Specifying code walking support

Michael Raskin, raskin@mccme.ru

Dept. of CS, TU Munich

April 1, 2019

Extending Common Lisp: Theory

Common Lisp is a programmable programming language

Major language features can be added via macros

No limits — code is data!

Conforming code is portable between implementations

Extending Common Lisp: Practice

CLOS was a library
Cells, ContextL

If necessary, one can code walk!
Iterate

Implementation-specific code walking code bitrots

Agnostic-Lizard: 2017, portable — still not too late to the game?
Test suite to break the other portable code walkers...

Extending Common Lisp: Practice

CLOS was a library
Cells, ContextL

If necessary, one can code walk!
Iterate

Implementation-specific code walking code bitrots

Agnostic-Lizard: 2017, portable — still not too late to the game?
Test suite to break the other portable code walkers...

Code walking support

Standard is not enough

  opaque lexical environments

Common Lisp: the Language (2 ed.) *is* enough

Implementations have enough functions... with unique names

Interesting expansions of standard macros

Standard macros

```
(defun f (x) (1+ x))
```

is currently allowed to expand to any of the following:

```
; CLISP
(impl::define-function
  'f (function f '(lambda (x) (block f (1+ x)))))
; SBCL
(impl::define-function
  'f (impl::named-lambda f (x) (block f (1+ x))))
; Please no
; Bonus «no» if define-function is special operator
(impl::define-function 'f "x -> (1+ x)")
; Please?
(impl::define-function
  'f (labels ((f (x) (1+ x))) (function f)))
```

Can we agree on what implementations can signal?

CDR?

CDR-NN package for necessary functionality?
Either `with-augmented-environment`, or `augment-environment`, or `environment-entry-names`, or `macroexpand-all`.

:CDR-NN-EXPANSIONS feature for standard-conforming code in the expansions?

Goals:
Unified naming for function/macro aliases — cheap for implementations
Full code-walking friendliness — cheap enough to have a chance...

Current draft: https://gitlab.common-lisp.net/mraskin/cdr-walkability

A Short Note on Tries and Compressed Data Structures

– Andrew Lawson

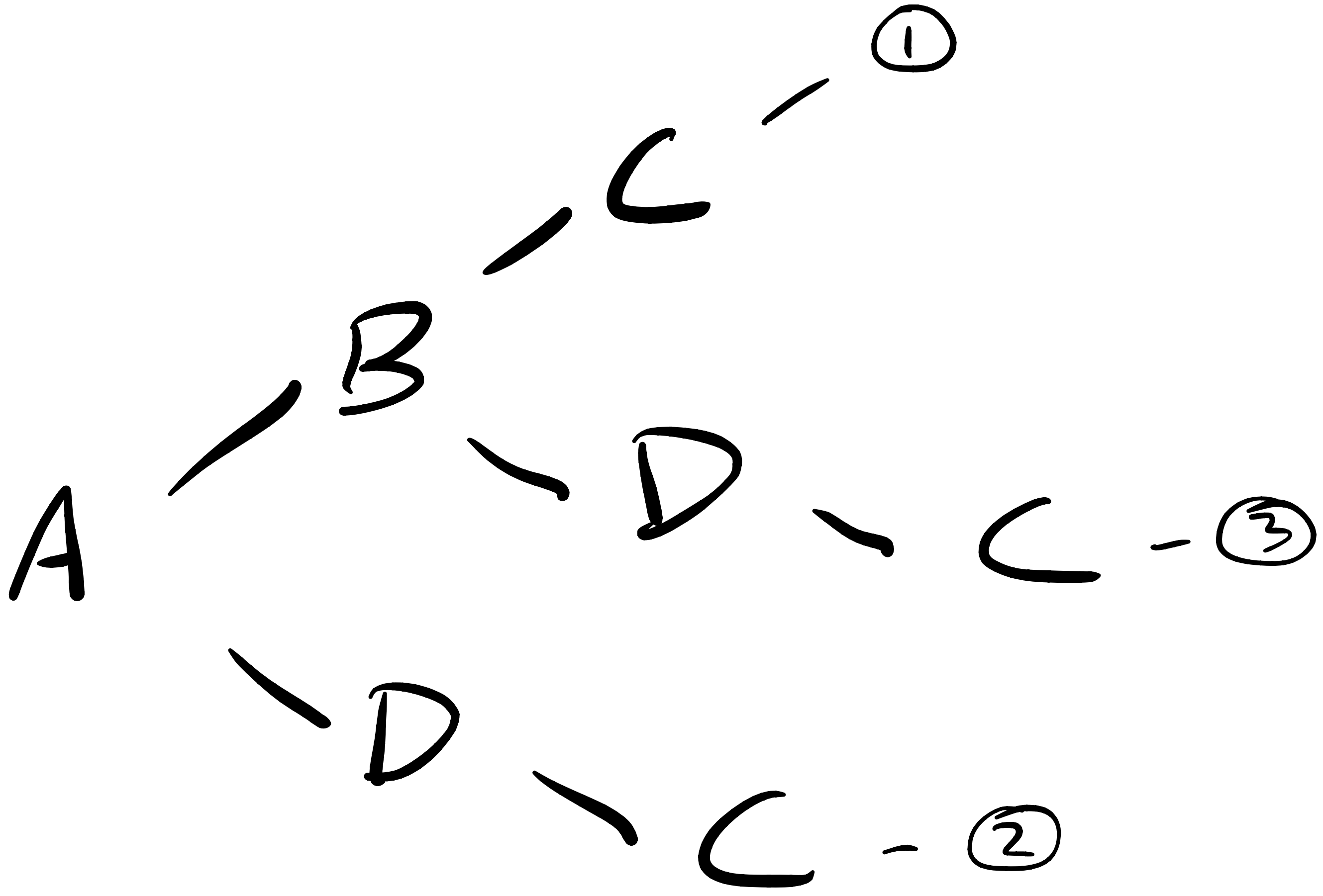
(RavenPack)

A – B – C –> RESULT ①

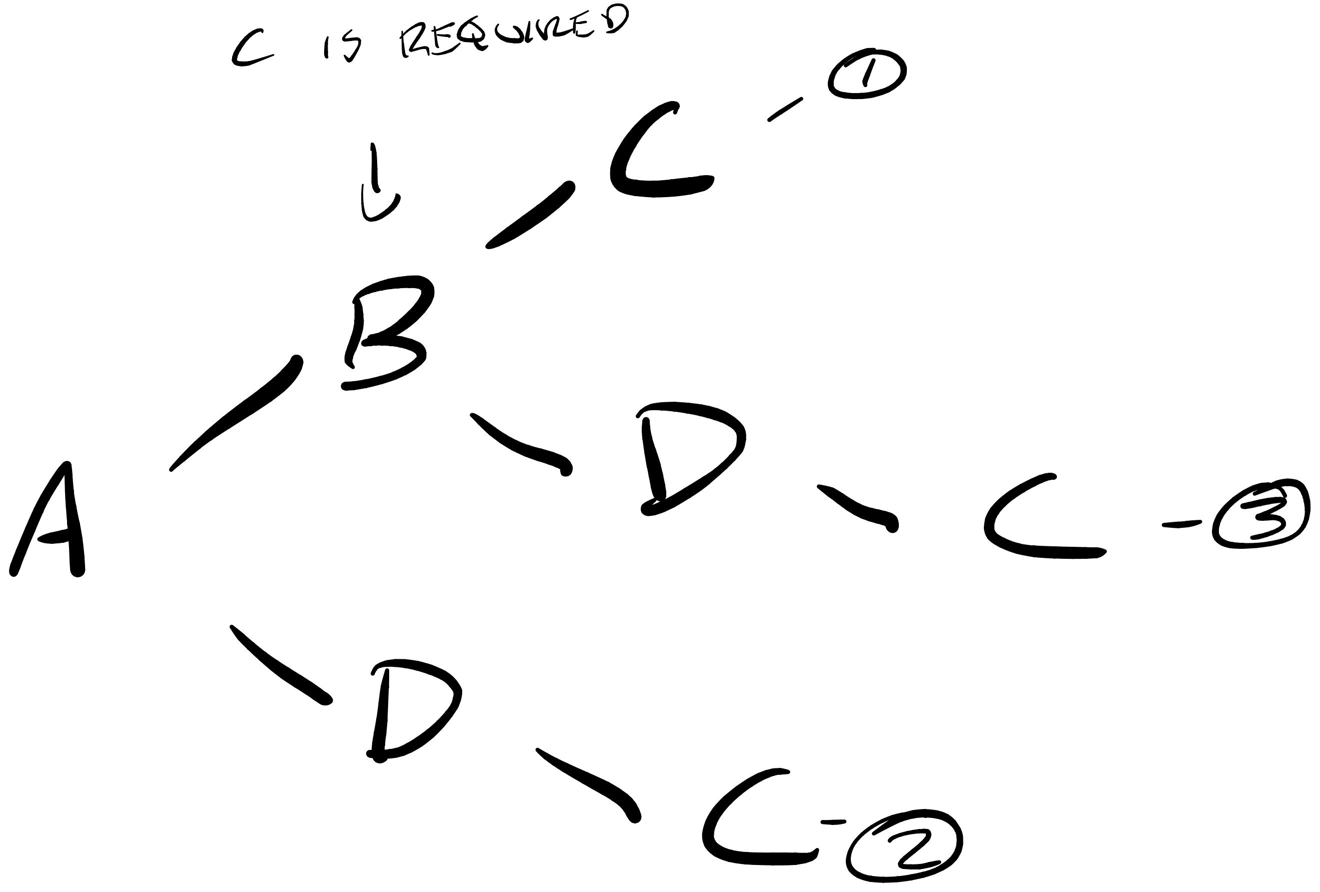
A – D – C –> RESULT ②

A – B – D – C –> RESULT ③

WE HAVE SEQUENCES OF TOKENS THAT INDICATE A PARTICULAR RESULT.

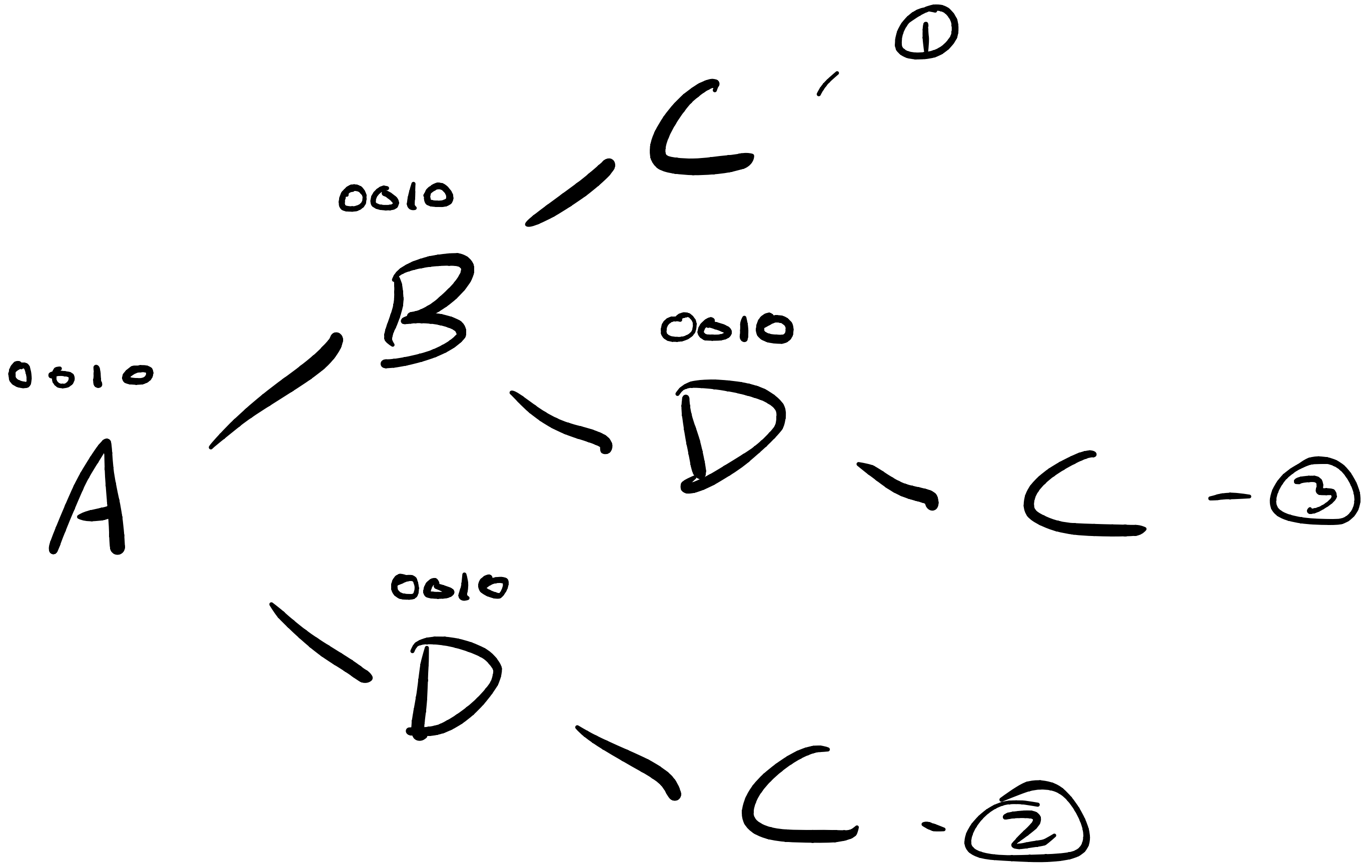


We Build a Trie and can Easily write a Function That Checks For Matches



CAN WE PRUNE THE TREE?

AT ANY NODE WE CAN ENCODE THE SYMBOLS NEEDED IN THE FOLLOWING NODES IN A BIT ARRAY. WITH 100,000 SYMBOLS WE BUCKET BY FIRST LETTER, STICKING TO 4 LETTERS WE GET $\longrightarrow$



WORK OUT THE BIT ARRAY FOR THE CANDIDATE SEQUENCE

A - B - C ⟹ #(1 1 1 0)

BUT WE DON'T WANT TO BUCKET BY ONLY FIRST LETTER.

IN THE REAL WORLD WE ARE LIMITED BY MEMORY FOR BIT VECTORS, OR THE INTEGER SIZE LIMITS

$\longrightarrow$ BLOOM FILTERS

(NEW DISCOVERY FOR ME, I'M NOT A COMPUTER SCIENTIST)

WIKIPEDIA

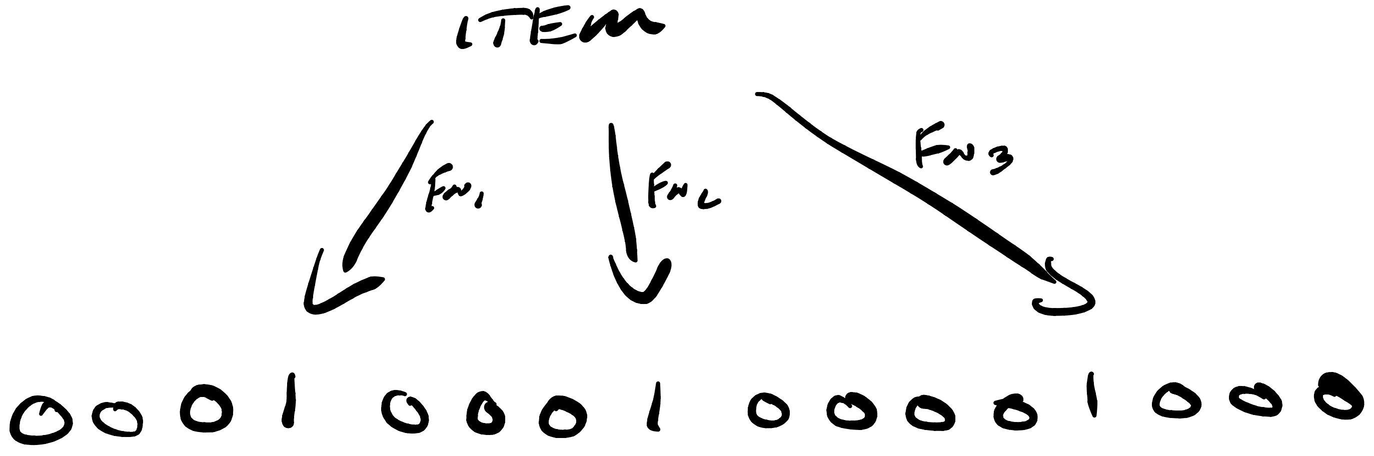
↓

A Bloom filter is a space-efficient probabilistic data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set. False positive matches are possible, but false negatives are not

WE REPRESENT OUR BLOOM FILTER AS A BIT ARRAY AND A SET OF HASH FUNCTIONS

ADDING AN ELEMENT:

WE APPLY OUR HASH FUNCTIONS AND SET THE INDICATED BITS



QUERYING AN ELEMENT

RUN THE HASH FUNCTIONS AGAIN

NOT ALL BITS SET → DEFINITELY NOT IN THE SET

ALL BITS SET → PROBABLY* IN THE SET

* VARIES DEPENDING ON NO. BITS, NO. HASH FUNCTIONS

So:

- Compact Representation using Same Comparison Logic Etc
- False Positives are fine, just means that we pass to the next Node
- Can have a very Granular Filter
- Now working on tuning & Measuring

BUT:

SURELY THERE ARE BETTER WAYS TO DO ALL OF THIS (TREE PRUNING).

REMINDER:

RAVENPACK IS HIRING
MULTIPLE LISP PROGRAMMERS
RIGHT NOW

→ COME TALK TO
(OR ANDREW NICK KYLE)

DAVE COOPER <DAVE@GEN.WORKS>

COMMON LISP FOUNDATION UPDATE 2019

BACKGROUND

- Ten years of advocating Common Lisp (founded 2009)
- Volunteers working in “copious” spare time to promote Common Lisp
- Provides resources for development common-lisp.net including GitLab, mailing lists, project pages, and continuous integration

COMMON-LISP.NET

- Complete site revamp (late 2018)
  - Engaged a team of volunteers spearheaded by Marinano Montone (over 72 discrete contributions)
  - Site built automatically from every commit to the repository
  - Eating our own dogfood: transitioned from Ruby tooling to Common Lisp site generator
  - Test server is using portable Allegro Serve

FUNDRAISING

- ASDF Appreciation Fundraiser
  - Sponsors contributed $5000 matching funds
  - Currently achieved over half this goal
  - Fundraiser has been back online for ELS2019
    - https://payments.common-lisp.net/asdf

FUTURE DIRECTIONS

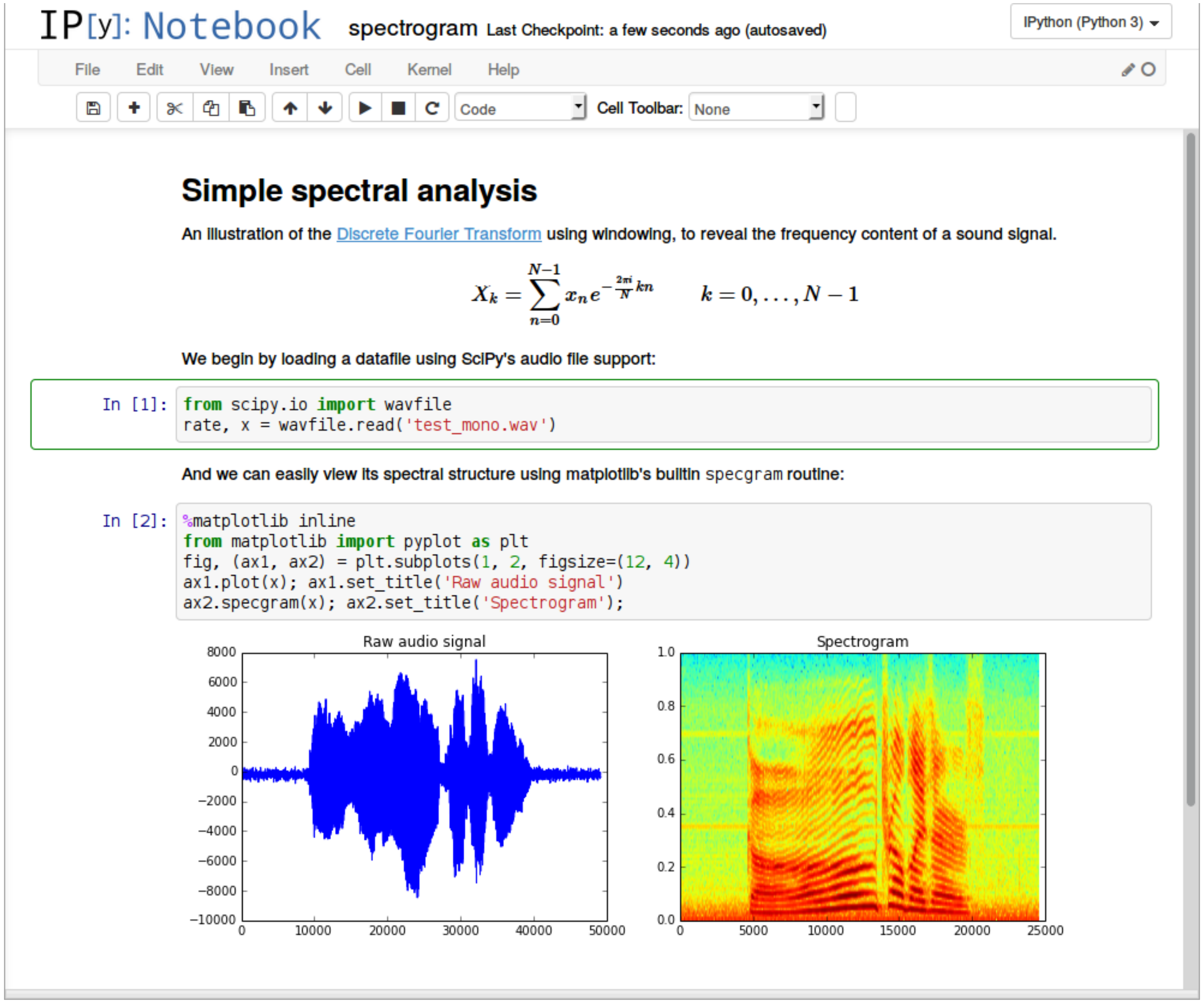
- Fundraising
  - We would like to allow each project hosted with common-lisp.net to activate their own fundraiser
- Services
  - Would like to provide convenient ways for projects to utilize continuous integration services across CL implementations
  - Hosting of source code escrow with particular support for Common Lisp based projects and products

flight-recorder

Vsevolod Dyomkin

European Lisp Symposium ’2019

Inspiration



Simple spectral analysis

An illustration of the Discrete Fourier Transform using windowing, to reveal the frequency content of a sound signal.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \dots, N-1$$

We begin by loading a datafile using SciPy's audio file support:

```
In [1]: from scipy.io import wavfile
        rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin `specgram` routine:

```
In [2]: %matplotlib inline
        from matplotlib import pyplot as plt
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
        ax1.plot(x); ax1.set_title('Raw audio signal')
        ax2.specgram(x); ax2.set_title('Spectrogram');
```

But only

* text-based
* Emacs-based

Not a poor-man’s REPL, but
A robust history recorder

slime-history.eld

- custom sexp-based format — not human-readable

```
;; -*- coding: utf-8-unix -*-
;; History for SLIME REPL. Automatically written.
;; Edit only if you know what you're doing
("(push \"~/prj/lisp/cl-nlp/\" asdf:*central-registry*)
(push \"~/prj/lisp/wiki-lang-detect/\" asdf:*central-
registry*)
(push \"~/prj/lisp/crawlik/\" asdf:*central-registry*)
(ql:quickload :cl-nlp)
(ql:quickload :crawlik)
(in-package :nlp-user)"
…
```

slime-history.eld

- custom sexp-based format — not human-readable
- not robust
- hard to control
- etc.

Robustness Requirements

* recording history from concurrently running sessions
* keeping unlimited history
* identifying the time of the record and its context

frlog

```
* client-server application:
  CL server, EL/CL/… client
* logs to human-readable *.md
```

```
### nlp-user (14) 2018-10-23_19:09:21

    (drakma:http-request "http://schema.org/" :accept "application/ld+json")

;;; 2018-10-23_19:09:27

    ((:ACCESS-CONTROL-ALLOW-ORIGIN . "*") (:CONTENT-TYPE . "application/ld+json")
                                          (:VARY . "Accept, Accept-Encoding")
                                          (:ETAG . "d61d9aceb39f0342f4dc4c54cb30baf4")
                                          (:LAST-MODIFIED . "Thu, 27 Sep 2018 13:34:07 GMT")
                                          (:X-CLOUD-TRACE-CONTEXT . "f1823f5633d337de3df2369fe1e86d1c")
                                          (:DATE . "Tue, 23 Oct 2018 16:08:59 GMT") (:SERVER . "Google Frontend")
                                          (:CONTENT-LENGTH . "136698") (:CACHE-CONTROL . "public, max-age=600") ...)

;;; 2018-10-23_19:09:27

    200

;;; 2018-10-23_19:09:27

    #<FLEXI-STREAMS:FLEXI-IO-STREAM {101D8B3AA3}>

;;; 2018-10-23_19:09:27

    T

;;; 2018-10-23_19:09:27

    #<PURI:URI https://schema.org/docs/jsonldcontext.json>

;;; 2018-10-23_19:09:27

    "OK"
```

.EL problem

It uses Emacs native indenting function:

```
(with-temp-buffer
  (lisp-mode)
  (insert text)
  (let ((inhibit-message t))
    (indent-region 0 (point))
    (string-trim (buffer-string)))))
```

But after some time we get:

```
error in process sentinel: url-http-idle-sentinel: Lisp nesting exceeds ‘max-
lisp-eval-depth’
error in process sentinel: Lisp nesting exceeds max-lisp-eval-depth
```

Thanks! Read more

http://lisp-univ-etc.blogspot.com/2018/09/ann-flight-recorder-robust-repl-logging.html