Il était une fois un langage fonctionnel

Alice BRENON

2025-02-18

Il était une fois. . .

Le problème (python)

c.f. https://issues.guix.gnu.org/75400

Le 6 janv. 2025 dernier:

`guix pull`

→ guix @ *7a7c01b393ecb20dff142b9ea9bf8317e994bef2*

`guix shell -m manuscrit.scm` (∋ `python-spacy`)

→ `python-jose` est cassé

. . . et à la fin <*SPOILER*>

```
commit db9b46ad53a5c3bfa2e4ba59af24611fddb853da
Author: Ricardo Wurmus <rekado@elephly.net>
Date:   Thu Jan 2 20:13:16 2025 +0100

    gnu: python-jose: Fix build.

    * gnu/packages/python-web.scm (python-jose)[build-system]: Use
    pyproject-build-system.
    [arguments]: Disable some failing tests; remove custom 'check phase.
    [native-inputs]: Add python-setuptools and python-wheel.

    Change-Id: I39831da68beb6e7e80cd97df04310676e2cdf92a
```

La bonne solution c'était juste de désactiver les tests : /

Les ennuis

/var/log/guix/drvs/fx/d67zlmgm6mh90pdc4db8j22l47r15l-python-jose-3.3.0.drv.gz (`zless`)

- cause: erreur bête dans un test
- on désactive les tests:
  `guix shell --without-tests=python-jose python-jose`
- ça marche !
  (/gnu/store/3bzawvwsfy2rsalx5kqr580jqix4wd1n-python-jose-3.3.0)
- on réeesaye `python-spacy`:
  `guix shell --without-tests=python-jose python-spacy`

→ en fait `python-aws-smart-open` est cassé aussi:

/var/log/guix/drvs/9d/zkzchs5hm9f9dy2akgxfdwgl3a7bmi-python-aws-xray-sdk-2.12.0.drv.gz

Un peu de recul

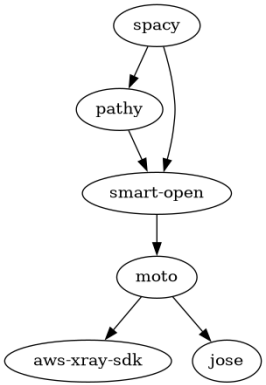

Figure 1: Quelques dépendances de spacy

Les libs python

- `python-jose`: test peu robuste
- `python-aws-xray-sdk`: erreurs avec `sqlalchemy` ?!
- `guix graph --path python-{spacy,jose}`

Que signifient ces erreurs pour moi ?

- `smart-open`: interface vers des clouds douteux
- `moto`: mock d'une lib python (donc tests)
- ... et `spacy` en fait ?
  - tokenization pour
  - ma lib `geopyck`
  - qui génère de la visualisation de mon manuscrit

Première tentative

idée retirer ces paquets de l'arbre des dépendances

- on va dans les sources de guix[1]
- on va éditer les paquets:
  `./pre-inst-env guix edit python-moto`
- on rebuild: `make`
- on utilise la nouvelle version pour construire les paquets:
  `./pre-inst-env guix build python-moto`

problème ça n'est pas très utilisable tel quel

→ on voudrait pouvoir définir *notre* version du paquet

[1] https://guix.gnu.org/manual/devel/en/html_node/Building-from-Git.html

Guile

Un langage «LISP»

Principes

- « LISt Processing »
- → tout est construit sur les listes (chaînées)
- listes notées entre parenthèses

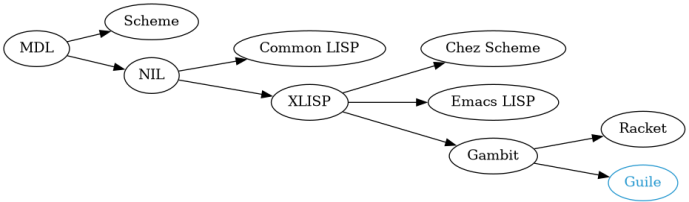

Figure 3: Une famille de langages

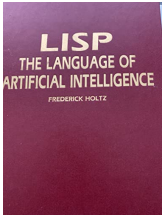

Figure 2: LISP, langage de l'IA et du futur (dans les années 1970)

Le cœur de ces langages

- `'()` / `(list)`
- `cons`
- `car` («address»)
- `cdr` («decrement»)

Comment se familiariser avec guile ?

- Utiliser le **R**ead **E**val **P**rint **L**oop: `guile` !
- Un peu plus ergonomique: `rlwrap guile`
- avec tous les modules `guix` et leurs dépendances: `guix repl`
- version confort (couleur, auto-complétion)
  - `guix shell guile{,-{readline,colorized}}`
  - `(use-modules (ice-9 readline) (ice-9 colorized))`
  - `(activate-readline)`
  - `(activate-colorized)`
  - `(> ~/.guile)`

astuce le REPL nomme tous les résultats d'évaluation `$n` (on peut les réutiliser !)

Le langage

Types primitifs

- booléens
- nombres (entiers, flottants...)
- caractères, chaînes

Paires

- operateur `.`
- base des listes ! (c'est `cons`)
- ... ou de toute structure

Application

- `(<FUNCTION> <ARG0> <ARG1> ... <ARGN>)`
- `()` → erreur de syntaxe !

Exemples

- `(string-append "café" " " "guix")`
- `(if <BOOL> <THEN> <ELSE>)`

Capacités d'abstraction

Deux opérateurs pour «retenir» l'évaluation:

- guillemet: `'`
- guillemet arrière: `` ` `` (acolyte: `,`)

Applications

- liste vide `'()`
- symboles `'guix` (≠ string)
- remettre l'évaluation: `` `(,$5) ``
- à part ça, équivalent:
  `(eq? 'guix `guix)`

Bon, et les fonctions ?

- lambda-calcul → «(lambda-)abstractions»
- évidemment, une liste !
- une valeur comme les autres

```
(lambda <ARG> <BODY>)
```

- `<ARG>`: une liste (formellement, une variable ou une liste)
- `<BODY>`: une expression

Ça permet

- `fold`: «reduce» (gauche)
  `(a -> b -> b) -> b -> [a] -> b`
- `map`: «transform»
  `(a -> b) -> [a] -> [b]`

Où est le manuel ?

https://www.gnu.org/software/guile/manual/html_node/

Ressources

- **S**cheme **R**equest **F**or **I**mplementation
  - `(srfi srfi-1)` : constructeurs de listes
  - `(srfi srfi-4)` : vecteurs de nombres
  - ...
- ICE-9
  - `(ice-9 match)` : pattern matching
  - `(ice-9 ftw)` : système de fichier
- + ce qui est défini dans Guix

Mise en œuvre

Changer la déclaration d'un paquet

1ère idée recopier / éditer / renommer dans un channel local

→ trop d'imports

mieux l'héritage

Héritage

- `inherit`
- puis déclarations des champs normalement
- «getter»: `<STRUCTURE>-<CHAMP>`

```
(package
  (inherit python-moto)
  (version
    (package-version python-moto)))
```

Édition «à la main»

srfi-1[2] fournit `remove`: *pred list*

```
(propagated-inputs
  (remove (lambda (pkg) (equal? pkg python-jose))
          (package-propagated-inputs python-moto)))
```

→ ça ne marche pas !

Autrefois…

```
(inputs `(("libffi" ,libffi)))
```

[2] https://www.gnu.org/software/guile/manual/html_node/SRFI_002d1-Filtering-and-Partitioning.html

Édition «à la main»

srfi-1[3] fournit `remove`: *pred list*

```
(propagated-inputs
  (remove (lambda (pkg) (equal? (car pkg) "python-jose"))
          (package-propagated-inputs python-moto)))
```

[3] https://www.gnu.org/software/guile/manual/html_node/SRFI_002d1-Filtering-and-Partitioning.html

Et maintenant

- «brancher» notre `python-moto` dans `python-smart-open`
- faire pareil pour `python-pathy`
- on regarde dans `guix`: `modify-inputs`
  - `delete`
  - `replace`

```
(define-public python-smart-open-with-custom-moto
  (package
    (inherit python-smart-open)
    (native-inputs
      (modify-inputs (package-native-inputs python-smart-open)
        (replace "python-moto" python-moto-sans-tests)))))
```

Et enfin

- plus qu'à faire ça récursivement
- on re-regarde dans `guix`: `package-input-rewriting`

```
(define-public python-spacy-tmp-fix
  (package
    (inherit
      ((package-input-rewriting
         `(,python-smart-open . ,python-smart-open-with-custom-mo
       python-spacy))))
```

Conclusion

Guile, un langage

- très simple
- un peu dur à cerner ?
- mais flexible à manipuler
- très pratique pour des contournements temporaires