GF + MMT = GLF

From Language to Semantics through LF

Michael Kohlhase **Jan Frederik Schaefer**

Friedrich-Alexander-Universität Erlangen-Nürnberg

LFMTP

Vancouver, June 22, 2019

"*Mary runs and John is happy.*"	$\text{run}'(\text{mary}') \wedge \text{happy}'(\text{john}')$
"*Everyone loves Mary.*"	$\forall x.\text{love}'(x, \text{mary}')$
"*He loves her.*"	$\exists X_{\mathbb{M}}, Y_{\mathbb{F}}.\text{love'}(X_{\mathbb{M}}, Y_{\mathbb{F}})$
"*John isn't allowed to run.*"	$\neg \Diamond \text{run}'(\text{john}')$

Definition

NL semantics studies the meaning of NL utterances

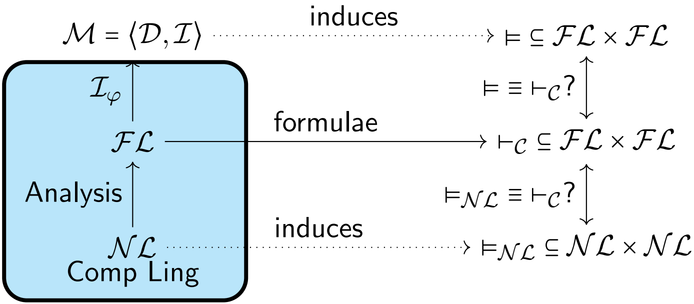
How could we do this?

Look at a fragment of English and define a suitable logic [Mon70]

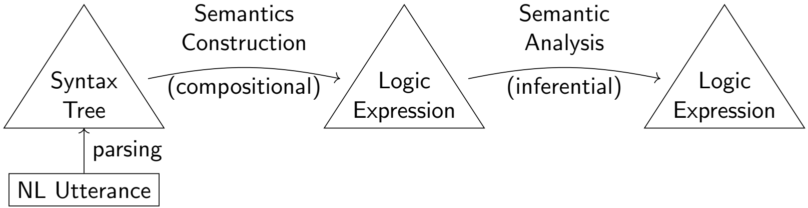
$\rightsquigarrow$ we could cheat a little:

"*Mary runs. She is happy.*" $\text{run}'(\text{mary}') \wedge \text{happy}'(\text{mary}')$

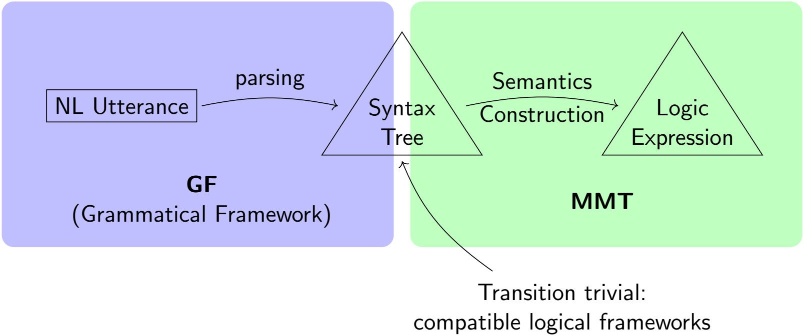
$\rightsquigarrow$ describe the translation as well


$\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$
induces
$\models \subseteq \mathcal{FL} \times \mathcal{FL}$
$\mathcal{I}_\varphi$
$\models \equiv \vdash_{\mathcal{C}}$?
$\mathcal{FL}$
formulae
$\vdash_{\mathcal{C}} \subseteq \mathcal{FL} \times \mathcal{FL}$
Analysis
$\models_{\mathcal{NL}} \equiv \vdash_{\mathcal{C}}$?
induces
$\mathcal{NL}$
$\models_{\mathcal{NL}} \subseteq \mathcal{NL} \times \mathcal{NL}$
Comp Ling

Natural Language Understanding (NLU) Systems



The Grammatical Logical Framework (GLF)



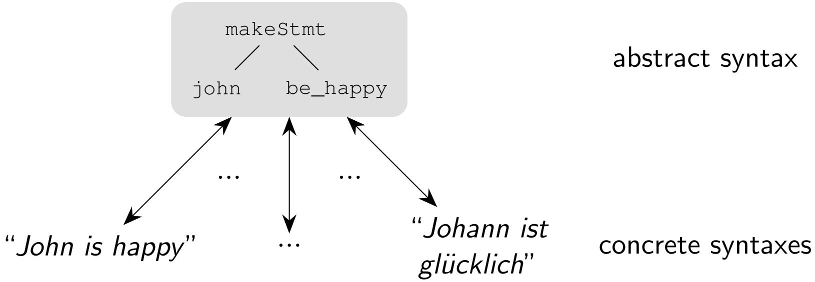
	GF	= **grammar** *development framework*
+	**MMT**	= **logic** *development framework*
	GLF	= **semantics** *development framework*

The Example

"*Everyone runs.*"	$\forall x.\text{run}'(x)$
"*Someone is happy.*"	$\exists x.\text{happy}'(x)$
"*John and Mary are happy.*"	$\text{happy}'(\text{john}') \wedge \text{happy}'(\text{mary}')$
Fragment of English	Target logic: FOL

The Grammatical Framework (GF) [Ran11]

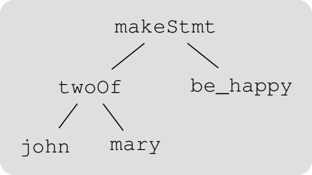
- GF is a programming language for multilingual grammar applications
- **Abstract syntax**: describes parse trees
- **Concrete syntaxes**: language-specific linearization rules



Describing the Fragment in GF – Abstract Syntax

```
abstract Gossip = {
  cat
    Actor; Action; Stmt;
  fun
    everyone : Actor;
    someone  : Actor;
    makeStmt : Actor -> Action -> Stmt;
    twoOf    : Actor -> Actor -> Actor;
}
```

```
abstract GossipLex = Gossip ** {
  fun
    john, mary : Actor;
    run        : Action;
    be_happy   : Action;
}
```




```
makeStmt (twoOf john mary) be_happy
```

Describing the Fragment in GF – Concrete Syntax

```
concrete GossipEng of Gossip = {
  lincat
    Actor = Str; Action = Str; Stmt = Str;
  lin
    everyone                = "everyone";
    someone                 = "someone";
    makeStmt actor action = actor ++ action;
    twoOf a b               = a ++ "and" ++ b;
}

concrete GossipLexEng of GossipLex = GossipEng ** {
  lin
    john     = "John";
    mary     = "Mary";
    run      = "runs";
    be_happy = "is happy";
}
```

Concrete Syntax for English (first attempt)

In [50]:
```
concrete GossipEng0 of Gossip = {
  lincat
    Actor = Str; Action = Str; Stmt = Str;
  lin
    everyone                  = "everyone";
    someone                   = "someone";
    makeStmt actor action = actor ++ action;
    and a b                   = a ++ "and" ++ b;
}
```

Abstract changed, previous concretes discarded.

In [51]:
```
concrete GossipLexEng0 of GossipLex = GossipEng0 ** {
  lin
    john     = "John";
    mary     = "Mary";
    run      = "runs";
    be_happy = "is happy";
}
```

Abstract changed, previous concretes discarded.

Let's try it out!

In [52]:
```
parse -lang=Eng0 -cat=Stmt "John runs"
```

makeStmt john run

In [53]:
```
parse -lang=Eng0 -cat=Stmt "John and Mary are happy"
```

The parser failed at token 4: "are"

In [54]:
```
linearize makeStmt (and john mary) be_happy
```

John and Mary is happy

Describing the Fragment in GF – Concrete Syntax

Problem

"*John* **is** *happy*" vs "*John and Mary* **are** *happy*"

Solutions

- More sophisticated grammar rules
- Use the *resource grammar library*

Describing the Fragment in GF – Concrete Syntax

```
concrete GossipEng of Gossip = {
  param
    Plurality = Sg | Pl;
  lincat
    Actor  = {s : Str; p :  Plurality};
    Action = Plurality => Str;
    Stmt   = Str;
  lin
    everyone  = {s = "everyone"; p = Sg};
    someone   = {s = "someone"; p = Sg};
    makeStmt actor action = actor.s ++ action ! actor.p;
    twoOf a b = {s = a.s ++ "and" ++ b.s; p = Pl};
}
```

Let's try it out!

```
In [47]: parse -lang=Engl -cat=Stmt "John runs"
```

```
makeStmt john run
```

```
In [48]: parse -lang=Engl -cat=Stmt "John and Mary are happy"
```

```
makeStmt (and john mary) be_happy
```

```
In [49]: parse -lang=Engl -cat=Stmt "John and Mary is happy"
```

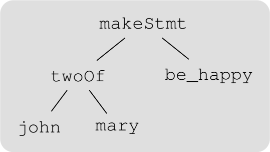
```
The parser failed at token 4: "is"
```

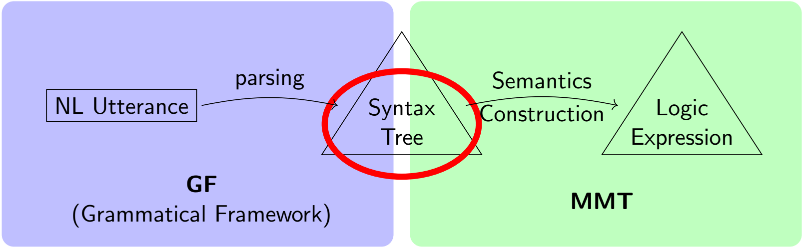
Describing the Fragment in GF – Concrete Syntax

Resource Grammar Library: grammar rules for 36 languages

```
concrete GossipEng of Gossip = open SyntaxEng, DictEng in {
  lincat
    Actor  = NP;
    Action = VP;
    Stmt   = S;
  lin
    everyone  = everyone_NP;
    someone   = someone_NP;
    makeStmt actor action = mkS (mkCl actor action);
    twoOf a b = mkNP and_Conj a b;
}
```

"John and Mary are happy" ↦





MMT – "anything you can do we can do meta" [RK13]

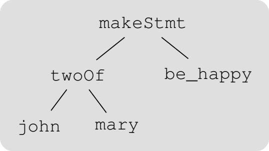
- You may remember "Rapid Prototyping Formal Systems in MMT: 5 Case Studies" [MR19]
- Meta meta theories/meta meta tool set
- Little theories
- Bring your own logic
- Logic development environment
- Foundation-independent

Abstract syntax (GF)

```
abstract Gossip = {
  cat
    Actor;
    Action;
    Stmt;
  fun
    everyone : Actor;
    someone  : Actor;
    makeStmt :
          Actor->Action->Stmt;
    twoOf:Actor->Actor->Actor;
}
```

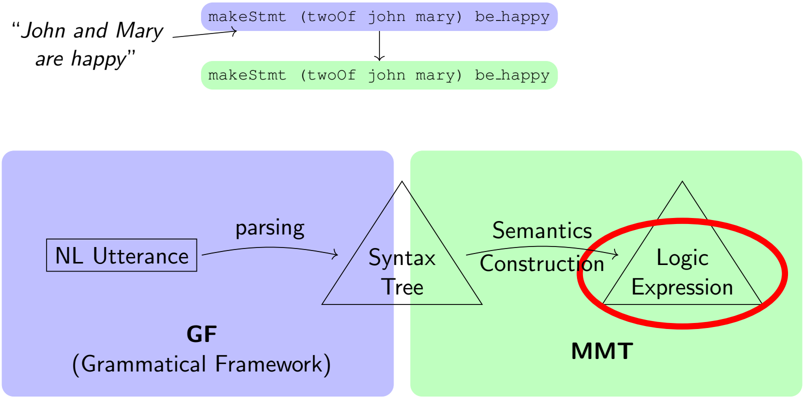
Language theory (MMT)

```
theory Gossip : ur:?LF =

  Actor  : type ‖
  Action : type ‖
  Stmt   : type ‖

  everyone : Actor ‖
  someone  : Actor ‖
  makeStmt :
        Actor → Action → Stmt‖
  twoOf:Actor → Actor → Actor‖
❚
```



↦ `makeStmt (twoOf john mary) be_happy`

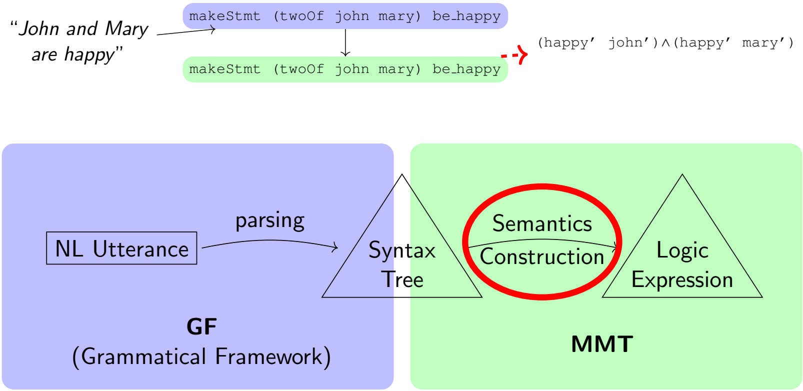
Where are we?



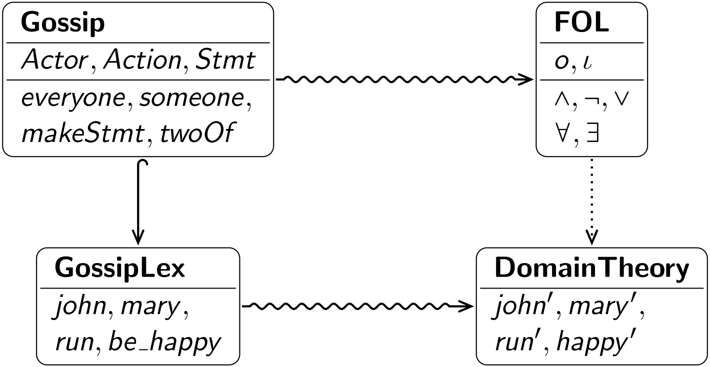
"John and Mary are happy"
makeStmt (twoOf john mary) be_happy
makeStmt (twoOf john mary) be_happy
NL Utterance
parsing
Syntax Tree
Semantics Construction
Logic Expression
GF
(Grammatical Framework)
MMT

```
theory FOL : ur:?LF =
    prop    : type            | # o        ∥
    and     : o → o → o       | # 1 ∧ 2    ∥
    neg     : o → o           | # ¬ 1      ∥
    or      : o → o → o       | # 1 ∨ 2 |
        = [x,y] ¬ ( (¬ x) ∧ (¬ y) )    ∥


    ind     : type            | # ι        ∥
    forall  : (ι → o) → o     | # ∀ 1      ∥
    exists  : (ι → o) → o     | # ∃ 1 |
        = [p] ¬ (∀ [x] (¬ p x))        ∥
❚

theory DomainTheory : ?FOL =
    mary  : ι      | # mary'     ∥
    john  : ι      | # john'     ∥
    run   : ι → o  | # run' 1    ∥
    happy : ι → o  | # happy' 1  ∥
❚
```

Where are we?



"John and Mary are happy"
makeStmt (twoOf john mary) be_happy
makeStmt (twoOf john mary) be_happy
(happy' john')∧(happy' mary')
NL Utterance
parsing
Syntax Tree
Semantics Construction
Logic Expression
GF
(Grammatical Framework)
MMT


Gossip
Actor, Action, Stmt
everyone, someone,
makeStmt, twoOf
FOL
o, ι
∧, ¬, ∨
∀, ∃
GossipLex
john, mary,
run, be_happy
DomainTheory
john′, mary′,
run′, happy′

```
view GossipSem : ?Gossip -> ?FOL =
    Stmt     = o ‖
    Action   = ι → o ‖
    Actor    = ι ‖

    everyone = ??? ‖
    someone  = ??? ‖
    makeStmt = [a,φ] φ a ‖
    twoOf    = ??? ‖
❚

view GossipLexSem : ?GossipLex -> ?DomainTheory =
    include ?GossipSem ‖
    john     = john’ ‖
    mary     = mary’ ‖
    run      = [x] run’ x ‖
    be_happy = [x] happy’ x ‖
❚
```

Problem

`Actor` $=$ ι

`everyone :  Actor = ?`

Solution

`Actor` $= (\iota \to o) \to o$

`john` $= [\varphi]\ \varphi$ `john'`

`everyone` $= [\varphi]\ \forall\ [x]\ (\varphi\ x) = [\varphi]\ \forall\ \varphi$

Example

"*everyone runs*" $\mapsto$ $([\varphi]\ \forall\ [x]\ (\varphi\ x))$ `run'` $\rightsquigarrow_\beta$ $\forall\ [x]$ (`run'` x)

```
view GossipSem : ?Gossip -> ?FOL =
    Stmt      = o ‖
    Action    = ι → o ‖
    Actor     = (ι → o) → o ‖

    everyone = [φ] ∀ φ ‖
    someone  = [φ] ∃ φ ‖
    makeStmt = [a,φ] a φ ‖
    twoOf    = [a1,a2] [φ] (a1 φ) ∧ (a2 φ) ‖
❚

view GossipLexSem : ?GossipLex -> ?DomainTheory =
    include ?GossipSem ‖
    john     = [φ] φ john’ ‖
    mary     = [φ] φ mary’ ‖
    run      = [x] run’ x ‖
    be_happy = [x] happy’ x ‖
❚
```

```
view GossipSem : ?Gossip -> ?FOL =
    Stmt      = o ‖
    Action    = ι → o ‖
    Actor     = (ι → o) → o ‖

    everyone = [φ] ∀ φ ‖
    someone  = [φ] ∃ φ ‖
    makeStmt = [a,φ] a φ ‖
    twoOf    = [a1,a2] [φ] (a1 φ) ∧ (a2 φ) ‖
❚
```

These views are described in NL semantics papers like [Mon74]:

Rules of conjunction and disjunction

T11. If $\phi, \psi \in \mathrm{P_t}$ 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 $\gamma, \delta \in \mathrm{P_{IV}}$ 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 $\alpha, \beta \in \mathrm{P_T}$ and α, β translate into α', β' respectively, then α **or** β translates into $\hat{\mathrm{P}}[\alpha'(\mathrm{P}) \vee \beta'(\mathrm{P})]$.

Example

"*John and Mary are happy*"

↓ parse

```
makeStmt (twoOf john mary) be_happy
```

↓ semantics construction

```
([a,φ] a φ)
  (([a1,a2] [φ] (a1 φ) ∧ (a2 φ)) ([φ] φ john') ([φ] φ mary'))
  ([x] happy' x)
```

↓ simplify

```
([a1,a2,φ] (a1 φ) ∧ (a2 φ)) ([φ] φ john') ([φ] φ mary') happy'
```

↓ simplify

```
([φ] (φ john') ∧ (φ mary')) happy'
```

↓ simplify

```
(happy' john') ∧ (happy' mary')
```

Adding Transitive Verbs ($\leadsto$ more type raising)

"*John and Mary love everyone*"

$\downarrow$

`∀[x:ι](love' john' x)∧(love' mary' x)`

(Multi) Modal Logic

Modalities:

- *deontic* – something is obligatory ($[\![$d$]\!]$) or permissible ($\langle\!\langle$d$\rangle\!\rangle$)
- *epistemic* – someone believes something is true ($[\![$e john'$]\!]$) or possible ($\langle\!\langle$e john'$\rangle\!\rangle$).

"*John doesn't believe that Mary has to run*"

$\downarrow$

`¬⟦(e john')⟧⟦d⟧(run' mary').`

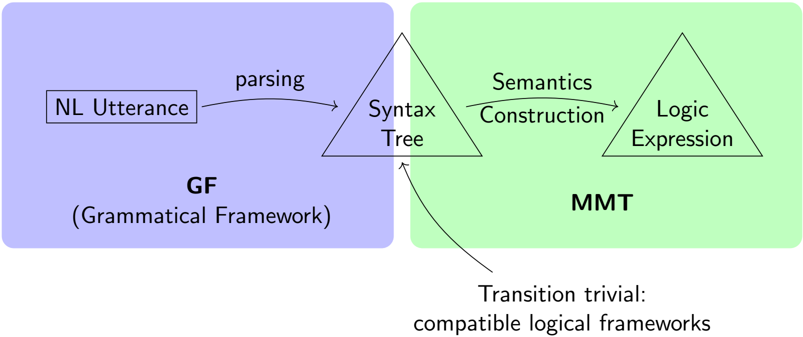
```
Please enter a sentence: John isn’t allowed to run
I got the following interpretations:
¬⟪d⟫(run’ john’)
Please enter a sentence: Mary believes that John doesn’t have to run
I got the following interpretations:
⟦(e mary’)⟧¬⟦d⟧(run’ john’)
Please enter a sentence:
```

Conclusions

- GLF $=$ tool to implement NLU system
- would have been great in the 90's to avoid pen-and-paperness
- previous versions used for teaching NL semantics

- work on a Jupyter kernel for GLF
- generic tableau calculus for semantic analysis?

The Grammatical Logical Framework (GLF)



GF	= **grammar** *development framework*
+ **MMT**	= **logic** *development framework*
GLF	= **semantics** *development framework*

Concrete Syntax for English (first attempt)

In [50]:
```
concrete GossipEng0 of Gossip = {
  lincat
    Actor = Str; Action = Str; Stmt = Str;
  lin
    everyone               = "everyone";
    someone                = "someone";
    makeStmt actor action = actor ++ action;
    and a b                = a ++ "and" ++ b;
}
```
Abstract changed, previous concretes discarded.

In [51]:
```
concrete GossipLexEng0 of GossipLex = GossipEng0 ** {
  lin
    john     = "John";
    mary     = "Mary";
    run      = "runs";
    be_happy = "is happy";
}
```
Abstract changed, previous concretes discarded.

Let's try it out!

In [52]:
```
parse -lang=Eng0 -cat=Stmt "John runs"
```
makeStmt john run

In [53]:
```
parse -lang=Eng0 -cat=Stmt "John and Mary are happy"
```
The parser failed at token 4: "are"

In [54]:
```
linearize makeStmt (and john mary) be_happy
```
John and Mary is happy

Problem:
"*John owns a book. It is red.*"
$(\exists x.\text{own'}(\text{john'}, x) \wedge \text{book'}(x)) \wedge \text{red}(x)$

Solution: Define more suitable logic (e.g. discourse representation theory)

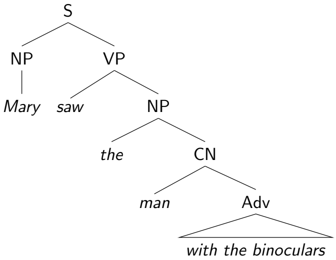
"*Mary works at a bank*"

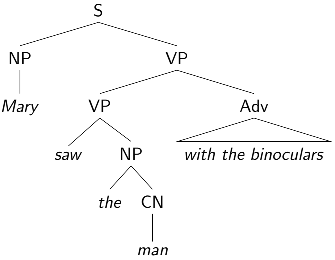
⇝ river bank or bank institute?

⇝ two parse trees:

- `work_at mary bank_institute`
- `work_at mary bank_river`

"*Mary saw the man with the binoculars*"





R. Montague.
English as a Formal Language, chapter Linguaggi nella Societa e nella Tecnica, B. Visentini et al eds, pages 189–224.
Edizioni di Communita, Milan, 1970.
Reprinted in [**?**], 188–221.

Richard Montague.
The proper treatment of quantification in ordinary English.
In R. Thomason, editor, *Formal Philosophy. Selected Papers*.
Yale University Press, New Haven, 1974.

Dennis Müller and Florian Rabe.
Rapid prototyping formal systems in mmt: Case studies.
2019.

Aarne Ranta.
Grammatical Framework: Programming with Multilingual Grammars.
CSLI Publications, Stanford, 2011.
ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).

Florian Rabe and Michael Kohlhase.
A scalable module system.
Information & Computation, 0(230):1–54, 2013.