# miniAdapton

## A Minimal Implementation of Incremental Computation in Scheme

Dakota Fisher, Matthew Hammer, William E. Byrd, Matthew Might

September 17, 2016

This material is partially based on research sponsored by DARPA under agreement number AFRL FA8750-15-2-0092 and by NSF under CAREER grant 1350344. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

# Memoization

- Remember (i.e. "make a memo of") previous results
- Classic example: fibonacci

$$fib(0) = 1; fib(1) = 1; fib(n) = fib(n-1) + fib(n-2)$$

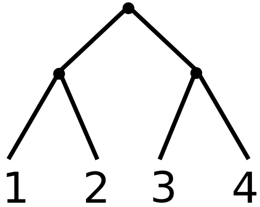
- Naively-implemented fibonacci is exponential
- Using only memoization, fibonacci can be made linear
- Memoization can yield algorithmic speedups
- Memoization forbids mutation

# A Memoized Function

```
(define-memo (max-tree t)
  (cond
    ((pair? t)
     (max (max-tree (car t))
          (max-tree (cdr t))))
    (else t)))
```

# A Memoized Function

```
>(max 1 2 3 4)
4
>(max-tree '((1 . 2) . (3 . 4))
4
```



# User Session with Memoization

```
> (define some-tree '((1 . 2) . (3 . 4)))
> (max 1 2 3 4)
4
> (max-tree some-tree)
4

> (set-cdr! some-tree 5)
> some-tree
((1 . 2) . 5)
> (max 1 2 5)
5
> (max-tree some-tree)
4
```

# What is incremental computation?

- Reuse previous results/computations (like memoization)
- ... specifically for changing inputs

# What is Adapton?

- a general, language-based approach to incremental computation
- “memoization supporting mutation”
- How: remember not just the result of a computation, but also keep track of dependencies between computations
- Specifically, Adapton creates a dependency graph called the DCG (or demanded computation graph).

# What is Adapton?

By analogy to thunks (zero-argument procedures) and promises (memoized thunks)

| Feature | Thunk | Promise | Adapton "Promise" |
|---|---|---|---|
| Stored | closure | + result | + dependencies |
| Avoids Recomputation | no | yes | when correct |
| Supports Mutation | yes | no | yes |

# Aside: Why Mutation?

We live in a temporal world full of mutation, some programs dealing with mutation:

# Aside: Why Mutation?

We live in a temporal world full of mutation, some programs dealing with mutation:

- Make

# Aside: Why Mutation?

We live in a temporal world full of mutation, some programs dealing with mutation:

- Make
- Spreadsheets

# Aside: Why Mutation?

We live in a temporal world full of mutation, some programs dealing with mutation:

- Make
- Spreadsheets
- Databases

# Aside: Why Mutation?

We live in a temporal world full of mutation, some programs dealing with mutation:

- Make
- Spreadsheets
- Databases
- Interpreters

# What is miniAdapton?

- a minimal version of Adapton
- try to be readable
- try to be portable
- try to be small

# What is miniAdapton?

- a minimal version of Adapton
- try to be readable
- try to be portable
- try to be small
- try to be used

# What is miniAdapton?

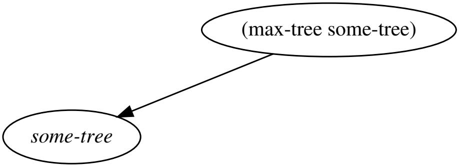
- a minimal version of Adapton
- try to be readable
- try to be portable
- try to be small
- try to be used
- try to be abused

# Visualization of max-tree in Adapton

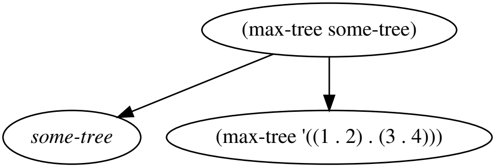
# Visualization of max-tree in Adapton

(max-tree some-tree)

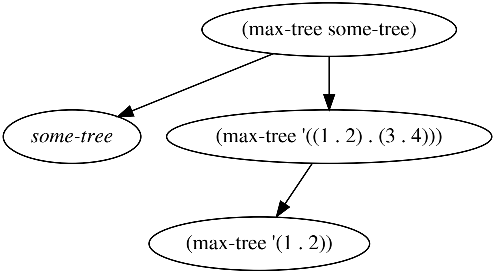
# Visualization of max-tree in Adapton



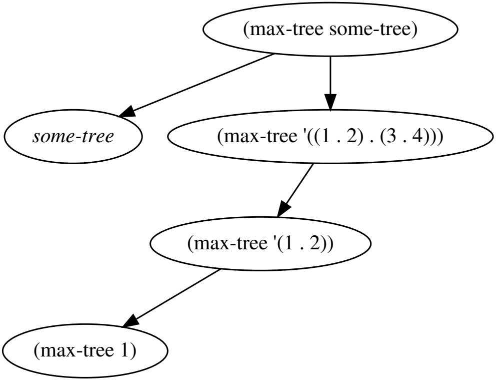
# Visualization of max-tree in Adapton



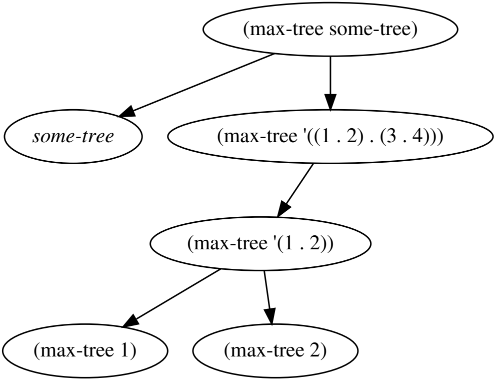
# Visualization of max-tree in Adapton



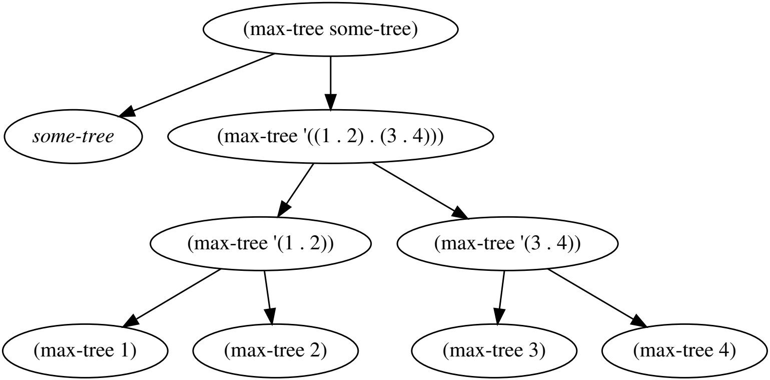
# Visualization of max-tree in Adapton



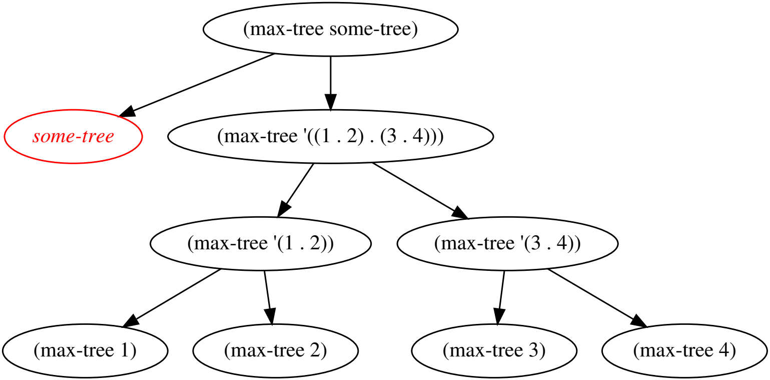
# Visualization of max-tree in Adapton



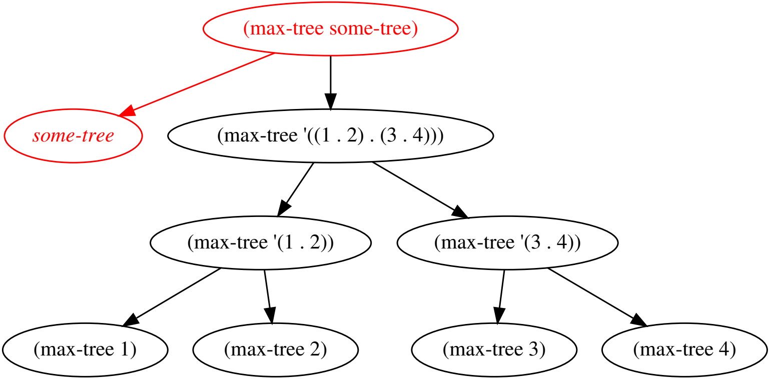
# Visualization of max-tree in Adapton



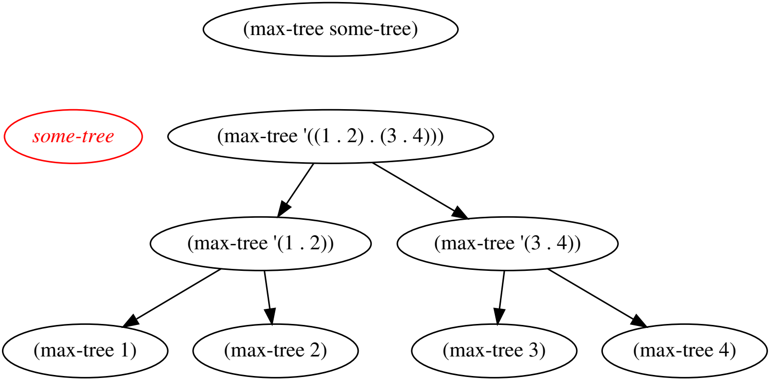
# Visualization of max-tree in Adapton



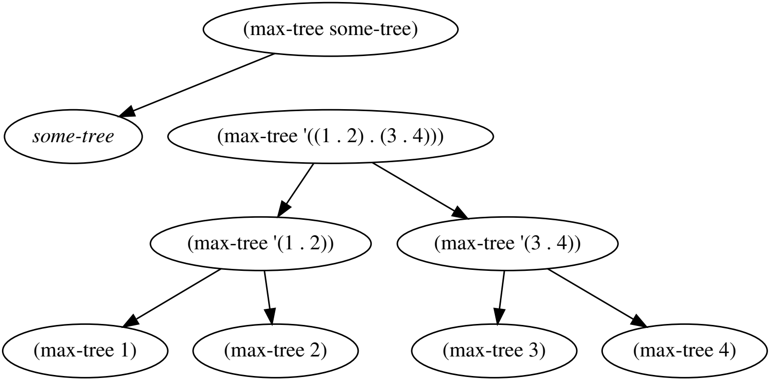
# Visualization of max-tree in Adapton



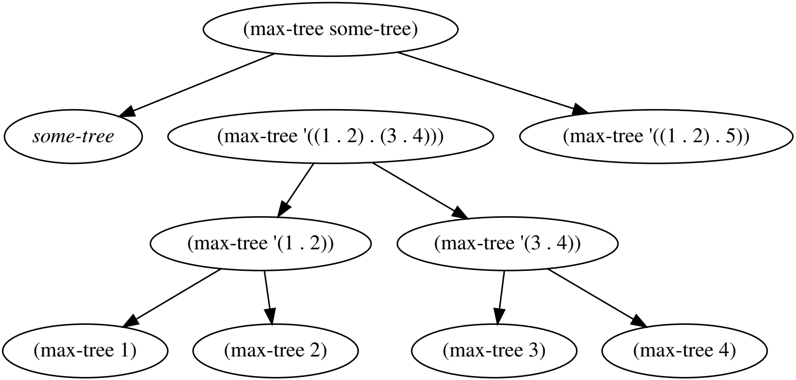
# Visualization of max-tree in Adapton



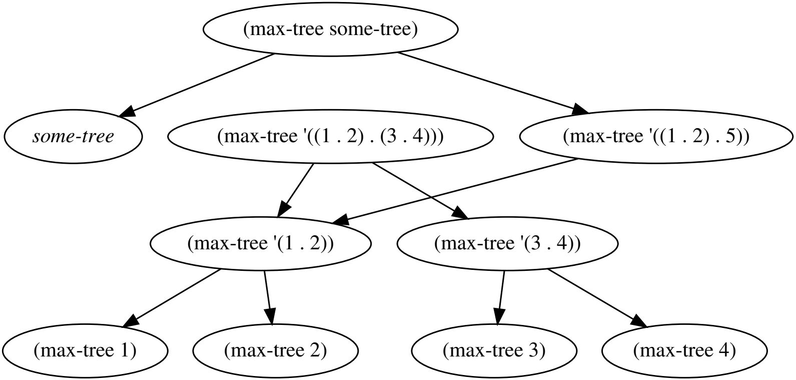
# Visualization of max-tree in Adapton



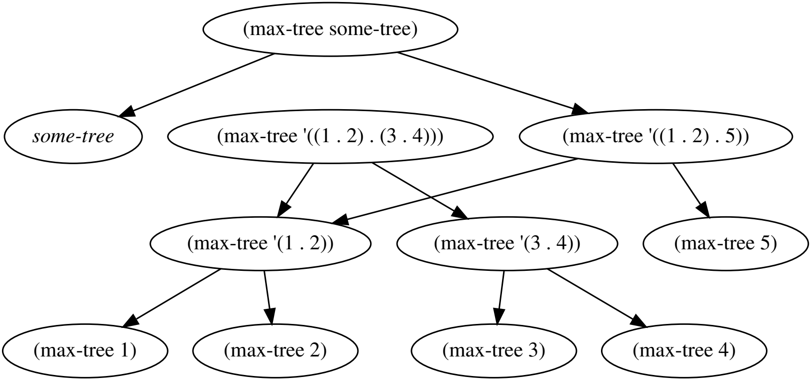
# Visualization of max-tree in Adapton



# Visualization of max-tree in Adapton



# Visualization of max-tree in Adapton



# What's in a node?

```
(define-record-type
  (adapton adapton-cons adapton?)
  (fields
```

# What's in a node?

```
(define-record-type
  (adapton adapton-cons adapton?)
  (fields
   thunk
```

# What's in a node?

```
(define-record-type
  (adapton adapton-cons adapton?)
  (fields
   thunk
   (mutable result)
```

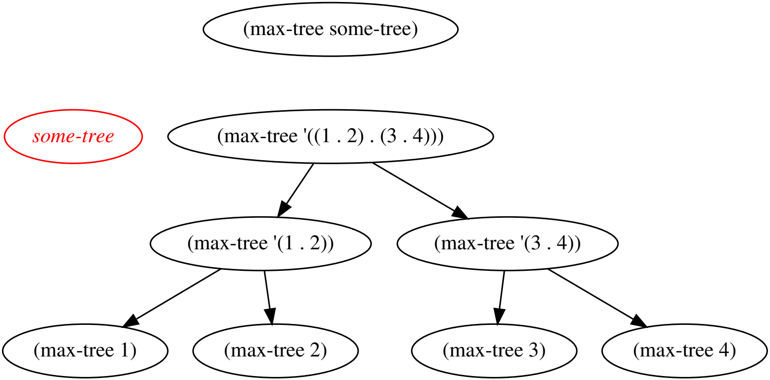
# What's in a node?

```
(define-record-type
  (adapton adapton-cons adapton?)
  (fields
   thunk
   (mutable result)
   (mutable sub)
   (mutable super)
```

# What's in a node?

```
(define-record-type
  (adapton adapton-cons adapton?)
  (fields
   thunk
   (mutable result)
   (mutable sub)
   (mutable super)
   (mutable clean?)))
```

# Nodes



(max-tree some-tree)

*some-tree*

(max-tree '((1 . 2) . (3 . 4)))

(max-tree '(1 . 2))

(max-tree '(3 . 4))

(max-tree 1)

(max-tree 2)

(max-tree 3)

(max-tree 4)

# miniAdapton Interfaces

- Adapton thunks (“athunks”) and Adapton references (“arefs”)
  - adapton-ref
  - adapton-ref-set!
  - adapt
  - adapton-force

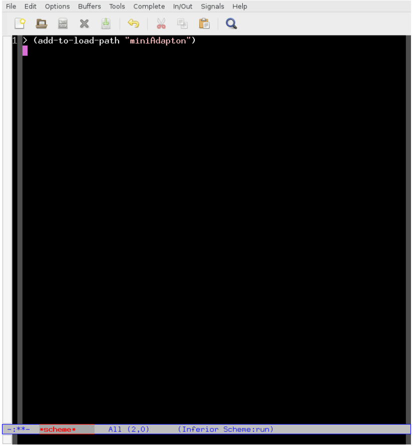
# miniAdapton Interfaces

- Adapton thunks (“athunks”) and Adapton references (“arefs”)
  - adapton-ref
  - adapton-ref-set!
  - adapt
  - adapton-force
- Adapton memoization (“amemo”)
  - adapton-memoize, adapton-memoize-l
  - define-amemo, define-amemo-l

# miniAdapton Interfaces

- Adapton thunks (“athunks”) and Adapton references (“arefs”)
    - adapton-ref
    - adapton-ref-set!
    - adapt
    - adapton-force
- Adapton memoization (“amemo”)
    - adapton-memoize, adapton-memoize-l
    - define-amemo, define-amemo-l
- Adapton variables (“avar”)
    - define-avar
    - avar-get
    - avar-set!

# Interface Demo

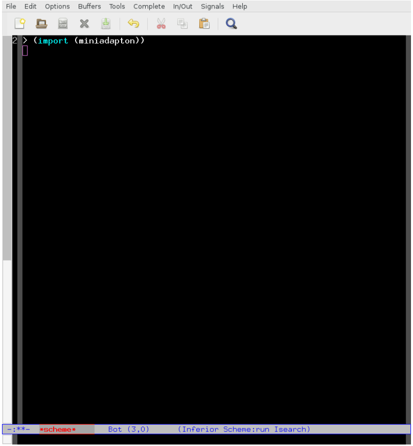


File Edit Options Buffers Tools Complete In/Out Signals Help

```
1 > (add-to-load-path "miniAdapton")
```

-:**- *scheme* All (2,0) (Inferior Scheme:run)

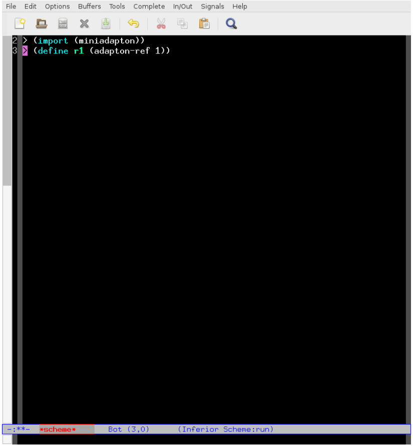
# Interface Demo


File Edit Options Buffers Tools Complete In/Out Signals Help

```
2 > (import (miniadapton))
```

```
-:**-  *scheme*     Bot (3,0)      (Inferior Scheme:run Isearch)
```

# Interface Demo

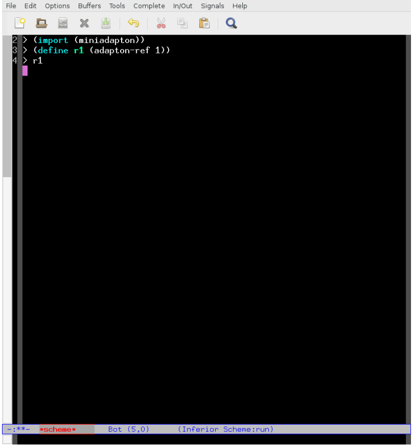


File Edit Options Buffers Tools Complete In/Out Signals Help

```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
```

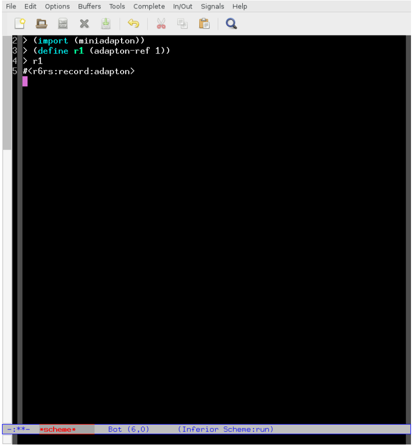
-:**- *scheme* Bot (3,0) (Inferior Scheme:run)

# Interface Demo



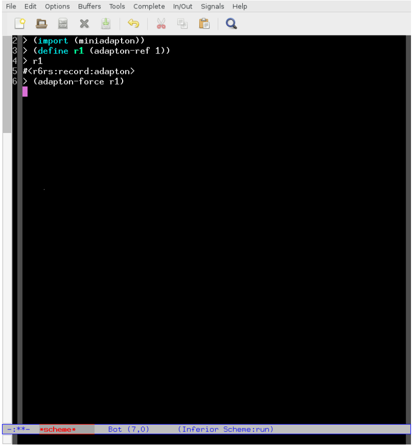
```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
```

# Interface Demo



```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
```

# Interface Demo

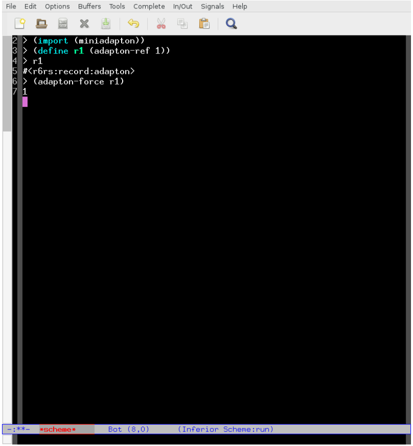


File Edit Options Buffers Tools Complete In/Out Signals Help

```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
> (adapton-force r1)
```

```
-:**-  *scheme*   Bot (7,0)    (Inferior Scheme:run)
```

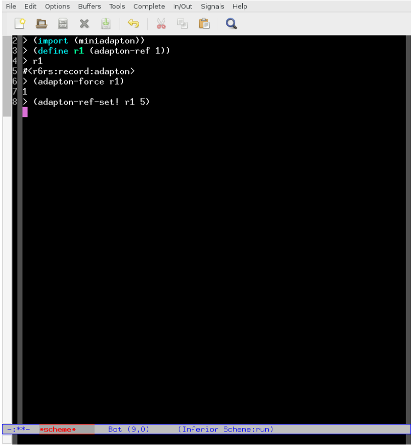
# Interface Demo


File Edit Options Buffers Tools Complete In/Out Signals Help

```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
> (adapton-force r1)
1
```

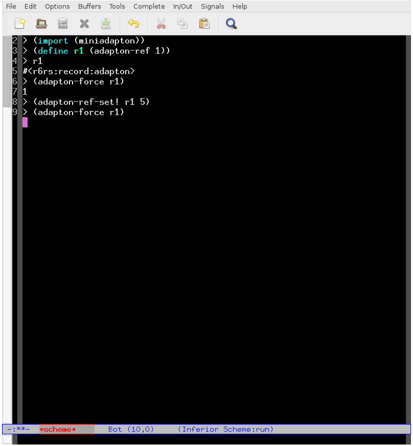
-:**-  *scheme*    Bot (8,0)     (Inferior Scheme:run)

# Interface Demo



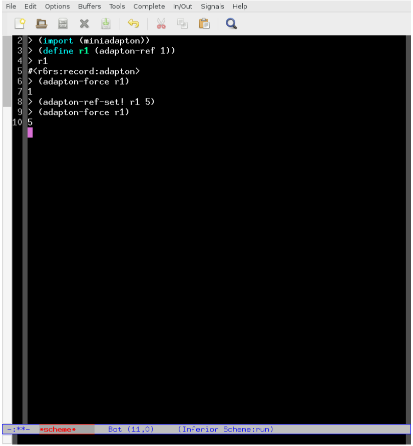
```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
> (adapton-force r1)
1
> (adapton-ref-set! r1 5)
```

# Interface Demo



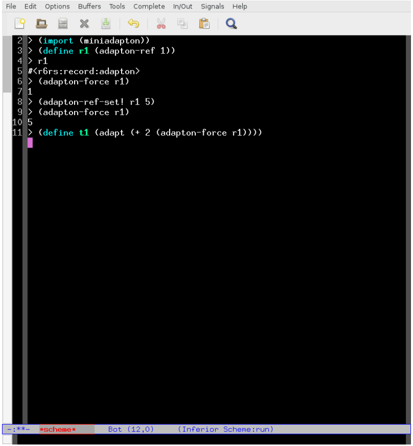
```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
> (adapton-force r1)
1
> (adapton-ref-set! r1 5)
> (adapton-force r1)
```

# Interface Demo


File Edit Options Buffers Tools Complete In/Out Signals Help
-:**- *scheme* Bot (11,0) (Inferior Scheme:run)

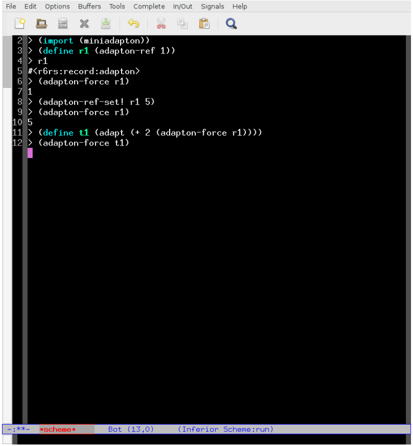
```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
> (adapton-force r1)
1
> (adapton-ref-set! r1 5)
> (adapton-force r1)
5
```

# Interface Demo



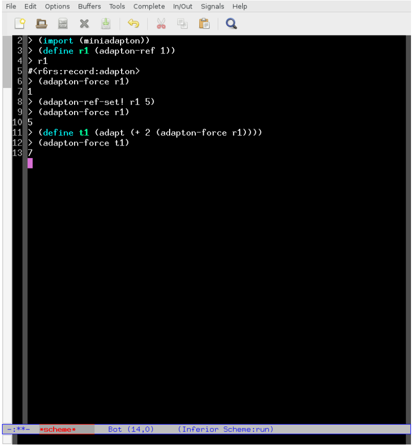
```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
> (adapton-force r1)
1
> (adapton-ref-set! r1 5)
> (adapton-force r1)
5
> (define t1 (adapt (+ 2 (adapton-force r1))))
```

# Interface Demo



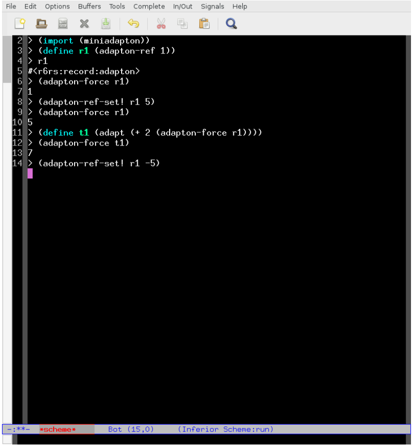
```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
> (adapton-force r1)
1
> (adapton-ref-set! r1 5)
> (adapton-force r1)
5
> (define t1 (adapt (+ 2 (adapton-force r1))))
> (adapton-force t1)
```

# Interface Demo


File Edit Options Buffers Tools Complete In/Out Signals Help
-:**- *scheme* Bot (14,0) (Inferior Scheme:run)

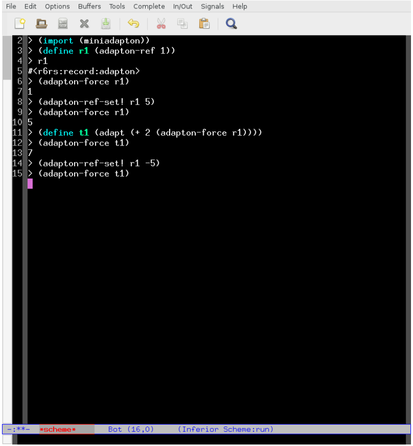
```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
> (adapton-force r1)
1
> (adapton-ref-set! r1 5)
> (adapton-force r1)
5
> (define t1 (adapt (+ 2 (adapton-force r1))))
> (adapton-force t1)
7
```

# Interface Demo


File Edit Options Buffers Tools Complete In/Out Signals Help
-:**- *scheme* Bot (15,0) (Inferior Scheme:run)

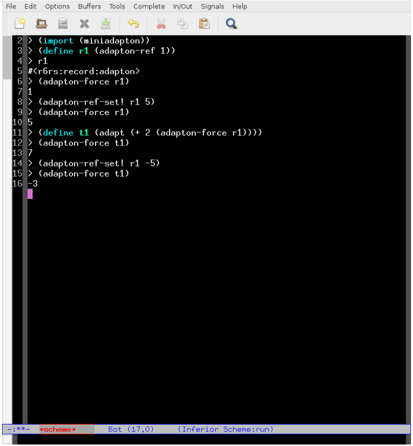
```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
> (adapton-force r1)
1
> (adapton-ref-set! r1 5)
> (adapton-force r1)
5
> (define t1 (adapt (+ 2 (adapton-force r1))))
> (adapton-force t1)
7
> (adapton-ref-set! r1 -5)
```

# Interface Demo



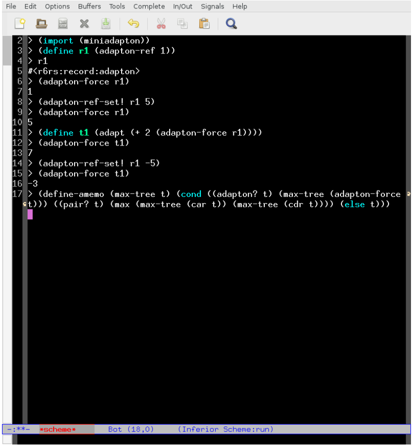
```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
> (adapton-force r1)
1
> (adapton-ref-set! r1 5)
> (adapton-force r1)
5
> (define t1 (adapt (+ 2 (adapton-force r1))))
> (adapton-force t1)
7
> (adapton-ref-set! r1 -5)
> (adapton-force t1)
```

# Interface Demo



```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
> (adapton-force r1)
1
> (adapton-ref-set! r1 5)
> (adapton-force r1)
5
> (define t1 (adapt (+ 2 (adapton-force r1))))
> (adapton-force t1)
7
> (adapton-ref-set! r1 -5)
> (adapton-force t1)
-3
```

# Interface Demo

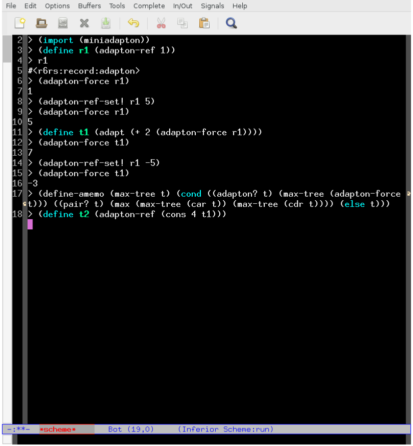


File Edit Options Buffers Tools Complete In/Out Signals Help

```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
> (adapton-force r1)
1
> (adapton-ref-set! r1 5)
> (adapton-force r1)
5
> (define t1 (adapt (+ 2 (adapton-force r1))))
> (adapton-force t1)
7
> (adapton-ref-set! r1 -5)
> (adapton-force t1)
-3
> (define-amemo (max-tree t) (cond ((adapton? t) (max-tree (adapton-force t))) ((pair? t) (max (max-tree (car t)) (max-tree (cdr t)))) (else t)))
```

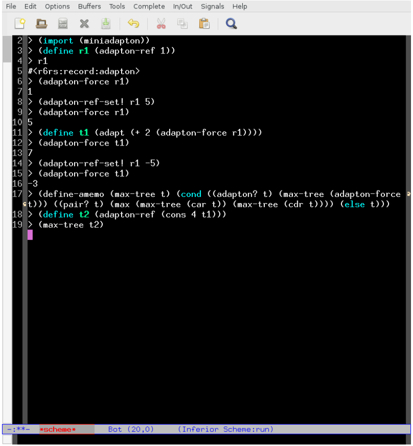
-:**-  *scheme*  Bot (18,0)  (Inferior Scheme:run)

# Interface Demo



```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
> (adapton-force r1)
1
> (adapton-ref-set! r1 5)
> (adapton-force r1)
5
> (define t1 (adapt (+ 2 (adapton-force r1))))
> (adapton-force t1)
7
> (adapton-ref-set! r1 -5)
> (adapton-force t1)
-3
> (define-amemo (max-tree t) (cond ((adapton? t) (max-tree (adapton-force
t))) ((pair? t) (max (max-tree (car t)) (max-tree (cdr t)))) (else t)))
> (define t2 (adapton-ref (cons 4 t1)))
```

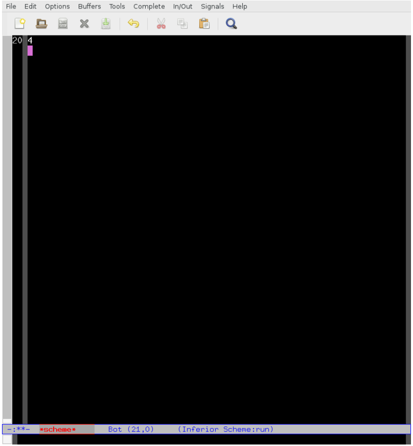
# Interface Demo



```
> (import (miniadapton))
> (define r1 (adapton-ref 1))
> r1
#<r6rs:record:adapton>
> (adapton-force r1)
1
> (adapton-ref-set! r1 5)
> (adapton-force r1)
5
> (define t1 (adapt (+ 2 (adapton-force r1))))
> (adapton-force t1)
7
> (adapton-ref-set! r1 -5)
> (adapton-force t1)
-3
> (define-amemo (max-tree t) (cond ((adapton? t) (max-tree (adapton-force
t))) ((pair? t) (max (max-tree (car t)) (max-tree (cdr t)))) (else t)))
> (define t2 (adapton-ref (cons 4 t1)))
> (max-tree t2)
```

```
-:**-  *scheme*    Bot (20,0)    (Inferior Scheme:run)
```

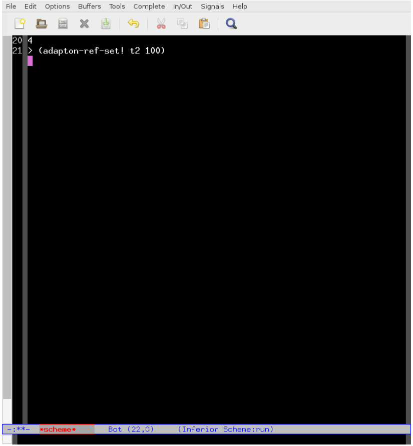
# Interface Demo



```
20 4
```

```
-:**-  *scheme*     Bot (21,0)     (Inferior Scheme:run)
```

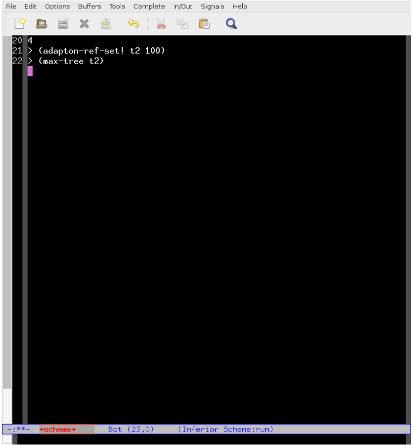
# Interface Demo


File Edit Options Buffers Tools Complete In/Out Signals Help

```
20 4
21 > (adapton-ref-set! t2 100)
```

```
-:**-  *scheme*   Bot (22,0)    (Inferior Scheme:run)
```

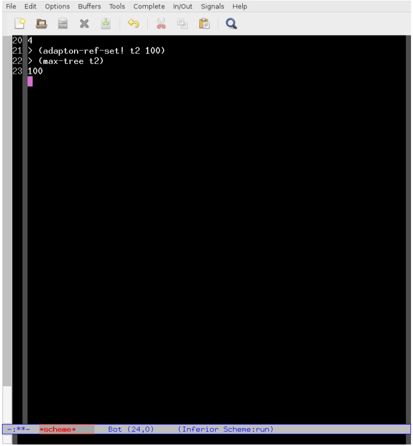
# Interface Demo



```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
```

```
-:**-  *scheme*    Bot (23,0)     (Inferior Scheme:run)
```

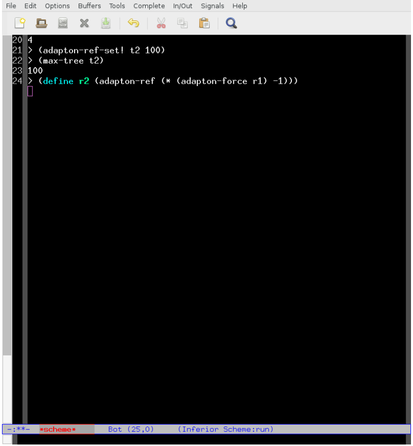
# Interface Demo


File Edit Options Buffers Tools Complete In/Out Signals Help

```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
```

-:**-  *scheme*   Bot (24,0)    (Inferior Scheme:run)

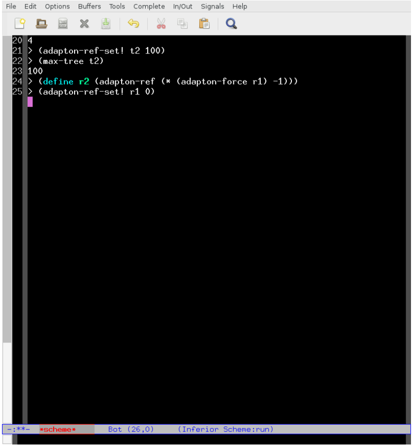
# Interface Demo


File Edit Options Buffers Tools Complete In/Out Signals Help

```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
```

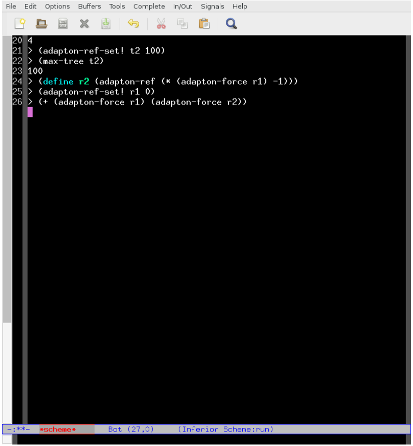
-:**-  *scheme*   Bot (25,0)   (Inferior Scheme:run)

# Interface Demo



```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
```

# Interface Demo

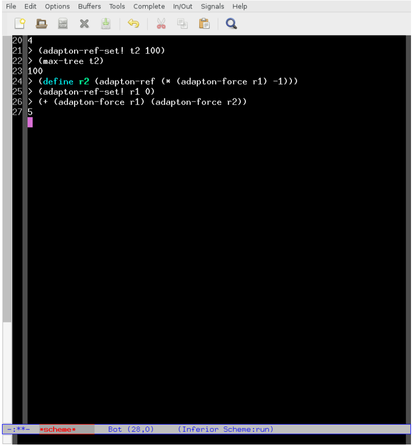


File Edit Options Buffers Tools Complete In/Out Signals Help

```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
> (+ (adapton-force r1) (adapton-force r2))
```

```
-:**-  *scheme*     Bot (27,0)     (Inferior Scheme:run)
```

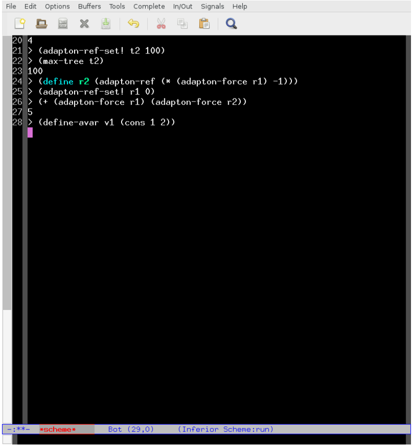
# Interface Demo


File Edit Options Buffers Tools Complete In/Out Signals Help

```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
> (+ (adapton-force r1) (adapton-force r2))
5
```

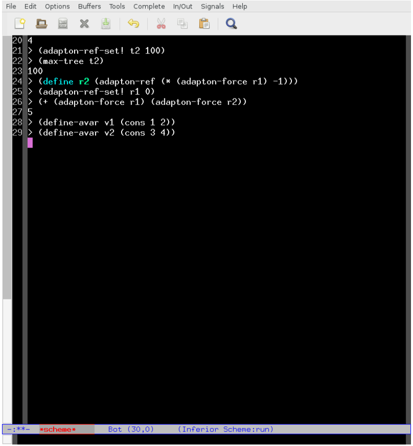
-:**-  *scheme*   Bot (28,0)   (Inferior Scheme:run)

# Interface Demo



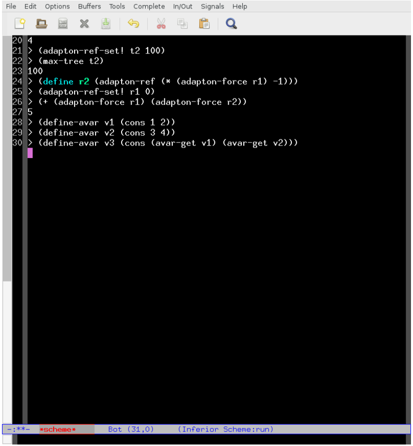
```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
> (+ (adapton-force r1) (adapton-force r2))
5
> (define-avar v1 (cons 1 2))
```

# Interface Demo


File Edit Options Buffers Tools Complete In/Out Signals Help
-:**- *scheme* Bot (30,0) (Inferior Scheme:run)

```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
> (+ (adapton-force r1) (adapton-force r2))
5
> (define-avar v1 (cons 1 2))
> (define-avar v2 (cons 3 4))
```

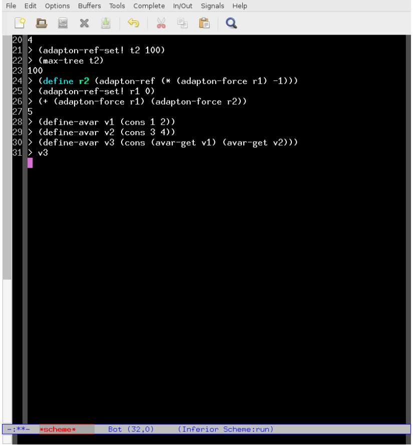
# Interface Demo



```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
> (+ (adapton-force r1) (adapton-force r2))
5
> (define-avar v1 (cons 1 2))
> (define-avar v2 (cons 3 4))
> (define-avar v3 (cons (avar-get v1) (avar-get v2)))
```

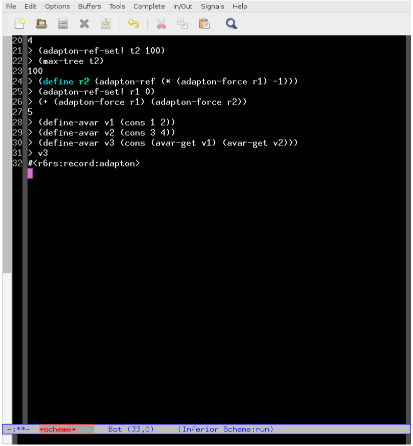
```
-:**-  *scheme*      Bot (31,0)     (Inferior Scheme:run)
```

# Interface Demo


File Edit Options Buffers Tools Complete In/Out Signals Help
-:**- *scheme* Bot (32,0) (Inferior Scheme:run)

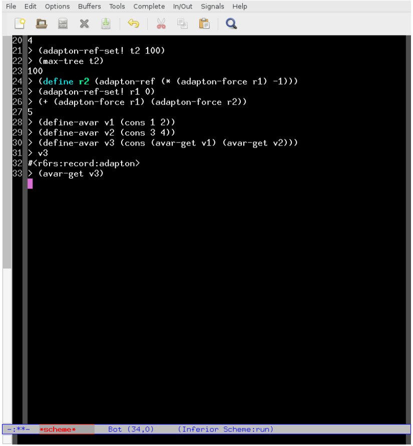
```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
> (+ (adapton-force r1) (adapton-force r2))
5
> (define-avar v1 (cons 1 2))
> (define-avar v2 (cons 3 4))
> (define-avar v3 (cons (avar-get v1) (avar-get v2)))
> v3
```

# Interface Demo


File Edit Options Buffers Tools Complete In/Out Signals Help
-:**-  *scheme*  Bot (33,0)  (Inferior Scheme:run)

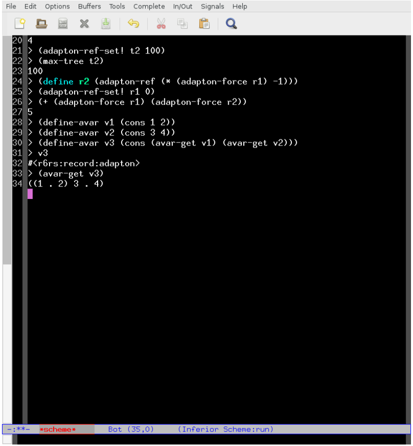
```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
> (+ (adapton-force r1) (adapton-force r2))
5
> (define-avar v1 (cons 1 2))
> (define-avar v2 (cons 3 4))
> (define-avar v3 (cons (avar-get v1) (avar-get v2)))
> v3
#<r6rs:record:adapton>
```

# Interface Demo



```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
> (+ (adapton-force r1) (adapton-force r2))
5
> (define-avar v1 (cons 1 2))
> (define-avar v2 (cons 3 4))
> (define-avar v3 (cons (avar-get v1) (avar-get v2)))
> v3
#<r6rs:record:adapton>
> (avar-get v3)
```

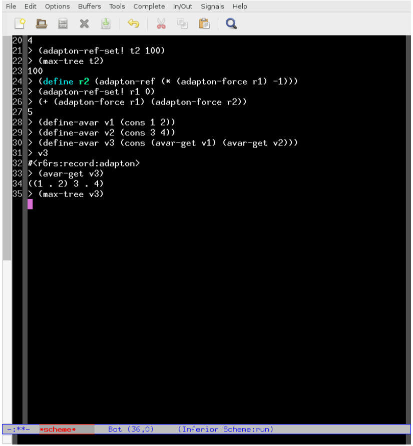
# Interface Demo


File Edit Options Buffers Tools Complete In/Out Signals Help

```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
> (+ (adapton-force r1) (adapton-force r2))
5
> (define-avar v1 (cons 1 2))
> (define-avar v2 (cons 3 4))
> (define-avar v3 (cons (avar-get v1) (avar-get v2)))
> v3
#<r6rs:record:adapton>
> (avar-get v3)
((1 . 2) 3 . 4)
```

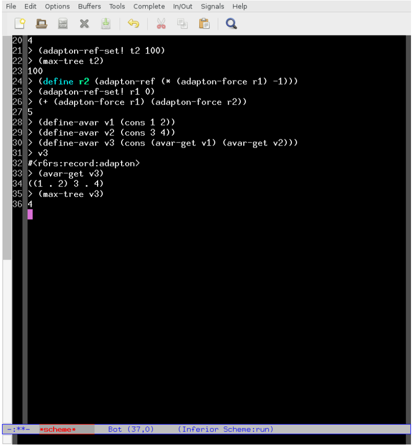
-:**-  *scheme*   Bot (35,0)   (Inferior Scheme:run)

# Interface Demo



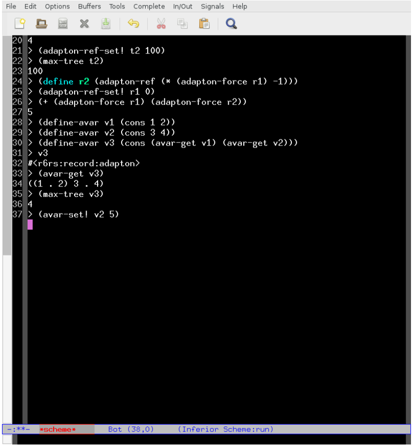
```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
> (+ (adapton-force r1) (adapton-force r2))
5
> (define-avar v1 (cons 1 2))
> (define-avar v2 (cons 3 4))
> (define-avar v3 (cons (avar-get v1) (avar-get v2)))
> v3
#<r6rs:record:adapton>
> (avar-get v3)
((1 . 2) 3 . 4)
> (max-tree v3)
```

# Interface Demo



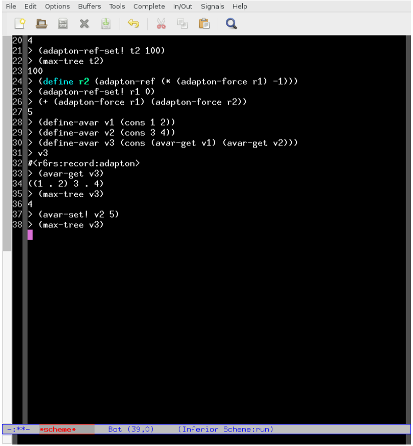
```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
> (+ (adapton-force r1) (adapton-force r2))
5
> (define-avar v1 (cons 1 2))
> (define-avar v2 (cons 3 4))
> (define-avar v3 (cons (avar-get v1) (avar-get v2)))
> v3
#<r6rs:record:adapton>
> (avar-get v3)
((1 . 2) 3 . 4)
> (max-tree v3)
4
```

# Interface Demo



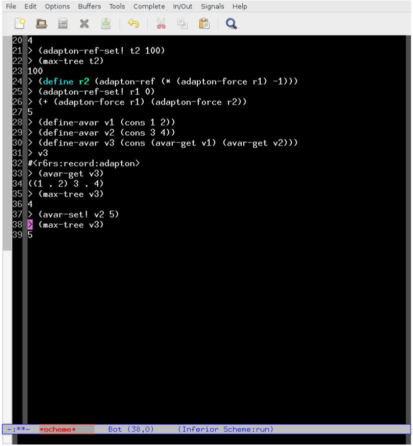
```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
> (+ (adapton-force r1) (adapton-force r2))
5
> (define-avar v1 (cons 1 2))
> (define-avar v2 (cons 3 4))
> (define-avar v3 (cons (avar-get v1) (avar-get v2)))
> v3
#<r6rs:record:adapton>
> (avar-get v3)
((1 . 2) 3 . 4)
> (max-tree v3)
4
> (avar-set! v2 5)
```

# Interface Demo



```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
> (+ (adapton-force r1) (adapton-force r2))
5
> (define-avar v1 (cons 1 2))
> (define-avar v2 (cons 3 4))
> (define-avar v3 (cons (avar-get v1) (avar-get v2)))
> v3
#<r6rs:record:adapton>
> (avar-get v3)
((1 . 2) 3 . 4)
> (max-tree v3)
4
> (avar-set! v2 5)
> (max-tree v3)
```

# Interface Demo



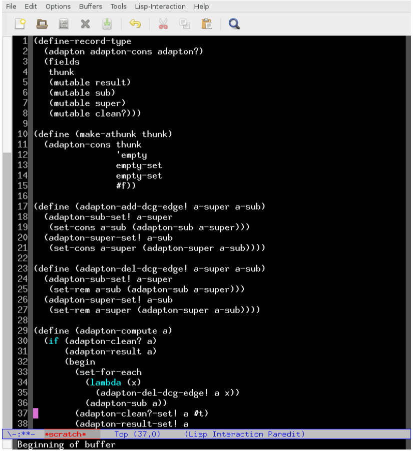
```
4
> (adapton-ref-set! t2 100)
> (max-tree t2)
100
> (define r2 (adapton-ref (* (adapton-force r1) -1)))
> (adapton-ref-set! r1 0)
> (+ (adapton-force r1) (adapton-force r2))
5
> (define-avar v1 (cons 1 2))
> (define-avar v2 (cons 3 4))
> (define-avar v3 (cons (avar-get v1) (avar-get v2)))
> v3
#<r6rs:record:adapton>
> (avar-get v3)
((1 . 2) 3 . 4)
> (max-tree v3)
4
> (avar-set! v2 5)
> (max-tree v3)
5
```

```
-:**-  *scheme*      Bot (38,0)     (Inferior Scheme:run)
```

# microAdapton: the core of miniAdapton

- inspired by microKanren for implementing miniKanren
- implements core operations for miniAdapton
- avoids implicit DCG construction
- miniAdapton builds implicit DCG construction on top of microAdapton

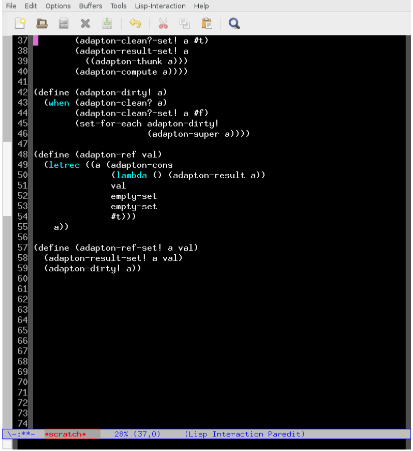
# Implementation - microAdapton



```
(define-record-type
  (adapton adapton-cons adapton?)
  (fields
   thunk
   (mutable result)
   (mutable sub)
   (mutable super)
   (mutable clean?)))

(define (make-athunk thunk)
  (adapton-cons thunk
                'empty
                empty-set
                empty-set
                #f))

(define (adapton-add-dcg-edge! a-super a-sub)
  (adapton-sub-set! a-super
   (set-cons a-sub (adapton-sub a-super)))
  (adapton-super-set! a-sub
   (set-cons a-super (adapton-super a-sub))))

(define (adapton-del-dcg-edge! a-super a-sub)
  (adapton-sub-set! a-super
   (set-rem a-sub (adapton-sub a-super)))
  (adapton-super-set! a-sub
   (set-rem a-super (adapton-super a-sub))))

(define (adapton-compute a)
  (if (adapton-clean? a)
      (adapton-result a)
      (begin
        (set-for-each
          (lambda (x)
            (adapton-del-dcg-edge! a x))
          (adapton-sub a))
        (adapton-clean?-set! a #t)
        (adapton-result-set! a
```

```
\-:**-  *scratch*    Top (37,0)     (Lisp Interaction Paredit)
Beginning of buffer
```

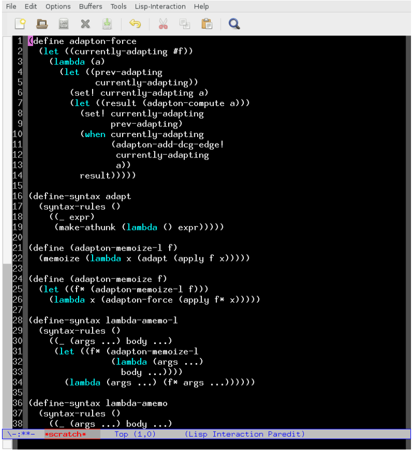
# Implementation - microAdapton


File
Edit
Options
Buffers
Tools
Lisp-Interaction
Help

```
        (adapton-clean?-set! a #t)
        (adapton-result-set! a
          ((adapton-thunk a)))
        (adapton-compute a))))

(define (adapton-dirty! a)
  (when (adapton-clean? a)
        (adapton-clean?-set! a #f)
        (set-for-each adapton-dirty!
                      (adapton-super a))))

(define (adapton-ref val)
  (letrec ((a (adapton-cons
               (lambda () (adapton-result a))
               val
               empty-set
               empty-set
               #t)))
    a))

(define (adapton-ref-set! a val)
  (adapton-result-set! a val)
  (adapton-dirty! a))
```

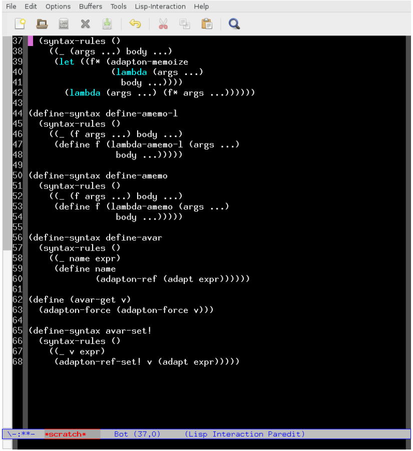
-:**-  *scratch*     28% (37,0)     (Lisp Interaction Paredit)

# Implementation - miniAdapton


File
Edit
Options
Buffers
Tools
Lisp-Interaction
Help
-:**-  *scratch*    Top (1,0)     (Lisp Interaction Paredit)

```scheme
(define adapton-force
  (let ((currently-adapting #f))
    (lambda (a)
      (let ((prev-adapting
             currently-adapting))
        (set! currently-adapting a)
        (let ((result (adapton-compute a)))
          (set! currently-adapting
                prev-adapting)
          (when currently-adapting
                (adapton-add-dcg-edge!
                 currently-adapting
                 a))
          result)))))

(define-syntax adapt
  (syntax-rules ()
    ((_ expr)
     (make-athunk (lambda () expr)))))

(define (adapton-memoize-l f)
  (memoize (lambda x (adapt (apply f x)))))

(define (adapton-memoize f)
  (let ((f* (adapton-memoize-l f)))
    (lambda x (adapton-force (apply f* x)))))

(define-syntax lambda-amemo-l
  (syntax-rules ()
    ((_ (args ...) body ...)
     (let ((f* (adapton-memoize-l
                (lambda (args ...)
                  body ...))))
       (lambda (args ...) (f* args ...))))))

(define-syntax lambda-amemo
  (syntax-rules ()
    ((_ (args ...) body ...)
```

# Implementation - miniAdapton



```
  (syntax-rules ()
    ((_ (args ...) body ...)
     (let ((f* (adapton-memoize
                (lambda (args ...)
                  body ...))))
       (lambda (args ...) (f* args ...))))))

(define-syntax define-amemo-l
  (syntax-rules ()
    ((_ (f args ...) body ...)
     (define f (lambda-amemo-l (args ...)
                 body ...)))))

(define-syntax define-amemo
  (syntax-rules ()
    ((_ (f args ...) body ...)
     (define f (lambda-amemo (args ...)
                 body ...)))))

(define-syntax define-avar
  (syntax-rules ()
    ((_ name expr)
     (define name
             (adapton-ref (adapt expr))))))

(define (avar-get v)
  (adapton-force (adapton-force v)))

(define-syntax avar-set!
  (syntax-rules ()
    ((_ v expr)
     (adapton-ref-set! v (adapt expr)))))
```

```
\-:**-  *scratch*   Bot (37,0)    (Lisp Interaction Paredit)
```

# Conclusion

- Adapton implemented in a more minimal form
- A minimal implementation encourages hackability

# Conclusion - Play with it

- Incremental computation that you can play with RIGHT NOW

# Conclusion - Play with it

- Incremental computation that you can play with RIGHT NOW
- We want you to use this as soon as possible

# Conclusion - Play with it

- Incremental computation that you can play with RIGHT NOW
- We want you to use this as soon as possible
- Play with this

# Conclusion - Play with it

- Incremental computation that you can play with RIGHT NOW
- We want you to use this as soon as possible
- Play with this
- This toy we made is neat and everyone should play with it:

# Conclusion - Play with it

- Incremental computation that you can play with RIGHT NOW
- We want you to use this as soon as possible
- Play with this
- This toy we made is neat and everyone should play with it:
- `git clone 'https://github.com/fisherdj/miniAdapton'`

# Conclusion - Play with it

- Incremental computation that you can play with RIGHT NOW
- We want you to use this as soon as possible
- Play with this
- This toy we made is neat and everyone should play with it:
- `git clone 'https://github.com/fisherdj/miniAdapton'`
- ...

# Conclusion - Play with it

- Incremental computation that you can play with RIGHT NOW
- We want you to use this as soon as possible
- Play with this
- This toy we made is neat and everyone should play with it:
- `git clone 'https://github.com/fisherdj/miniAdapton'`
- ...
- `git clone 'https://github.com/fisherdj/miniAdapton'`

# Conclusion - Play with it

- Incremental computation that you can play with RIGHT NOW
- We want you to use this as soon as possible
- Play with this
- This toy we made is neat and everyone should play with it:
- `git clone 'https://github.com/fisherdj/miniAdapton'`
- ...
- `git clone 'https://github.com/fisherdj/miniAdapton'`
- More details about the code are in the paper

# Conclusion - Play with it

- Incremental computation that you can play with RIGHT NOW
- We want you to use this as soon as possible
- Play with this
- This toy we made is neat and everyone should play with it:
- `git clone 'https://github.com/fisherdj/miniAdapton'`
- ...
- `git clone 'https://github.com/fisherdj/miniAdapton'`
- More details about the code are in the paper

# Challenges

Modifying miniAdapton:

- Avoid recomputation when answers to subcomputations don't change (full Adapton)
- Add debugging information and/or visualization
- miniAdapton in other languages

Using miniAdapton:

- Adapton data structures
- Adapton for interactive applications

# Acknowledgements and Related Work

- Thanks to Jason Hemann and Dan Friedman for microKanren, a huge inspiration and motivation for miniAdapton
- Incremental Computing via Function Caching, Pugh and Teitelbaum POPL 1986 (still a good inspiration for data structures using Adapton)
- The Adapton Project, Hammer et al OOPSLA 2015 and PLDI 2014 (http://adapton.org)
- Self-Adjusting Computation, Acar et al; (http://www.umut-acar.org/self-adjusting-computation)