

# 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)))
```

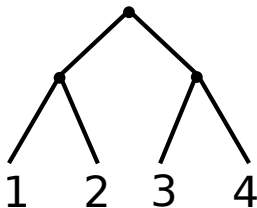
# A Memoized Function

```
>(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

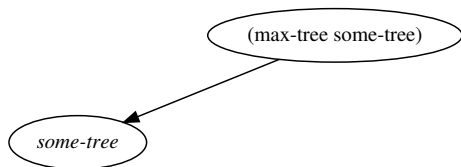


# Visualization of max-tree in Adapton

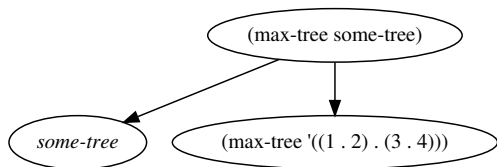
# Visualization of max-tree in Adapton

(max-tree some-tree)

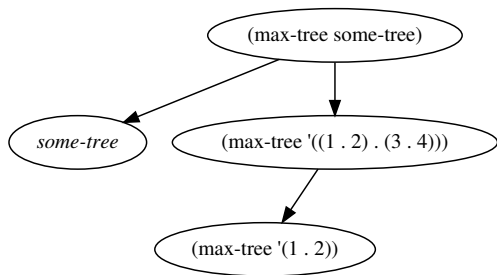
# Visualization of max-tree in Adapton



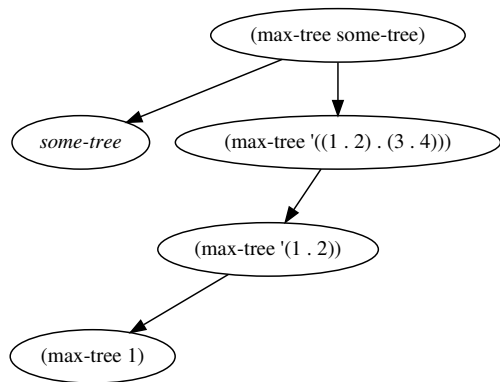
# Visualization of max-tree in Adapton



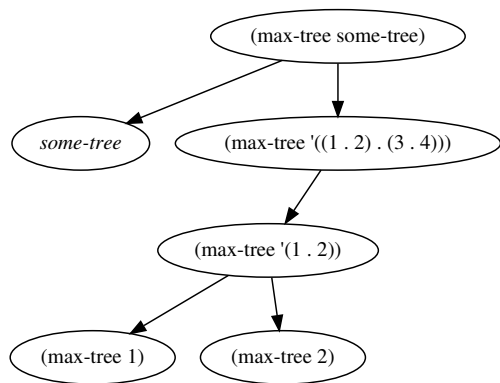
# Visualization of max-tree in Adapton



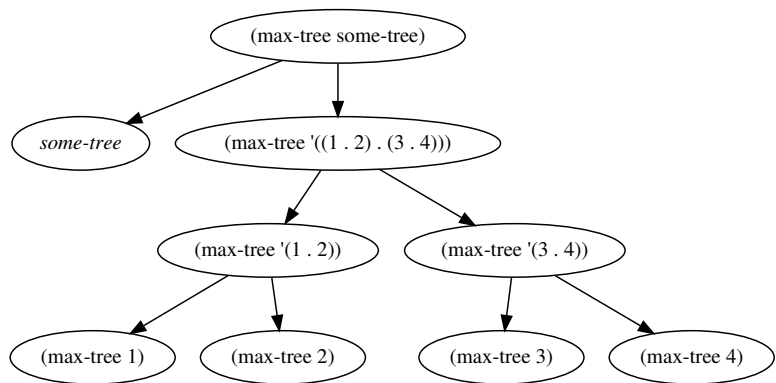
# Visualization of max-tree in Adaption



# Visualization of max-tree in Adapton

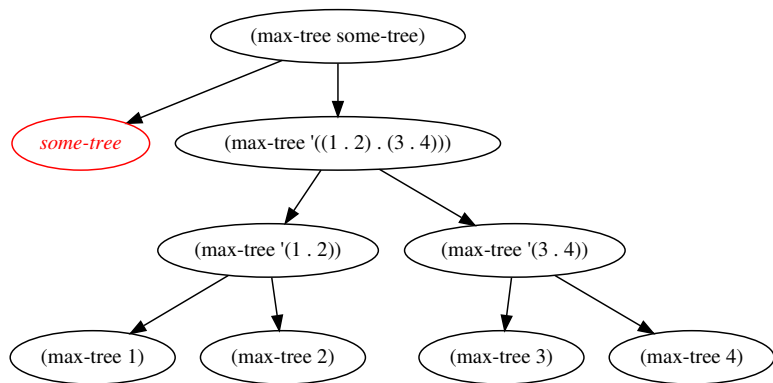


# Visualization of max-tree in Adaption

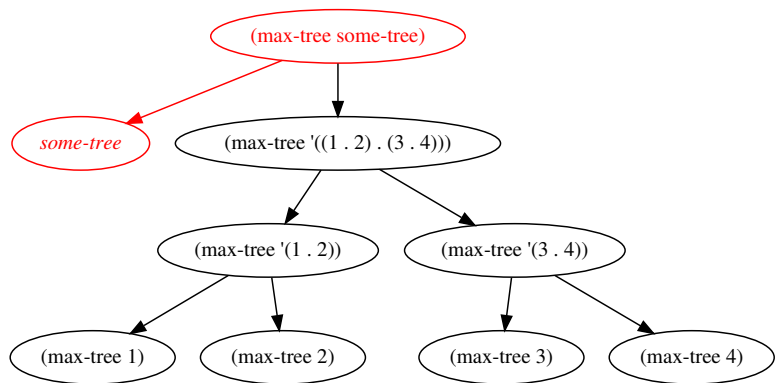




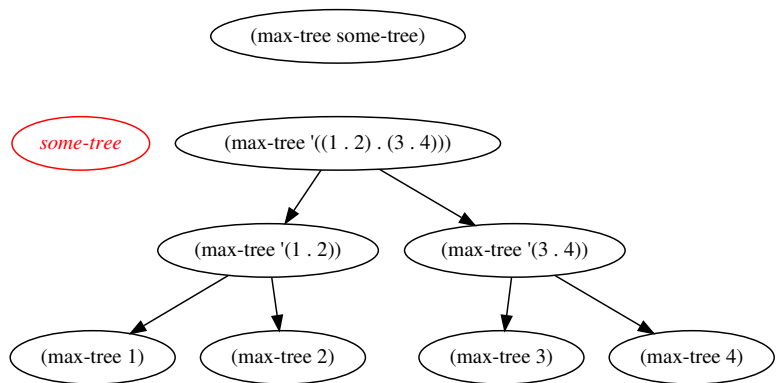
# Visualization of max-tree in Adaption



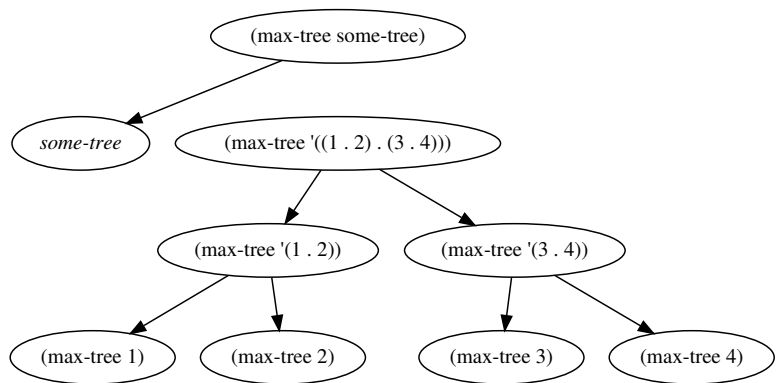
# Visualization of max-tree in Adapton



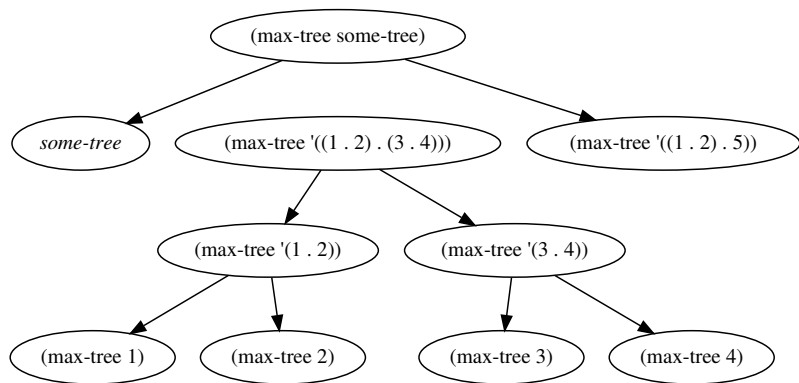
# Visualization of max-tree in Adaption



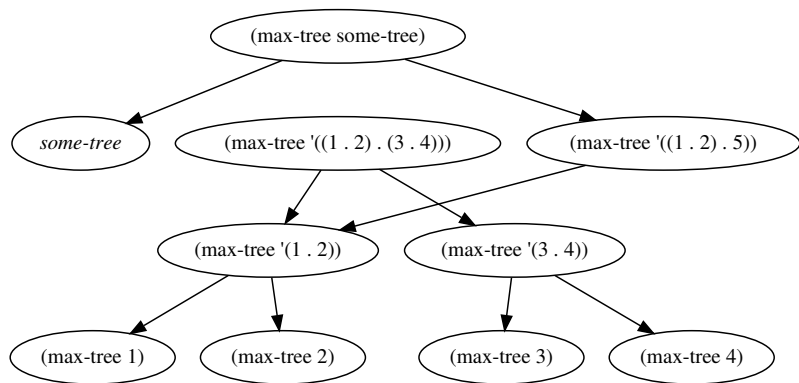
# Visualization of max-tree in Adapton



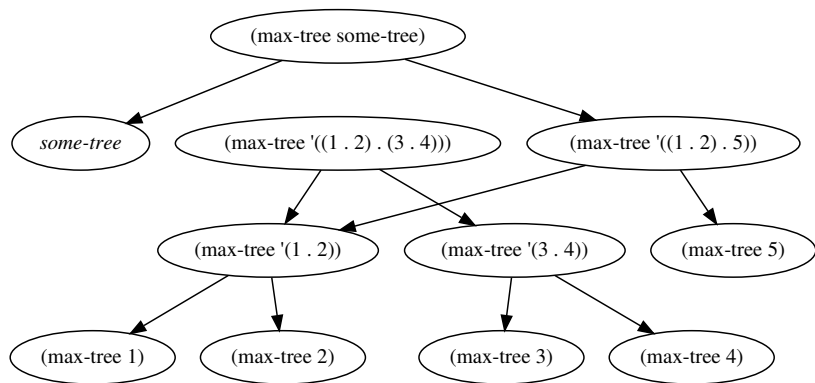
# Visualization of max-tree in Adaption



# 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))
```

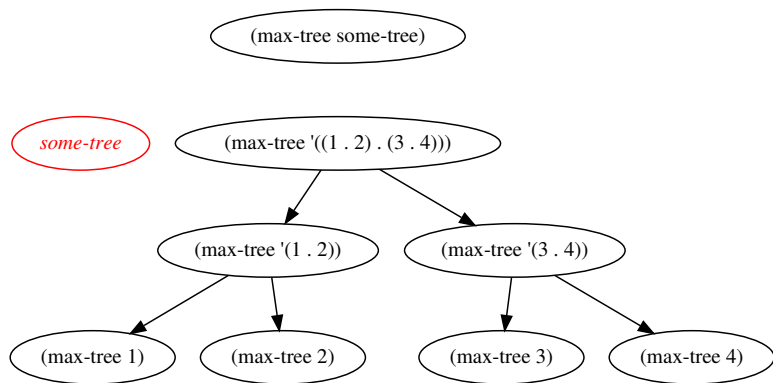
## What's in a node?

```
(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



# miniAdapton Interfaces

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

# miniAdapton Interfaces

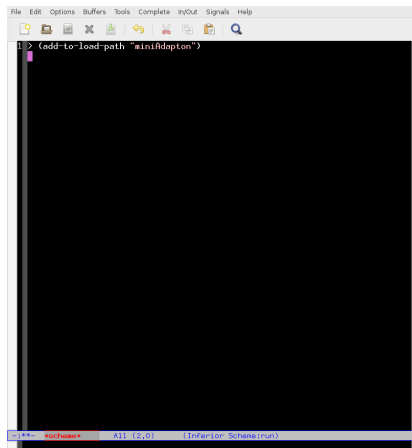
- ▶ Adapton thunks (“athunks”) and Adapton references (“arefs”)
  - ▶ adapton-ref
  - ▶ adapton-ref-set!
  - ▶ adapt
  - ▶ adapton-force
- ▶ Adapton memoization (“amemo”)
  - ▶ adapton-memoize, adapton-memoize-l
  - ▶ define-amemo, define-amemo-l

# 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!



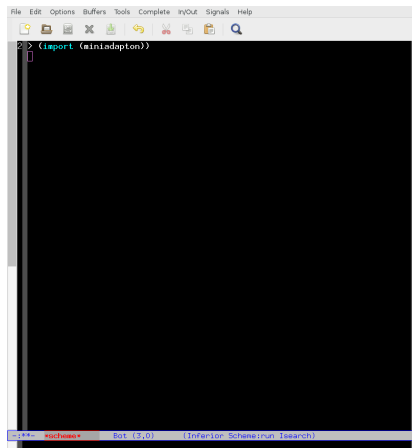
# Interface Demo



A screenshot of a terminal window with a light gray title bar and menu bar. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Complete', 'In/Out', 'Signals', and 'Help'. Below the menu bar is a toolbar with icons for file operations and search. The main area is a black terminal with a white prompt 'I >' and a command '(add-to-load-path "minifadpton")'. A pink cursor is positioned at the end of the command. The bottom status bar shows '10:44', 'minifadpton', 'A11 (2,0)', and '[Inferior Scheme (run)]'.

```
I > (add-to-load-path "minifadpton")
```

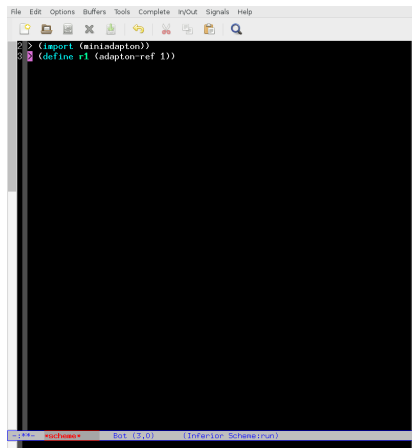
# Interface Demo



The image shows a screenshot of a REPL window titled "Inferior Scheme (run: Teasarch)". The window has a menu bar with "File", "Edit", "Options", "Buffers", "Tools", "Complete", "In/Out", "Signals", and "Help". Below the menu bar is a toolbar with various icons. The main area is a black terminal with a white prompt character ">" on the first line. The text "(import (miniadapt))" has been entered on the second line, and a white cursor is positioned at the end of the line. The status bar at the bottom of the window displays "Bot (2,0) [Inferior Scheme (run: Teasarch)]".

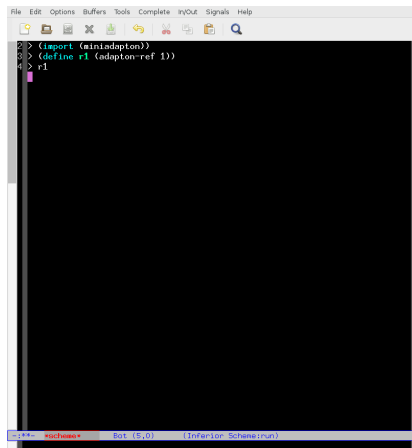
```
> (import (miniadapt))
```

# Interface Demo



The image shows a screenshot of a code editor window. The window has a menu bar at the top with the following items: File, Edit, Options, Buffers, Tools, Complete, In/Out, Signals, and Help. Below the menu bar is a toolbar with various icons for file operations and editing. The main area of the window is a dark background with two lines of code in a light color. The first line is: `2 > (import (miniadapton))`. The second line is: `3 > (define r1 (adapton-ref 1))`. At the bottom of the window, there is a status bar with the text: `Bot (2,0) [Inferior Scheme (run)]`.

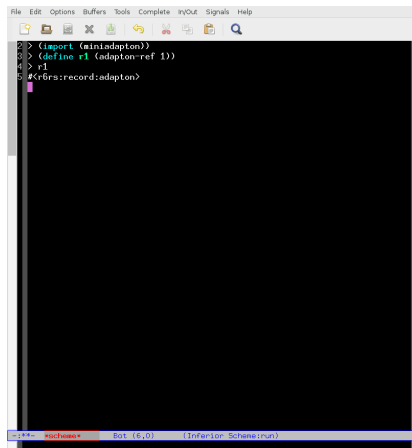
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
```

Bot [E,0] [Inferior Scheme (run)]

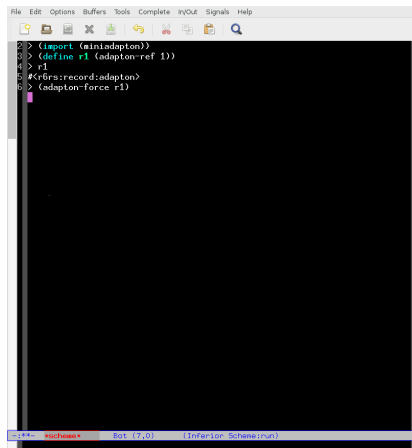
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
2 > (import (miniadapt))
3 > (define r1 (adapt-ref 1))
4 > r1
5 #<r6rs:record:adapt>
```

Bot (E, 0) [Inferior Scheme (run)]

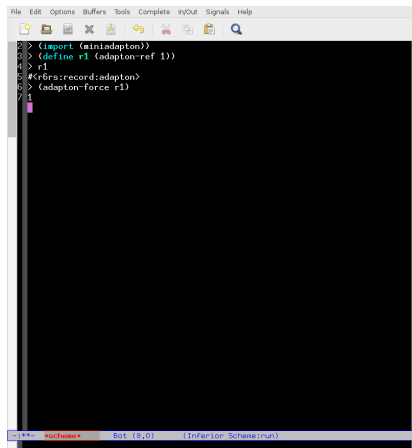
# Interface Demo



A screenshot of a Scheme REPL window. The window has a menu bar with 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Complete', 'In/OUT', 'Signals', and 'Help'. Below the menu bar is a toolbar with icons for file operations and search. The main area is a black terminal with white text. The text shows a sequence of commands: line 2: > (import (miniadapton)), line 3: > (define r1 (adapton-ref 1)), line 4: > r1, line 5: #<r0rs:record:adapton>, and line 6: > (adapton-force r1). A pink cursor is visible at the end of line 6. At the bottom of the window, a status bar shows '12:44', '402/4000', 'Bot: (7,0)', and '(Inferior Scheme (run))'.

```
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
5 #<r0rs:record:adapton>
6 > (adapton-force r1)
```

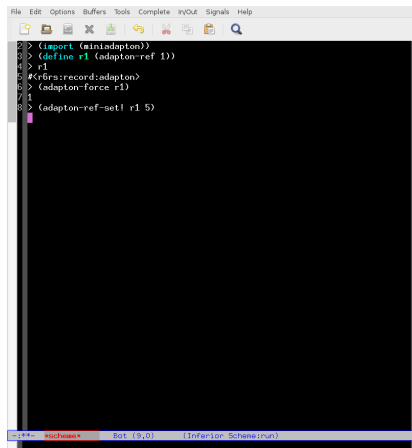
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
5 #<r0rs:record:adapton>
6 > (adapton-force r1)
7 1
```

\*\*\* Bot (8,0) [Inferior Scheme (run)]

# Interface Demo

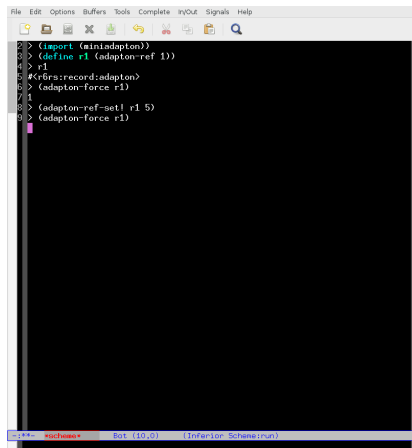


```
File Edit Options Buffers Tools Complete In/OUT Signals Help
[Icons]
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
5 #<r0rs:record:adapton>
6 > (adapton-force r1)
7 1
8 > (adapton-ref-set! r1 5)
```

\*\*\* +-----+ Bot (9,0) [Inferior Scheme (run)]



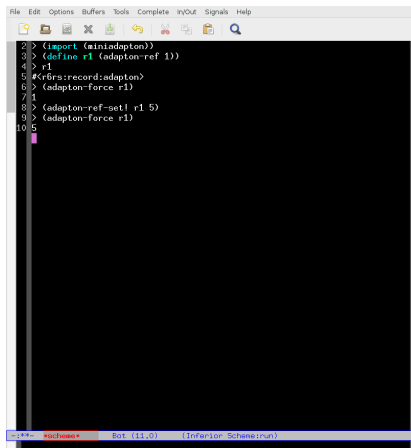
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
5 #<r0rs:record:adapton>
6 > (adapton-force r1)
7 1
8 > (adapton-ref-set! r1 5)
9 > (adapton-force r1)
```

\*\*\*- #<Scheme> Bot: (10,0) [Inferior Scheme (run)]

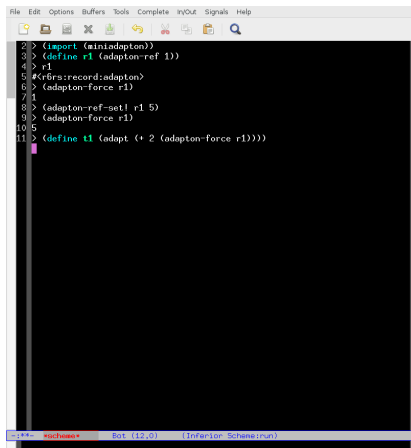
# Interface Demo



```
File Edit Options Buffers Tools Complete In/OUT Signals Help
[Icons]
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
5 #<r6rs:record:adapton>
6 > (adapton-force r1)
7 4
8 > (adapton-ref-set! r1 5)
9 > (adapton-force r1)
10 5
```

At the bottom of the window, the status bar shows: `12:14`, `***Chassis***`, `Bot (11,0)`, and `[Inferior Scheme (run)]`.

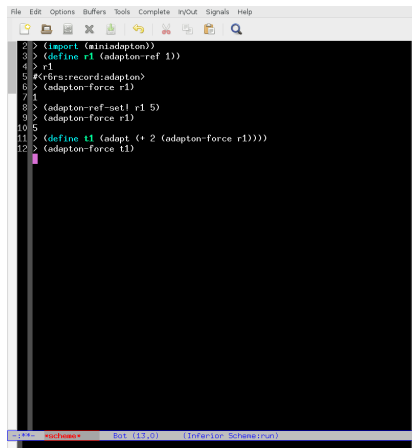
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
5 #r6rs:record:adapton
6 > (adapton-force r1)
7 4
8 > (adapton-ref-set! r1 5)
9 > (adapton-force r1)
10 5
11 > (define t1 (adapton (+ 2 (adapton-force r1))))
```

11:44 - #C[main] - Bot (12,0) | Inferior Scheme (run)

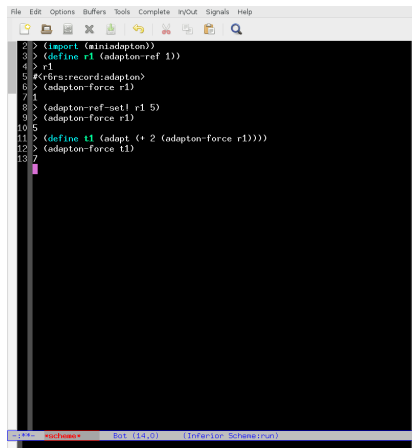
# Interface Demo



```
File Edit Options Buffers Tools Complete In/OUT Signals Help
[Icons]
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
5 #<rs:record:adapton>
6 > (adapton-force r1)
7 4
8 > (adapton-ref-set! r1 5)
9 > (adapton-force r1)
10 5
11 > (define t1 (adapt (+ 2 (adapton-force r1))))
12 > (adapton-force t1)
```

10:14 -- #C[main] Bot (1,1,0) [Inferior Scheme (run)]

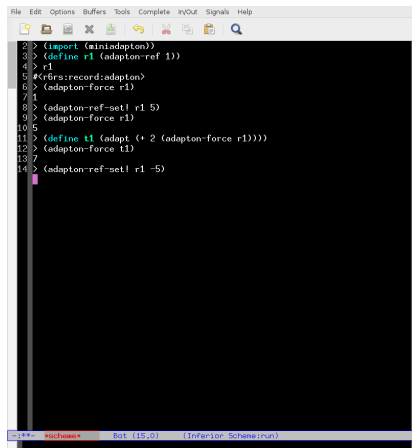
# Interface Demo



```
File Edit Options Buffers Tools Complete In/OUT Signals Help
[Icons]
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
5 #<rs:record:adapton>
6 > (adapton-force r1)
7 4
8 > (adapton-ref-set! r1 5)
9 > (adapton-force r1)
10 5
11 > (define t1 (adapt (+ 2 (adapton-force r1))))
12 > (adapton-force t1)
13 7
```

14:44 - #C[main] - Bot: [1,0] | Inferior Scheme (run)

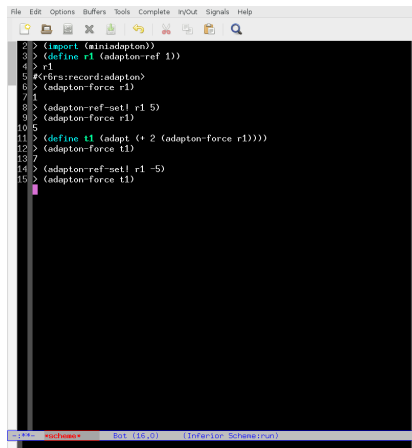
# Interface Demo



```
File Edit Options Buffers Tools Complete In/OUT Signals Help
[Icons]
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
5 #<rs:record:adapton>
6 > (adapton-force r1)
7 1
8 > (adapton-ref-set! r1 5)
9 > (adapton-force r1)
10 5
11 > (define t1 (adapt (+ 2 (adapton-force r1))))
12 > (adapton-force t1)
13 7
14 > (adapton-ref-set! r1 -5)
[Cursor]
```

14:14 - #<Scheme> Bot (15,0) | Inferior Scheme (run)

# Interface Demo



```
File Edit Options Buffers Tools Complete In/OUT Signals Help
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
5 #<rs:record:adapton>
6 > (adapton-force r1)
7 1
8 > (adapton-ref-set! r1 5)
9 > (adapton-force r1)
10 5
11 > (define t1 (adapt (+ 2 (adapton-force r1))))
12 > (adapton-force t1)
13 7
14 > (adapton-ref-set! r1 -5)
15 > (adapton-force t1)
```

15:14+ - #<Scheme> Bot: (16,0) | Inferior Scheme (run)

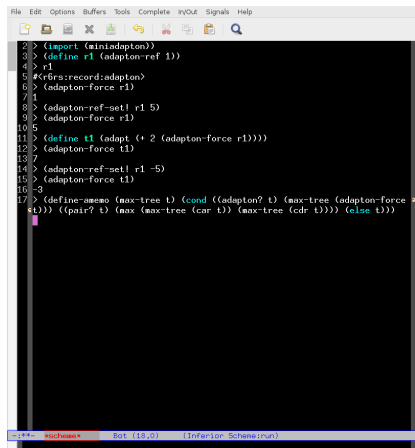
# Interface Demo

```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
5 #r6rs:record:adapton
6 > (adapton-force r1)
7 1
8 > (adapton-ref-set! r1 5)
9 > (adapton-force r1)
10 5
11 > (define t1 (adapt (+ 2 (adapton-force r1))))
12 > (adapton-force t1)
13 7
14 > (adapton-ref-set! r1 -5)
15 > (adapton-force t1)
16 -3
```

100% - #C100000 Bot (17,0) | Inferior Scheme (run)



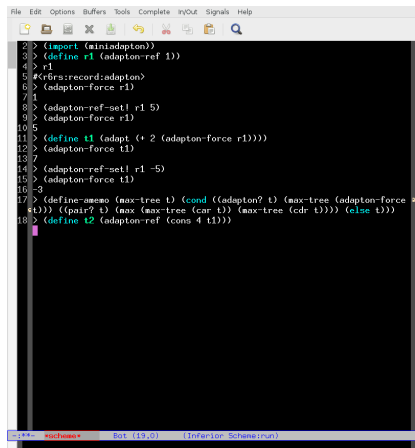
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
5 #r6rs:record:adapton
6 > (adapton-force r1)
7 1
8 > (adapton-ref-set! r1 5)
9 > (adapton-force r1)
10 5
11 > (define t1 (adapt (+ 2 (adapton-force r1))))
12 > (adapton-force t1)
13 7
14 > (adapton-ref-set! r1 -5)
15 > (adapton-force t1)
16 -3
17 > (define-memo (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 v8.10.0.0 Bot (18,0) [Inferior Scheme (run)]

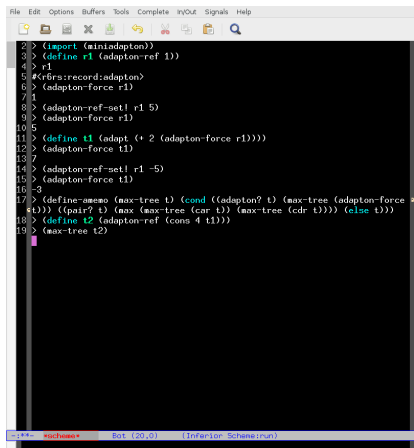
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
5 #r6rs:record:adapton
6 > (adapton-force r1)
7 1
8 > (adapton-ref-set! r1 5)
9 > (adapton-force r1)
10 5
11 > (define t1 (adapt (+ 2 (adapton-force r1))))
12 > (adapton-force t1)
13 7
14 > (adapton-ref-set! r1 -5)
15 > (adapton-force t1)
16 -3
17 > (define-memo (max-tree t) (cond ((adapton? t) (max-tree (adapton-force
18 > (define t2 (adapton-ref (cons 4 t1)))
```

100% Emacs Bot (19,0) | Inferior Scheme (run)

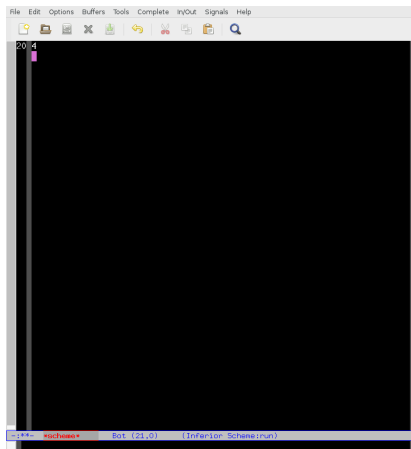
# Interface Demo



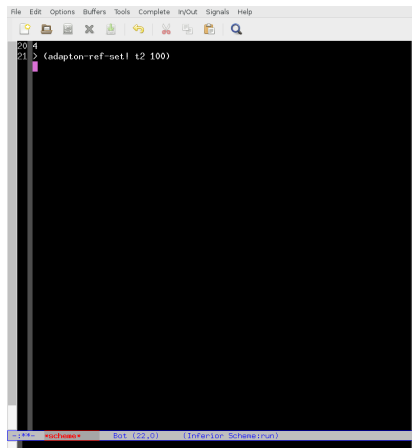
```
File Edit Options Buffers Tools Complete In/Out Signals Help
2 > (import (miniadapton))
3 > (define r1 (adapton-ref 1))
4 > r1
5 #r6rs:record:adapton
6 > (adapton-force r1)
7 1
8 > (adapton-ref-set! r1 5)
9 > (adapton-force r1)
10 5
11 > (define t1 (adapton (+ 2 (adapton-force r1))))
12 > (adapton-force t1)
13 7
14 > (adapton-ref-set! r1 -5)
15 > (adapton-force t1)
16 -3
17 > (define-memo (max-tree t) (cond ((adapton? t) (max-tree (adapton-force
  t)))) ((pair? t) (max (max-tree (car t)) (max-tree (cdr t)))) (else t)))
18 > (define t2 (adapton-ref (cons 4 t1)))
19 > (max-tree t2)
```

SCHEME - #C#max - Bot (20,0) | Inferior Scheme (run)

# Interface Demo



# Interface Demo



A screenshot of a terminal window with a light gray title bar and menu bar. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Complete', 'In/Out', 'Signals', and 'Help'. Below the menu bar is a toolbar with various icons. The main area of the terminal is black with white text. The text shows line numbers 20 and 21. Line 20 contains the number '4'. Line 21 contains the command '> (adaption-ref-set! t2 100)'. A pink cursor is positioned at the end of line 21. At the bottom of the terminal window, there is a status bar with a blue background, displaying '10:44', '\*\*\*Errors\*\*\*', 'Bot: (22,0)', and '(Inferior Scheme (run))'.

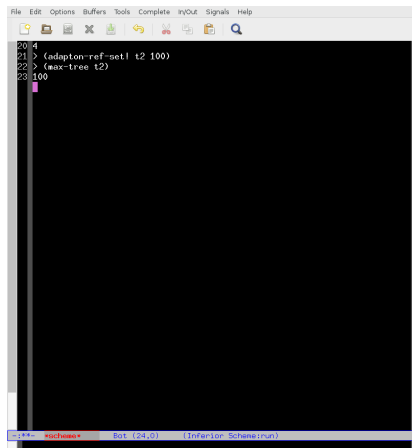
```
20 4
21 > (adaption-ref-set! t2 100)
```

# Interface Demo

```
20 4
21 > (adaption-ref-set! t2 100)
22 > (max-tree t2)
```

Bot: (27,0) [Inferior Scheme (run)]

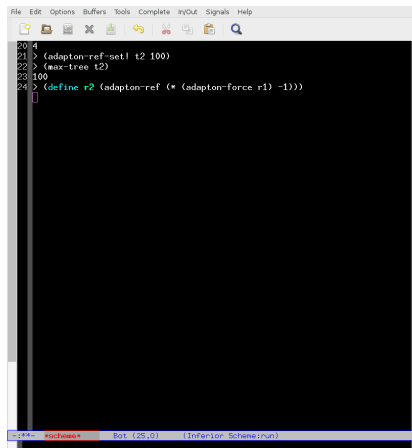
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
20 4
21 > (adaption-ref-set! t2 100)
22 > (max-tree t2)
23 100
```

At the bottom of the window, a status bar displays: `15:14 - #c[main] - Bot: (24,0) - [Inferior Scheme (run)]`

# Interface Demo

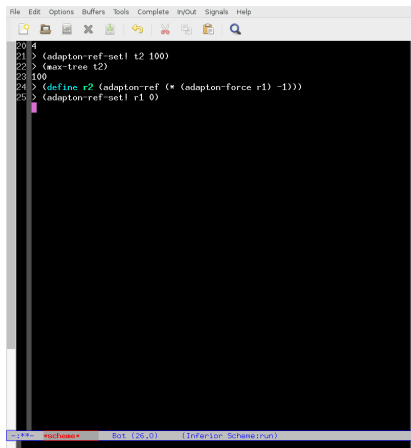


```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
[Cursor]
```

10:24 AM - 4/23/2025 - Bot (25, 0) - [Inferior Scheme (run)]



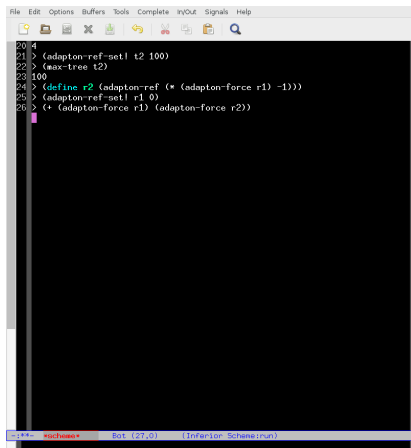
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
```

10:14 AM - 4/23/2016 - Bot [25,0] | Inferior Scheme (run)

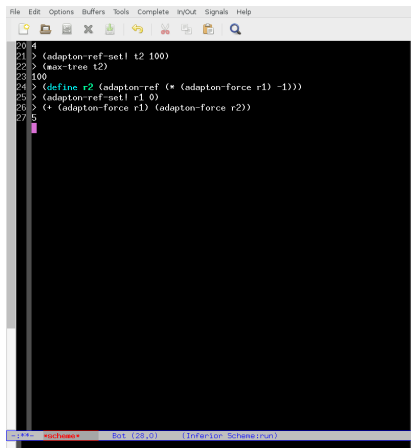
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
26 > (+ (adapton-force r1) (adapton-force r2))
```

10:14 AM - 4/23/2024 - Bot (27,0) | Inferior Scheme (run)

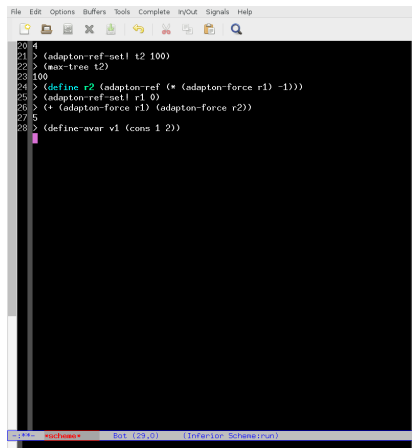
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
26 > (+ (adapton-force r1) (adapton-force r2))
27 5
```

10:14 AM - 4/23/2024 - Bot (28, 0) | Inferior Scheme (run)

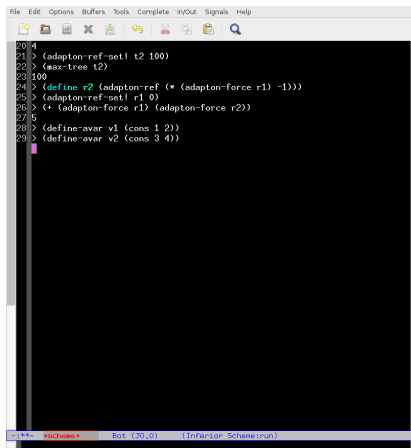
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
26 > (+ (adapton-force r1) (adapton-force r2))
27 5
28 > (define-avar v1 (cons 1 2))
```

15:44 - **inferior-scheme** - Bot (29,0) | Inferior Scheme (run)

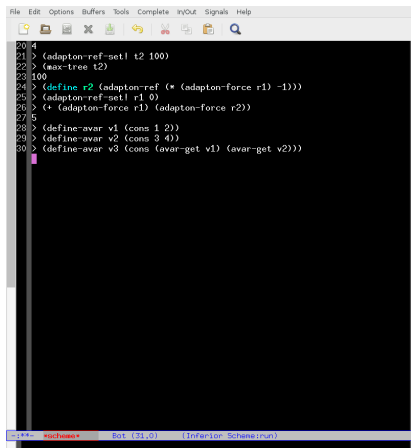
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
26 > (+ (adapton-force r1) (adapton-force r2))
27 5
28 > (define-avar v1 (cons 1 2))
29 > (define-avar v2 (cons 3 4))
```

15:44 - 402 Lines - Bot (30,0) | Inferior Scheme (run)

# Interface Demo

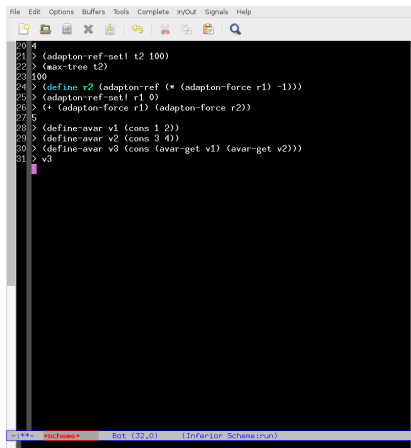


The image shows a screenshot of an Emacs editor window. The window title is "Inferior Scheme (run)". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "Complete", "In/Out", "Signals", and "Help". The toolbar contains icons for file operations and search. The main text area is black with white text, showing the following Scheme code:

```
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
26 > (+ (adapton-force r1) (adapton-force r2))
27 5
28 > (define-avar v1 (cons 1 2))
29 > (define-avar v2 (cons 3 4))
30 > (define-avar v3 (cons (avar-get v1) (avar-get v2)))
```

The status bar at the bottom shows "12:44", "emacs", "Buf: (31,0)", and "Inferior Scheme (run)".

# Interface Demo



The image shows a screenshot of an Emacs editor window. The title bar at the top reads "File Edit Options Buffers Tools Complete In/Out Signals Help". Below the title bar is a toolbar with various icons. The main editing area has a black background with white text. The text consists of several lines of Scheme code, with line numbers 20 through 31 on the left. The code includes function definitions and calls to `adapton-ref-set!`, `adapton-ref`, `adapton-force`, `define-avar`, and `avar-get`. A pink cursor is visible on line 31. At the bottom of the window, a status bar shows "12.144", "emacs", "Buf: (32,0)", and "(Inferior Scheme (run))".

```
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
26 > (+ (adapton-force r1) (adapton-force r2))
27 5
28 > (define-avar v1 (cons 1 2))
29 > (define-avar v2 (cons 3 4))
30 > (define-avar v3 (cons (avar-get v1) (avar-get v2)))
31 > v3
```

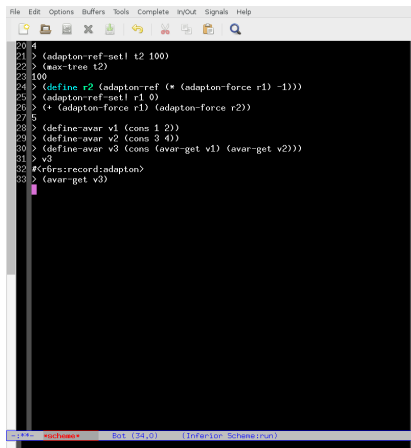
# Interface Demo

```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
26 > (+ (adapton-force r1) (adapton-force r2))
27 5
28 > (define-avar v1 (cons 1 2))
29 > (define-avar v2 (cons 3 4))
30 > (define-avar v3 (cons (avar-get v1) (avar-get v2)))
31 > v3
32 #<rbrs:record:adapton>
```

10:24 AM - 4/23/2016 - Bot: (33,0) | Inferior Scheme (run)



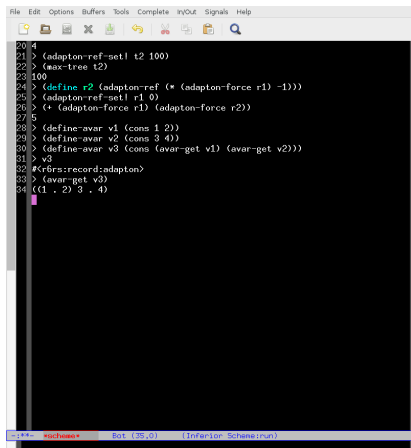
# Interface Demo



```
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
26 > (+ (adapton-force r1) (adapton-force r2))
27 5
28 > (define-avar v1 (cons 1 2))
29 > (define-avar v2 (cons 3 4))
30 > (define-avar v3 (cons (avar-get v1) (avar-get v2)))
31 > v3
32 #<rbrs:record:adapton>
33 > (avar-get v3)
```

12:14 PM - Emacs - Bot (34,0) | Inferior Scheme (run)

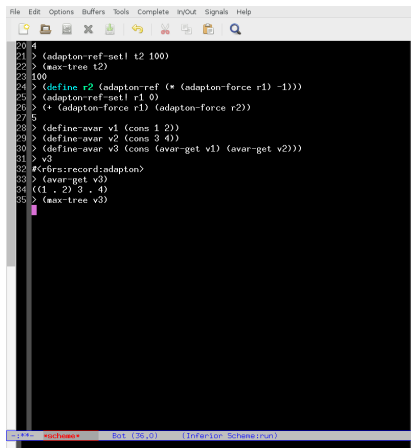
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
26 > (+ (adapton-force r1) (adapton-force r2))
27 5
28 > (define-avar v1 (cons 1 2))
29 > (define-avar v2 (cons 3 4))
30 > (define-avar v3 (cons (avar-get v1) (avar-get v2)))
31 > v3
32 #<rbrs:record:adapton>
33 > (avar-get v3)
34 ((1 . 2) 3 . 4)
```

12:44 ~\*~ Emacs ~ Bot (35,0) | Inferior Scheme (run)

# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
26 > (+ (adapton-force r1) (adapton-force r2))
27 5
28 > (define-avar v1 (cons 1 2))
29 > (define-avar v2 (cons 3 4))
30 > (define-avar v3 (cons (avar-get v1) (avar-get v2)))
31 > v3
32 #<rbrs:record:adapton>
33 > (avar-get v3)
34 (1 . 2) 3 . 4)
35 > (max-tree v3)
```

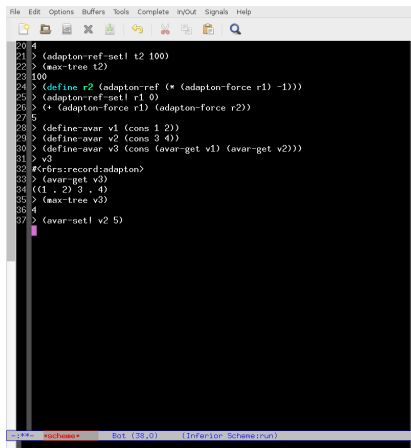
100% 1/31/2024 10:00 AM Bot (36,0) | Inferior Scheme (run)

# Interface Demo

```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
26 > (+ (adapton-force r1) (adapton-force r2))
27 5
28 > (define-avar v1 (cons 1 2))
29 > (define-avar v2 (cons 3 4))
30 > (define-avar v3 (cons (avar-get v1) (avar-get v2)))
31 > v3
32 #<rbrs:record:adapton>
33 > (avar-get v3)
34 (1 2) 3 4
35 > (max-tree v3)
36 4
```

10:24 AM - 10/2/2024 - Bot (37,0) | Inferior Scheme (run)

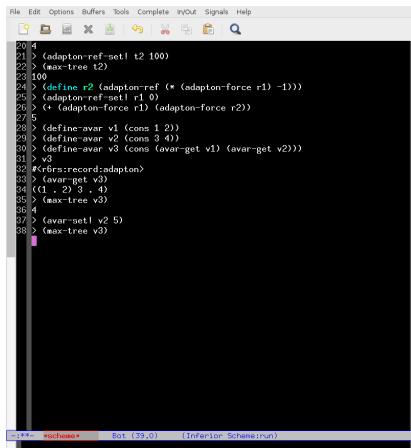
# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
26 > (+ (adapton-force r1) (adapton-force r2))
27 5
28 > (define-avar v1 (cons 1 2))
29 > (define-avar v2 (cons 3 4))
30 > (define-avar v3 (cons (avar-get v1) (avar-get v2)))
31 > v3
32 #<rbrs:record:adapton>
33 > (avar-get v3)
34 ((1 . 2) 3 . 4)
35 > (max-tree v3)
36 4
37 > (avar-set! v2 5)
```

100% 1/31/2024 10:00 AM Bot (35,0) | Inferior Scheme (run)

# Interface Demo



```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
26 > (+ (adapton-force r1) (adapton-force r2))
27 5
28 > (define-avar v1 (cons 1 2))
29 > (define-avar v2 (cons 3 4))
30 > (define-avar v3 (cons (avar-get v1) (avar-get v2)))
31 > v3
32 #<rbrs:record:adapton>
33 > (avar-get v3)
34 ((1 . 2) 3 . 4)
35 > (max-tree v3)
36 4
37 > (avar-set! v2 5)
38 > (max-tree v3)
```

SCHEME+ vscjmas Bot (39,0) | Inferior Scheme (run)

# Interface Demo

```
File Edit Options Buffers Tools Complete In/Out Signals Help
[Icons]
20 4
21 > (adapton-ref-set! t2 100)
22 > (max-tree t2)
23 100
24 > (define r2 (adapton-ref (* (adapton-force r1) -1)))
25 > (adapton-ref-set! r1 0)
26 > (+ (adapton-force r1) (adapton-force r2))
27 5
28 > (define-avar v1 (cons 1 2))
29 > (define-avar v2 (cons 3 4))
30 > (define-avar v3 (cons (avar-get v1) (avar-get v2)))
31 > v3
32 #<rbrs:record:adapton>
33 > (avar-get v3)
34 ((1 . 2) 3 . 4)
35 > (max-tree v3)
36 4
37 > (avar-set! v2 5)
38 > (max-tree v3)
39 5
```

10:24 AM - 10/2/2014 - Bot (35,0) | 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

```
File Edit Options Buffers Tools Lisp-Interaction Help
[Icons] [Search]
1 (define-record-type
2   (adapton adapton-cons adapton?)
3   (fields
4     think
5     (mutable result)
6     (mutable sub)
7     (mutable super)
8     (mutable clean?)))
9
10 (define (make-athunk think)
11   (adapton-cons think
12     'empty
13     empty-set
14     empty-set
15     #f))
16
17 (define (adapton-add-dcg-edge! a-super a-sub)
18   (adapton-sub-set! a-super
19     (set-cons a-sub (adapton-sub a-super))))
20   (adapton-super-set! a-sub
21     (set-cons a-super (adapton-super a-sub))))
22
23 (define (adapton-del-dcg-edge! a-super a-sub)
24   (adapton-sub-set! a-super
25     (set-rem a-sub (adapton-sub a-super))))
26   (adapton-super-set! a-sub
27     (set-rem a-super (adapton-super a-sub))))
28
29 (define (adapton-compute a)
30   (if (adapton-clean? a)
31     (adapton-result a)
32     (begin
33       (set-for-each
34         (lambda (x)
35           (adapton-del-dcg-edge! a x))
36         (adapton-sub a))
37       (adapton-clean?-set! a #t)
38       (adapton-result-set! a
39         (adapton-super a))))))
40
41 adapton-compute (top (27,0)) (lisp-Interaction Faredit)
Beginning of buffer
```

# Implementation - microAdapton

```
File Edit Options Buffers Tools Lisp-Interaction Help
[Icons]
37/ | (adapton-clean?-set! a #t)
38 | (adapton-result-set! a
39 | ((adapton-thank a)))
40 | (adapton-compute a)))
41 |
42 | (define (adapton-dirty! a)
43 | (when (adapton-clean? a)
44 | (adapton-clean?-set! a #f)
45 | (set-for-each adapton-dirty!
46 | (adapton-super a))))
47 |
48 | (define (adapton-ref val)
49 | (letrec ((a (adapton-cons
50 | (lambda () (adapton-result a))
51 | val
52 | empty-set
53 | empty-set
54 | #t)))
55 | a))
56 |
57 | (define (adapton-ref-set! a val)
58 | (adapton-result-set! a val)
59 | (adapton-dirty! a))
60 |
61 |
62 |
63 |
64 |
65 |
66 |
67 |
68 |
69 |
70 |
71 |
72 |
73 |
74 |
[Status Bar] scratch 2/3 (27,0) (Lisp-Interaction Faredit)
```

# Implementation - miniAdapton

```
File Edit Options Buffers Tools Lisp-Interaction Help
[Icons]
1 (define adapton-force
2   (let ((currently-adapting #f))
3     (lambda (a)
4       (let ((prev-adapting
5             currently-adapting))
6         (set! currently-adapting a)
7         (let ((result (adapton-compute a)))
8           (set! currently-adapting
9                 prev-adapting)
10          (when currently-adapting
11              (adapton-add-dcg-edge!
12               currently-adapting
13               a))
14          result))))))
15
16 (define-syntax adapt
17   (syntax-rules ()
18     (( _ expr)
19      (make-athunk (lambda () expr)))))
20
21 (define (adapton-memoize-1 f)
22   (memoize (lambda x (adapt (apply f x)))))
23
24 (define (adapton-memoize f)
25   (let ((f* (adapton-memoize-1 f)))
26     (lambda x (adapton-force (apply f* x))))))
27
28 (define-syntax lambda-memo-1
29   (syntax-rules ()
30     (( (args ...) body ...)
31      (let ((f* (adapton-memoize-1
32                (lambda (args ...)
33                  body ...))))
34        (lambda (args ...) (f* args ...))))))
35
36 (define-syntax lambda-memo
37   (syntax-rules ()
38     (( (args ...) body ...)
39      (make-athunk (lambda () (lambda-memo-1 (lambda (args ...) body ...)))))))
40
41 Emacs 24.3.1 (Lisp-Interaction Faredit)
```

# Implementation - miniAdapton

```
File Edit Options Buffers Tools Lisp-Interaction Help
[Icons]
37 (syntax-rules ()
38   (lambda (args ...) body ...)
39   (let ((f* (adapton-memoize
40             (lambda (args ...)
41               body ...))))
42     (lambda (args ...) (f* args ...))))))
43
44 (define-syntax define-amenol
45   (syntax-rules ()
46     ((f args ...) body ...)
47     (define f (lambda-amenol (args ...)
48                  body ...))))
49
50 (define-syntax define-ameno
51   (syntax-rules ()
52     ((f args ...) body ...)
53     (define f (lambda-ameno (args ...)
54                  body ...))))
55
56 (define-syntax define-avar
57   (syntax-rules ()
58     (define name
59       (adapton-ref (adapt expr))))))
60
61 (define (avar-get v)
62   (adapton-force (adapton-force v)))
63
64
65 (define-syntax avar-set!
66   (syntax-rules ()
67     ((v expr)
68      (adapton-ref-set! v (adapt expr))))))
69
[Scratch] Bot (27,0) (Lisp-Interaction Faredit)
```

# 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>)