GF as a programming language

partly inspired of Herbert Lange's "GF for Python programmers"

Arianna Masciolini

LT2214 Computational Syntax

VT25

From the GF website

What is GF?

GF, Grammatical Framework, is a programming language for **multilingual grammar applications**. It is

- a **special-purpose language for grammars**, like YACC, Bison, Happy, BNFC, but not restricted to programming languages
- a **functional programming language**, like Haskell, Lisp, OCaml, SML, Scheme, but specialized to grammar writing
- a **development platform for natural language grammars**, like LKB, XLE, Regulus, but based on functional programming and type theory
- a **categorial grammar formalism**, like ACG, CCG, but specialized for multilingual grammars,
- a **logical framework**, like Agda, Coq, Isabelle, but equipped with concrete syntax in addition to logic
- a **platform for machine translation**, like Moses, Apertium, but based on deep structural analysis (and usually applied for limited fragments of language).

Python vs GF

	Python	**GF**
applicability	general-purpose	domain-specific
paradigm	mostly procedural	functional
typing	duck-dynamic	static
documentation	almost overly abundant	sparse but high-quality

Striking syntactic differences

	Python	**GF**
comments	start with `#`	start with `--`
separators	tabs and newlines	`{}` and `;`
operators	`:`, `[]`, `+`	`=>`, `!`, `+` and `++`
function application	`f(p1, p2, ..., pn)`	`f p1 p2 ... pn`

(more on “functions” in the next slides)

Functions, `lins` and `opers`

2 GF constructs that resemble Python functions:

- **linearization rules** (`lins`), which specify how ASTs are linearized
- **operations** (`opers`), general-purpose “functions”

Operator definition (GF)

```
smartNoun : Str -> Noun = \sg -> case sg of {
  _ + ("s" | "ch" | "sh") => mkNoun sg (sg + "es") ;
  _ + ("ay" | "ey" | "oy" | "uy") => regNoun sg ;
  x + "y" => mkNoun sg (x + "ies") ;
  _ => regNoun sg
} ;
```

(example from lecture 3, module `MorphologyEng`)

Function definition (Python)

```
def smart_noun(sg):
  if sg.endswith("s") or sg.endswith("ch") or ...:
    return mk_noun(sg, sg + "es")
  else if sg.endswith("ay") or sg.endswith("ey") or ...:
    return reg_noun(sg)
  else if sg.endswith("y"):
    x = sg[:-1]
    return mk_noun(sg, x + "ies")
  else:
    return reg_noun(sg)
```

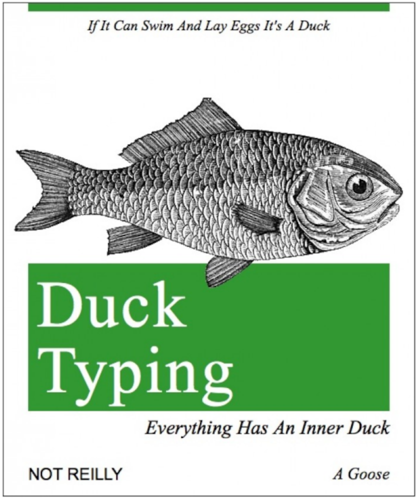
Variables

- in Python, `x = expr` creates a *variable* named `x`
- in GF, there are no variables (that vary), but you can name the result of an expression using the `let...in` syntax

```
irregVerb : (inf,past,pastpart : Str) -> Verb =
  \inf,past,pastpart ->
    let verb = smartVerb inf
    in mkVerb inf (verb.s ! PresSg3) past ... ;
```

(example from lab 1, module `MicroResEng`)

Duck typing


If It Can Swim And Lay Eggs It's A Duck
Duck Typing
Everything Has An Inner Duck
NOT REILLY
A Goose

"If it walks like a duck and it quacks like a duck, it must be a duck"

Types in Python

- **duck typing**
- **dynamic typing** (=type checking at runtime)
- **type inference** (+ optional type annotations)

```
>>> duck = Duck()
>>> person = Person()
>>> duck.walk() # ok
>>> duck.quack() # ok
>>> person.walk() # also ok
>>> person.quack()
AttributeError: 'Person' object has no attribute
'quack'
```

Types in GF

Almost the opposite of Python:

- **static typing**
- limited type inference, lots of **type declarations**
  - abstract modules are 100% made of type declarations

A simple example `abstract`

```
abstract Simple = {
    cat S ; NP ; VP ;
    fun PredVP : NP -> VP -> S ;
}
```

- `cat CatName` declares a new grammatical category called `CatName`
- `fun funName : Cat1 -> Cat2 -> ... -> CatN -> CatX` is the *type signature* of a function *funName*:
  - `Cat1 -> Cat2 -> ... -> CatN` are *parameter types*
  - `CatX` is `funName`'s *return type*

A simpl(istic) example `concrete`

In the simplest case, everything becomes a string:

```
concrete SimpleEng of Simple = {
    lincat S, NP, VP = Str ;
    lin PredVP np vp = np ++ vp ;
}
```

So, if `np = "the cat"` and `vp = "sees us"`,

```
> l PredVP np vp
the cat sees us
```

What about `resource` modules?

- reusable collections of `opers` and `params`
- can be opened ($\sim$ `imported`) in `concrete` modules
- in practice, `MicroResLan` is where you will implement most of your `Language`'s morphology

Custom types

In Python:

- everything is an **object**
- new types of objects are:
  - *defined* via **class definitions**
  - *instantiated* by calling their **constructors**

In GF:

- **grammatical categories** are:
  - *defined* by `cat` + `lincat` pairs
  - *instantiated* through `lins`
- **inflectional parameters** are defined as **algebraic data types** and used in tables

Parameters

```
-- example params for NPs in romance languages
param Gender = M | F ; -- + N if Romanian
param Number = Sg | Pl ;
param Agreement = Agr Gender Number ;
```

Tables

- usually represent *inflection tables*
- similar to Python dictionaries, but *total*
- created with `table { foo => bar }` (cf. Python's `{foo: bar}`)
- table cells are accessed with `table ! key` (cf. Python's `dict[key]`)

Tables - example

```
-- table for the Sicilian noun "boy"
table {
  Sg => "picciriddu" ;
  Pl => "picciriddi"
} ;
```

Records

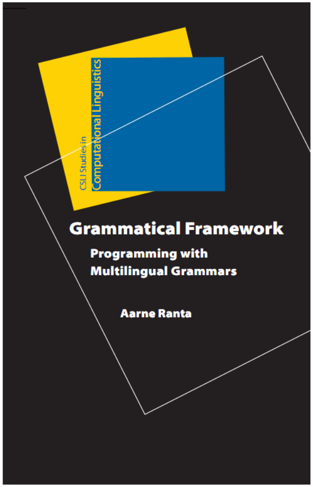
- usually used to keep track of subparts of phrases and *inherent features*
- similar to Python objects
- created with `{ foo = bar }`
- record fields are accessed with `record.key`

Records - example

```
-- lincat for nouns suitable for Romance languages
lincat Noun = {
  s : Number => Str;
  g : Gender
} ;

-- record for the Sicilian noun "boy"
{
  s = table {
    Sg => "picciriddu" ;
    Pl => "picciriddi"
  } ;
 g = M
} ;
```

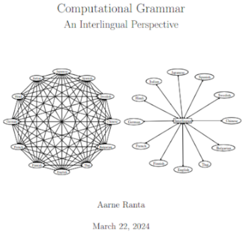
Books


CSLI Studies in Computational Linguistics

Grammatical Framework

Programming with Multilingual Grammars

Aarne Ranta


Computational Grammar

An Interlingual Perspective

Aarne Ranta

March 22, 2024

Online material

- official basic tutorial (`grammaticalframework.org/doc/tutorial/gf-tutorial.html`)
- original "GF for Python programmers" tutorial (`daherb.github.io/GF-for-Python-programmers/Tutorial.html`)
- GF programming reference manual (`grammaticalframework.org/doc/gf-refman.html`)
- shell reference (`grammaticalframework.org/doc/gf-shell-reference.html`)
- Inari's blog (`inariksit.github.io/blog`)
- Discord server (`discord.gg/EvfUsjzmaz`)
- StackOverflow (#gf tag)