Introduction to the Atlas software

Marc van Leeuwen

Laboratoire de Mathématiques et Applications
Université de Poitiers

Monday July 10, 2017 / Atlas workshop Utah

The Atlas software

Within the Atlas of Lie groups and Representations project, *software* to perform computations involving representations is central. The foundations for this software were laid (2004–2006) by Fokko du Cloux.

The heart of the software takes the form of two programs written in C++ : **`atlas`** and **`Fokko`**. While **`Fokko`** descends directly from Fokko's original program, we only discuss **`atlas`**.

Built on top of **`atlas`**, a collection of *scripts* provides high-level algorithms to complement the built-in fundamental ones.

A *Web-interface* is also under development, aimed at occasional users (therefore not further discussed here).

Structure of the `atlas` program

- An extensive library of classes and functions:
  - General purpose utilities (bitmaps, unbounded integers. . .)
  - Mathematical (combinatorial) structures (root data, $K\backslash G/B$, blocks, KLV matrices, . . .)
  - Computations with (parameters for) representations
- Interface: user types (`Param`, . . .), and functions on them
- Interpreter for the `axis` programming language:
  - Input processing (possible redirection from files, lexical scan, command isolation)
  - Syntactic analysis (parsing)
  - Type check and conversion (overloading, coercions)
  - Evaluation (computation, storage functions and values)
  - Output (possible redirection to files)
- Readline library for command line editing

Language structures in `axis`

- Function application $f(x)$ (or $x.f$; also $a \star b$ means $\star(a,b)$)
- Function abstraction $(\langle\text{type}\rangle\ \langle\text{pattern}\rangle)\langle\text{optional type}\rangle\texttt{:}\ \langle\text{expr}\rangle$
- Tuple formation $(\langle\text{expr}\rangle, \langle\text{expr}\rangle \dots)$
- Local definition $\texttt{let}\ \langle\text{pattern}\rangle{=}\langle\text{expr}\rangle\ \texttt{in}\ \langle\text{expr}\rangle$
  - Sugar: $\texttt{let}\ a{=}2,b{=}3\ \texttt{in}\dots$ means $\texttt{let}\ (a,b) = (2,3)\ \texttt{in}\dots$
  - $\texttt{let}\ f(T\ a,U\ b) = \dots$ means $\texttt{let}\ f{=}((T,U)\ (a,b))\ \texttt{:}\dots$
  - $\texttt{let}\ \langle\text{decl}\rangle\ \texttt{then}\ \langle\text{decl}\rangle\ \texttt{in}\dots$ means
    $\texttt{let}\ \langle\text{decl}\rangle\ \texttt{in}\ \texttt{let}\ \langle\text{decl}\rangle\ \texttt{in}\dots$
- Assignment $v := \langle\text{expr}\rangle$ or (parallel) $\texttt{set}\ \langle\text{pattern}\rangle := \langle\text{expr}\rangle$
- Sequencing $\langle\text{expr}\rangle;\ \langle\text{expr}\rangle$ and $\langle\text{expr}\rangle\ \texttt{next}\ \langle\text{expr}\rangle$
- Conditional $\texttt{if}\ \langle\text{cond}\rangle\ \texttt{then}\ \langle\text{expr}\rangle\ \texttt{else}\ \langle\text{expr}\rangle\ \texttt{fi}$
- Integer case $\texttt{case}\ \langle\text{expr}\rangle\ \texttt{in}\ \langle\text{expr}\rangle, \langle\text{expr}\rangle \dots\ \texttt{esac}$
- Row formation $[\langle\text{expr}\rangle, \langle\text{expr}\rangle \dots]$
- Row (and some other types) selection $\langle\text{expr}\rangle[\langle\text{expr}\rangle]$
- Slicing $\langle\text{expr}\rangle[\langle\text{expr}\rangle\texttt{:}\langle\text{expr}\rangle]$ and variants involving ~.
- Component assignment $v[\langle\text{expr}\rangle] := \langle\text{expr}\rangle$

More language structures in axis

- Counted loops: `for` *i*:⟨expr⟩ `from` ⟨expr⟩ `do` ⟨expr⟩ `od`
  - Variations: *i*, `from` optional, possible ~ before `do` and/or `od`
- Loops over values `for` ⟨pattern⟩@*i* `in` ⟨expr⟩ `do` ⟨expr⟩ `od`
  - Variations: possible ~ before `do` and/or `od`
- While loops `while` ⟨expr⟩ `do` ⟨expr⟩ `od`
  - Variations: possible ~ before `od` (reverses resulting list)
- Early exits: `break` (loops), and `return` ⟨expr⟩ (functions)

- Named tuple types, introduced by
  : ⟨typename⟩ = (⟨type⟩ ⟨name⟩,⟨type⟩ ⟨name⟩, . . .)
  - Field selection ⟨expr⟩.*field*
  - Field assignment *v*.*field* := ⟨expr⟩ (assigns new tuple to *v*)
- Named union types, introduced by
  : ⟨typename⟩ = (⟨type⟩ ⟨name⟩|⟨type⟩ ⟨name⟩| . . .)
  - Injection into the union ⟨expr⟩.*tag*
  - Case distinction clauses (on an expression of union type)
    `case` ⟨expr⟩ |⟨pattern⟩.*tag*: ⟨expr⟩ |⟨pattern⟩.*tag*: ⟨expr⟩ . . . `esac`