ForTheL and GF

Jan Frederik Schaefer

FAU Erlangen-Nürnberg

ERBN
December 1, 2020

ForTheL – Formal Theory Language

Controlled natural language for *System of Automated Deduction*

$\rightarrow$ Formal language

$\rightarrow$ Subset of natural language

```
Signature SetSort. A set is a notion.
Let S,T denote sets.

Signature ElmSort. An element of S is a notion.
Let x belongs to X stand for x is an element of X.

Definition DefEmpty. S is empty iff S has no elements.

Axiom ExEmpty. There exists an empty set.
```

ForTheL – Formal Theory Language

Controlled natural language for *System of Automated Deduction*

$\rightarrow$ Formal language

$\rightarrow$ Subset of natural language

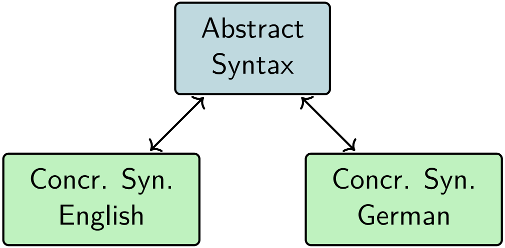
```
Signature SetSort. A set is a notion.
Let S,T denote sets.

Signature ElmSort. An element of S is a notion.
Let x belongs to X stand for x is an element of X.

Definition DefEmpty. S is empty iff S has no elements.

Axiom ExEmpty. There exists an empty set.
```

Semantics in first-order logic: $\exists x.set(x) \wedge empty(x)$.

GF – Grammatical Framework

"*A programming language for multilingual grammar applications*"

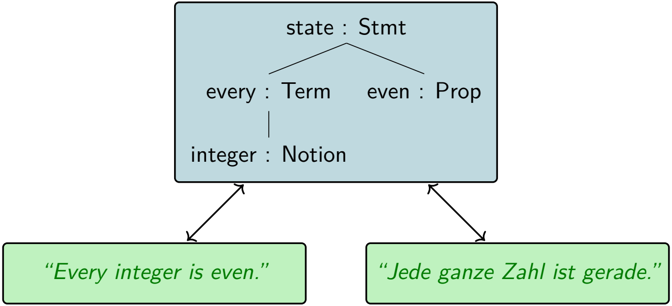
- Separate abstract and concrete syntax
- Often used for high-precision machine translation



GF – Grammatical Framework

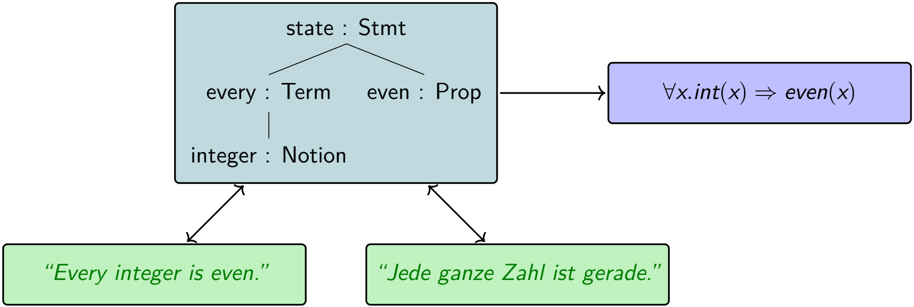
"*A programming language for multilingual grammar applications*"

- Separate abstract and concrete syntax
- Often used for high-precision machine translation



My Master's Project

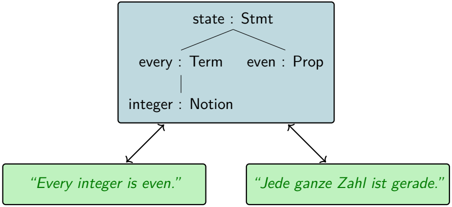
Use GLF = (GF + MMT) to parse ForTheL and create logical expressions



Conclusion (everything else is optional)

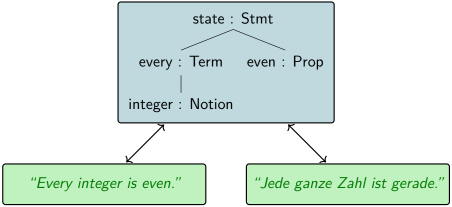
- ForTheL is a CNL for mathematics.
- It is a great case study for GLF ($=$ GF $+$ MMT).
- With GF multiple languages and translation can be easily added.

GF in More Detail: Abstract Syntax




```
abstract Grammar = {
  cat               -- "types of nodes"
    Stmt; Term; Notion; Prop;
  fun               -- production rules
    state      : Term -> Prop -> Stmt;
    every      : Notion -> Term;
    derivative : Term -> Term;
    integer    : Notion;
    even       : Prop;
}
```

GF in More Detail: Concrete Syntax




```
concrete GrammarEng of Grammar = {
  lincat
    Stmt = Str; Term = Str; Notion = Str; Prop = Str;
  lin
    state term prop = term ++ "is" ++ prop;
    every notion    = ("every"|"any") ++ notion;
    derivative term = "the derivative of" ++ term;
    integer         = "integer";
    even            = "even";
}
```

GF for ForTheL

“Let a, b be sets that aren’t empty”

```
letAssume : Names -> ClassNoun -> Assume;

-- ClassNoun = {pref : Plurality=>Str; suf : Plurality=>Str};
--    pref: set/sets
--    suf: that isn't empty/that aren't empty

letAssume names cn =
    "let" ++ names.s ++ ("be"|"denote"|"stand for") ++
    indefArt!names.p ++ cn.pref!names.p ++ cn.suf!names.p;
```

A Closer Look at Notions

Example notions: *"set"*, *"subgroup of D_8"*, *"set that isn't empty"*

On the logic side: $\mathit{set}(\cdot)$, $\mathit{subgroup}(\cdot, D_8)$, $\mathit{set}(\cdot) \wedge \neg \mathit{empty}(\cdot)$

Should *"subgroup H of D_8"* be a notion?

- Yes — *ForTheL does this*
  - \+ Notions are continuous strings
  - − Semantically tricky
  - − *"G is a subgroup H of D_8"* — *feels wrong...*
- No — *I did this*
  - \+ Semantically easy
  - \+ Definitely isn't only plural/singular yet
  - − Can't use Resource Grammar Library easily

Another Conclusion (everything else is optional)

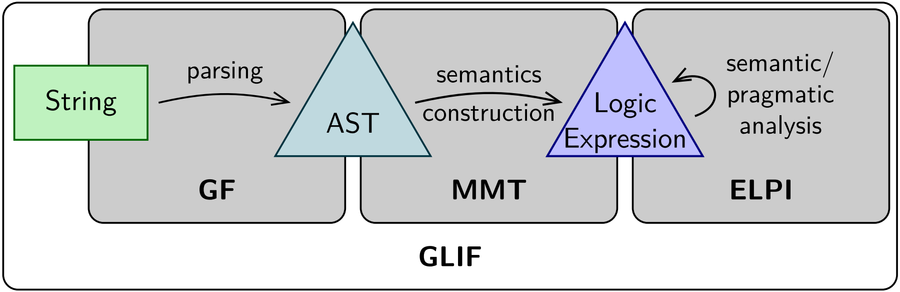
Good things:

- GF is great for parsing
- Can avoid much over-generation
- Relatively extensible
- Allows for quick prototyping

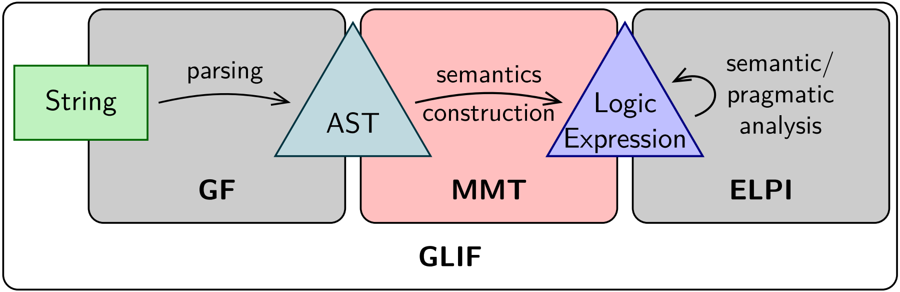
Bad things:

- Problems with using Resource Grammar Library
- No dynamic lexicon extension

Result of my Master's Thesis: GLIF



Result of my Master's Thesis: GLIF



MMT

One way to look at it:

- Use logics to represent knowledge
- Use logical frameworks to represent logics
- Use MMT to implement logical frameworks *Meta Meta Tool*

In GLIF:

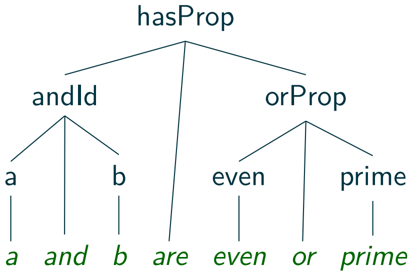
- Develop a logic *syntax, semantics, calculus*
- Describe semantics construction *map ASTs to logical expressions*

Logic in MMT

$$\begin{aligned}
o &: \mathit{type} \\
\iota &: \mathit{type} \\
\neg &: o \to o \\
\wedge &: o \to o \to o \\
\vee &: o \to o \to o \\
\forall &: (\iota \to o) \to o \\
\exists &: (\iota \to o) \to o
\end{aligned}$$

$$\begin{aligned}
\mathit{set} &: \iota \to o \\
\mathit{int} &: \iota \to o \\
\mathit{even} &: \iota \to o \\
\mathit{empty} &: \iota \to o \\
\mathit{subgroup} &: \iota \to \iota \to o \\
\mathit{derivative} &: \iota \to \iota
\end{aligned}$$

Semantics Construction in MMT



`hasProp (andId a b) (orProp even prime)`

Semantics Construction in MMT

Substitute every node in AST with lambda functions $\lambda x.M$ as $x \mapsto M$

```
hasProp (andId a b) (orProp even prime)
```

```
(λx,p.x(p))                          // haspProp
(
    (λx,y.λp.x(p)∧y(p))              // andId
    (λp.p(A))                        // a
    (λp.p(B))                        // b
)
(
    (λp,q.λx.p(x)∨q(x))              // orProp
    even                             // even
    prime                            // prime
)
```

Semantics Construction in MMT

Substitute every node in AST with lambda functions $\lambda x.M$ *as* $x \mapsto M$

```
hasProp (andId a b) (orProp even prime)
```

```
(λx,p.x(p))                // haspProp
(λp.p(A)∧p(B))             // andId a b
(λx.even(x)∨prime(x))      // orProp even prime
```

$$\rightsquigarrow_\beta$$

```
(even(A)∨prime(A))∧(even(B)∨prime(B))
```

ForTheL can be illustrated with Textual Transformations

“a and b are even or prime”

$\rightsquigarrow$ *“a is even or prime and b is even or prime”*

$\rightsquigarrow$ *“a is even or a is prime and b is even or b is prime”*

$\rightsquigarrow (\mathit{even}(a) \vee \mathit{prime}(a)) \wedge (\mathit{even}(b) \vee \mathit{prime}(b))$

Another Conclusion (everything else is optional)

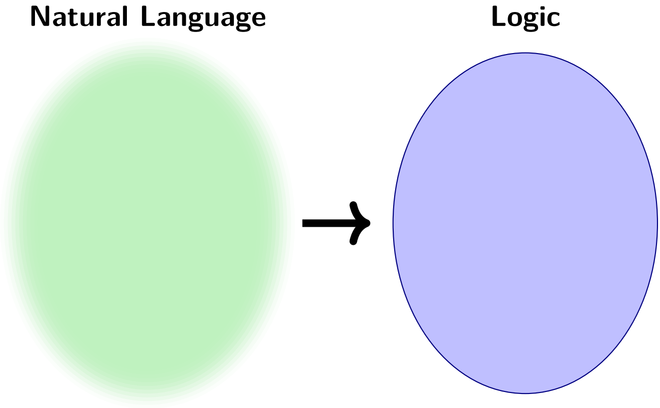
Good things:

- GF is great for parsing
- Can avoid much over-generation
- Relatively extensible
- Allows for quick prototyping
- MMT is a dedicated logic development framework

Bad things:

- Problems with using GF's Resource Grammar Library
- No dynamic lexicon extension
- Semantics construction can be tricky

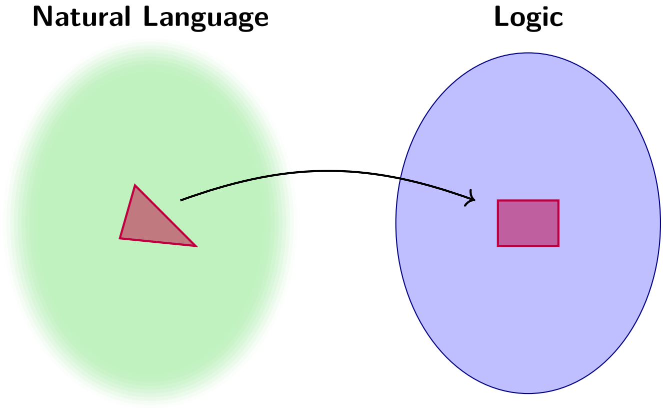
Method of Fragments



How do we get from messy language to formal logic?

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

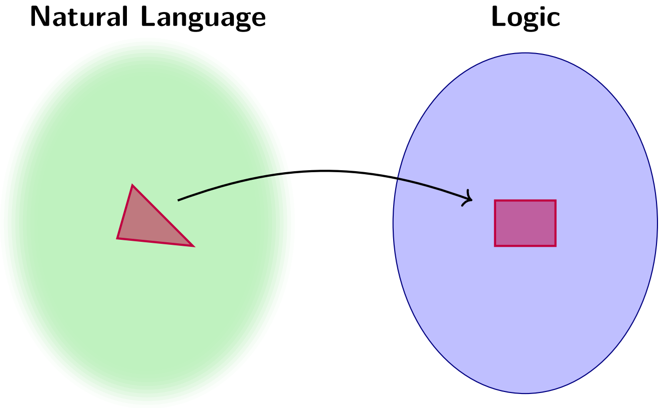
Method of Fragments



How do we get from messy language to formal logic?

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

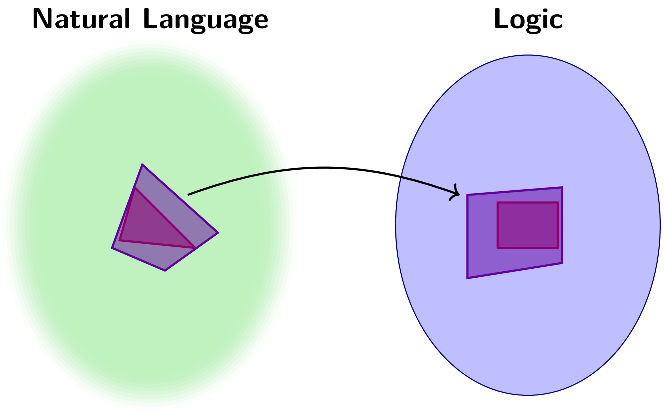
Method of Fragments



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

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

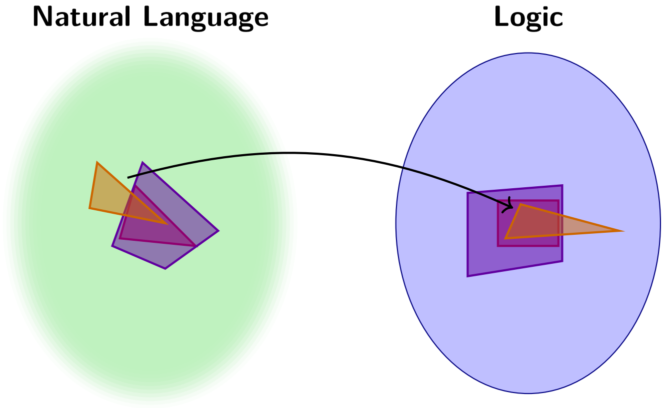
Method of Fragments



"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

If we only hand-wave, we gloss over problems:

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

Specify:

- Grammar *fixes NL subset*
- Target logic
- Semantics construction *maps parse trees to logic*

On paper [Mon74]: *difficult to scale*

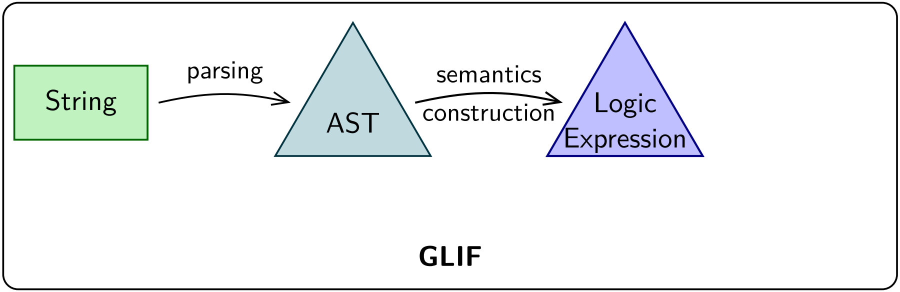
T11. If $\phi, \psi \in P_t$ and ϕ, ψ translate into ϕ', ψ' respectively, then ϕ **and** ψ translates into $[\phi \wedge \psi]$, ϕ **or** ψ translates into $[\phi \vee \psi]$.

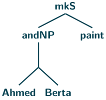
T12. If $\gamma, \delta \in P_{IV}$ and γ, δ translate into γ', δ' respectively, then γ **and** δ translates into $\hat{x}[\gamma'(x) \wedge \delta'(x)], \gamma$ **or** δ translates into $\hat{x}[\gamma'(x) \vee \delta'(x)]$.

T13. If $\alpha, \beta \in P_T$ and α, β translate into α', β' respectively, then α **or** β translates into $\hat{P}[\alpha'(P) \vee \beta'(P)]$.

GLIF: Grammatical Logical Inference Framework

We have a tool for this!

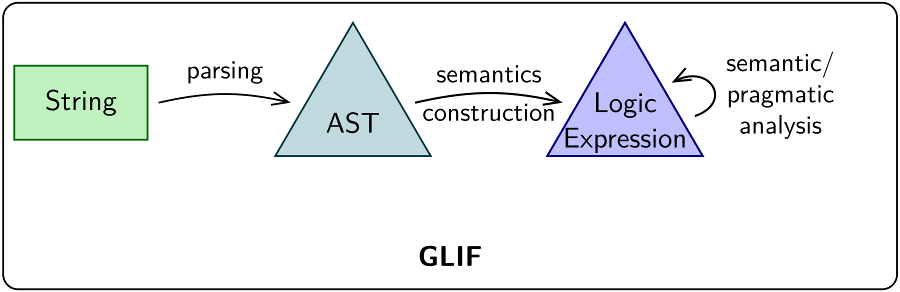


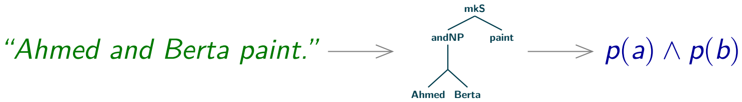
"Ahmed and Berta paint." ⟶ 

⟶ $p(a) \wedge p(b)$

GLIF: Grammatical Logical Inference Framework

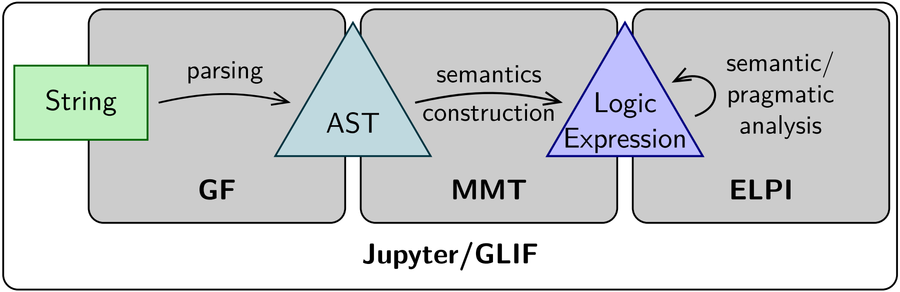
We have a tool for this!





GLIF: Grammatical Logical Inference Framework

It combines existing tools.



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

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

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

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

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

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

"Ahmed doesn't paint"

Example: Tableaux Machine

Background Knowledge

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

"John talks to Mary."

$talkto(j, m)$

"Sasha is sad."

$sad(s)$

$$\downarrow$$

$$talkto(j, m)$$

$$\downarrow$$

$$sad(s)$$

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."

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

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

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

Example: Tableaux Machine

Background Knowledge

"John talks to Mary."

$talkto(j, m)$

"Sasha is sad."

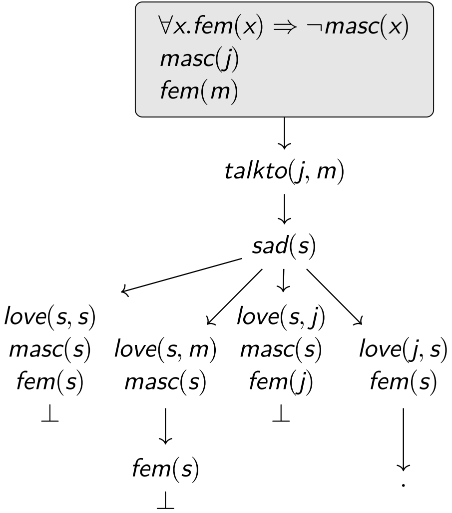
$sad(s)$

"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."

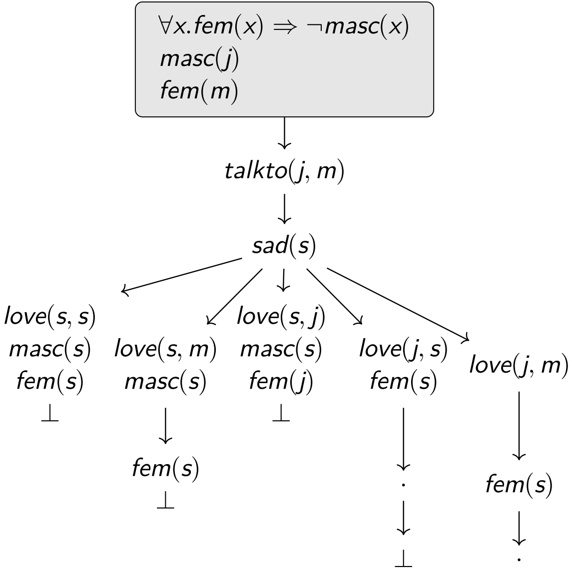
$\exists X.masc(X) \wedge$
$\quad \exists Y.fem(Y) \wedge love(X, Y)$

"Sasha is a woman."

$fem(s)$

"John doesn't love Sasha."

$\neg love(j, s)$



Example: 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)

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

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