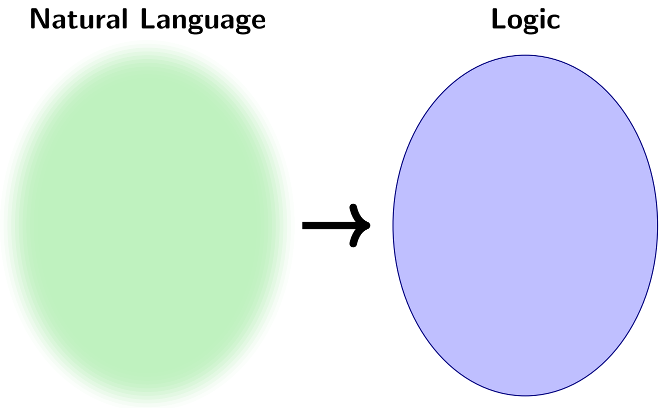
GLIF: A Framework for Prototyping Symbolic Natural Language Understanding

Jan Frederik Schaefer

FAU Erlangen-Nürnberg

Prospects of Formal Mathematics – Bridging between informal and formal
Hausdorff Research Institute for Mathematics
Bonn
July 9, 2024

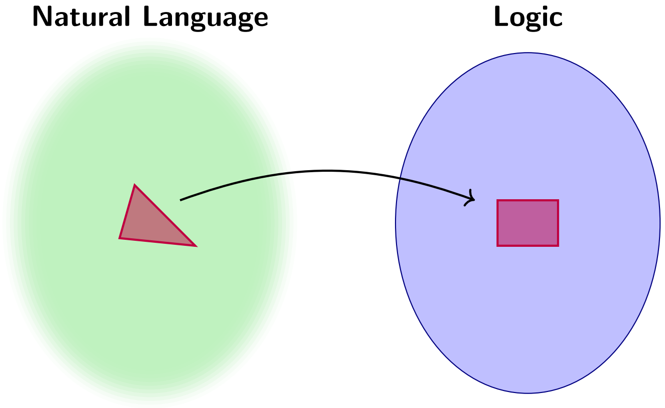
Method of Fragments



How do we get from messy language to formal logic?

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

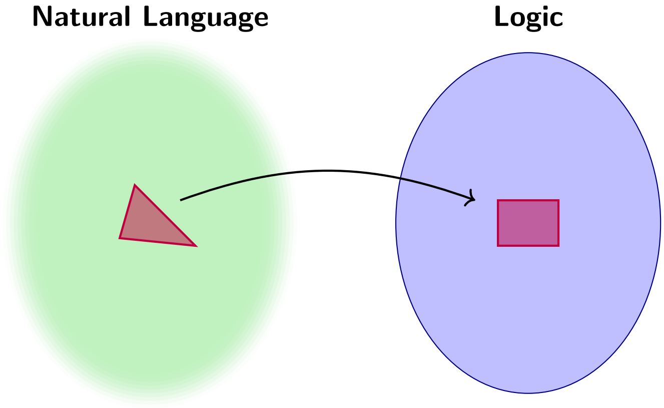
Method of Fragments



How do we get from messy language to formal logic?

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

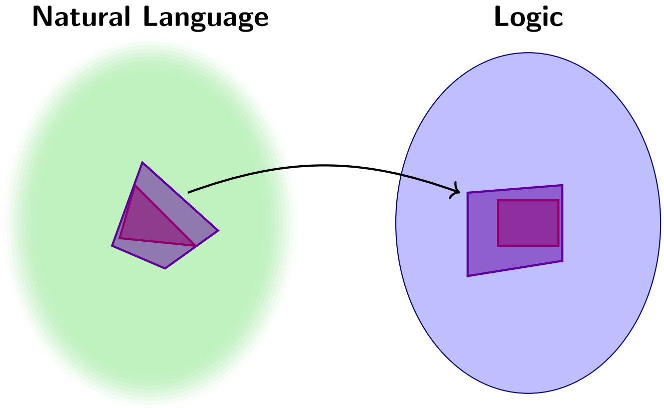
Method of Fragments



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

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

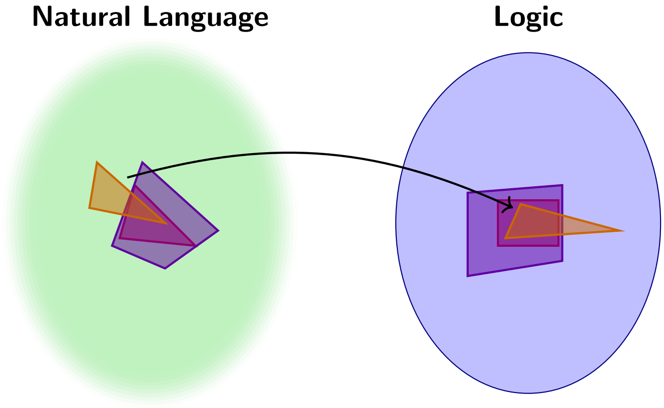
Method of Fragments



"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 $\phi, \psi \in P_t$ 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

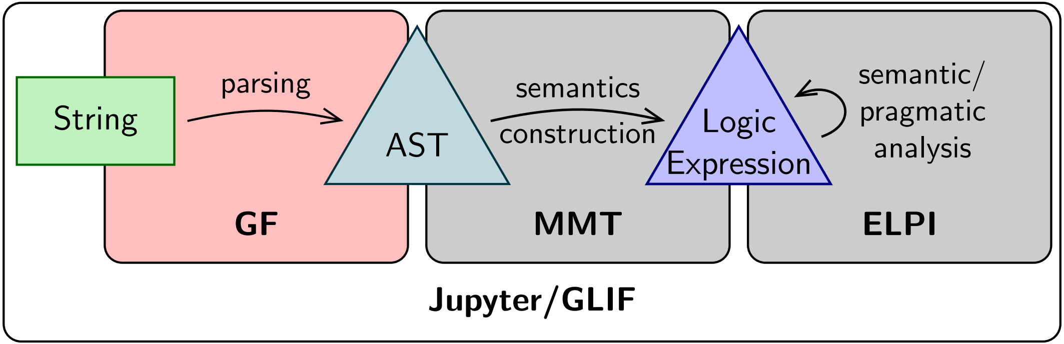
```
> my-translate "Every student paints and is quiet."
```
$\forall x.s(x) \Rightarrow (p(x) \wedge q(x))$

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

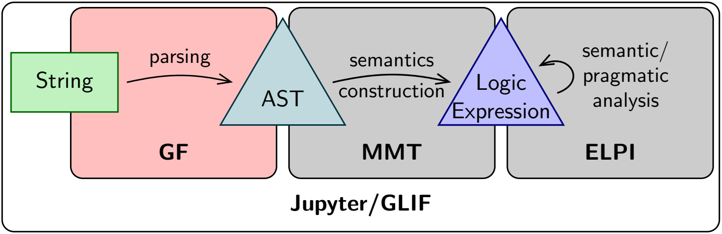
```
yes
```

- 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



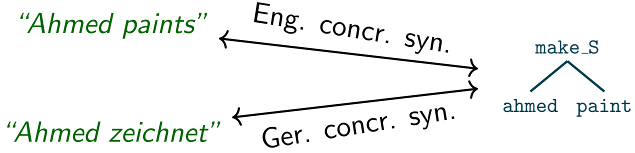
Components of GLIF: Grammatical Framework [GF]

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

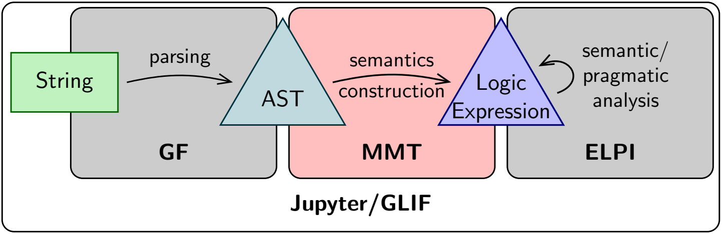
```
make_S : NP -> VP -> S;
make_S np vp = np.s ++ vp.s!np.n;
```

abstract / *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 $\longmapsto$ **MMT**

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

$$
\begin{array}{lll}
\mathtt{NP} & : & \mathit{type} \\
\mathtt{VP} & : & \mathit{type} \\
\mathtt{S} & : & \mathit{type} \\
\mathtt{make\_S} & : & \\
\quad \mathtt{NP} \to \mathtt{VP} \to \mathtt{S} & &
\end{array}
$$

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

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

Discourse Domain

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

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	$\mapsto$	$\lambda n.\lambda v.v\ n$
ahmed	$\mapsto$	ahmed

More advanced

NP	$\mapsto$	$(\iota \to o) \to o$
sentence	$\mapsto$	$\lambda n.\lambda v.n\ v$
everyone	$\mapsto$	$\lambda p.\forall\lambda x.p\ x$
berta	$\mapsto$	$\lambda p.p\ \text{berta}$

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`

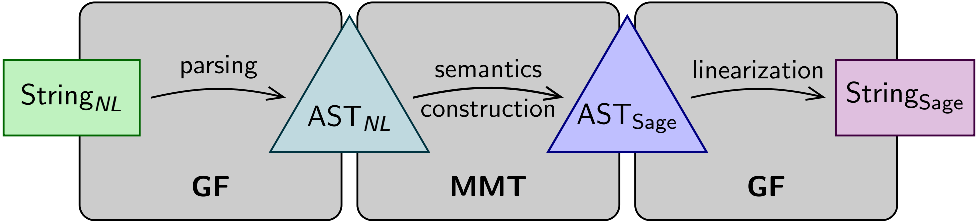
Example: Input Language for SageMath

- 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?"

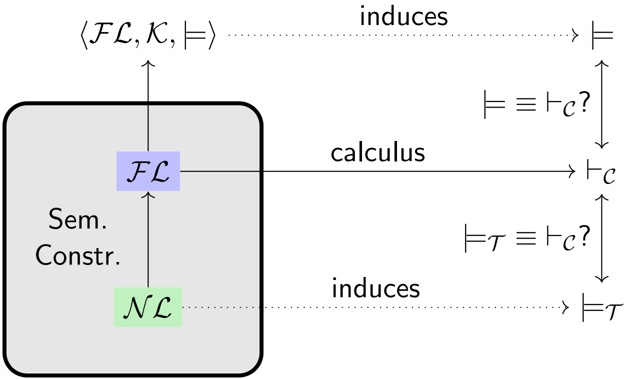
Example: Input Language for SageMath



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
```

Levels of inference

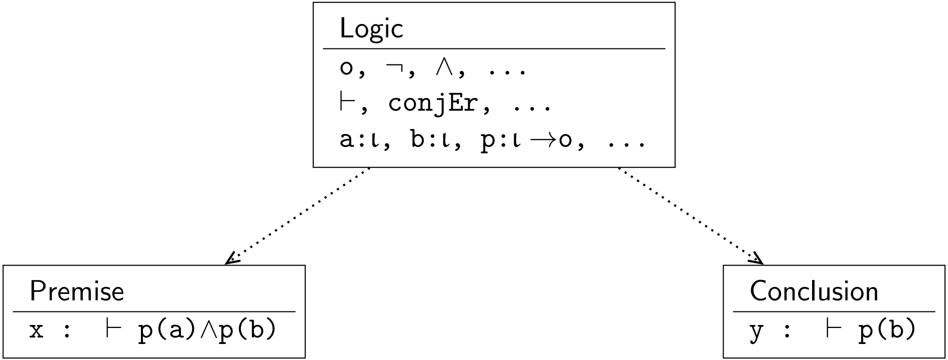


1. Test: Does *"Ahmed and Berta paint."* $\models_{\mathcal{T}}$ *"Berta paints."*?
2. Model prediction: Yes, because $p(a) \wedge p(b) \vdash_{\mathcal{C}} \mathrm{p(b)}$.
3. Correct result: Ask people.

Natural deduction in MMT: *“Judgments as types”*

```
⊢ : o → type

s1 : ⊢ p(a)∧p(b)

conjEr : {A:o} → {B:o} → ⊢A∧B → ⊢B

s2 : ⊢ p(b)
   = conjEr p(a) p(b) s1
```

$[\![$ *"Ahmed and Berta paint"* $]\!] \vdash_{\mathcal{ND}} [\![$ *"Berta paints"* $]\!]$



$[\![$ *“Ahmed and Berta paint”* $]\!] \vdash_{\mathcal{ND}} [\![$ *“Berta paints”* $]\!]$

Logic

`o, ¬, ∧, ...`

`⊢, conjEr, ...`

$\mathtt{a}:\iota$, $\mathtt{b}:\iota$, $\mathtt{p}:\iota\to\mathtt{o}$, ...

Premise

$\mathtt{x} \;:\; \vdash \mathtt{p(a)} \wedge \mathtt{p(b)}$

Conclusion

$\mathtt{y} \;:\; \vdash \mathtt{p(b)}$

$\boxed{\mathtt{y} \mapsto \mathtt{conjEr}\ \mathtt{p(a)}\ \mathtt{p(b)}\ \mathtt{x}}$

$[\![$ *"Ahmed paints"* $]\!]$, $[\![$ *"Berta knows Ahmed"* $]\!] \nvdash_{\mathcal{ND}} [\![$ *"Berta knows everyone who paints"* $]\!]$

Logic

...

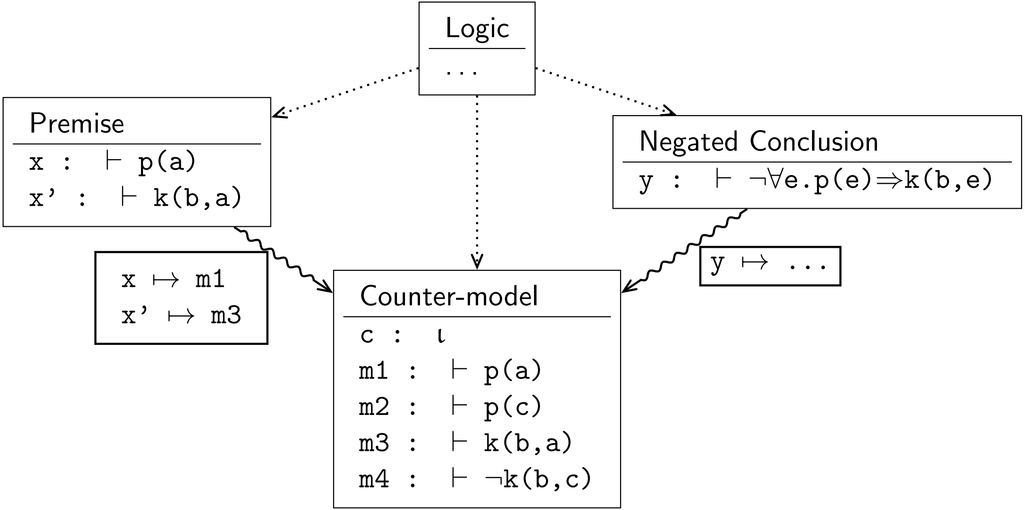
Premise

```
x  :  ⊢ p(a)
x' :  ⊢ k(b,a)
```

Negated Conclusion

```
y  :  ⊢ ¬∀e.p(e)⇒k(b,e)
```

$[\![$ *"Ahmed paints"* $]\!]$, $[\![$ *"Berta knows Ahmed"* $]\!]$ $\nvdash_{\mathcal{ND}}$ $[\![$ *"Berta knows everyone who paints"* $]\!]$

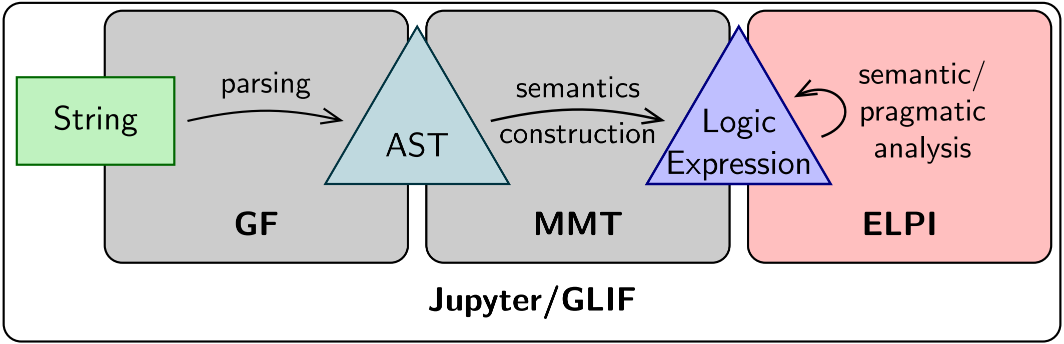


Mini summary

- Parsing with GF
- Logic syntax in MMT
- Semantics construction in MMT
- (Manual) inference in MMT

"Bring your own logic"

Components of GLIF: ELPI



Components of GLIF: ELPI

- Implementation and extension of λProlog $\approx$ *Prolog + HOAS*
- MMT can generate logic signatures
- 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 $\longrightarrow$ $\mathsf{mass}(\mathsf{theball}, \mathsf{quant}(5, \mathsf{kilo\ gram}))$

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 $\longrightarrow$ $\lambda x. E_{\mathrm{kin}}(x, \mathrm{quant}(12, \mathbf{milli\ Newton}))$

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

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 $\longrightarrow$ $\lambda x.E_{\mathrm{kin}}\cancel{(x, \mathrm{quant}(12, \mathbf{milli\ Newton}}))$

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

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: ForTheL

```
parse -cat=DefinitionStatement "a subset of S is a set T such that every element of T belongs to S"
```

```
∀[V_T:ι](subset V_T V_S)⇔(set V_T)∧∀[V_new:ι](element V_new V_T)∧⊤⇒(belongTo V_new V_S)∧⊤
```

```
parse -cat=Statement "there exists an empty set" | construct -v semantics/forthelUnsortedSem
```

```
∃[V_new:ι]((empty V_new)∧(set V_new))∧⊤
```

```
parse -cat=Statement "S is a subset of every set iff S is empty" | construct -v semantics/forthelUn
```

```
(∀[V_new:ι](set V_new)∧⊤⇒(subset V_S V_new)∧⊤)⇔(empty V_S)
```

Example: “pairwise disjoint”

“A, B and C are pairwise disjoint”

$\text{disjoint}(A, B) \wedge \text{disjoint}(A, C) \wedge \text{disjoint}(B, C)$

- **Approach 1**

  Semantics construction with lots of λs: *difficult!*

  $$\text{disjoint}(A, B) \wedge \text{disjoint}(A, C) \wedge \top \wedge \text{disjoint}(B, C) \wedge \top \wedge \top \wedge \top$$

  Simplify with ELPI:

  $$\text{disjoint}(A, B) \wedge \text{disjoint}(A, C) \wedge \text{disjoint}(B, C)$$

Example: "pairwise disjoint"

"A, B and C are pairwise disjoint"

$\text{disjoint}(A, B) \wedge \text{disjoint}(A, C) \wedge \text{disjoint}(B, C)$

- **Approach 1**
  Semantics construction with lots of λs: *difficult!*
  $\text{disjoint}(A, B) \wedge \text{disjoint}(A, C) \wedge \top \wedge \text{disjoint}(B, C) \wedge \top \wedge \top \wedge \top$
  Simplify with ELPI:
  $\text{disjoint}(A, B) \wedge \text{disjoint}(A, C) \wedge \text{disjoint}(B, C)$
- **Approach 2**
  Semantics construction creates preliminary expression:
  `relNT disjoint (cons A (cons B (cons C nil)))`
  Convert with ELPI: *easier*
  $\text{disjoint}(A, B) \wedge \text{disjoint}(A, C) \wedge \text{disjoint}(B, C)$

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	$\leadsto$	unknown

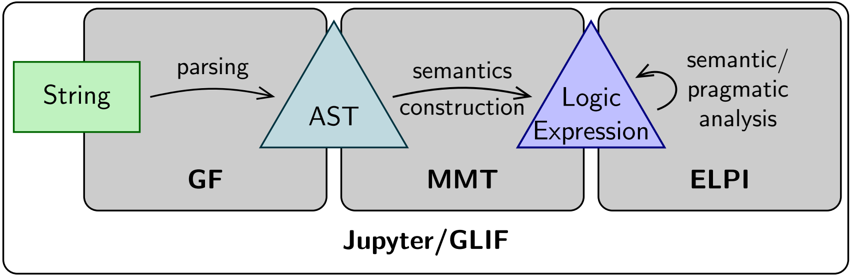
Conclusion

Summary:

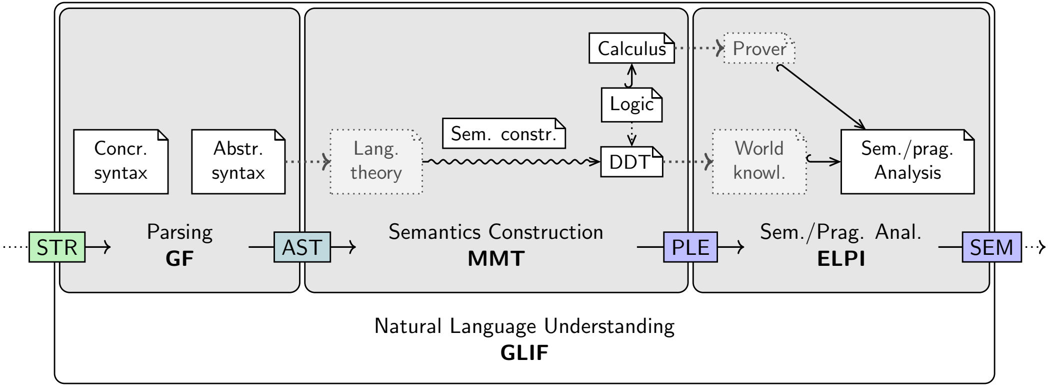
- GLIF = GF + MMT + ELPI
- Prototyping natural language understanding
- We use it for teaching

Examples:

1. *"What is the cardinality of G?"*
2. *"a kinetic energy of 12mN"*
3. *"A, B and C are pairwise disjoint"*
4. *"John knows that Eve has a dog"*



Pipeline Specification



Example: Tableaux Machine [**KohKol:ramgpm03**]

- 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 [**KohKol:ramgpm03**]

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

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

$$p(a)^T$$

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

$$p(a)^F$$

$$\perp$$

Example: Tableaux Machine [**KohKol:ramgpm03**]

- 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"

$$\begin{array}{ccc}
 & p(a) \vee p(b)^T & \\
p(a)^T & & p(b)^T \\
\neg p(a)^T & & \neg p(a)^T \\
p(a)^F & & p(a)^F \\
\bot & &
\end{array}$$

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."

$sad(s)$

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

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."

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

$$\downarrow$$
$$talkto(j, m)$$
$$\downarrow$$
$$sad(s)$$

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

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."

$fem(s)$

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

$\downarrow$

$talkto(j, m)$

$\downarrow$

$sad(s)$

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

Example: Tableaux Machine

Background Knowledge

"John talks to Mary."

$talkto(j, m)$

"Sasha is sad."

$sad(s)$

"He loves her."

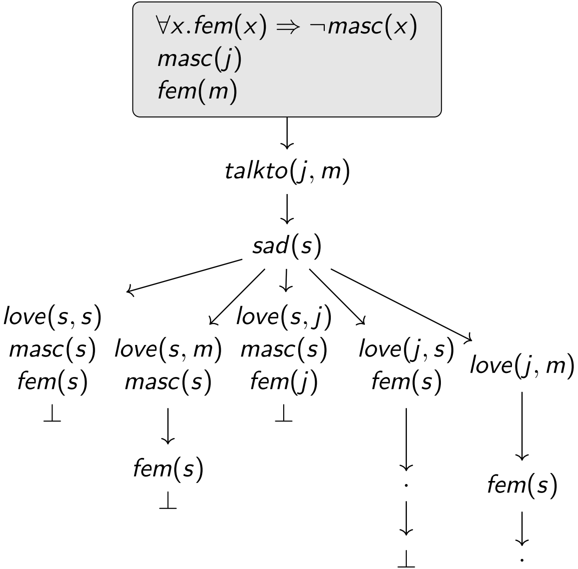
$\exists X.masc(X) \wedge$
$\quad \exists Y.fem(Y) \wedge love(X, Y)$

"Sasha is a woman."

$fem(s)$

"John doesn't love Sasha."

$\neg love(j, s)$



Example: 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”

AST_1

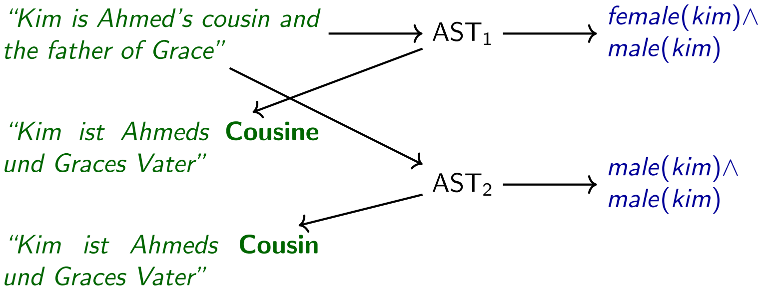
“Kim ist Ahmeds ***Cousine*** *und Graces Vater”*

AST_2

“Kim ist Ahmeds ***Cousin*** *und Graces Vater”*

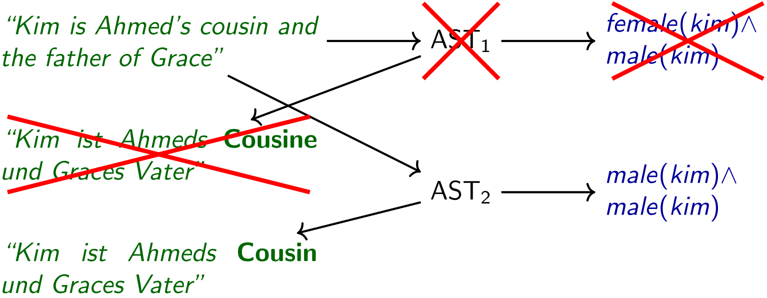
Example: Translation

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



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



Natural Deduction in MMT/LF

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

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

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

ELPI equivalent

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

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

Generating Provers in ELPI

LF rule $\wedge\mathtt{El} : \mathbf{\Pi}_{\mathtt{A:o}}\mathbf{\Pi}_{\mathtt{B:o}}\ \vdash\mathtt{A}\wedge\mathtt{B} \to \vdash\mathtt{A}$

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: $\vee\mathtt{E} : \mathbf{\Pi}_{\mathtt{A:o}}\mathbf{\Pi}_{\mathtt{B:o}}\mathbf{\Pi}_{\mathtt{C:o}}\ \vdash\mathtt{A}\vee\mathtt{B} \to (\vdash\mathtt{A} \to \vdash\mathtt{C}) \to (\vdash\mathtt{B} \to \vdash\mathtt{C}) \to \vdash\mathtt{C}$

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

Example: Forall-Introduction

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

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

$$\frac{A \wedge B^F}{A^F \mid B^F}\wedge^F \qquad\qquad \dfrac{A \wedge B^F \quad \begin{matrix}[A^F] \\ \vdots \\ \bot\end{matrix} \quad \begin{matrix}[B^F] \\ \vdots \\ \bot\end{matrix}}{\bot}\wedge^F$$

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

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

References I

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

[Mon70] R. Montague. "English as a Formal Language". In: Reprinted in [**Thomason:fp74**], 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.