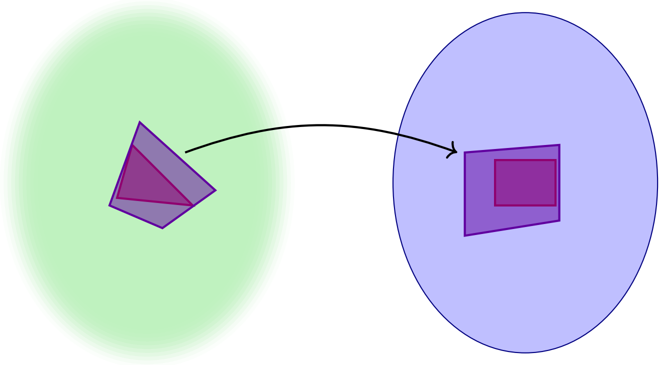


"Ahmed paints and Berta is quiet." $p(a) \wedge q(b)$

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

Method of Fragments

Natural Language **Logic**

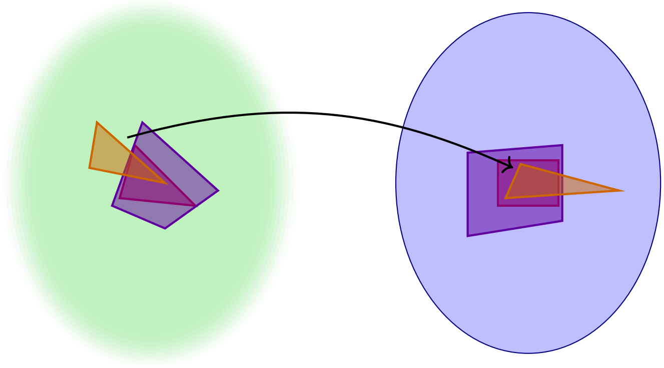


"Every student paints and is quiet." $\forall x. s(x) \Rightarrow (p(x) \wedge q(x))$

"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." $(\Box p(a)) \land \Box p(b)$

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 $\phi, \psi \in P_t$ 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 $\rightsquigarrow$ **We need prototyping!**

NLU Prototyping

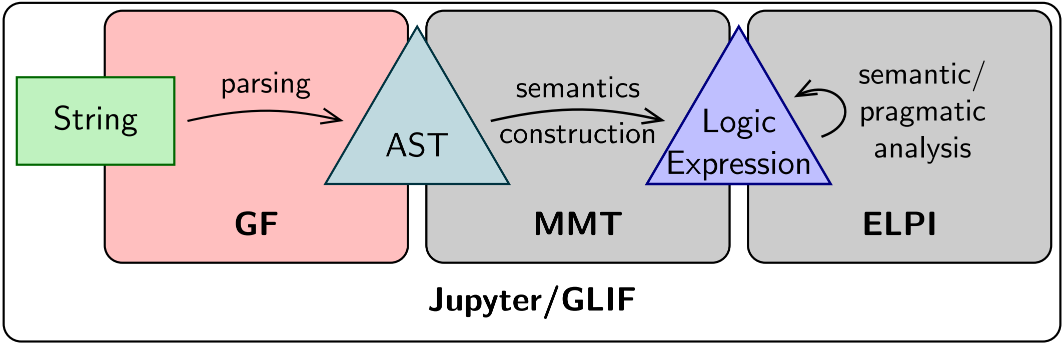
```
> translate "Every student paints and is quiet."
```
$\forall x.s(x) \Rightarrow (p(x) \wedge q(x))$

```
> answer "Every student is quiet.  John is a student.  Is John quiet?"
```
$\forall x.s(x) \Rightarrow q(x), s(j) \vdash^{?} q(j)$

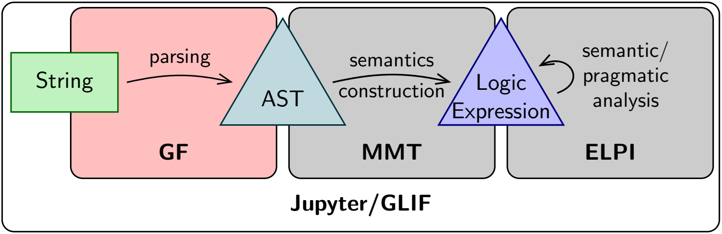
```
yes
```

- 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: GF

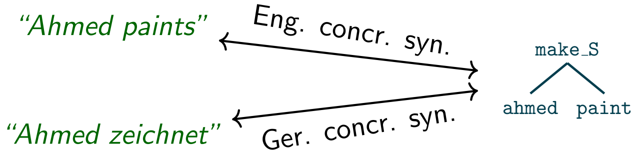


Components of GLIF: Grammatical Framework [GF]

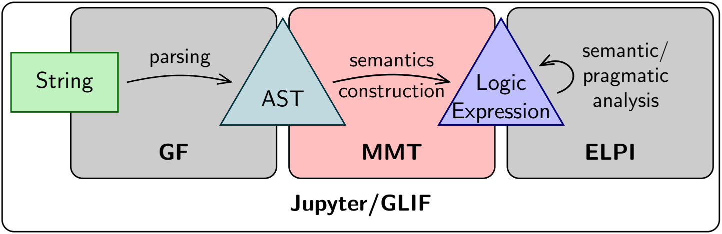
- Specialized for developing natural language grammars
- Separates abstract and concrete syntax

```
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 — $\geq$ 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

GF		**MMT**

`cat`
`  NP; VP; S;`
`fun`
`  make_S :`
`    NP -> VP -> S;`
	$\longmapsto$	NP : *type*
		VP : *type*
		S : *type*
		make_S :
		  NP $\rightarrow$ VP $\rightarrow$ 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

$$
\begin{array}{lcl}
o & : & \mathit{type} \quad \textit{//propositions} \\
\neg & : & o \to o \\
\wedge & : & o \to o \to o \\
\vee & : & o \to o \to o \\
 & & \\
\iota & : & \mathit{type} \quad \textit{//individuals} \\
\forall & : & (\iota \to o) \to o \\
\exists & : & (\iota \to o) \to o
\end{array}
$$

Discourse Domain

$$
\begin{array}{lcl}
\mathtt{paint} & : & \iota \to o \\
\mathtt{quiet} & : & \iota \to o \\
\mathtt{ahmed} & : & \iota \\
\mathtt{berta} & : & \iota
\end{array}
$$

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`	$\mapsto$	`o`
`NP`	$\mapsto$	ι
`VP`	$\mapsto$	$\iota \to$ `o`
`make_S`	$\mapsto$	λ`n.`λ`v.v n`
`ahmed`	$\mapsto$	`ahmed`

More advanced

`NP`	$\mapsto$	$(\iota \to$ `o`$) \to$ `o`
`sentence`	$\mapsto$	λ`n.`λ`v.n v`
`everyone`	$\mapsto$	λ`p.`$\forall\lambda$`x.p x`
`berta`	$\mapsto$	λ`p.p berta`

Example: Parsing + Semantics Construction

"Ahmed and Berta paint"

$\downarrow$parsing

make_S (andNP ahmed berta) paint

$\downarrow$semantics construction

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

$\downarrow\beta$-reduction

`paint ahmed ∧ 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)

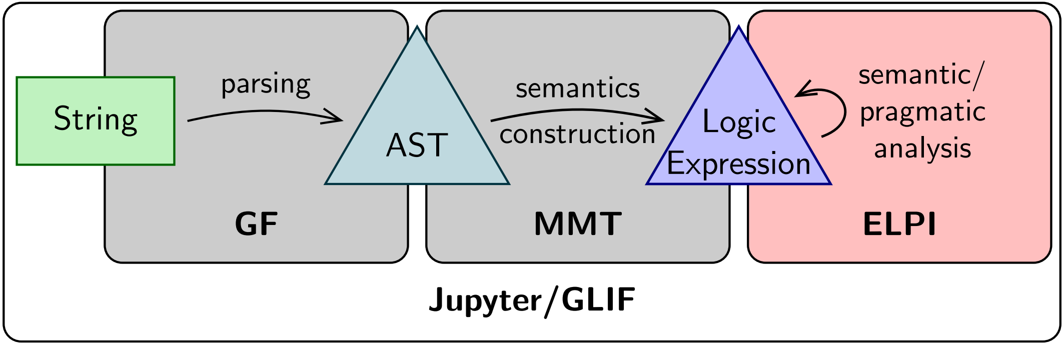
```
parse "Mary saw that John would kill the dog" | construct
```

P 👁mary👁F (kill john dog)

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

(run mary)∧👁john👁(run mary)

Components of GLIF: ELPI



Components of GLIF: ELPI

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

MMT	ELPI
$o : type$ *//propositions*	`kind o type.`
$\neg : o \to o$	`not : o -> o.`
$\wedge : o \to o \to o$	`and : o -> o -> o.`
$\vee : o \to o \to o$	`or  : o -> o -> o.`
$\iota : type$ *//individuals*	`kind i type.`
$\forall : (\iota \to o) \to o$	`type forall (i -> o) -> o.`
$\exists : (\iota \to o) \to o$	`type exists (i -> o) -> o.`

Example: Discard wrong readings in controlled natural language

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

Example: Discard wrong readings in controlled natural language

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

"a kinetic energy of 12mN" $\longrightarrow$ AST_1 $\longrightarrow$ $\lambda x.E_{\mathrm{kin}}(x, \mathrm{quant}(2, \mathbf{milli\ Newton}))$

"a kinetic energy of 12mN" $\longrightarrow$ AST_2 $\longrightarrow$ $\lambda x.E_{\mathrm{kin}}(x, \mathrm{quant}(2, \mathbf{meter{\cdot}Newton}))$

Example: Discard wrong readings in controlled natural language

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

"a kinetic energy of 12mN" $\longrightarrow$ AST_1 $\longrightarrow$ $\lambda x.E_{\mathsf{kin}}\cancel{(x, \mathsf{quant}(2, \mathbf{milli\ Newton}}))$

"a kinetic energy of 12mN" $\longrightarrow$ AST_2 $\longrightarrow$ $\lambda x.E_{\mathsf{kin}}(x, \mathsf{quant}(2, \mathbf{meter} \cdot \mathbf{Newton}))$

Example: Discard wrong readings in controlled natural language

In [20]:

```
parse "the ball has a mass of 5 k g and a kinetic energy of 12 m N" |
        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]:

```
parse "the ball has a mass of 5 k g and a kinetic energy of 12 m N" |
        construct | filter -predicate=filter_pred
```

(mass theball (quant 5 kilo gram)) ∧ (ekin theball (quant 12 meter·Newton))

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)

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

$S_1, S_2 \vdash_{S5_n} Q$	$\leadsto$	yes
$S_1, S_2 \vdash_{S5_n} \neg Q$	$\leadsto$	no
else	$\leadsto$	maybe

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"

$$
\begin{array}{ccc}
 & p(a) \vee p(b)^T & \\
 \swarrow & & \searrow \\
p(a)^T & & p(b)^T
\end{array}
$$

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

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

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

$p(a)^F$

$\perp$

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"

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

$$\begin{array}{cc} p(a)^T & p(b)^T \\ | & | \\ \neg p(a)^T & \neg p(a)^T \\ | & | \\ p(a)^F & p(a)^F \\ | & \\ \perp & \end{array}$$

"Ahmed doesn't paint"

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."

$sad(s)$

$$\begin{array}{c} \forall x.fem(x) \Rightarrow \neg masc(x) \\ masc(j) \\ fem(m) \\ \downarrow \\ talkto(j, m) \\ \downarrow \\ sad(s) \end{array}$$

Example: Tableaux Machine

Background Knowledge

"John talks to Mary."

$talkto(j, m)$

"Sasha is sad."

$sad(s)$

"He loves her."

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

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

$$\downarrow$$

$$talkto(j, m)$$

$$\downarrow$$

$$sad(s)$$

$$\swarrow \qquad \searrow$$

$$\begin{array}{cc} love(s, s) & \\ masc(s) & love(s, m) \\ fem(s) & masc(s) \\ \bot & \end{array}$$

Example: Tableaux Machine

Background Knowledge

"John talks to Mary."

$talkto(j, m)$

"Sasha is sad."

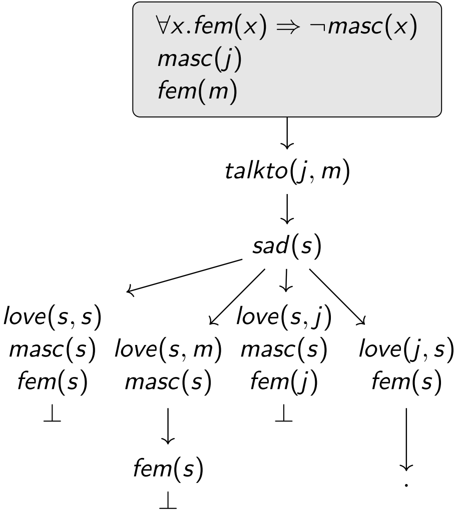
$sad(s)$

"He loves her."

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

"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."

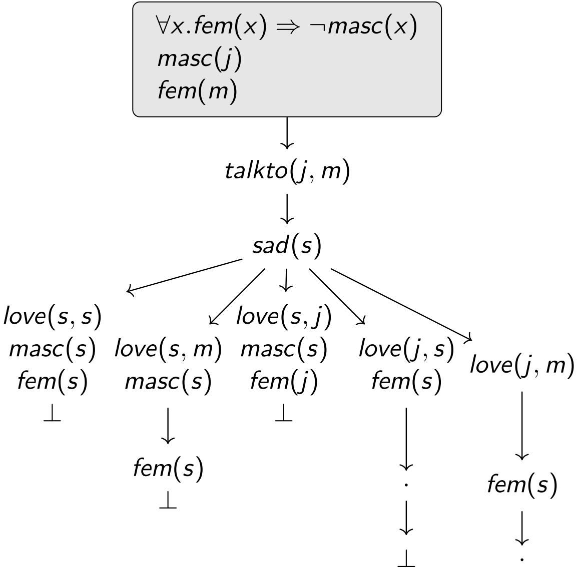
$\exists X.masc(X) \wedge$
$\quad \exists Y.fem(Y) \wedge love(X, Y)$

"Sasha is a woman."

$fem(s)$

"John doesn't love Sasha."

$\neg love(j, s)$



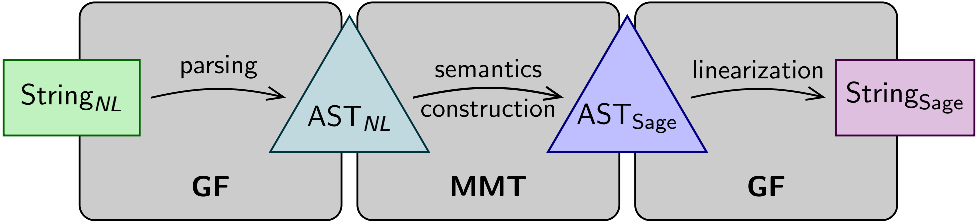
Example: Input Language for SageMath

- 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
```

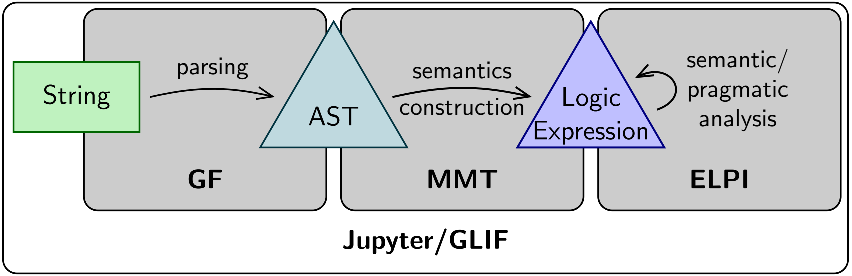
Conclusion

Summary:

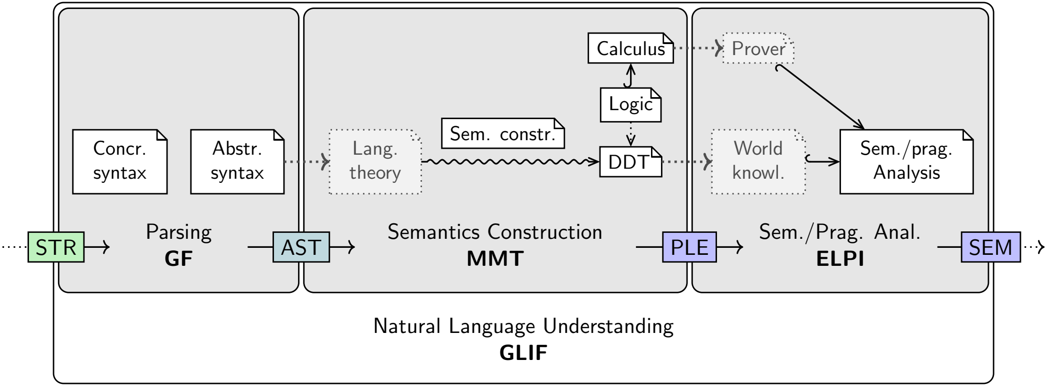
- GLIF = GF + MMT + ELPI
- Prototyping natural language semantics
- We use it for teaching

Examples:

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



Pipeline Specification



References I

[BP22] Louise Budzynski and Andrea Pagnani. *Small Coupling Expansion for Multiple Sequence Alignment*. 2022. DOI: 10.48550/ARXIV.2210.03463. URL: https://arxiv.org/abs/2210.03463.

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

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

[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.

References II

[MK] Tom Wiesing Michael Kohlhase. *Report on OpenDreamKit deliverable D4.9, in-place computation in active documents (context/computation)*. URL: https://raw.githubusercontent.com/OpenDreamKit/OpenDreamKit/master/WP4/D4.9/report-final.pdf (visited on 12/02/2022).

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

[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.

[Sch16] Jan Frederik Schaefer. "Declaration Spotting in Mathematical Documents". B. Sc. Thesis. Jacobs University Bremen, 2016. URL: https://gl.kwarc.info/supervision/BSc-archive/blob/master/2016/schaefer-frederick.pdf.

References III

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