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
```

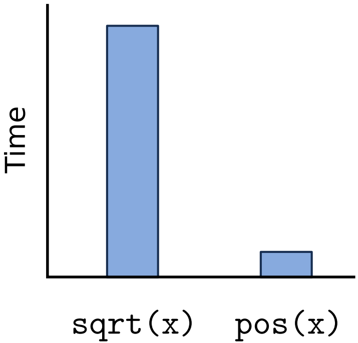
```
def pos(x)
  x > 0

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

Three Examples

1. Complexity disparity
2. "Infinite" domain
3. Uncertain inputs

Complexity Disparity (Example 1)

```
                  if pos(x)
sqrt(x)  ⟼          sqrt′(x)
                    else
                      error
```



Complexity Disparity (Example 1)

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

def search(lst)
  ...
```

$O(N)$

$O(\log N)$

Complexity Disparity (Example 1)

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

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

$O(N)$

$O(N)$

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

```
def make_fp(f, δ)
  let fp = ... in
    λx. let ans = fp(x) in
          if abs( ans - slope(f, x) ) <= δ
            ans
          else
            error
```

Infinite Domain (Example 2)

Accept Risk

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

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

Uncertain Inputs (Example 3)

RSA Key Generation

p, q must be prime. Checking primeness is expensive $\ddot{\frown}$

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
```

Standard PCF

$$
\begin{array}{llll}
\textbf{Types} & \tau & ::= & \mathrm{num} \mid \mathrm{bool} \mid \tau \to \tau \\
\textbf{Terms} & e & ::= & v \mid x \mid e\ e \mid e \oplus e \mid \mathrm{zero?}(e) \mid \mathrm{if}\ e\ e\ e \\
\textbf{Values} & v & ::= & \mathbb{N} \mid \mathbf{tt} \mid \mathbf{ff} \mid \lambda x^{\tau}.e \\
\textbf{Operators} & \oplus & ::= & + \mid - \mid \wedge \mid \vee \mid <
\end{array}
$$

Contract PCF (CPCF)

Types	τ	$::=$	$\mathrm{num} \mid \mathrm{bool} \mid \tau \to \tau \mid \mathrm{con}(\tau)$
Terms	e	$::=$	$v \mid x \mid e\ e \mid e \oplus e \mid \mathrm{zero?}(e) \mid \mathrm{if}\ e\ e\ e$
		$\mid$	$\mathrm{mon}^{l}(\kappa\ e) \mid \mathrm{error}(l\ v)$
Values	v	$::=$	$\mathbb{N} \mid \mathbf{tt} \mid \mathbf{ff} \mid \lambda x^{\tau}.e$
Operators	$\oplus$	$::=$	$+ \mid - \mid \wedge \mid \vee \mid <$
Contracts	κ	$::=$	$\mathrm{flat}(e)$

Deterministic Contracts (Definitely Yes)

```
mon("c1", flat(λx. x > 0), 10)
  ⟼ if (10 > 0) 10 else error("c1", 10)
  ⟼ 10

mon("c2", flat(λx. x > 0), -1)
  ⟼ if (-1 > 0) -1 else error("c2", -1)
  ⟼ 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(λx. sorted_p(x, 1)),
   [3, 2, 1])
```

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

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

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

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

Probabilistic CPCF (PCPCF)

$$
\begin{array}{llll}
\textbf{Types} & \tau & ::= & \mathrm{num} \mid \mathrm{bool} \mid \tau \rightarrow \tau \mid \mathrm{con}(\tau) \\
\textbf{Terms} & e & ::= & v^{A} \mid x \mid e\ e \mid e \oplus e \mid \mathrm{zero?}(e) \mid \mathrm{if}\ e\ e\ e \\
 & & \mid & \mathrm{mon}^{l}(\kappa\ e) \mid \mathrm{error}(l\ v) \\
 & & \mid & \mathrm{error}(l\ v\ \bar{l}) \mid \mathrm{random}(e\ e) \\
\textbf{Values} & v & ::= & \mathbb{N} \mid \mathbf{tt} \mid \mathbf{ff} \mid \lambda x^{\tau}.e \\
\textbf{Operators} & \oplus & ::= & + \mid - \mid \wedge \mid \vee \mid < \\
\textbf{Contracts} & \kappa & ::= & \mathrm{flat}_{D}(e) \mid \mathrm{flat}_{P}(e)
\end{array}
$$

Tracking Attestors (1/3)

Operators and Pairs

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

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

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

Example 1 Revisited

```
mon("c1", flatP(λx. sorted_p(x)), [2 3 1])
   ⟼ if (sorted_p([2 3 1])) [2 3 1]^{c1}
       else error("c1", [2 3 1])

   ⟼ [2 3 1]^{c1}
...
min = car([2 3 1]^{c1})
   ⟼* min = 2^{c1}
...
mon("c2", ..., min^{c1})
   ⟼* error("c2", 2, {"c1"})
```

Tracking Attestors (2/3)

Functions and Control Flow

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

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

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

Example 2 Revisited

$$\texttt{make\_fp(...)} \overset{*}{\longmapsto} \texttt{fp}^{\{\texttt{make\_fp}\}}\texttt{(x)} \overset{*}{\longmapsto} \texttt{v}^{\{\texttt{make\_fp}\}}$$

```
def make_fp(f, δ, N)
  let fp = ... in

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

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

Tracking Attestors (3/3)

Flat Contracts

$$\frac{e_1 \hookrightarrow {v_1}^{A_1} \qquad e_2 \hookrightarrow {v_2}^{A_2} \qquad ({v_1}^{A_1}\ {v_2}^{A_2}) \hookrightarrow {v_3}^{A_3}}{E\,[\mathrm{mon}^{l}(\mathrm{flat}_D(e_1)\ e_2)] \hookrightarrow E\,[\mathrm{if}\ ({v_3}^{A_3})\ {v_2}^{A_2 \cup A_3}\ \mathrm{error}(l\ v_2\ \{A_2 \cup A_3\})]}$$

$$\frac{e_1 \hookrightarrow {v_1}^{A_1} \qquad e_2 \hookrightarrow {v_2}^{A_2} \qquad ({v_1}^{A_1}\ {v_2}^{A_2}) \hookrightarrow {v_3}^{A_3}}{E\,[\mathrm{mon}^{l}(\mathrm{flat}_P(e_1)\ e_2)] \hookrightarrow E\,[\mathrm{if}\ ({v_3}^{A_3})\ {v_2}^{A_2 \cup A_3 \cup l}\ \mathrm{error}(l\ v_2\ \{A_2 \cup A_3\})]}$$

Example 3 Revisited

```
# Generates prime with 1−2^−c probability
def get_prime(c)
  return mon("get_prime",
             flatP(λx. 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
```

Formal Statement

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

Formal Statement

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 $\kappa' \in A$

Future Work

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

Questions?