Computational Contracts

Christophe Scholliers
Wolfgang De Meuter
Software Languages Lab
Vrije Universiteit Brussel

Éric Tanter
PLEIAD Lab
University of Chile

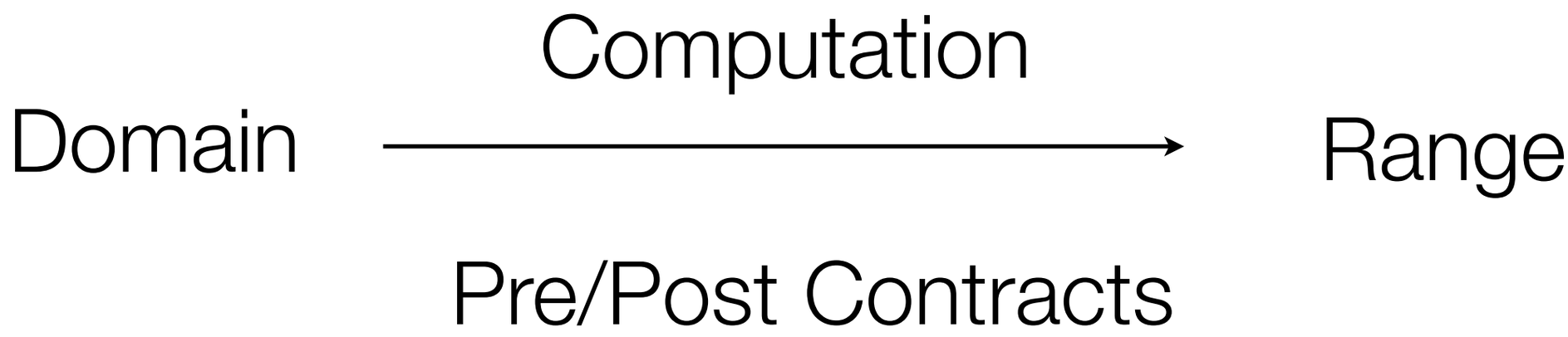
Contracts

Agreement

Exchanged Values

2

Overview



3

Overview

Domain → Range

Computation

Pre/Post Contracts

3

Overview

Domain → Computation → Range

Computational Contracts

A computational contracts ***verifies a (sequence of) events*** during the execution of a contracted entity.

3

Contracts in Racket

4

Pre/Post Contracts

provider.rkt

```
(define (sqrt x)
  ...)
(provide/contract
  [sqrt (-> positive? positive?)])
```

```
Welcome to DrRacket, version 5.1.3 [3m].
Language: scheme [custom]; memory limit: 128 MB.
> (sqrt -2)
lascheme-api.rkt:219:34: Sqrt: contract violation, expected <positive?>, given: -2
  contract from
    /Users/cfscholl/Desktop/Research/DCC-SVN/LAScheme/provider.rkt
  blaming
    /Users/cfscholl/Desktop/Research/DCC-SVN/LAScheme/user.rkt
  contract: (-> positive? positive?)
       at: /Users/cfscholl/Desktop/Research/DCC-SVN/LAScheme/provider.rkt:174.2
```

5

Pre/Post Contracts

provider.rkt

```
(define (sqrt x)
  ...)
(provide/contract
  [sqrt (-> positive? positive?)])
```

```
Welcome to DrRacket, version 5.1.3 [3m].
Language: scheme [custom]; memory limit: 128 MB.
> (sqrt -2)
lascheme-api.rkt:219:34: Sqrt: contract violation, expected <positive?>, given: -2
contract from
  /Users/cfscholl/Desktop/Research/DCC-SVN/LAScheme/provider.rkt
blaming
  /Users/cfscholl/Desktop/Research/DCC-SVN/LAScheme/user.rkt
contract: (-> positive? positive?)
      at: /Users/cfscholl/Desktop/Research/DCC-SVN/LAScheme/provider.rkt:174.2
```


5

Higher-Order Contracts

provider.rkt

```
(define (map-pos f l)
  (map f l))

(provide/contract
 [map-pos (->
           (-> positive? positive?)
           (listof positive?)
           (listof positive?))])
```


6

Higher-Order Contracts

provider.rkt

```
(define (map-pos f l)
  (map f l))

(provide/contract
 [map-pos (->
           (-> positive? positive?)
           (listof positive?)
           (listof positive?))])
```


6

Higher-Order Contracts

provider.rkt

```
(define (map-pos f l)
  (map f l))

(provide/contract
 [map-pos (->
           (-> positive? positive?)
           (listof positive?)
           (listof positive?))])
```


6

Higher-Order Contracts

provider.rkt

```
(define (map-pos f l)
  (map f l))

(provide/contract
 [map-pos (->
           (-> positive? positive?)
           (listof positive?)
           (listof positive?))])
```


6

Higher-Order Contracts

provider.rkt

```
(define (map-pos f l)
  (map f l))

(provide/contract
 [map-pos (->
           (-> positive? positive?)
           (listof positive?)
           (listof positive?))])
```


6

Higher-Order Contracts

provider.rkt

```
(define (map-pos f l)
  (map f l))

(provide/contract
 [map-pos (->
           (-> positive? positive?)
           (listof positive?)
           (listof positive?))])
```


6

Higher-Order Contracts

provider.rkt

```
(define (map-pos f l)
  (map f l))

(provide/contract
 [map-pos (->
           (-> positive? positive?)
           (listof positive?)
           (listof positive?))])
```


6

Higher-Order Contracts

provider.rkt

```
(define (map-pos f l)
  (map f l))

(provide/contract
 [map-pos (->
           (-> positive? positive?)
           (listof positive?)
           (listof positive?))])
```

user.rkt

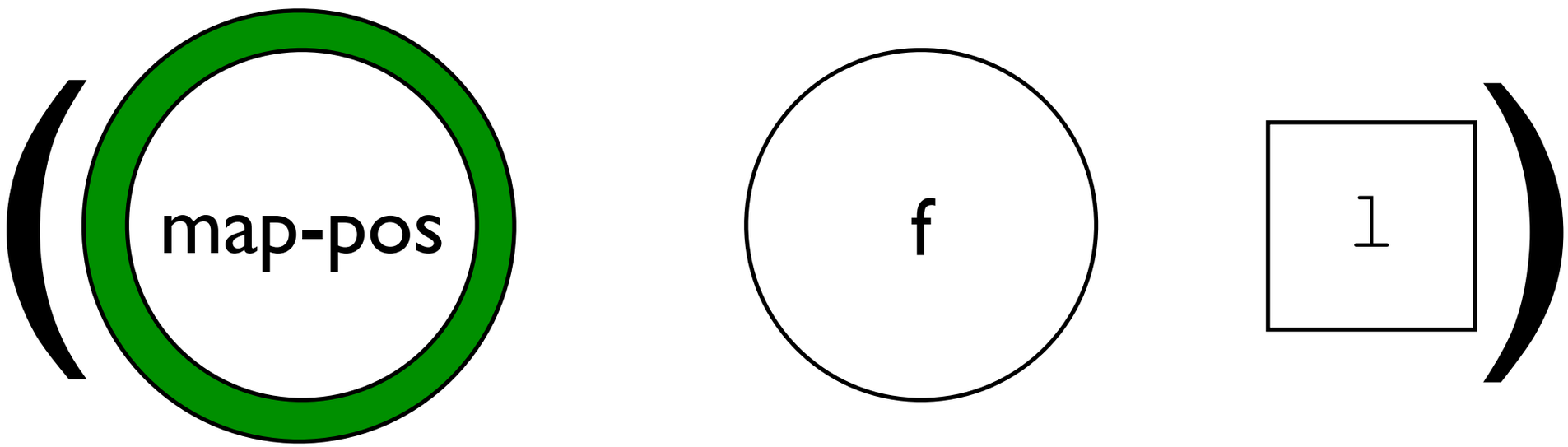
```
(map-pos integer? '(1 2 3))
```

Higher-Order Contracts

provider.rkt

```
contract violation: expected <integer?>, given: #t
  contract on map-pos from
    (file .../provider.rkt)
  blaming
    (file .../user.rkt)
  contract:
    (->
     (-> positive? positive?)
     (listof positive?)
     (listof positive?))
```

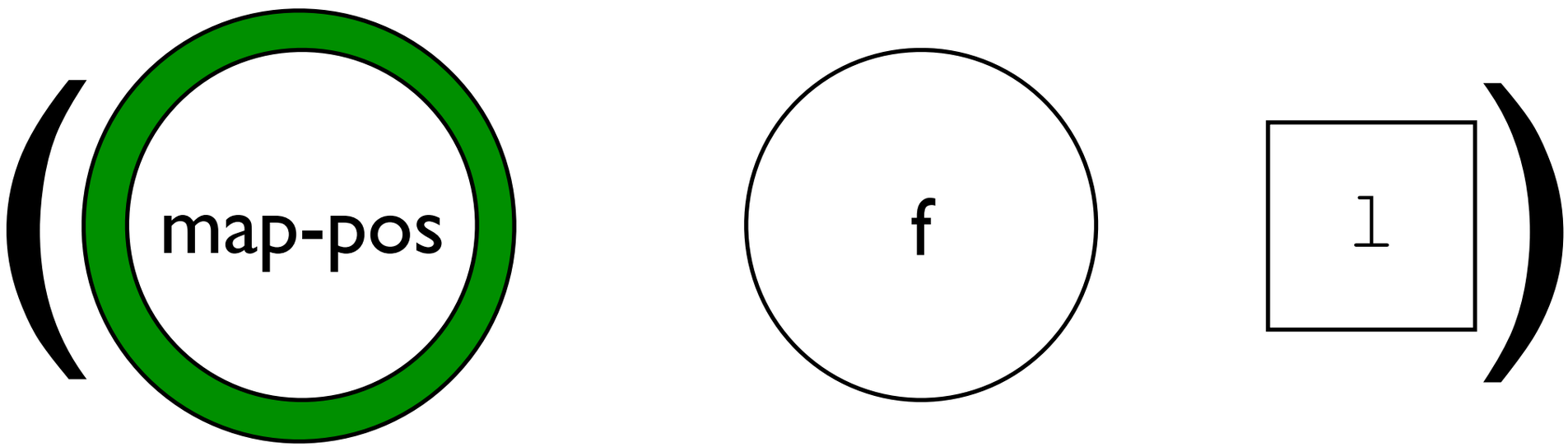
Principle of HOCs



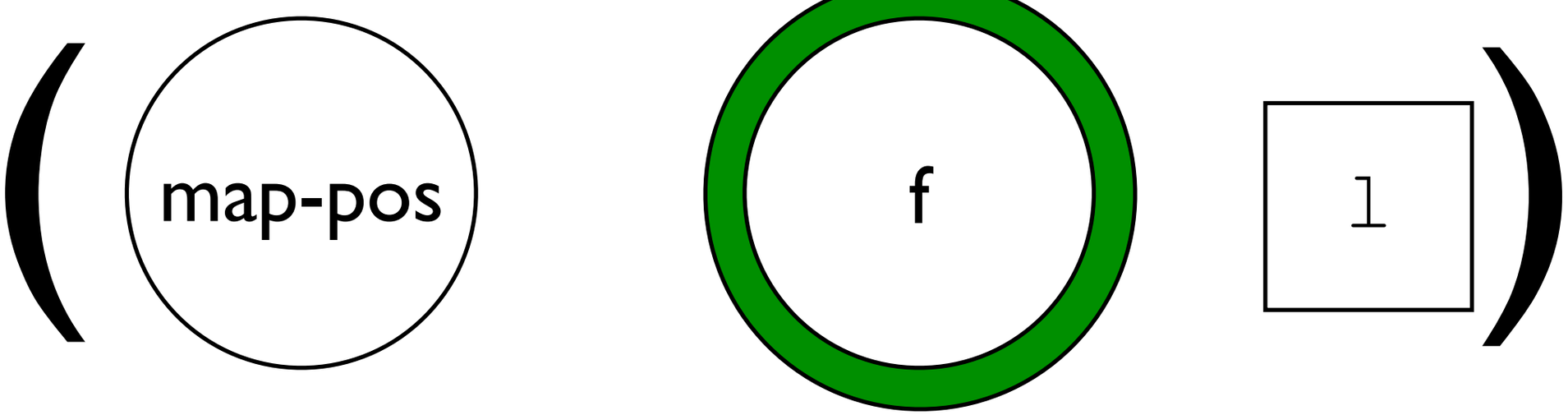
There is no simple predicate that can verify the contract defined over f.

7

Principle of HOCs



There is no simple predicate that can verify the contract defined over f.



7

Computational Contracts

8

Computational Contracts

Safety

Prohibit contract

Liveness

Promise contract

9

Computational Contracts

Safety

Prohibit contract

Liveness

Promise contract

9

Prohibit Contract

Computational Contracts over a single event

provider.rkt

```
 (define (map-silent f l)
   (map f l))

(provide/contract
 [map-silent (->
              (prohibit/c  (call display))
              (listof any?)
              (listof any?))])
```

10

Prohibit Contract

Computational Contracts over a single event

provider.rkt

```
 (define (map-silent f l)
   (map f l))

(provide/contract
 [map-silent (->
              (prohibit/c  (call display))
              (listof any?)
              (listof any?))])
```

10

Prohibit Contract

Computational Contracts over a single event

provider.rkt

```
 (define (map-silent f l)
   (map f l))

(provide/contract
 [map-silent (->
              (prohibit/c  (call display))
              (listof any?)
              (listof any?))])
```

10

Prohibit Contract

Computational Contracts over a single event

provider.rkt

```
 (define (map-silent f l)
   (map f l))

(provide/contract
 [map-silent (->
              (prohibit/c  (call display))
              (listof any?)
              (listof any?))])
```

10

Prohibit Contract

Computational Contracts over a single event

provider.rkt

```
 (define (map-silent f l)
   (map f l))

(provide/contract
 [map-silent (->
              (prohibit/c  (call display))
              (listof any?)
              (listof any?))])
```

10

Prohibit Contract

Computational Contracts over a single event

provider.rkt

```
 (define (map-silent f l)
   (map f l))

(provide/contract
 [map-silent (->
              (prohibit/c  (call display))
              (listof any?)
              (listof any?))])
```

10

Prohibit Contract

Computational Contracts over a single event

provider.rkt

```
 (define (map-silent f l)
   (map f l))

(provide/contract
 [map-silent (->
              (prohibit/c  (call display))
              (listof any?)
              (listof any?))])
```

10

Prohibit Contract

Computational Contracts over a single event

provider.rkt

```
 (define (map-silent f l)
   (map f l))

(provide/contract
 [map-silent (->
              (prohibit/c  (call display))
              (listof any?)
              (listof any?))])
```

user.rkt

```
(define (inc x) (display x) (+ x 1))
(map-silent inc  '(1 2 3 4))
```

Prohibit Contract

Computational Contracts over a single event

provider.rkt

```
(define (map-silent f l)
  (map f l))
```

(file user.rkt)
broke the contract
(->
prohibit-call-display
(listof any?)
(listof any?))
on map-silent; computational contract violation ...

```
(map-silent inc '(1 2 3 4))
```

Promise Contract

Computational Contracts over a single event

provider.rkt

```
(define (set-x! x) ...)
(provide/contract
 [set-x!
  (and/c (promise/c (call notify))
         (-> integer? any?))])
```

11

Promise Contract

Computational Contracts over a single event

provider.rkt

```
(define (set-x! x) ...)
(provide/contract
 [set-x!
  (and/c (promise/c (call notify))
         (-> integer? any?))])
```

11

Promise Contract

Computational Contracts over a single event

(file .../provider.rkt)
broke the contract
 (and/c
  promise-call-notify
  (-> integer? any?))
on set-x!;
computational contract violation ....

11

Computational Contracts

Sequence of Events

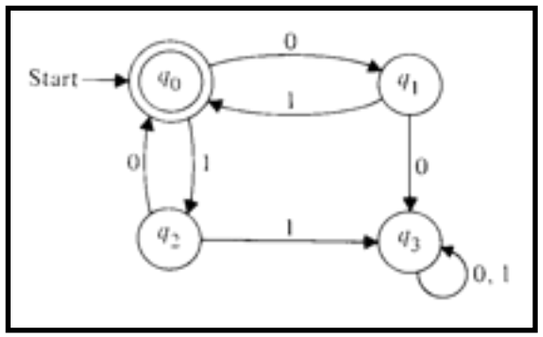
Prohibit Contract

Promise Contract

12

Computational Contracts

Sequence of Events



Prohibit Contract

Promise Contract

12

Sequence of Events

Computational Contracts

Promise:

A file that is opened will eventually be closed

13

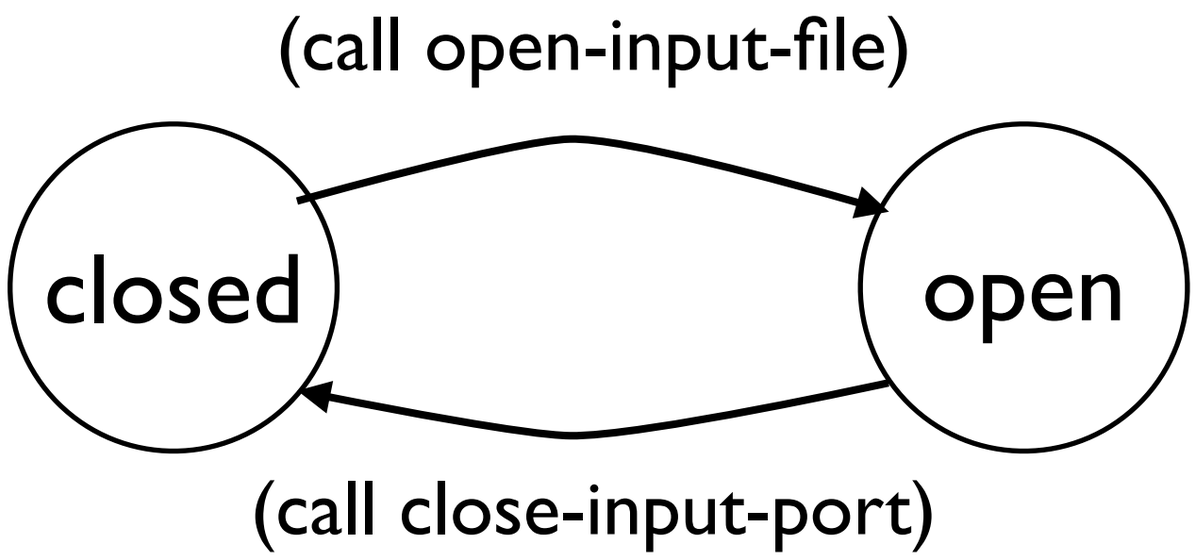
Sequence of Events

Computational Contracts

Promise:

A file that is opened will eventually be closed

file-protocol



13

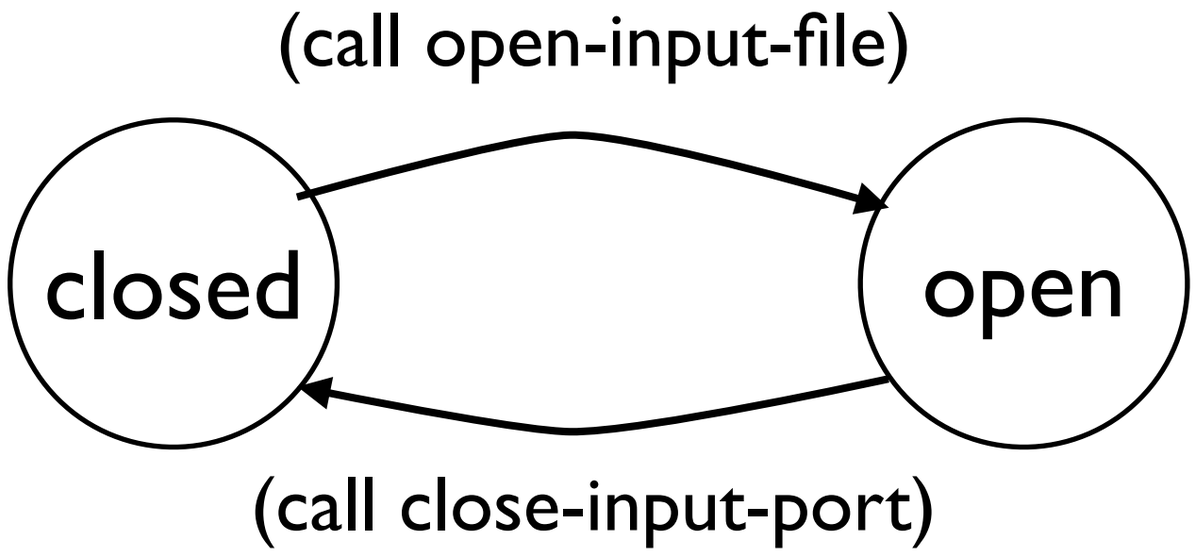
Sequence of Events

Computational Contracts

Promise:

A file that is opened will eventually be closed

file-protocol



end state SHOULD be reached

13

Sequence of Events

Computational Contracts

provider.rkt

```
(define (get-content reader)
  (reader "myfile.rkt"))
(provide/contract
 [get-content
  (promise/c file-protocol) -> string?])
```


14

Sequence of Events

Computational Contracts

provider.rkt

```
(define (get-content reader)
  (reader "myfile.rkt"))
(provide/contract
 [get-content
  (promise/c file-protocol) -> string?])
```

user.rkt

```
(get-content bad-reader)
```


14

Sequence of Events

Computational Contracts

provider.rkt

```
(define (get-content reader)
```

(file .../user.rkt)
broke the contract
 (-> promise-protocol/c string?)
on get-content; promise-protocol/c
violated, not in end state, last transition
after 'open-input-file

14

Sequence of Events

Computational Contracts

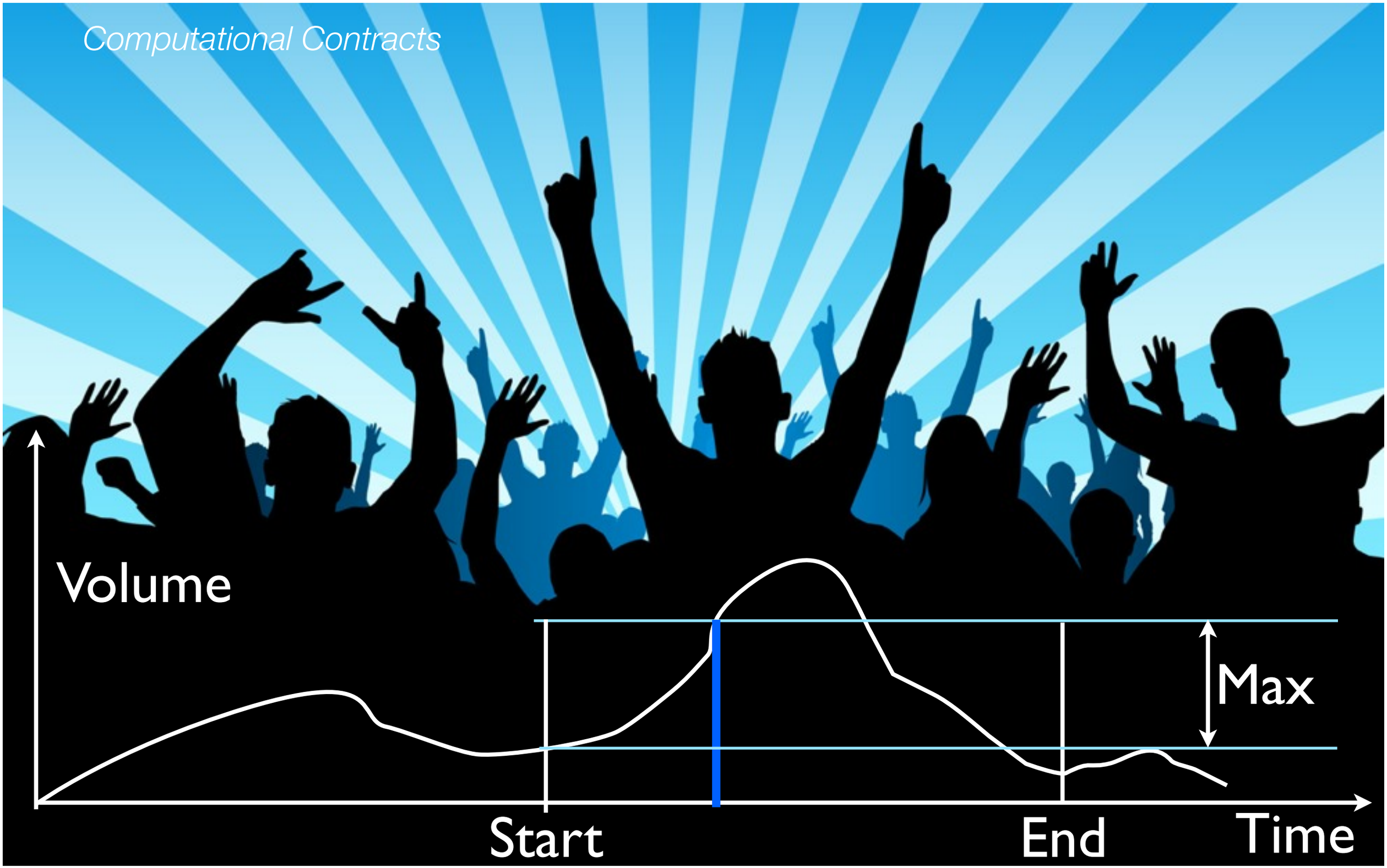
provider.rkt

```
(define file-protocol
  (protocol end
            [end : ((call open-input-file)
                                  -> open)]
            [open : ((call close-input-port)
                                  -> end)]))
```

15

Stateful Monitors

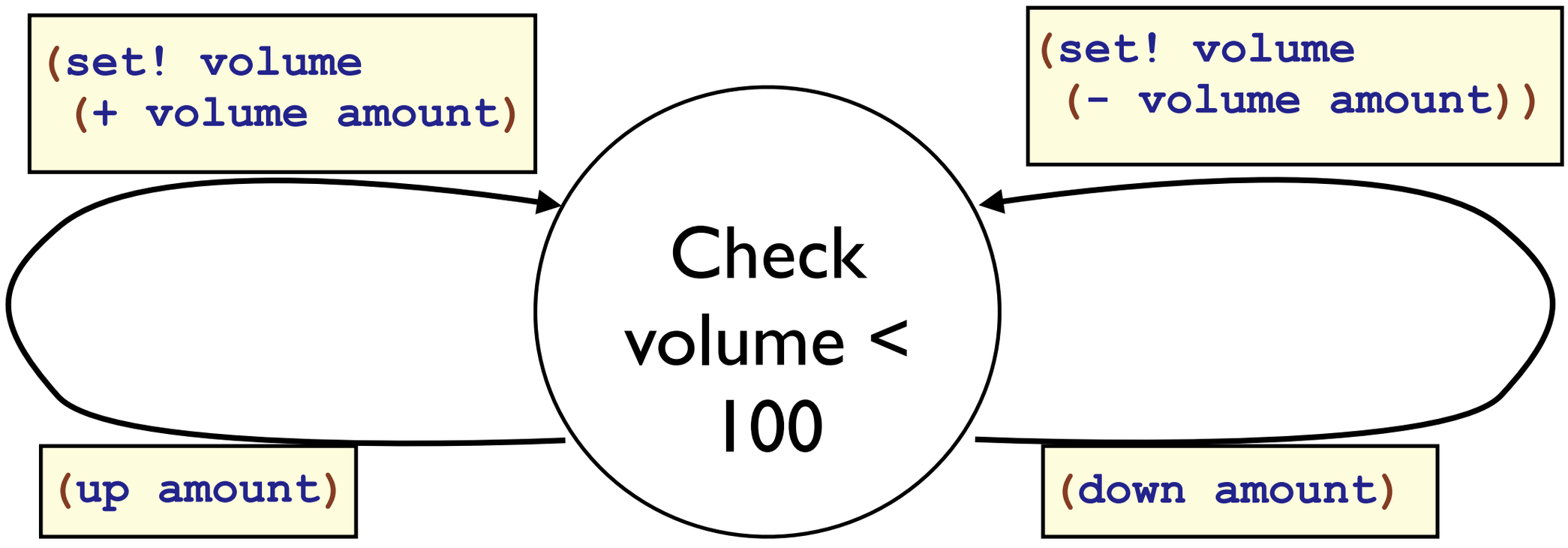
Computational Contracts



Stateful Monitors

Computational Contracts

sound-monitor



17

Stateful Monitors

Computational Contracts

provider.rkt

```
(define (sound-monitor max-volume)
  (monitor
   [(volume 0)]
   [(< volume max-volume)]
   [((up amount)   ->  (set! volume (+ volume amount)))
    ((down amount) ->  (set! volume (- volume amount)))]))
(provide/contract
 [speak
  (and/c (promise/c (sound-monitor 100))
         (-> string? any?))])
```

Stateful Monitors

Computational Contracts

provider.rkt

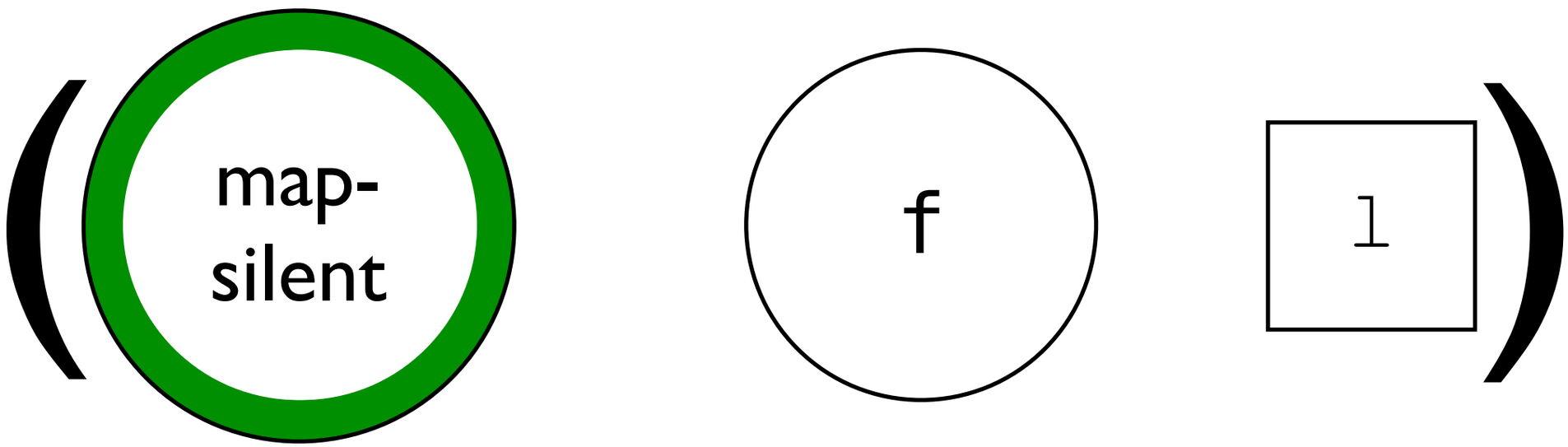
(file .../provider.rkt)
broke the contract
promise-monitor/c
on speak;
monitor computational contract violation
#<procedure:...user.rkt:138:53>

18

Implementation

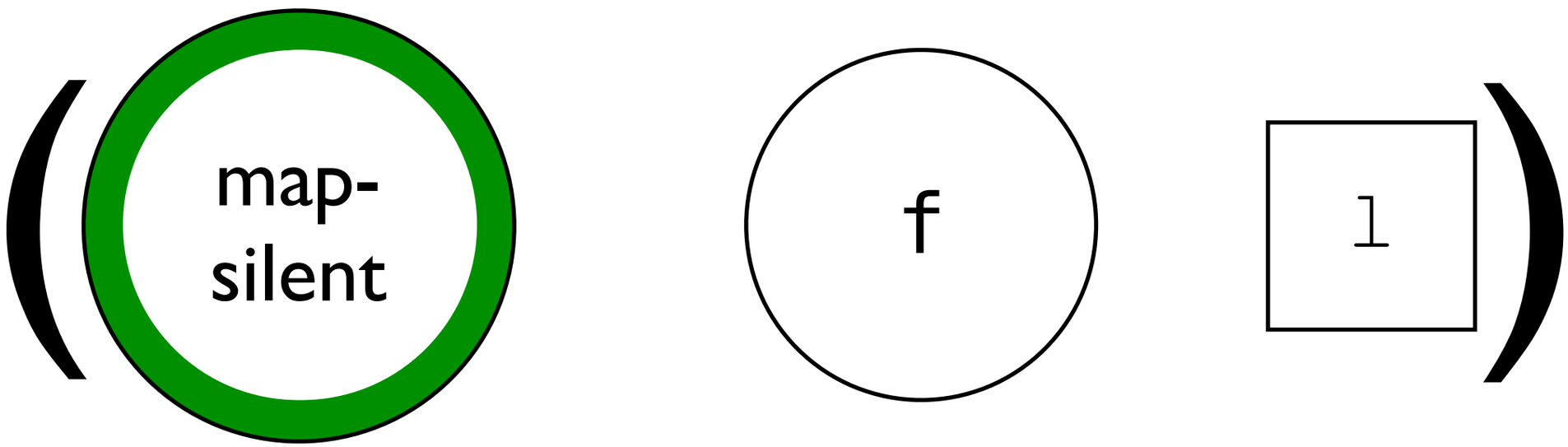
19

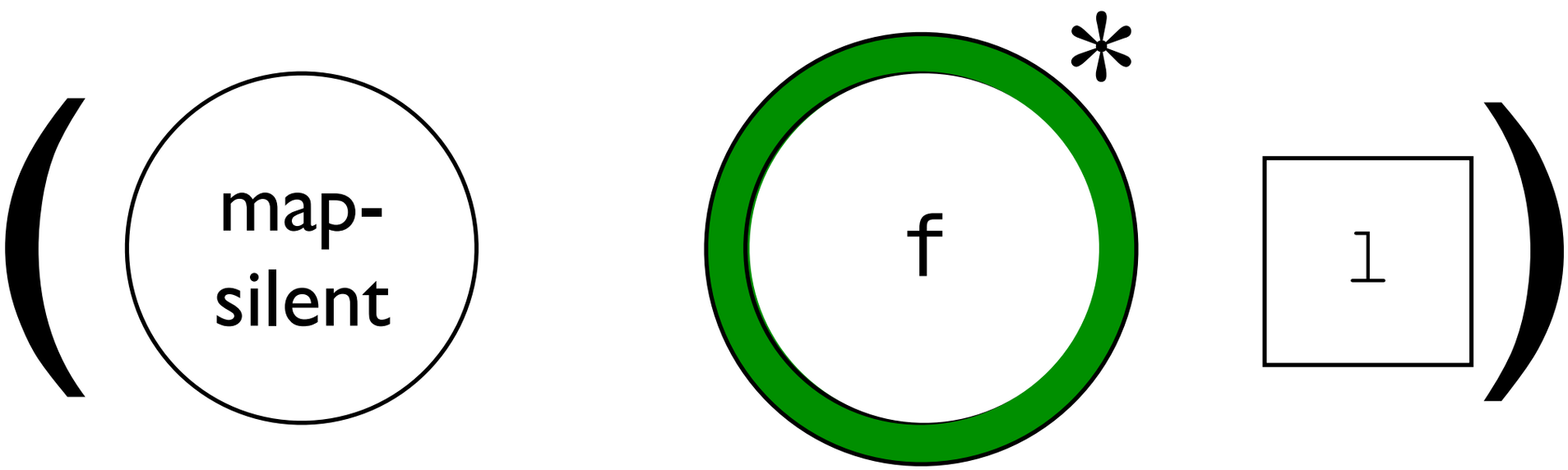
Computational Contracts



20

Computational Contracts





20

Contract Representation

```
(provide/contract
 [map-silent (->
              (prohibit/c  (call display))
              (listof any?)
              (listof any?))])
```


21

Contract Representation

```
(provide/contract
 [map-silent (->
              (prohibit/c  (call display))
              (listof any?)
              (listof any?))])
```

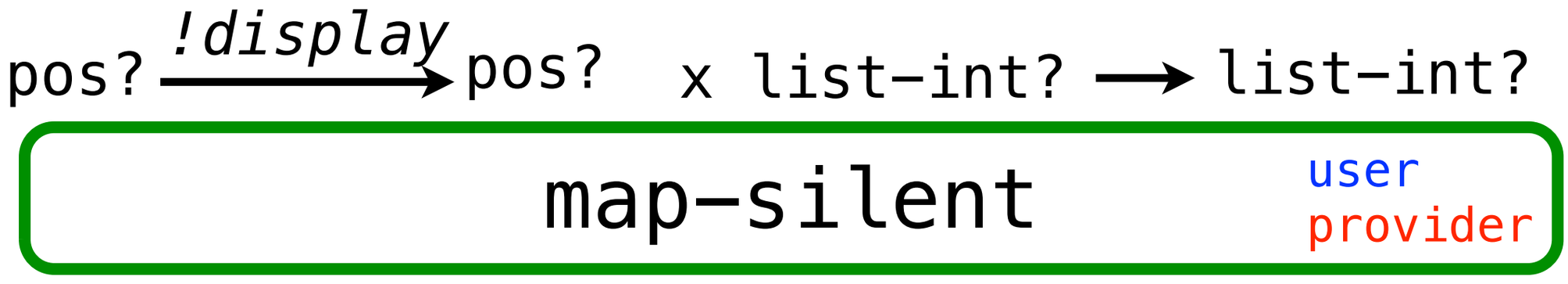
pos? —*!display*→ pos? x list-int? → list-int?

map-silent

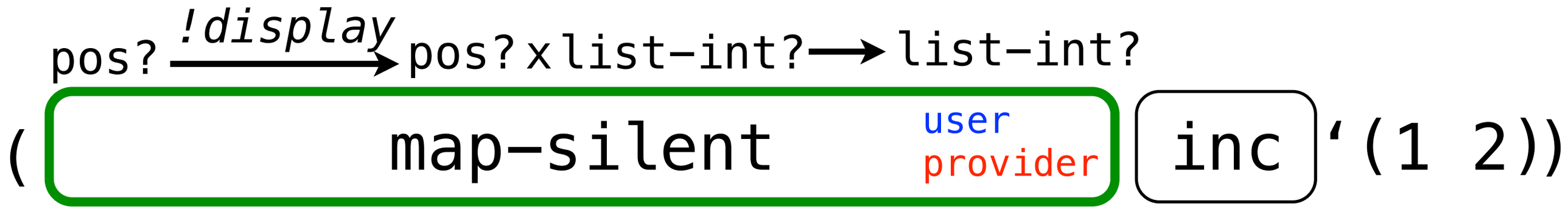
21

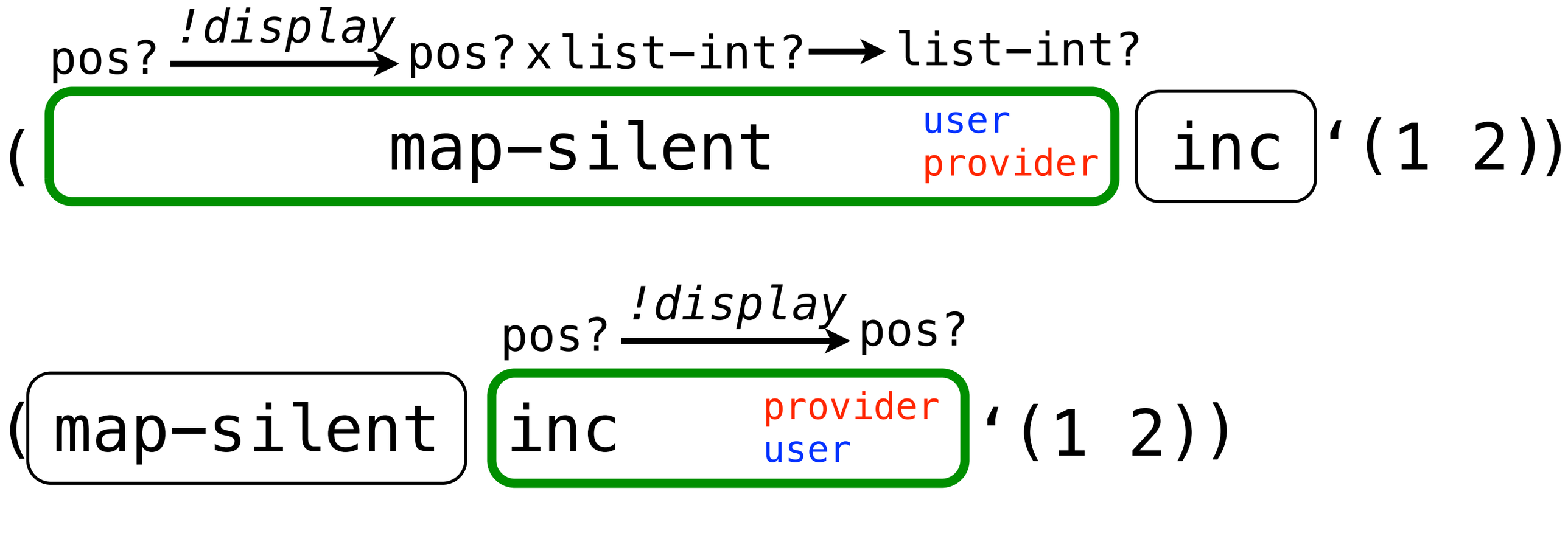
Contract Representation

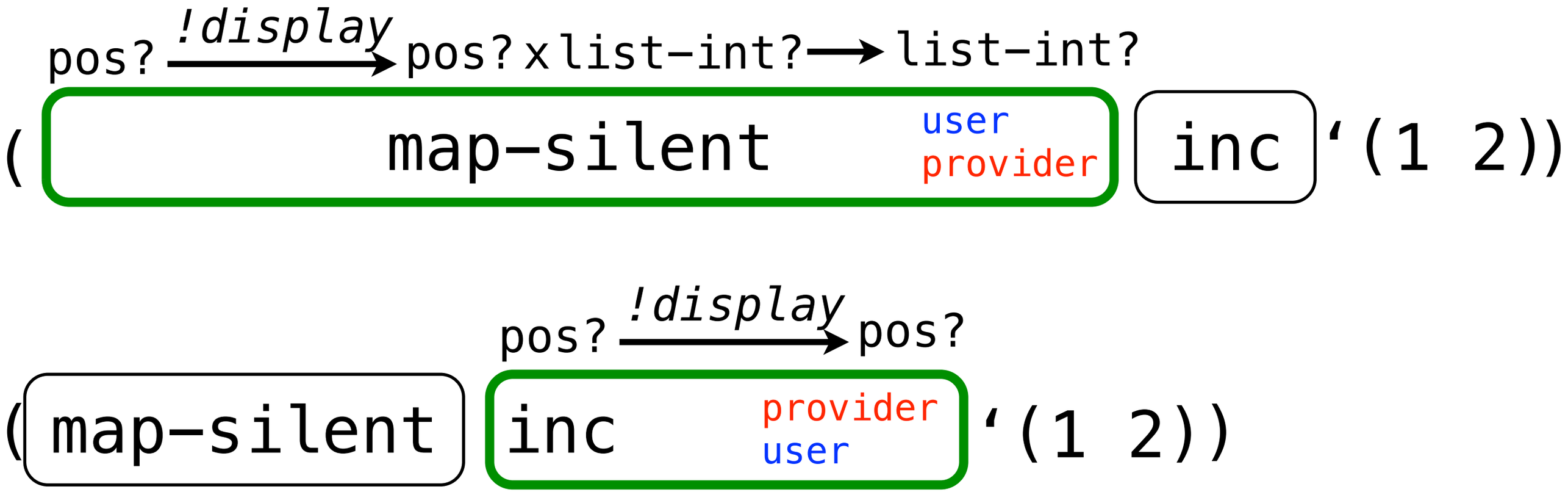
```
(provide/contract
 [map-silent (->
              (prohibit/c  (call display))
              (listof any?)
              (listof any?))])
```

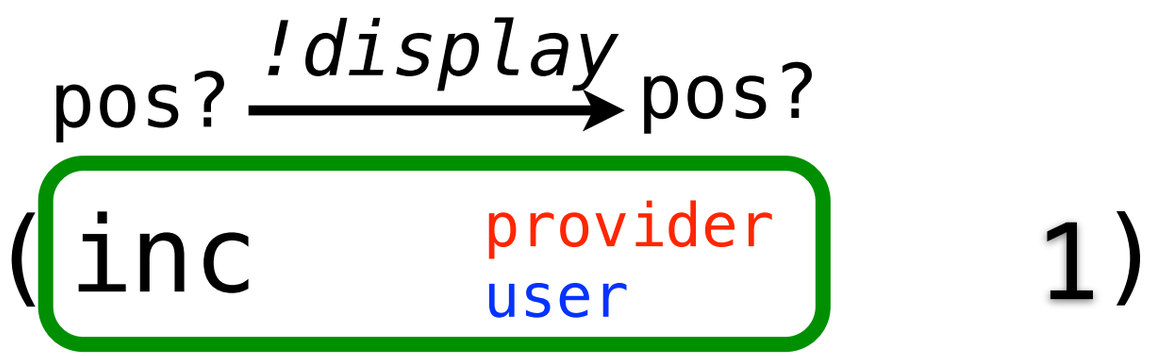


21


pos?
!display
pos? x list-int?
list-int?
(
map-silent
user
provider
inc
'(1 2))


pos?
!display
pos? x list-int?
list-int?
(
map-silent
user
provider
inc
'(1 2))
pos?
!display
pos?
(
map-silent
inc
provider
user
'(1 2))


pos?
!display
pos? x list-int? → list-int?
(
map-silent
user
provider
inc
'(1 2))
pos?
!display
pos?
(
map-silent
inc
provider
user
'(1 2))


pos?
!display
pos?
(
inc
provider
user
1)

pos? —!display→ pos? x list-int? → list-int?

(map-silent user provider inc '(1 2))

pos? —!display→ pos?

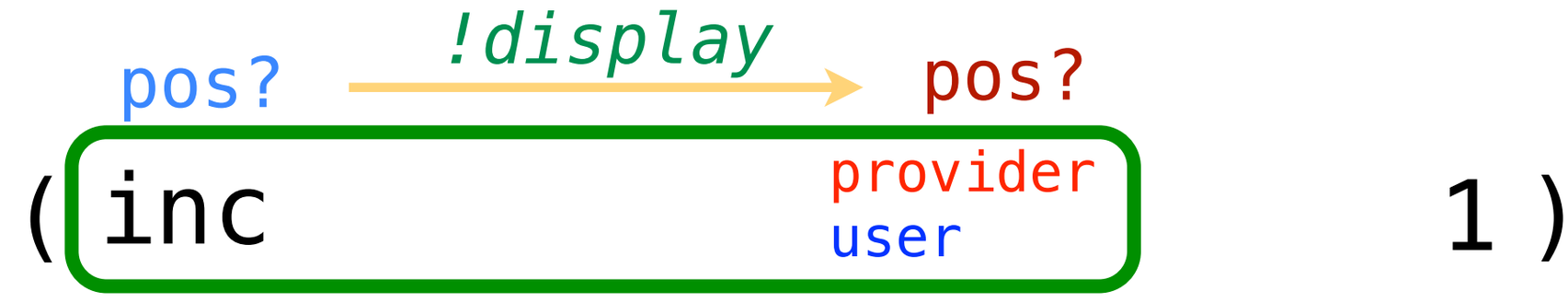
(map-silent inc provider user '(1 2))

pos? —!display→ pos?

(inc provider user 1)

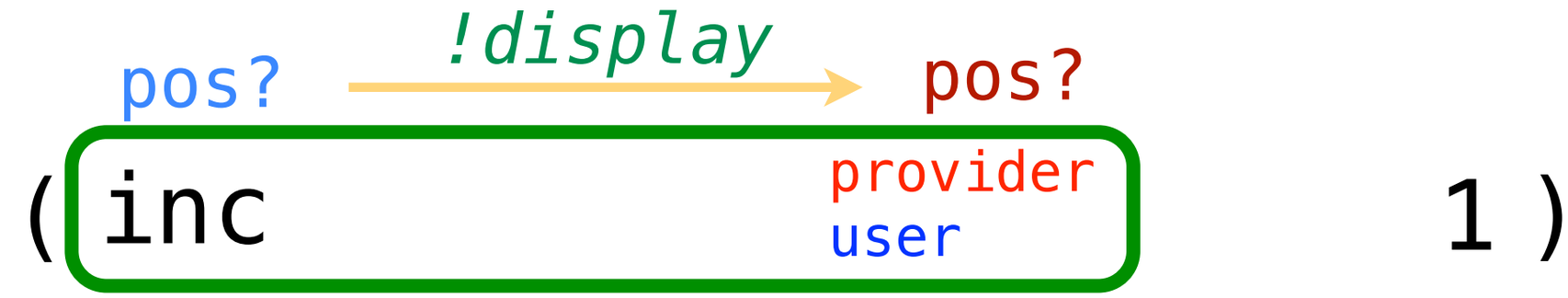
(inc 1)!display user

Implementation Overview



23

Implementation Overview

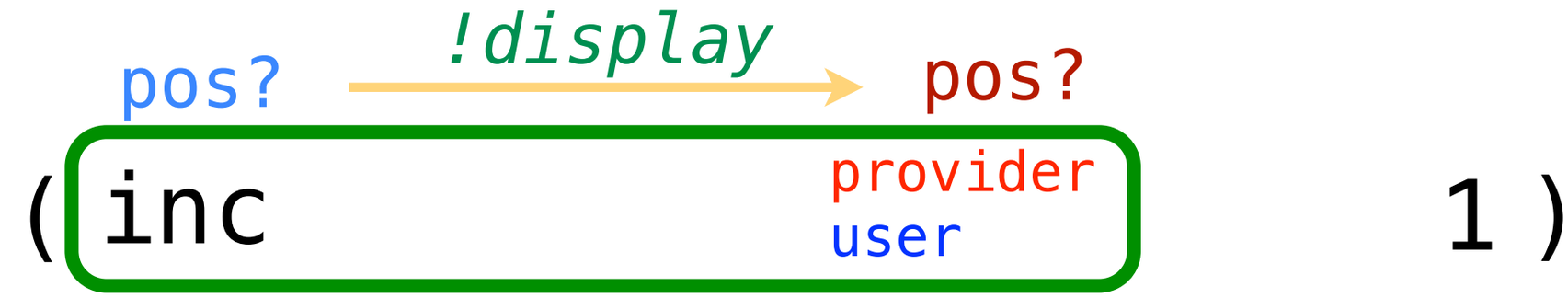


Domain

```
(if(pre? arg)
  (computation)
  (blame `provider)))
```


23

Implementation Overview



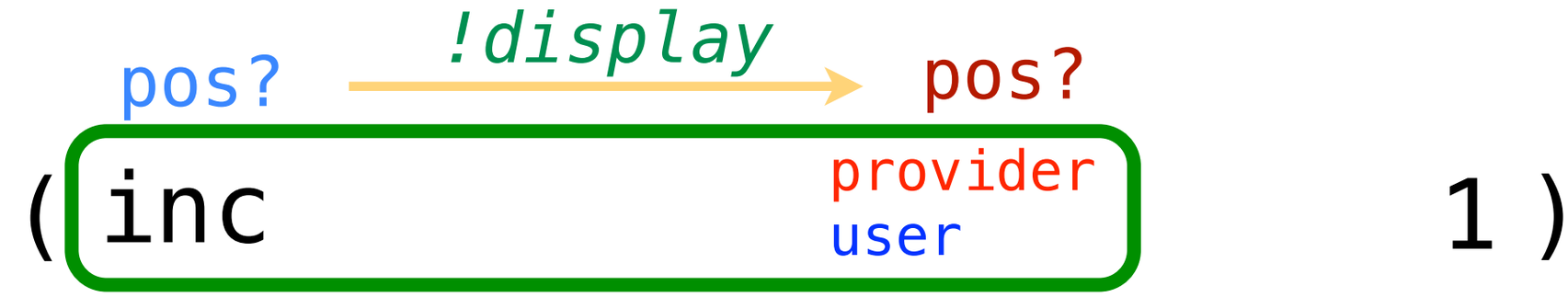
Domain

```
(if(pre? arg)
  (computation)
  (blame `provider)))
```

Range

```
  (if(post? result))
     result
    (blame `user))
```

Implementation Overview



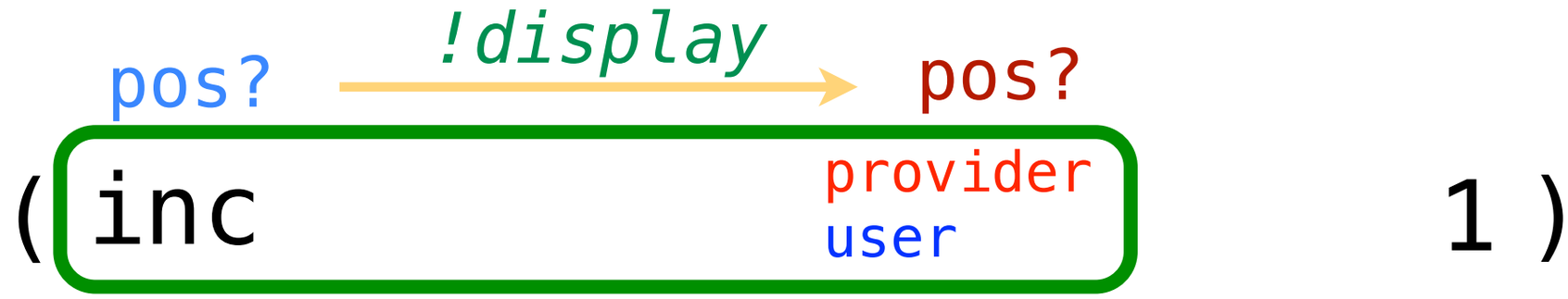
Domain Predicate

```
(if(pre? arg)
  (computation)
  (blame `provider)))
```

Range Predicate

```
  (if(post? result))
     result
    (blame `user))
```

Implementation Overview



Domain Predicate

```
(if(pre? arg)
  (computation)
  (blame `provider)))
```

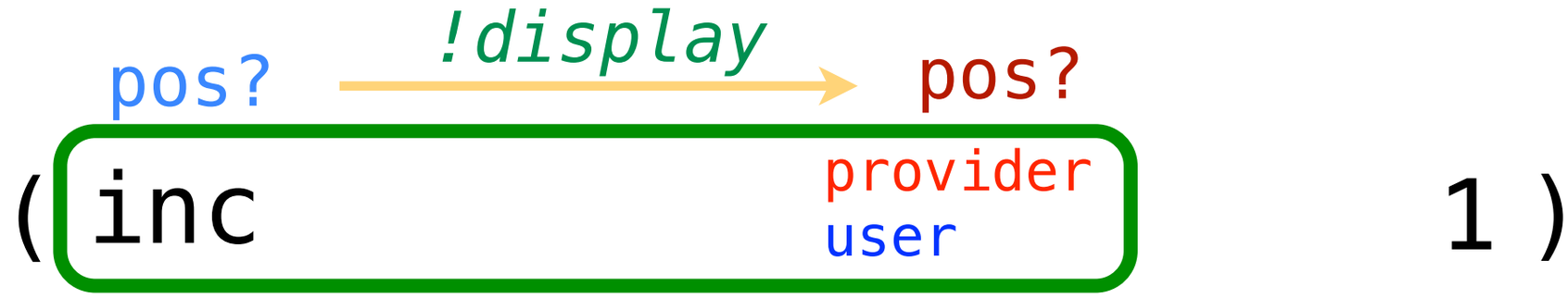
Computation

```
(deploy-fluid
  (call display)
  (blame-adv `user)
 (inc arg))
```

Range Predicate

```
  (if(post? result))
     result
    (blame `user))
```

Implementation Overview



Domain Predicate

```
(if(pre? arg)
  (computation)
  (blame `provider)))
```

Computation Dynamic aspect

```
(deploy-fluid
  (call display)
  (blame-adv `user)
 (inc arg))
```

Range Predicate

```
(if(post? result))
    result
  (blame `user))
```

Status

- Implemented in Racket
- using LAScheme
- integrated with Racket's contract system

24

Status

- Implemented in Racket
- using LAScheme
- integrated with Racket's contract system

24

Applications

25

Applications

- Existing computational contracts in disguise
  - QoS contracts, access permission contracts, ...

25

Applications

- Existing computational contracts in disguise
  - QoS contracts, access permission contracts, ...
- Side-effect free contract for futures (Racket)

25

Applications

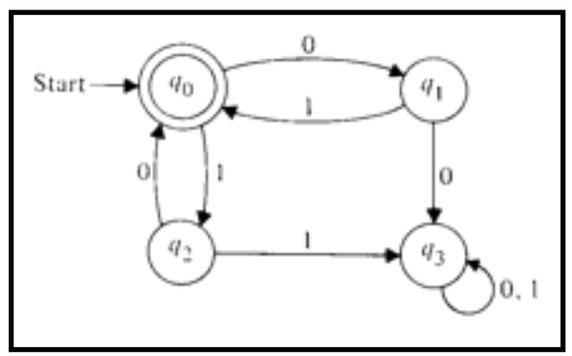
- Existing computational contracts in disguise
  - QoS contracts, access permission contracts, ...
- Side-effect free contract for futures (Racket)
- Protocols
  - different kinds of protocols [Beckman+11]: initialization, deactivation, redundant operation, ...
  - looking into Racket's libraries

25

Computational Contracts

Domain ⟶ Range

Computational Contracts



26

Temporal Higher-Order Contracts

```
(define (free x) 2)
(define (malloc) (free 2) 2)
(provide/contract
 [malloc (monitor/c memmon 'malloc (-> number?))]
 [free   (monitor/c memmon 'free (-> number? void))])
```

```
(define allocated (make-weak-hasheq))
(define memmon
  (match-lambda
    [(monitor:return 'malloc _ _ _ _ _ _ (list addr))
     (begin
       (display "Malloc called \n")
       (hash-set! allocated addr #t))
     #t]
    [(monitor:call 'free _ _ _ _ _ (list addr))
     (begin (display "calling free")
            (hash-has-key? allocated addr))]
    [(monitor:return 'free _ _ _ _ _ (list addr) _)
     (hash-remove! allocated addr)
     #t]
    [_
     #t]))
```

Tim Disney, Cormac Flanagan, and Jay McCarthy.

“Temporal Higher-Order Contracts".

International Conference on Functional Programming, 2011.

27

Temporal Higher-Order Contracts

```
(define (free x) 2)
(define (malloc) (free 2) 2)
(provide/contract
 [malloc (monitor/c memmon 'malloc (-> number?))]
 [free   (monitor/c memmon 'free (-> number? void))])
```

```
(define allocated (make-weak-hasheq))
(define memmon
  (match-lambda
    [(monitor:return 'malloc _ _ _ _ _ _ (list addr))
     (begin
       (display "Malloc called \n")
       (hash-set! allocated addr #t))
     #t]
    [(monitor:call 'free _ _ _ _ _ (list addr))
     (begin (display "calling free")
            (hash-has-key? allocated addr))]
    [(monitor:return 'free _ _ _ _ _ (list addr) _)
     (hash-remove! allocated addr)
     #t]
    [_
     #t]))
```

Tim Disney, Cormac Flanagan, and Jay McCarthy.

“Temporal Higher-Order Contracts".

International Conference on Functional Programming, 2011.

27

Temporal Higher-Order Contracts

```
(define (free x) 2)
(define (malloc) (free 2) 2)
(provide/contract
 [malloc (monitor/c memmon 'malloc (-> number?))]
 [free   (monitor/c memmon 'free (-> number? void))])
```

```
(define allocated (make-weak-hasheq))
(define memmon
  (match-lambda
    [(monitor:return 'malloc _ _ _ _ _ _ (list addr))
     (begin
       (display "Malloc called \n")
       (hash-set! allocated addr #t))
     #t]
    [(monitor:call 'free _ _ _ _ _ (list addr))
     (begin (display "calling free")
            (hash-has-key? allocated addr))]
    [(monitor:return 'free _ _ _ _ _ (list addr) _)
     (hash-remove! allocated addr)
     #t]
    [_
     #t]))
```

```
Welcome to DrRacket, version 5.1.3 [3m].
Language: racket [custom]; memory limit: 128 MB.
> (malloc)
Malloc called
2
>
```

Tim Disney, Cormac Flanagan, and Jay McCarthy.

“Temporal Higher-Order Contracts".

International Conference on Functional Programming, 2011.

27