TiCL: the prototype

Star TEX: the Next Generation (Season 2)

Didier Verna

didier@lrde.epita.fr
@didierverna
facebook/didier.verna
http://www.lrde.epita.fr/˜didier

TUG 2013, October 23 – 26

TEX
The `[final]` *frontier.*
These are the voyages,
Of a software enterprise.
Its continuing mission:
To explore new tokens,
To seek out a new life,
New forms of implementation…

A modernized TEX
Previously, on Star TEX TNG…

TEX's strength is in the quality of its typesetting, *not* in its programmatic interface.

Keep the typesetting functionality but provide…

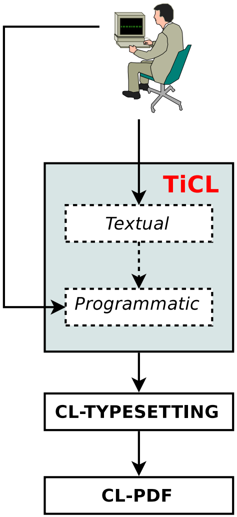
- A more modern and consistent API
- Real programming capabilities
- Still simple to use (at least for simple things)
- Extensibility / customizability

Fitness of Lisp

Previously, on Star TEX TNG. . .

- Existing approaches are heterogeneous
- What about a fully integrated approach ?
- Industrial-scale general purpose language
  - Multi-paradigm
  - Highly optimizable
  - Pletora of libraries
- Scripting / extension language
  - Highly dynamic
  - Highly reflexive
  - Easy to learn (no syntax)

TiCL: the prototype
Current architecture



This is where we start

Not very TEXy…

cl-typesetting Hello World

```
(defun first-document (&key (file "/tmp/texput.pdf"))
  (tt:with-document ()
    (let ((content (tt:compile-text ()
                     (tt:paragraph () "This␣is␣some␣text."))))
      (tt:draw-pages content)
      (when pdf:*page* (typeset:finalize-page pdf:*page*))
      (tt:write-document file))))
```

LaTeX-like programmatic layer: step 1
Still full of Lisp idioms

- Global variables: `*title*`, `*author*` *etc.*
- Functions: `document-class`, `make-title` *etc.*
  Note: keywords arguments
- Macros: `with-document`, `with-section` *etc.*

LaTeX-like programmatic layer: step 2
Less full of Lisp idioms

- Non lispy accessors (although for constants)
- Non lispy function (macro) names
- Raw symbols instead of keywords
  - for constant arguments
  - Requires a macro layer

Example

```
;; Before:
(document-class :article :paper :letter :pt 12)

;; After:
(documentclass article :paper letter :pt 12)
```

LaTeX-like programmatic layer: step 3
Even less full of Lisp idioms

- Symbol macros
  Turning 0-ary function calls into mere symbols
- More non lispy function (macro) names

Example

```
;; Before:
(make-title)
(table-of-contents)

;; After:
maketitle
tableofcontents
```

LaTeX-like programmatic layer: step 4
Getting rid of some `with-*` idioms

- A `(par)` command instead of the `with-par` macro
- And the `par` symbol macro that goes with it.

Example

```
;; Before:
(with-par "bla␣bla␣bla")
(with-par "bla␣bla␣bla")

;; After:
"bla␣bla␣bla" par "bla␣bla␣bla"
```

LaTeX-like programmatic layer: step 5

Getting rid of some other `with-*` idioms

- Standalone sectionning commands
- Need to get rid of underlying macros
- Require explicit state management
  e.g. PDF outline levels

Example

```
;; Before:
(with-section "Title"
  "bla␣bla␣bla" par
  "bla␣bla␣bla")

;; After:
(section "Title")
"bla␣bla␣bla" par
"bla␣bla␣bla"
```

LaTeX-like programmatic layer: step 6

The “empty lines” trick

- Replacing `par` with empty lines in Lisp strings
- Requires overriding `cl-typesetting`’s behavior
- But that’s easy (it’s Lisp) !

Example

```
;; Before:
(section "Title")
"bla␣bla␣bla" par
"bla␣bla␣bla")

;; After:
(section "Title")
"bla␣bla␣bla

bla␣bla␣bla"
```

LaTeX-like programmatic layer: step 7
LaTeX-like named environments

- When impossible to get rid of the macro layer
- Syntax extension through (read-time) macro characters

Example

```
;; Before:
(with-document
  "bla␣bla␣bla"
  "bla␣bla␣bla")

;; After:
{begin document}
  "bla␣bla␣bla"
  "bla␣bla␣bla"
{end document}
```

LaTeX-like textual layer
The ultimate goal

- **Idea**
  - Remain text-driven instead of program-driven
  - Convert automatically to the programmatic layer
- **Implementation**
  - Use \ as an escape (to Lisp) character
  - Everything else is accumulated into Lisp strings

Example

```
;; Before:
(section "Lorem␣Ipsum")
"Lorem␣" (textbf "ipsum") "␣" (textit "dolor") "␣sit␣amet,␣...

;;␣After:
\(section␣"Lorem Ipsum")
Lorem␣\(textbf␣"ipsum")␣\(textit␣"dolor")␣sit␣amet,␣...
```

The merits of extensibility
By example

- **Rivers detection**
  - Requires introspection (boxes internal representation)
  - But that’s easy (it’s Lisp, and OO) !
- **Implementation**
  - New kind of box (subclass of `vbox`)
  - Collects content subject to rivers detection
  - Additional stroking method for this new box
  - User-level “rivers” environment

Conclusion

- Last year: fitness of Lisp for a TEX implementation
- This year: proof of concept
- Remember the objectives ?
  - A more modern and consistent API
  - Real programming capabilities
  - Still simple to use (at least for simple things)
  - Extensibility / customizability

These were the voyages,
Of a software enterprise.
Its continuing mission:
To explore new tokens,
To seek out a new life,
New forms of implementation.
To `\textbf{go}`,
Where no TEX has gone before!

Live long and prosper!

Questions?