Steel Bank Common Lisp

A Sanely-Bootstrappable Common Lisp

Christophe Rhodes

Goldsmiths, University of London

Thursday 15th May

Common Lisp

- Draws together the best bits of 1980s-era Lisps;
- ANSI-standardized language;
- Multiply implemented (11 counted in a recent survey);
- A programmable programming langauge;
- Quite good really.

Steel Bank Common Lisp

History

- early 1980s: SPICE Lisp, DARPA "Research on Parallel Computing" contract (CMU);
- 1994: After contract (and funding) ended, CMUCL supported by volunteers (including Rob MacLachlan);
- early 1999: Cadabra (later `GoTo.com`) hires Bill Newman for CMUCL consulting;
- December 1999: Steel Bank Common Lisp announced;
- September 2000: uploaded to `SourceForge.net`
- March 2002: first extra committer;
- June 2002: first build under unrelated host Lisp;
- May 2003: first build from C (using CLISP).

Steel Bank Common Lisp

Features

- native code compiler;
- efficient hardware arithmetic;
- SMP-capable threading (on some platforms);
- generational (but stop-the-world) garbage collection;
- Full CLOS and full MOP (with documented exceptions);
- Language Extensions:
  - practically-oriented: event handling, OS bindings;
  - more theoretical: extensible sequence protocol, extensible specializers.

Self-sustainability

Better Go Programs?

Bill Newman's particular interest:

- hard problem: strategy *and* tactics;
- large search space: $3^{361} \sim 10^{172}$ board configurations;
- no cries of victory yet.

Lisp as alternative to C++:

- We would like to write our programs in Lisp...
- ... including our Lisp compilers, interpreters and runtimes.

Self-sustainability

So what, then?

Software that modifies itself:

- Fun to develop;
- Rapid development;
- Rapid feature inclusion.

User and developer communities:

- Future-proof ("bus-proof");
- Provide confidence to users;
- Train users to become developers;
- Control over one's own destiny.

Overall Philosopy

Building a new thing from a blueprint

versus

Clone, mutate, and do transplant surgery
(but keep the patient alive!)

Building the system

- Install a (sufficiently reasonable) Common Lisp;
  - Supported: ABCL, CMUCL, OpenMCL (ClozureCL), SBCL, XCL;
  - Sometimes works: CLISP;
  - Unsupported: Allegro, Corman CL, Lispworks, Lisp500, GCL;
- `sh ./make.sh '<lisp> arg1 arg2'`;
- Wait (but not too long):
  - compile the sources in the host lisp: builds an SBCL cross-compiler;
  - compile the sources using the cross-compiler;
  - 'genesis': build a memory image from the cross-compiled sources;
  - finish up the build.

An example problem

`defstruct` metacircularity

```
(defstruct defstruct-defscription
  (name (missing-arg) :type symbol :read-only t)
  (doc nil :type (or string null))
  (slots () :type list)
  ...)
```

December 2002 (SBCL 0.7.10.9):

New slot in `defstruct-description`, to handle aliasing of accessors by subclasses.

An example problem

`defstruct` metacircularity

```
(defstruct defstruct-defscription
  (name (missing-arg) :type symbol :read-only t)
  (doc nil :type (or string null))
  (slots () :type list)
  ...)
```

Self-modifying system (traditional Lisps):

- Place new slot at the end of `defstruct-description` structure;
- Build once, accepting modification of existing structure;
- Now write code that *uses* new slot;
- Build again.

An example problem

`defstruct` metacircularity

```
(defstruct defstruct-defscription
  (name (missing-arg) :type symbol :read-only t)
  (doc nil :type (or string null))
  (slots () :type list)
  ...)
```

SBCL:

- Place new slot anywhere in `defstruct-description` structure;
- Write code that uses new slot;
- Build and enjoy.

Hacking on the system

Convenience

Self-sustaining code:

- most of the system sources can be live-edited and patched directly;
- SLIME makes this convenient; special logic to help package translation work;
- there are some exceptions to live patching;
- changes don't trigger recompilation of dependents:
  - macros;
  - code generation templates.

Hacking on the system

Reliability

Self-sustaining community:

- build results predictable;
- no dependencies of the result on the build environment.

Future Work

- Address architectural issues:
  - CLOS (PCL) in the cross-compile;
  - allow clearer expression of compiler algorithms.
- Support other Common Lisp implementations:
  - CLISP, ECL
    - Building SBCL stresses many parts of a CL implementation;
    - Good for finding bugs!
    - (Both in SBCL and elsewhere)
- Make SBCL a better Common Lisp:
  - well-designed extensions;
  - good support for libraries;
  - helpful developer tools;
  - grow user and developer community.

Conclusions

- Self-sustaining systems are fun to work on;
- Growing a maintainer community is hard;
- Software and its developer community as the whole system.

Thanks!

- Bill Newman;
- Dan Barlow;
- Alexey Dejneka, Kevin Rosenberg, Rudi Schlatte, Patrik Nordebo, Nikodemus Siivola, Andreas Fuchs, Nathan Froyd, Juho Snellman, Brian Mastenbrook, Brian Downing, Gábor Melis, Paul F. Dietz, Thiemo Seufer, Cyrus Harmon, Teemu Kalvas, NIIMI Saitoshi, Alastair Bridgewater, Paul Khuong, Miles Egan, Dave Roberts
- ... and that's just the 'committers'! 100 or so other contributors, too;
- CMUCL: Raymond Toy, Pierre Mai, Eric Marsden, Gerd Möllmann;

Thanks!

- Bill Newman;
- Dan Barlow;
- Alexey Dejneka, Kevin Rosenberg, Rudi Schlatte, Patrik Nordebo, Nikodemus Siivola, Andreas Fuchs, Nathan Froyd, Juho Snellman, Brian Mastenbrook, Brian Downing, Gábor Melis, Paul F. Dietz, Thiemo Seufer, Cyrus Harmon, Teemu Kalvas, NIIMI Saitoshi, Alastair Bridgewater, Paul Khuong, Miles Egan, Dave Roberts
- ... and that's just the 'committers'! 100 or so other contributors, too;
- CMUCL: Raymond Toy, Pierre Mai, Eric Marsden, Gerd Möllmann;