How to be a good host: miniKanren as a case study

Jason Hemann and Dan Friedman
Indiana University
case study: a detailed account giving information about the development of a person, group, or thing, especially in order to show general principles
Outline

1. miniKanren is a particularly interesting case
2. what makes a desirable miniKanren host?
3. draw general lessons from our answers
miniKanren

shallow embedding

Logic programming
miniKanren

shallow embedding

Logic programming

The Reasoned Schemer
“A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.”

–Deursen, Klint, and Visser
Embedded Domain Specific Language
Embedded Domain Specific Language

embedded in a host

inherits some of host’s infrastructure

Building Domain-Specific Embedded Languages

Paul Hudak

Yale University, Department of Computer Science
Box 208285, New Haven, CT 06520, USA
hudak@cs.yale.edu,
http://www.cs.yale.edu/users/hudak-paul.html

Abstract: A short treatise on how and why we should design domain-specific embedded languages to solve the problem of building and gracefully evolving large and complex software.
A first example

```
(define-relation (appendo l s o)
  (conde
   ((== '() l) (== s o))
   (fresh (a d)
     ((== `(,a ,d) l)
      (fresh (r)
        ((== `(,a ,r) o)
         (appendo d s r))))))

(run 1 (q) (appendo '((b c d) '(e f) q))
  '((b c d e f)))
```
Why miniKanren?
Practical / Commercial
Practical / Commercial

- malware detection
- processing database queries
- web testing
Practical / Commercial

- malware detection
- processing database queries
- web testing
But mostly, PL awesomeness
But mostly, PL awesomeness

exponentiation / logarithm

type checker / inhabiter

CL reducer / Y generator

interpreters / Quines
Intrinsic Reasons
< 100 lines of Racket code (1/4 for ==)
< 100 lines of Racket code (1/4 for ==)

50+ implementations in at least 30 languages
< 100 lines of Racket code (1/4 for ==)

50+ implementations in at least 30 languages

Programs are derived from host language programs ...
< 100 lines of Racket code (1/4 for ==)

50+ implementations in at least 30 languages

Programs are derived from host language programs …

… and get superpowers!
<REPL>
Critical Features
closures
closures

hygienic macros
closures
hygienic macros
S-expressions
closures
hygienic macros
S-expressions
garbage collection
... you’re just listing features of Scheme
closures
hygienic macros
S-expressions
garbage collection
closures
hygienic macros
S-expressions
garbage collection
Closures
``Foolish pupil - objects are merely a poor man’s closures.”
``Foolish pupil - objects are merely a poor man’s closures.”

``When will you learn? Closures are a poor man’s object.”
``Foolish pupil -
objects are merely a poor man’s closures.”

``When will you learn?
Closures are a poor man’s object.”

… At that moment, Anton became enlightened.
static class DelayedGoal implements Goal {
    private final GoalFn f;
    private final Var v;
    public DelayedGoal(GoalFn f, Var v) {
        this.f = f;
        this.v = v;
    }
    public Stream run(final State s) {
        return new ImmatureStream() {
            Stream realize() {
                return f.call(v).run(s);
            }
        };
    }
}
Hygienic Macros
Hygiene prevents capture during macro expansion
Hygiene prevents capture during macro expansion

Originated in Scheme (Kohlbecker ’86)
Hygienic Macro Expansion

Eugene Kohlbecker, Daniel P. Friedman, Matthias Felleisen, Bruce Duba

Computer Science Department
Lindley Hall 101
Indiana University
Bloomington, Indiana 47405 USA
(defmacro disj-macro
  [disj G] -> G
  [disj G0 G1 G2 | G*] ->
  [disj G0 [disj G1 G2 | G*]]
)

(defmacro conj-macro
  [conj G] -> G
  [conj G0 G1 G2 | G*] ->
  [conj G0 [conj G1 G2 | G*]]
)

(defmacro inverse-eta-delay-macro
  [inverse-eta-delay G] ->
  (let TEMP (intern (str (gensym s/c)))
    [lambda TEMP TEMP [freeze [G TEMP]]])
)
S-expressions
1. Change the term language
(* represent a logic term *)

type logic_term =
  | Var of var_id
  | Constant of const_value
  | List of (logic_term list)
  | Cons of logic_term * logic_term
2. implement cons, etc. if lacking from host
class Cons
  attr_reader :car, :cdr

  # Returns a Cons cell (read: instance) that is also marked an
  # later identification.
  def initialize(car, cdr)
    @car, @cdr = car, cdr
  end

  # Converts Lisp AST to a String. Algorithm is a recursive in
  # http://www.mat.uc.pt/~pedro/cientificos/funcional/lisp/gcl
  def to_s(cons_in_cdr = false)
    str = cons_in_cdr ? '' : '('
  end
Garbage Collection
“Any sufficiently complicated C or Fortran program contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.”

—Philip Greenspun
<table>
<thead>
<tr>
<th>C#</th>
<th>Erlang</th>
<th>Java</th>
<th>nu</th>
<th>Racket</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clojure</td>
<td>Extempore</td>
<td>Javascript</td>
<td>OCaml</td>
<td>Ruby</td>
</tr>
<tr>
<td>Coffeescript</td>
<td>F#</td>
<td>Lua</td>
<td>PHP</td>
<td>Scala</td>
</tr>
<tr>
<td>Common Lisp</td>
<td>Go</td>
<td>miniKanren</td>
<td>Prolog</td>
<td>Scheme</td>
</tr>
<tr>
<td>Dylan</td>
<td>Groovy</td>
<td>ML</td>
<td>Purescript</td>
<td>Shen</td>
</tr>
<tr>
<td>Elixir</td>
<td>Haskell</td>
<td>Moxie</td>
<td>Python</td>
<td>Smalltalk</td>
</tr>
<tr>
<td>C#</td>
<td>Erlang</td>
<td>Java</td>
<td>nu</td>
<td>Racket</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>---------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>Clojure</td>
<td>Extempore</td>
<td>Javascript</td>
<td>OCaml</td>
<td>Ruby</td>
</tr>
<tr>
<td>Coffeescript</td>
<td>F#</td>
<td>Lua</td>
<td>PHP</td>
<td>Scala</td>
</tr>
<tr>
<td>CommonLisp</td>
<td>Go</td>
<td><strong>miniKanren</strong></td>
<td>Prolog</td>
<td>Scheme</td>
</tr>
<tr>
<td>Dylan</td>
<td>Groovy</td>
<td>ML</td>
<td>Purescript</td>
<td>Shen</td>
</tr>
<tr>
<td>Elixir</td>
<td>Haskell</td>
<td>Moxie</td>
<td>Python</td>
<td>Smalltalk</td>
</tr>
</tbody>
</table>
It can be done
It can be done
... sort of
COME BACK WITH A WARRANT.
Other advice
**cons** cells ≠ lists, emulate or code around

explicit delays in relations avoid the macro

lists of goals, nums avoid the rest of the macros

complete search is tricky, try visualizing
Why not more?
Why not more?

- APL / J / K (array)
Why not more?

- APL / J / K (array)
- REFAL (tree)
Why not more?

- APL / J / K (array)
- REFAL (tree)
- SetL (set)
Why not more?

- APL / J / K (array)
- REFAL (tree)
- SetL (set)
- LOLcode (meme)
Why not more?

- APL / J / K (array)
- REFAL (tree)
- SetL (set)
- LOLcode (meme)
• Consider evaluation model
• Consider evaluation model

• Similar data structures helps transformation
• Consider evaluation model

• Similar data structures helps transformation

• Limit infrastructure development
“A programming language is low level when its programs require attention to the irrelevant.”

—Alan Perlis
• Consider evaluation model

• Similar data structures helps transformation

• Limit infrastructure development
• Consider evaluation model

• Similar data structures helps transformation

• Limit infrastructure development

• Usually lots of flexibility in design
• Consider evaluation

• Similar data structures helps transformation

• Limit infrastructure development

• Usually lots of flexibility in design

Thanks!

@jhemann    @dfried00

(miniKanren.org)
Photo Credits

- Kaya Kupferschmidt
- Plasmatorium (https://www.etsy.com/shop/Plasmatorium)
- Rob Silvers & Philip Greenspun (philip.greenspun.com)
- Eagle County Sheriff’s Office
- Ambir Kelo (https://ambirkelo.wordpress.com/)
- Wikimedia foundation
- @ScreenInvasion
- Saturday Morning Breakfast Cereal