GLIF with Jupyter Notebooks

In [1]:

```
put_string "Hello world!"
```

Hello world!

Grammatical Framework: Intro

In [2]:

```
abstract MiniExample = {
  cat
    NP;  -- noun phrases
    VP;  -- verb phrases
    S;   -- sentences
  fun
    make_S : NP -> VP -> S;
    and_S : S -> S -> S;

    -- lexicon:
    ahmed : NP;
    berta : NP;
    paint : VP;
    be_quiet : VP;
}
```

Successfully imported MiniExample.gf

In [3]:

```
concrete MiniExampleEng of MiniExample = {
  lincat
    NP = Str;
    VP = Str;
    S = Str;
```

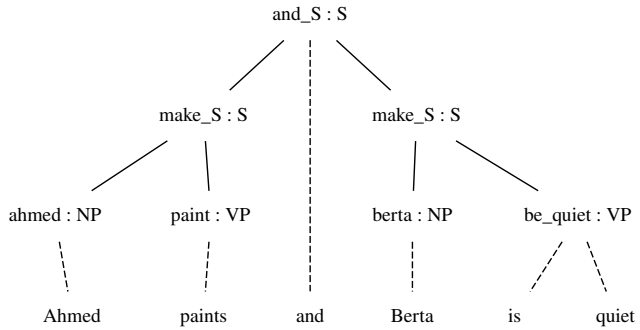
```
  lin
    make_S np vp = np ++ vp;
    and_S s1 s2 = s1++"and"++s2;

    ahmed = "Ahmed";
    berta = "Berta";
    paint = "paints";
    be_quiet = "is quiet";
}
```

Successfully imported MiniExampleEng.gf

In [4]:

```
parse \
  "Ahmed paints and Berta is quiet" | vp -showfun
```

1.0. and_S (make_S ahmed paint) (make_S berta be_qui



In [5]:

```
concrete MiniExampleGer of MiniExample = {
  lincat
    NP = Str;
    VP = Str;
    S = Str;
  lin
    make_S np vp = np ++ vp;
    and_S s1 s2 = s1++"und"++s2;
```

```
    ahmed = "Ahmed";
    berta = "Berta";
    paint = "zeichnet";
    be_quiet = "ist leise";
}
```

Successfully imported MiniExampleGer.gf

In [6]:

```
parse -lang=Eng "Ahmed is quiet" |
  linearize -lang=Ger
```

Ahmed ist leise

Bigger Example

Allow `Ahmed and Berta paint` (not `paints` !)

In [7]:

```
abstract BiggerExample =
        MiniExample ** {
  fun
    and_NP : NP -> NP -> NP;

    everyone : NP;
}
```

Successfully imported BiggerExample.gf

In [8]:

```
concrete BiggerExampleEng of BiggerExample = {
  param
    Number = Sg | Pl;
  lincat
    NP = {s: Str; n: Number};
    VP = {s: Number => Str};
```

```
    S = Str;
  lin
    make_S np vp =
       np.s ++ vp.s ! np.n;
    and_S s1 s2 = s1++"and"++s2;
    and_NP np1 np2 =
       {s=np1.s++"and"++np2.s; n=Pl};

    ahmed = {s="Ahmed"; n=Sg};
    berta = {s="Berta"; n=Sg};
    paint = {s=table{
       Sg=>"paints";Pl=>"paint"}};
    be_quiet = {s=table{
       Sg=>"is quiet";Pl=>"are quiet"}};
    everyone = {s="everyone"; n=Sg};
}
```

Successfully imported BiggerExampleEng.gf

In [9]:

```
parse "Ahmed and Berta paint"
```

make_S (and_NP ahmed berta) paint

In [10]:

```
parse "Ahmed and Berta paints"
```

Errors
The parser failed at token 4: "paints"

In [11]:

```
theory MiniExample_MMT : ur:?LF =
  NP : type ❙
  VP : type ❙
  S : type ❙

  make_S : NP → VP → S ❙
```

```
  and_S : S → S → S |

  ahmed : NP |
  berta : NP |
  paint : VP |
  be_quiet : VP |
|
```

Successfully imported MiniExample_MMT.mmt

In [12]:

```
theory logic : ur:?LF =
  o: type |
  not: o → o | # ¬ 1 prec 80 |
  and: o → o → o | # 1 ∧ 2 prec 70 |
  or:  o → o → o | # 1 ∨ 2 prec 60 |

  ι: type |
  forall: (ι → o) → o | # ∀ 1 |
  exists: (ι → o) → o | # ∃ 1 |
|
```

Successfully imported logic.mmt

In [13]:

```
theory ddt : ?logic =
  ahmed: ι |
  berta: ι |
  paint: ι → o |
  quiet: ι → o |
|
```

Successfully imported ddt.mmt

In [14]:

```
view MiniSemConstr : ?MiniExample_MMT -> ?ddt =
  S = o |
  NP = ι |
```

```
  VP = ι → o ❙

  // make_S : NP → VP → S ❙
  make_S = [n, v] v n ❙
  and_S = [s1, s2] s1 ∧ s2 ❙

  ahmed = ahmed ❙
  berta = berta ❙
  paint = paint ❙
  be_quiet = quiet ❙
❙
```

Successfully imported MiniSemConstr.mmt

In [15]:

```
parse "Ahmed paints"
```

make_S ahmed paint

In [16]:

```
parse "Ahmed paints" | construct -no-simplify
```

([n,v]v n) ahmed paint

In [17]:

```
parse "Ahmed paints" | construct
```

paint ahmed

In [18]:

```
parse "Ahmed paints and Berta is quiet"
```

and_S (make_S ahmed paint) (make_S berta be_quiet)

In [19]:

```
parse "Ahmed paints and Berta is quiet" | construct -
```

([s1,s2]s1 ∧ s2) (([n,v]v n) ahmed paint) (([n,v]v n)
berta quiet)

In [20]:

```
parse "Ahmed paints and Berta is quiet" |
  construct
```

(paint ahmed) ∧ (quiet berta)

In [21]:

```
view BiggerExampleSem : http://mathhub.info/tmpGLIF/d
  S = o |
  NP = (ι → o) → o |
  VP = ι → o |

  make_S = [n, v] n v |
  and_S = [s1, s2] s1 ∧ s2 |
  and_NP = [n1, n2] [v] n1 v ∧ n2 v |

  ahmed = [v] v ahmed |
  berta = [v] v berta |
  paint = paint |
  be_quiet = quiet |
  everyone = [v] ∀ [x] v x |
|
```

Successfully imported BiggerExampleSem.mmt

In [22]:

```
parse "Ahmed and Berta paint" |
  construct
```

(paint ahmed) ∧ (paint berta)

In [23]:

```
parse "Ahmed paints and everyone is quiet" |
  construct
```

(paint ahmed) ∧ ∀[x]quiet x