A Symbolic Framework for Mathematical Language Understanding

Jan Frederik Schaefer

FAU Erlangen-Nürnberg

SIGMathLing Seminar
virtual event
January 18, 2021

A Case for Symbolic Approaches

"Every integer is even." $\rightsquigarrow$ $\forall x.int(x) \Rightarrow even(x)$

+ No need for training data
+ No need for resource-heavy training
+ Verifiable, predictable, accurate
− Coverage very limited

Sometimes the pros outweigh the cons:

- Need for high reliability
  - Proof verification
  - Fabstracts
  - ...
- Prototyping

CNLs

A Case for Symbolic Approaches

"Every integer is even." $\rightsquigarrow$ $\forall x.int(x) \Rightarrow even(x)$

- \+ No need for training data
- \+ No need for resource-heavy training
- \+ Verifiable, predictable, accurate
- − Coverage very limited

Sometimes the pros outweigh the cons:

- Need for high reliability *CNLs*
    - Proof verification
    - Fabstracts
    - ...
- Prototyping

GLIF: A framework for prototyping symbolic NLU

Teaser: Input Language for SageMath

```
Enter command: Let G be the dihedral group of order 8.
gVar = DihedralGroup(int(8)//2)
```

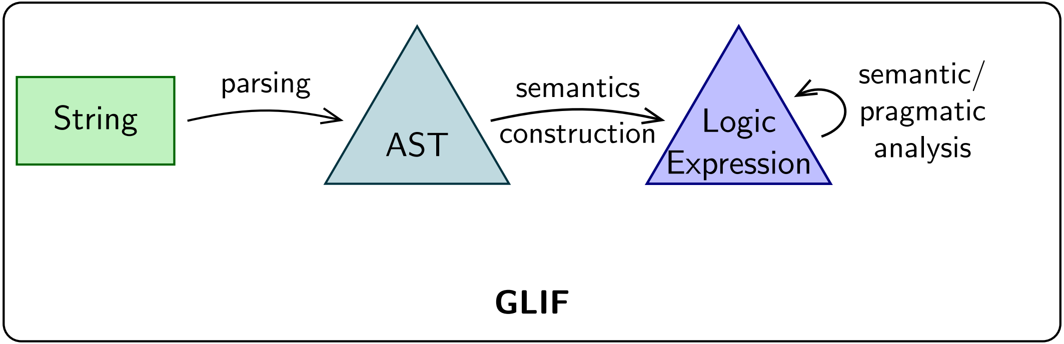
```
Enter command: Let A_N be a notation for the alternating group on N symbols.
def aVar(nVar): return AlternatingGroup(nVar)
```

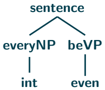
```
Enter command: What are the cardinalities of G and A_5?
print(gVar.cardinality())
print(aVar(int(5)).cardinality())
sage: 8
sage: 60
```

GLIF: Prototyping Symbolic NLU

- Claim: Prototyping NLU is important but requires much work
- GLIF as a dedicated, declarative framework for NLU prototyping
- Montague's approach:
  1. Parsing
  2. Compositional semantics construction *lots of λs*
- We also need
  3. Semantic/pragmatic analysis *disambiguation, ...*

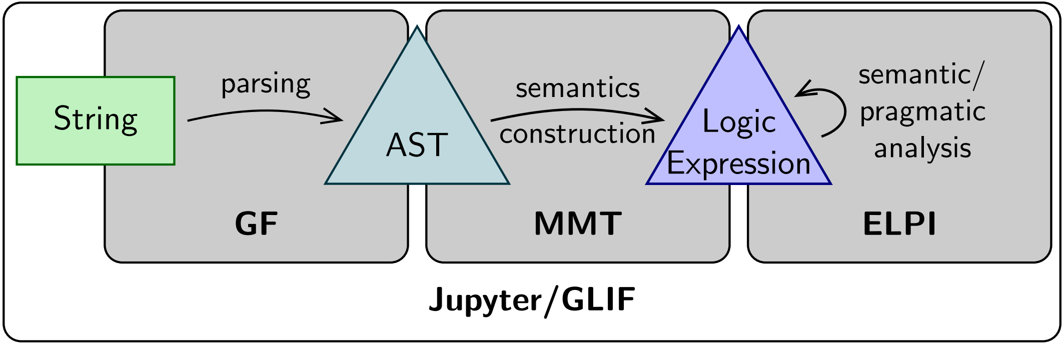
GLIF: Grammatical Logical Inference Framework



"Every integer is even" $\longrightarrow$ 

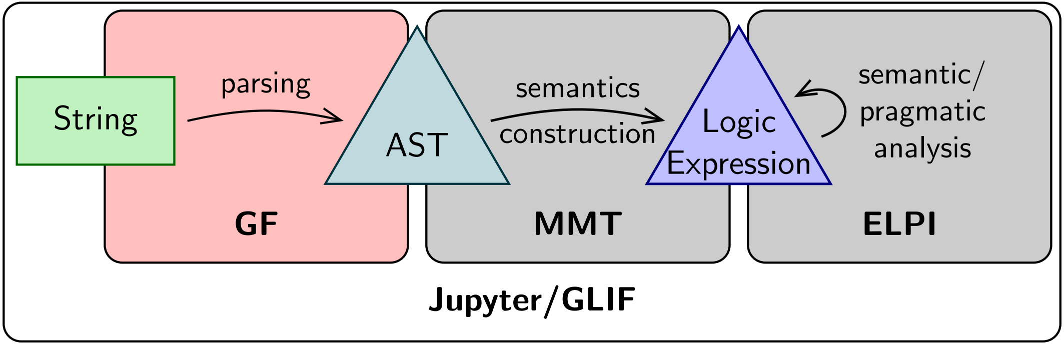
$\longrightarrow$ $\forall x.int(x) \Rightarrow even(x)$

GLIF: Grammatical Logical Inference Framework



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

Components of GLIF: GF

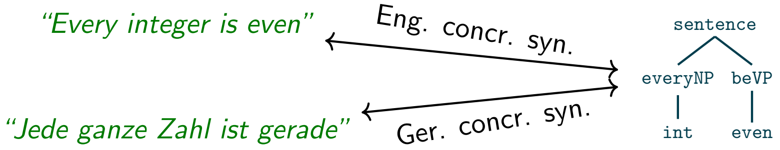


Components of GLIF: Grammatical Framework [GF]

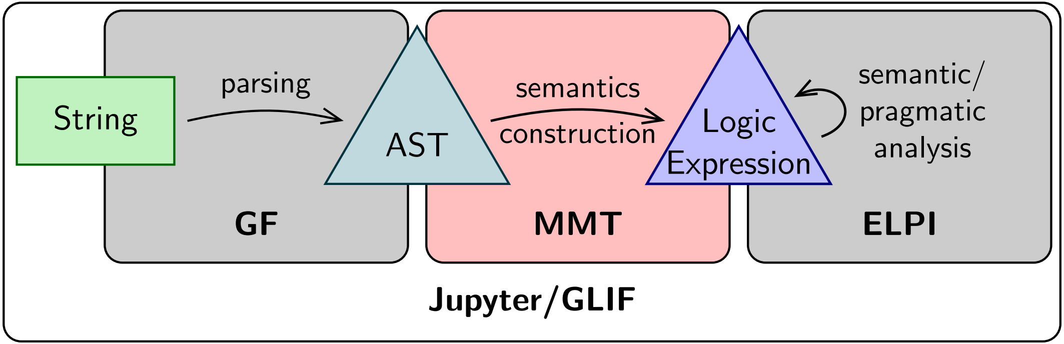
- Specialized for developing natural language grammars
- Separates abstract and concrete syntax

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

- Abstract syntax based on LF
- Comes with large library — *$\geq$ 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:
  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 representation
- 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

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

$\mapsto$

MMT

$$\begin{aligned}
&\mathtt{NP} : \mathit{type}\\
&\mathtt{VP} : \mathit{type}\\
&\mathtt{S} : \mathit{type}\\
&\mathtt{sentence} :\\
&\quad \mathtt{NP} \rightarrow \mathtt{VP} \rightarrow \mathtt{S}
\end{aligned}$$

Components of GLIF: MMT

- Modular logic development and knowledge representation
- 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 Theory

```
even : ι → o
int  : ι → o
sqrt : ι → ι
// etc.
```

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:
  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
sentence	$\mapsto$	λn.λv.v n
even	$\mapsto$	even

More advanced

NP	$\mapsto$	$(\iota \to$ o$) \to$ o
sentence	$\mapsto$	λn.λv.n v
everyNP	$\mapsto$	λn.λp.$\forall\lambda$x.n x $\Rightarrow$ p x
someNP	$\mapsto$	λn.λp.$\exists\lambda$x.n x $\wedge$ p x

Example: Parsing + Semantics Construction

"Every integer is even"

$\downarrow$parsing

sentence (everyNP int) (beVP even)

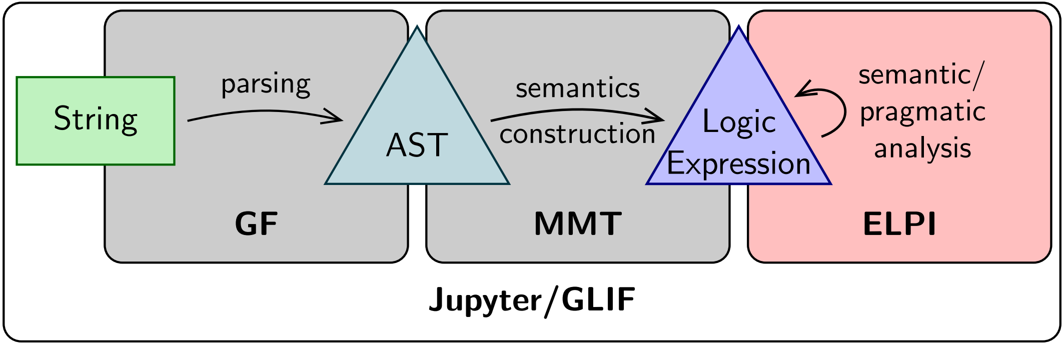
$\downarrow$semantics construction

$$\texttt{(}\lambda\texttt{n.}\lambda\texttt{v.n v) ((}\lambda\texttt{n.}\lambda\texttt{p.}\forall\lambda\texttt{x.n x} \Rightarrow \texttt{p x) int) ((}\lambda\texttt{a.a) even)}$$

$\downarrow\beta$-reduction

$$\forall\lambda\texttt{x.int x} \Rightarrow \texttt{even x}$$

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
- Goal: Use it for semantic/pragmatic analysis

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

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:

$$\text{“a kinetic energy of 12mN”} \nearrow \text{AST}_1 \longrightarrow \lambda x.E_{\text{kin}}(x, \text{quant}(2, \textbf{milli Newton}))$$
$$\text{“a kinetic energy of 12mN”} \searrow \text{AST}_2 \longrightarrow \lambda x.E_{\text{kin}}(x, \text{quant}(2, \textbf{meter}\cdot\textbf{Newton}))$$

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:

$$\begin{array}{lcl} & \nearrow & \mathrm{AST}_1 \longrightarrow \cancel{\lambda x.E_{\mathrm{kin}}(x, \mathrm{quant}(2, \mathbf{milli\ Newton}))} \\ \textit{"a kinetic energy of 12mN"} & & \\ & \searrow & \mathrm{AST}_2 \longrightarrow \lambda x.E_{\mathrm{kin}}(x, \mathrm{quant}(2, \mathbf{meter}\cdot\mathbf{Newton})) \end{array}$$

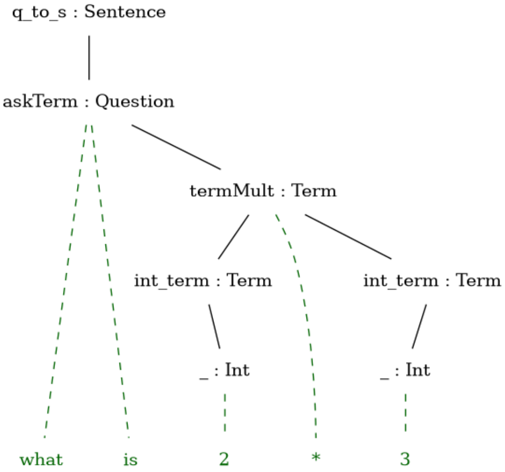
Example: Input Language for SageMath

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

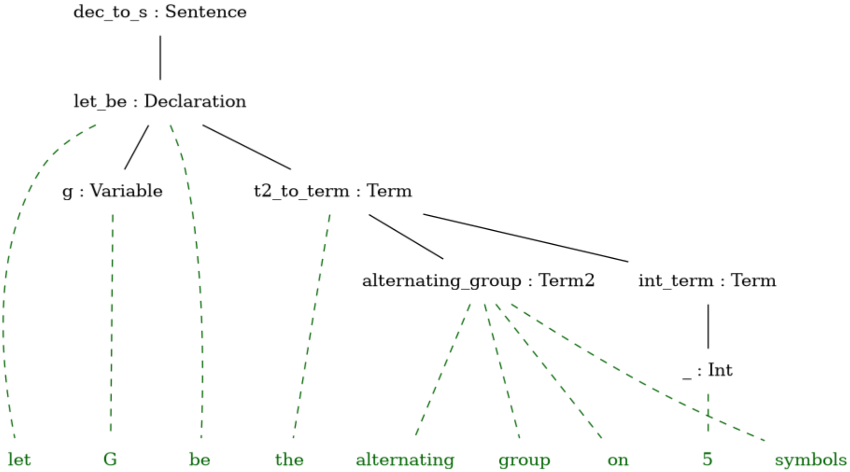
“Let G be the alternating group on 5 symbols. What is the cardinality of G?”

- Can we make a natural input language for SageMath? *WolframAlpha-like*
- GLIF Prototype:
  - Parsing
  - Semantics construction translates to SageMath command (not logic)

Example: Input Language for SageMath – Grammar



Example: Input Language for SageMath – Grammar



Example: Input Language for SageMath – Semantics Construction

- Target logic: Python/SageMath commands
- Can experiment with ideas (e.g. notations)

"let G be the alternating group on 5 symbols"

$\downarrow$

assign gVar (alternating_group (int_term 5))

```
g = AlternatingGroup(int(5))
```

Example: Input Language for SageMath – Semantics Construction

- Target logic: Python/SageMath commands
- Can experiment with ideas (e.g. notations)

"let G be the alternating group on 5 symbols"

$\downarrow$

assign gVar (alternating_group (int_term 5))

```
g = AlternatingGroup(int(5))
```

"let | G | be a notation for the cardinality of G"

$\downarrow$

```
def bar(gVar):  return gVar.cardinality()
```

*"let D_N be a notation for the dihedral group of order 2 * N"*

Example: Input Language for SageMath

`Enter command:` What are the Cayley tables of the alternating groups on 2 and 3 symbols?

```
print(AlternatingGroup(int(2)).cayley_table())
print(AlternatingGroup(int(3)).cayley_table())
sage:
*  a
 +--
a| a

sage:
*  a b c
 +------
a| a b c
b| b c a
c| c a b
```

Example: Input Language for SageMath

- Took just a few hours to create prototype
- Maybe useful for teaching?
- GF made it easy to support another language (German)
- $\rightsquigarrow$ can also translate automatically:

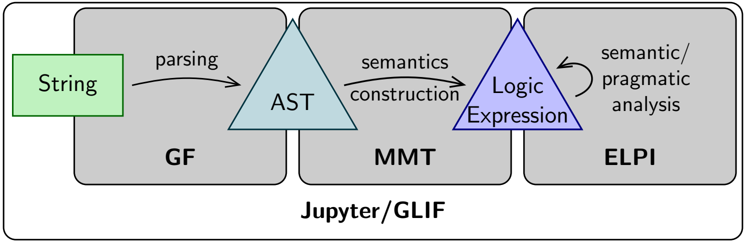
"What are the Cayley tables of the alternating groups on 2 and 3 symbols?"

$$\downarrow$$

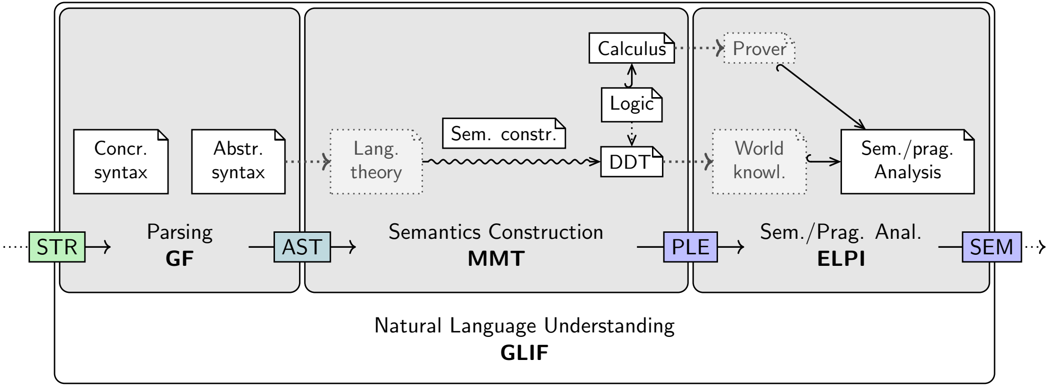
"Was sind die Verknüpfungstafeln der alternierenden Gruppen über 2 und 3 Elemente?"

Conclusion

- GLIF: Declarative framework for prototyping NLU
- Used in a 1-semester course on logic-based NL semantics
- First experiments with mathematical language



Specification of a GLIF Prototype



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.