Computational Contracts

Christophe Scholliers
Wolfgang De Meuter
Software Languages Lab
Vrije Universiteit Brussel

Éric Tanter
PLEIAD Lab
University of Chile
Contracts

Agreement

Exchanged Values
Overview

Domain \[\rightarrow\] Computation \[\rightarrow\] Range

Pre/Post Contracts
Overview

Domain -> Computation -> Range

Pre/Post Contracts
Overview

A computational contract verifies a (sequence of) events during the execution of a contracted entity.
Contracts in Racket
Pre/Post Contracts

```
(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)

  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
```
Pre/Post Contracts

(define (sqrt x)
  ...)
(provide/contract
  [sqrt (-> positive? positive?)])
Higher-Order Contracts

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

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

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

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

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

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

```scheme
(define (map-pos f l)
  (map f l))
(provide/contract
  [map-pos (->
    (-> positive? positive?)
    (listof positive?))])
```
Higher-Order Contracts

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

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

(provider.rkt)

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

(provide/contract
  [map-pos (->
    (-> positive? positive?)
    (listof positive?)
    (listof positive?))])
Higher-Order Contracts

\[
\begin{align*}
&\text{(define (map-pos f l)} \nonumber \\
&\quad \text{(map f l))} 
\end{align*}
\]

\[
\begin{align*}
\text{(provide/contract} \\
&\quad [\text{map-pos} \rightarrow \\
&\quad \text{(\rightarrow positive? positive?)} \\
&\quad \text{(listof positive?)})] 
\end{align*}
\]

provider.rkt
Higher-Order Contracts

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

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

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

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?))
There is no simple predicate that can verify the contract defined over f.
There is no simple predicate that can verify the contract defined over \( f \).
Computational Contracts
Computational Contracts

Safety
Prohibit contract

Liveness
Promise contract
Computational Contracts

Safety
Prohibit contract

Liveness
Promise contract
**Prohibit Contract**

*Computational Contracts over a single event*

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

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

*Computational Contracts over a single event*

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

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

Computational Contracts over a single event

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

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

Computational Contracts over a single event

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

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

Computational Contracts over a single event

\[
\begin{align*}
&\text{(define (map-silent f l)} \\
&\hspace{1cm} (\text{map f l}))
\end{align*}
\]

\[
\begin{align*}
&\text{(provide/contract [map-silent (->)} \\
&\hspace{1cm} (\text{prohibit/c (call display)}) \\
&\hspace{1.5cm} (\text{listof any?}) \\
&\hspace{2.5cm} (\text{listof any?}))])
\end{align*}
\]
Prohibit Contract

Computational Contracts over a single event

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

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

*Computational Contracts over a single event*

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

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

Computational Contracts over a single event

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

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

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

*Computational Contracts over a single event*

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

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

Computational Contracts over a single event

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

Prohibit Contract

Promise Contract

Sequence of Events
Computational Contracts

Sequence of Events

Prohibit Contract

Promise Contract
Sequence of Events

Computational Contracts

Promise:
A file that is opened will eventually be closed
Sequence of Events

Computational Contracts

Promise:
A file that is opened will eventually be closed

file-protocol

(called open-input-file)

closed

open

(called close-input-port)
Sequence of Events

Computational Contracts

Promise:
A file that is opened will eventually be closed

(file-protocol)

end state SHOULD be reached
Sequence of Events
Computational Contracts

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

(get-content bad-reader)
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

(user.rkt)
Sequence of Events

Computational Contracts

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

(provider.rkt)
Stateful Monitors

Computational Contracts

Volume

Start

End

Time

Max
Stateful Monitors

Computational Contracts

sound-monitor

Check volume < 100

(set! volume (+ volume amount))

(set! volume (- volume amount))

(up amount)

(down amount)
Stateful Monitors

Computational Contracts

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

(map-silent f 1)
Computational Contracts

(map-silent f 1)

(map-silent f 1)

(map-silent f 1)
Contract Representation

(provide/contract
  [map-silent (->
    (prohibit/c (call display))
    (listof any?)
    (listof any?)))")
Contract Representation

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

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

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

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

map-silent user provider
\[ \text{pos?} \xrightarrow{\!display} \text{pos?} \times \text{list-int?} \rightarrow \text{list-int?} \]

\((\text{map-silent user provider inc} \ '(1\ 2))\)
\[
\text{pos?} \xrightarrow{\text{ inc \ '}(1 \ 2)} \text{pos?} \times \text{list-int?} \rightarrow \text{list-int?}
\]

\[
\text{map-silent} \quad \text{user} \quad \text{provider}
\]

\[
\text{pos?} \xrightarrow{\text{ inc \ '}(1 \ 2)} \text{pos?}
\]

\[
\text{map-silent} \quad \text{provider} \quad \text{user}
\]
(map-silent (inc '(1 2))

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

(inc provider user 1)
Implementation Overview

(pos? \rightarrow !\text{display} \rightarrow \text{pos})

(\text{inc provider user})
Implementation Overview

Domain

\[
\begin{align*}
&\text{if} (\text{pre? arg}) \\
&\quad (\text{computation}) \\
&\quad (\text{blame } \text{`provider}))
\end{align*}
\]
Implementation Overview

Domain

\[(\text{if}(\text{pre? } \text{arg})\ \ (\text{computation})
\ \ \ (\text{blame} \ \ `\text{provider})))\]

Range

\[(\text{if}(\text{post? } \text{result}))\]
\nresult
\n(\text{blame} \ \ `\text{user})\]
Implementation Overview

Domain Predicate

\[
\text{if}\ (\text{pre? arg})
\]
\[
\ (\text{computation})
\]
\[
\ (\text{blame } \text{`provider}))
\]

Range Predicate

\[
\text{if}\ (\text{post? result}))
\]
\[
\text{result}
\]
\[
\ (\text{blame } \text{`user}))
\]
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   Computation   Dynamic aspect

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

(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
Status

• Implemented in Racket
  • using LAScheme
  • integrated with Racket’s contract system
Applications
Applications

• Existing computational contracts in disguise
  • QoS contracts, access permission contracts, ...
Applications

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

Domain  
Computational Contracts  
Range

Start  
$q_0$  
$q_1$  
$q_2$  
$q_3$

0 1 0 1

0, 1
Temporal Higher-Order Contracts

Tim Disney, Cormac Flanagan, and Jay McCarthy.

“Temporal Higher-Order Contracts”.

Temporal Higher-Order Contracts

Tim Disney, Cormac Flanagan, and Jay McCarthy.

“Temporal Higher-Order Contracts".

Temporal Higher-Order Contracts

Tim Disney, Cormac Flanagan, and Jay McCarthy.

“Temporal Higher-Order Contracts".