

Contracts in a World of Uncertainty

Michael Hansen, Amal Ahmed, Amr Sabry

PL-WONKS Talk

April 8, 2011



Assertions to Contracts

```
def sqrt(x)
  if x > 0
    ...
  else
    error
```

```
def pos(x)
  x > 0

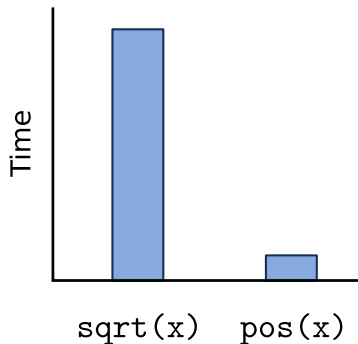
@contract(pos, x)
def sqrt(x)
  ...
```

Three Examples

1. Complexity disparity
2. “Infinite” domain
3. Uncertain inputs

Complexity Disparity (Example 1)

```
sqrt(x)  $\mapsto$  if pos(x)  
                 sqrt'(x)  
               else  
                 error
```



Complexity Disparity (Example 1)

```
def sorted(lst)
  for i in length(lst)
    ...
```

$O(N)$

```
def search(lst)
  ...
```

$O(\log N)$

Complexity Disparity (Example 1)

```
def sorted(lst)
  for i in length(lst)
    ...
```

$O(N)$

```
@contract(sorted, lst)
def search(lst)
  ...
```

$O(N)$

Complexity Disparity (Example 1)

Accept Risk

Only check a random subset of list.

```
def sorted_p(lst, N)
    for n in range(N)
        i = random(length(lst) - 2)
        if lst[i] > lst[i + 1]
            return False
    return True
```

Infinite Domain (Example 2)

Description

A function `make-fp` maps a real-valued function `f` onto another real-valued function `fp`.

Post-Condition

The slope of `f` at `x` is within δ of the value of `fp` at `x`.

Infinite Domain (Example 2)

Possible Solutions

- ▶ Check $\text{fp}(x)$ for all $x \in \mathbb{R}$
 - ▶ Technically finite!
- ▶ Verify post-condition for each call to fp
 - ▶ Save time with constraints, memoization, etc.

```
def make_fp(f,  $\delta$ )  
  let fp = ... in  
     $\lambda x$ . let ans = fp(x) in  
          if abs( ans - slope(f, x) ) <=  $\delta$   
            ans  
          else  
            error
```

Infinite Domain (Example 2)

Accept Risk

Only check a random subset of \mathbb{R} .

```
def make_fp(f,  $\delta$ , N)
  let fp = ... in
  for n in range(N)
    x = random()
    if abs( fp(x) - slope(f, x) ) >  $\delta$ 
      error
  return fp
```

Uncertain Inputs (Example 3)

RSA Key Generation

p, q must be prime. Checking primeness is expensive ☹

Accept Risk

Miller-Rabin algorithm.

```
# Generates prime with  $1 - 2^{-c}$  probability
def get_prime(c)
    ...
def gen_keys()
    let p = get_prime(),
        q = get_prime() in
    ...
    decode(encode(m, pub_key),
           priv_key) == m
```

Types $\tau ::= \text{num} \mid \text{bool} \mid \tau \rightarrow \tau$

Terms $e ::= v \mid x \mid e e \mid e \oplus e \mid \text{zero?}(e) \mid \text{if } e e e$

Values $v ::= \mathbb{N} \mid \mathbf{tt} \mid \mathbf{ff} \mid \lambda x^\tau. e$

Operators $\oplus ::= + \mid - \mid \wedge \mid \vee \mid <$

Contract PCF (CPCF)

Types $\tau ::= \text{num} \mid \text{bool} \mid \tau \rightarrow \tau \mid \text{con}(\tau)$

Terms $e ::= v \mid x \mid e e \mid e \oplus e \mid \text{zero?}(e) \mid \text{if } e e e$
 $\mid \text{mon}^l(\kappa e) \mid \text{error}(l v)$

Values $v ::= \mathbb{N} \mid \mathbf{tt} \mid \mathbf{ff} \mid \lambda x^\tau . e$

Operators $\oplus ::= + \mid - \mid \wedge \mid \vee \mid <$

Contracts $\kappa ::= \text{flat}(e)$

Deterministic Contracts (Definitely Yes)

```
mon("c1", flat( $\lambda x. x > 0$ ), 10)  
   $\mapsto$  if (10 > 0) 10 else error("c1", 10)  
   $\mapsto$  10
```

```
mon("c2", flat( $\lambda x. x > 0$ ), -1)  
   $\mapsto$  if (-1 > 0) -1 else error("c2", -1)  
   $\mapsto$  error("c2", -1)
```

Probabilistic Contracts (Maybe Yes)

```
def sorted_p(lst, N)
  for n in range(N)
    i = random(0, length(lst) - 2)
    if lst[i] > lst[i + 1]
      return False
  return True
```

Probabilistic Contracts (Maybe Yes)

```
mon("c1", flat( $\lambda$ x. sorted_p(x, 1)),  
    [3, 2, 1])
```

```
 $\xrightarrow{*}$  error("c1", [3, 2, 1])
```

```
mon("c2", flat( $\lambda$ x. sorted_p(x, 1)),  
    [2, 3, 1])
```

```
 $\xrightarrow{*}$  error("c2", [2, 3, 1])
```

```
 $\xrightarrow{*}$  [2, 3, 1]
```


Probabilistic CPCF (PCPCF)

Types $\tau ::= \text{num} \mid \text{bool} \mid \tau \rightarrow \tau \mid \text{con}(\tau)$

Terms $e ::= v^A \mid x \mid e e \mid e \oplus e \mid \text{zero?}(e) \mid \text{if } e e e$
 $\mid \text{mon}^l(\kappa e) \mid \text{error}(l v)$
 $\mid \text{error}(l v \bar{l}) \mid \text{random}(e e)$

Values $v ::= \mathbb{N} \mid \mathbf{tt} \mid \mathbf{ff} \mid \lambda x^\tau. e$

Operators $\oplus ::= + \mid - \mid \wedge \mid \vee \mid <$

Contracts $\kappa ::= \text{flat}_D(e) \mid \text{flat}_P(e)$

Operators and Pairs

$$\frac{e_1 \hookrightarrow v_1^{A_1} \quad e_2 \hookrightarrow v_2^{A_2}}{E[e_1 \oplus e_2] \hookrightarrow v_3^{A_1 \cup A_2}} \quad \frac{e_1 \hookrightarrow v_1^{A_1}}{E[\text{zero?}(e_1)] \hookrightarrow v_2^{A_1}}$$

$$\frac{e_1 \hookrightarrow v_1^{A_1} \quad e_2 \hookrightarrow v_2^{A_2}}{E[\text{cons}(e_1 \ e_2)] \hookrightarrow \text{pair}(v_1^{A_1} \ v_2^{A_2})^{A_1 \cup A_2}}$$

$$\frac{e_1 \hookrightarrow \text{pair}(v_1^{A_1} \ v_2^{A_2})^{A_3}}{E[\text{car}(e_1)] \hookrightarrow v_1^{A_1 \cup A_3}}$$

Example 1 Revisited

```
mon("c1", flatP( $\lambda$ x. sorted_p(x)), [2 3 1])  
   $\mapsto$  if (sorted_p([2 3 1])) [2 3 1]{c1}  
        else error("c1", [2 3 1])  
  
   $\mapsto$  [2 3 1]{c1}  
...  
min = car([2 3 1]{c1})  
   $\xrightarrow{*}$  min = 2{c1}  
...  
mon("c2", ..., min{c1})  
   $\xrightarrow{*}$  error("c2", 2, {"c1"})
```

Functions and Control Flow

$$\frac{e_2 \hookrightarrow v_2^{A_2} \quad e_1[x := v_2^{A_2}] \hookrightarrow v_3^{A_3}}{E[(\lambda x^\tau. e_1)^{A_1} e_2] \hookrightarrow v_3^{A_1 \cup A_3}}$$

$$\frac{e_1 \hookrightarrow v_1^{A_1}}{E[\text{if } (e_1) e_2 e_3] \hookrightarrow E[e_2^{A_1}] \text{ if } v_1 \text{ else } E[e_3^{A_1}]}$$

$$\frac{e_1 \hookrightarrow \{v_i^{A_i}\}^{A_1} \quad e_2[x := \{v_i^{A_i}\}^{A_1}] \xrightarrow{*} v_2^{A_2}}{E[\text{for } (x \text{ in } e_1) e_2] \hookrightarrow v_2^{A_1 \cup \{A_i\} \cup A_2}}$$

Example 2 Revisited

$$\text{make_fp}(\dots) \xrightarrow{*} \text{fp}^{\{\text{make_fp}\}}(x) \xrightarrow{*} v^{\{\text{make_fp}\}}$$

```
def make_fp(f,  $\delta$ , N)
  let fp = ... in

  def post_cond(ignore)
    for n in range(N)
      x = random()
      if abs( fp(x) - slope(f, x) ) >  $\delta$ 
        return False
    return True

  return mon("make_fp",
            flatP(post_cond), fp)
```

Flat Contracts

$$\frac{e_1 \hookrightarrow v_1^{A_1} \quad e_2 \hookrightarrow v_2^{A_2} \quad (v_1^{A_1} \ v_2^{A_2}) \hookrightarrow v_3^{A_3}}{E[\text{mon}'(\text{flat}_D(e_1) \ e_2)] \hookrightarrow E[\text{if}(v_3^{A_3}) \ v_2^{A_2 \cup A_3} \ \text{error}(! \ v_2 \ \{A_2 \cup A_3\})]}$$

$$\frac{e_1 \hookrightarrow v_1^{A_1} \quad e_2 \hookrightarrow v_2^{A_2} \quad (v_1^{A_1} \ v_2^{A_2}) \hookrightarrow v_3^{A_3}}{E[\text{mon}'(\text{flat}_P(e_1) \ e_2)] \hookrightarrow E[\text{if}(v_3^{A_3}) \ v_2^{A_2 \cup A_3 \cup !} \ \text{error}(! \ v_2 \ \{A_2 \cup A_3\})]}$$

Example 3 Revisited

```
# Generates prime with  $1 - 2^{-c}$  probability
def get_prime(c)
  return mon("get_prime",
            flatP( $\lambda x.$  is_prime(x)), ...)

def gen_keys()
  let p = get_prime(),
      q = get_prime() in
  ... p{get_prime} ...
  ... q{get_prime} ...
  decode(encode(m, pub_key),
         priv_key) == m
```

Bad Values

For every test contract κ

- ▶ Define B_κ as the set of values which cause κ to signal an error
- ▶ For κ to be a “proper” test contract, B_κ must not depend on the program state

Influence

A value v influences v' if changing v can also change v'

Invalid Values

A value v is invalid if

- ▶ v has passed some κ and $v \in B_\kappa$ or
- ▶ v has been influenced by v' and v' is invalid

Influence

A value v influences v' if changing v can also change v'

Invalid Values

A value v is invalid if

- ▶ v has passed some κ and $v \in B_{\kappa}$ or
 - ▶ v has been influenced by v' and v' is invalid
-

Statement

If a contract κ fails on v^A , then either

- ▶ $v \in B_{\kappa}$ or
- ▶ v is invalid with respect to some $\kappa' \in A$

- ▶ Need a static approximation for tracking
 - ▶ Dependency correctness
- ▶ Higher-order contracts
 - ▶ Should they behave differently?
- ▶ Different probabilities for $flatp$
 - ▶ High, low, exact?
- ▶ Recovery from an error
 - ▶ Find most likely culprit
 - ▶ Restart with new values

Questions?