Bounded Refinement Types

Niki Vazou, Alexander Bakst, Ranjit Jhala

UC San Diego
Given: Library

\[ \text{incr2} \ x = x + 2 \]
Goal: Verify Client

```python
p = incr2 40
assert (p == 42)

n = incr2 (-50)
assert (n < 0)
```
Automatic Verification via SMT
*From Decidable Logic
Specify

\[
\text{incr2 } x = x + 2
\]

Refinement Type

\[
x: \text{Int} \rightarrow \{r: \text{Int} \mid r = x+2\}
\]
... & Verify

\[ p = \text{incr2} \ 40 \]

\[ \text{assert} \ (p == 42) \]

Via Type

\[ p = 40 + 2 \]
... & Verify

\[
p = \text{incr2} \ 40
\]

\[
\text{assert} \ (p \ == \ 42)
\]

SMT: Is Valid? Yes!

\[
p = 40 + 2 \Rightarrow p = 42
\]
Automatic Verification via SMT
I ♥ SMT

SLAM, BLAST,..
ESC, BOOGIE,..
DART, KLEE...
Liquid, F*...
I have a problem with SMT

Decidable Logic

Quantifier-Free, First-Order Logic

Specifications Not “Modular”
Goal: Verify Client

\[ f = \text{compose} \left( \lambda a \rightarrow a + 1 \right) \left( \lambda b \rightarrow b + 1 \right) \]

\[ p = f \ 40 \]

\textcolor{red}{\text{assert} \ (p == 42)}
Given: Library

\[
\text{compose } f \ g \ x = f \ (g \ x)
\]

Specification?
Given: Library

First-Order Specification

```
compose :: (y:b -> {z:c | z = y+1})
  -> (x:a -> {y:b | y = x+1})
  -> (x:a -> {z:c | z = x+2})
```

`compose f g x = f (g x)`

Not Modular
Given: Library

compose \( f \, g \, x \) \( \equiv \) \( f \, (g \, x) \)

Higher-Order Specification

ensures \( \forall f \, g \, x. \ r = f \, (g \, x) \)

Not Decidable
Problem

Automatic Verification vs. Modular Specifications

\[ y = x + 1 \quad \quad \quad \quad r = f(g(x)) \]
Goal

**Automatic Verification of Modular Specifications**
Key Idea

Observe: Refinements are Relations

\[ x : \text{Int} \rightarrow \{ y : \text{Int} \mid y = x + 2 \} \]

Relation between input \( x \) and output \( y \)
Key Idea

Observe: Refinements are Relations

Step 1: Abstract Relations

\[ x: \text{Int} \rightarrow \{ y: \text{Int} \mid y = x + 2 \} \]
Key Idea

Observe: Refinements are Relations

Step 1: Abstract Relations

$$x: \text{Int} \rightarrow \{y: \text{Int} \mid \mathsf{p} \ x \ y\}$$

In SMT $\mathsf{p}$ is “Uninterpreted Function”
Key Idea

Step 1: Abstract Relations

\[ x: \text{Int} \rightarrow \{ y: \text{Int} | p \times x \times y \} \]

In SMT, \( p \) is “Uninterpreted Function”

\[ \forall x, \overline{y}. \overline{x} = \overline{y} \Rightarrow (p \overline{x}) = (p \overline{y}) \]
Key Idea

Observe: Refinements are Relations

Step 1: Abstract Relations

Step 2: Constrain Relations

To specify properties of abstract
Step 1: Abstract Relations

```plaintext
compose :: forall a, b, c.
  f:(y:b-> {z:c|z = y+1})
  g:(x:a-> {y:b|y = x+1})
  (x:a-> {z:c|z = x+2})

compose f g x = let y = g x in
  let z = f y in z
```
**Step 1: Abstract Relations**

\[ \text{compose :: forall } a, b, c. \]
\[ f: (y : b \rightarrow \{ z : c | z = y + 1 \}) \]
\[ \rightarrow g: (x : a \rightarrow \{ y : b | y = x + 1 \}) \]
\[ \rightarrow (x : a \rightarrow \{ z : c | z = x + 2 \}) \]

\[ \text{compose } f \ g \ x = \text{ let } y = g \ x \ \text{ in } \]
\[ \text{ let } z = f \ y \ \text{ in } z \]
Step 1: Abstract Relations

\[
\text{compose} :: \forall a, b, c, p, q, r. \\
f : (y : b \rightarrow \{ z : c | p \ y \ z \}) \\
\rightarrow \ g : (x : a \rightarrow \{ y : b | q \ x \ y \}) \\
\rightarrow (x : a \rightarrow \{ z : c | r \ x \ z \}) \\
\text{compose } f \ g \ x = \text{let } y = g \ x \ \text{in} \\
\quad \text{let } z = f \ y \ \text{in} \ z
\]
Wait! Is this specification correct?

```
compose :: forall p, q, r.
  f:(y:b-> {z:c|p y z})
  -> g:(x:a-> {y:b|q x y})
  -> (x:a-> {z:c|r x z})
compose f g x = let y = g x in
               let z = f y in z
```
Wait! Is this specification correct?

Assume

Prove
**Wait! Is this specification correct?**

```plaintext
compose :: forall p, q, r.
  f:(y:b-> {z:c|p y z})
  -> g:(x:a-> {y:b|q x y})
  -> (x:a-> {z:c|r x z})

compose f g x = let y = g x in
                let z = f y in z
```

**Assume**

**Prove**
Wait! Is this specification correct?

\[
\text{compose} :: \forall p, q, r. \\
\quad f:(y:b\rightarrow \{z:c|p\ y\ z\}) \\
\rightarrow g:(x:a\rightarrow \{y:b|q\ x\ y\}) \\
\rightarrow (x:a\rightarrow \{z:c|r\ x\ z\}) \\
\text{compose}\ f\ g\ x = \text{let}\ y = g\ x\ \text{in} \\
\quad \text{let}\ z = f\ y\ \text{in}\ z
\]
Wait! Is this specification correct?

\[
\text{compose :: forall } p, q, r.
\]

\[
f: (y: b \to \{z: c | p \ y \ z\}) \\
\rightarrow g: (x: a \to \{y: b | q \ x \ y\}) \\
\rightarrow (x: a \to \{z: c | r \ x \ z\})
\]

\[
\text{compose } f \ g \ x = \text{ let } y = g \ x \text{ in} \\
\text{ let } z = f \ y \text{ in } z
\]

Assume

\[
y: \{y: b | q \ x \ y\}
\]

Prove
compose :: forall p, q, r.
  f:(y:b-> \{z:c| p y z\})
  -> g:(x:a-> \{y:b| q x y\})
  -> (x:a-> \{z:c|r x z\})
compose f g x = let y = g x in
              let z = f y in z

Assume
y:{y:b|q x y}
z:{z:c|p y z}

Prove
Wait! Is this specification correct?

\[
\text{compose} :: \forall p, q, r.
\]
\[
f : (y : b \to \{ z : c \mid p \ y \ z \})
\]
\[
\to g : (x : a \to \{ y : b \mid q \ x \ y \})
\]
\[
\to (x : a \to \{ z : c \mid r \ x \ z \})
\]
\[
\text{compose } f \ g \ x = \text{let } y = g \ x \ \text{in}
\]
\[
\text{let } z = f \ y \ \text{in} \ z
\]

Assume

\[
y : \{ y : b \mid q \ x \ y \}
\]
\[
z : \{ z : c \mid p \ y \ z \}
\]

Prove

\[
z : \{ z : c \mid r \ x \ z \}
\]
Wait! Is this specification correct?

```
compose :: forall p, q, r.
f:(y:b-> {z:c|p y z})
    -> g:(x:a-> {y:b|q x y})
    -> (x:a-> {z:c|r x z})
compose f g x = let y = g x in
    let z = f y in z
```

Assume

- `y: {y:b|q x y}`
- `z: {z:c|p y z}`

Prove

- `z: {z:c|r x z}`
Wait! Is this specification correct?

```
compose :: forall p, q, r.
     f:(y:b-> {z:c|p y z})
     -> g:(x:a-> {y:b|q x y})
     -> (x:a-> {z:c|r x z})
compose f g x = let y = g x in
               let z = f y in z
```

Is SMT Valid? No!

\[ q \land p y z \Rightarrow r x z \]

Specification is too general!
Key Idea

Step 1: Abstract Relations

Specification is too general!

Need: $q(x,y) \land p(y,z) \implies r(x,z)$
Key Idea

Step 1: Abstract Relations

Specification is too general!

Step 2: Constrain Relations

Bound: $q \land p \Rightarrow r$
Key Idea

Step 1: Abstract Relations

Specification is too general!

Step 2: Constrain Relations

\textbf{bound Chain} \ p \ q \ r = \ \forall x y z \rightarrow \\
q \ x \ y \land p \ y \ z \Rightarrow r \ x \ z
Step 2: Constrain Relations

bound Chain p q r = \( \forall x y z \rightarrow q x y \land p y z \Rightarrow r x z \)

Type (Bad)

compose :: forall p, q, r.
\( (y:b \rightarrow \{ z:c | p y z \}) \)
\( \rightarrow (x:a \rightarrow \{ y:b | q x y \}) \)
\( \rightarrow (x:a \rightarrow \{ z:c | r x z \}) \)
Step 2: Constrain Relations

bound Chain p q r = \x y z ->
q x y \land p y z \Rightarrow r x z

Type (Fixed)

compose :: (Chain p q r)
=> (y:b-> {z:c|p y z})
-> (x:a-> {y:b|q x y})
-> (x:a-> {z:c|r x z})
**Step 2: Constrain Relations**

```
compose :: (Chain p q r)
  => (y:b-> {z:c|p y z})
  -> (x:a-> {y:b|q x y})
  -> (x:a-> {z:c|r x z})

compose f g x = let y = g x in
                let z = f y in z
```

Assume

<table>
<thead>
<tr>
<th>q x y</th>
</tr>
</thead>
<tbody>
<tr>
<td>p y z</td>
</tr>
</tbody>
</table>

Prove

| r x z |
Step 2: Constrain Relations

compose :: (Chain p q r)
  => (y:b-> {z:c|p y z})
  -> (x:a-> {y:b|q x y})
  -> (x:a-> {z:c|r x z})

compose f g x = let y = g x in
  let z = f y in z

Assume
q x y
p y z
Chain p q r

Prove
r x z
Step 2: Constrain Relations

\[
\text{compose} \quad :: \quad (\text{Chain}\ p\ q\ r)
\Rightarrow (y:b\rightarrow \{z:c\mid p\ y\ z\})
\rightarrow (x:a\rightarrow \{y:b\mid q\ x\ y\})
\rightarrow (x:a\rightarrow \{z:c\mid r\ x\ z\})
\]

\[
\text{compose}\ f\ g\ x\ =\ \text{let}\ y = g\ x\ \text{in}
\quad \text{let}\ z = f\ y\ \text{in}\ z
\]

Is SMT Valid? \ Yes(!) \ 

\[
q\ x\ y
\wedge\ p\ y\ z
\wedge(q\ x\ y\ \wedge\ p\ y\ z \Rightarrow r\ x\ z) \Rightarrow r\ x\ z
\]
Step 2: Constrain Relations

compose :: (Chain p q r)
  => (y:b-> {z:c|p y z})
  -> (x:a-> {y:b|q x y})
  -> (x:a-> {z:c|r x z})

compose f g x = let y = g x in
                let z = f y in z
Step 2: Constrain Relations

compose :: (Chain p q r) => (y:b-> {z:c|p y z}) -> (x:a-> {y:b|q x y}) -> (x:a-> {z:c|r x z})

compose f g x = let y = g x in
               let z = f y in z

... but is Specification Modular?

Next: Let's Verify a Client
Client Verification

\[
\text{compose} :: (\text{Chain } p \ q \ r) \\
\Rightarrow (y:b\to \{z:c|p \ y \ z\}) \\
\rightarrow (x:a\to \{y:b|q \ x \ y\}) \\
\rightarrow (x:a\to \{z:c|r \ x \ z\})
\]

\[
p, \ q ::= \ \lambda x \ z \rightarrow z=x+1 \\
r ::= \ \lambda x \ z \rightarrow z=x+2
\]

\[
\text{incr2} :: x:\text{Int} \rightarrow \{v:\text{Int} \mid v = x+2\} \\
\text{incr2} = \text{compose } (+1) (+1)
\]
Client Verification

\[
\text{bound Chain } p \ q \ r = \forall x \ y \ z \rightarrow q\ x\ y \land p\ y\ z \Rightarrow r\ x\ z
\]

\[
p, q ::= \forall x \ z \rightarrow z = x + 1
\]
\[
r ::= \forall x \ z \rightarrow z = x + 2
\]

\[
\text{incr2} :: x : \text{Int} \rightarrow \{ v : \text{Int} \mid v = x + 2 \}
\]
\[
\text{incr2} = \text{compose} (+1) (+1)
\]
bound \textbf{Chain} = \forall x \ y \ z \rightarrow
y=x+1 \ \land \ \ z=y+1 \implies z=x+2 \ \text{Valid}

\textbf{p, q} ::= \forall x \ z \rightarrow z=x+1
\textbf{r} ::= \forall x \ z \rightarrow z=x+2

\textbf{incr2} ::= x:\text{Int} \rightarrow \{v:\text{Int} \mid v = x+2\}
\text{incr2} = \text{compose} \ (\ +1) \ (\ +1) \ \text{OK}
Key Idea

Observe: Refinements are Relations

Step 1: Abstract Relations @ Lib

Step 2: Constrain Relations @ Lib

Step 3: Instantiate Relations @ Clt
Goal

Automatic Verification of Modular Specifications
Goal
Automatic Verification of Modular Specifications

Key Idea
Constrained Abstract Refinements
Goal
Automatic Verification of Modular Specifications

Key Idea
Constrained Abstract Refinements

Applications
Applications

Higher-Order Functions

Database Schema

Floyd-Hoare Logic

Capability Safe IO Monad
Higher-Order Functions

Examples

compose, foldr, filter, ...
Higher-Order Functions

Examples
compose, foldr, filter, ...
foldr \( f \ b \ (x:xs) = f \ x \ (foldr \ op \ b \ xs) \)
foldr \( f \ b \ [] = b \)
Specification

\[
\begin{align*}
\text{foldr} \ f \ b \ (x:xs) &= f \ x \ (\text{foldr} \ \text{op} \ b \ xs) \\
\text{foldr} \ f \ b \ [] &= b
\end{align*}
\]

\[
\begin{align*}
\text{inv} \ xs \ b &= \text{list } xs \text{ and value } b \\
\text{stp} \ x \ b \ b' &= \text{inputs and output of } f
\end{align*}
\]

\[
\begin{align*}
\text{bound} \ \text{Ind} \ \text{inv} \ \text{stp} &= \{x \ xs \ b \ b'\} \\
\text{inv} \ xs \ b &\Rightarrow \text{stp} \ x \ b \ b' \\
&\Rightarrow \text{inv} \ (x:xs) \ b'
\end{align*}
\]
Specification

\[ \text{bound Ind inv stp} = \lambda x\, xs\, b\, b' \rightarrow \]
\[ \text{inv} \, xs\, b \Rightarrow \text{stp} \, x\, b\, b' \]
\[ \Rightarrow \text{inv} \,(x:\!xs)\, b' \]

Refinement Type

\[ \text{foldr} :: (\text{Ind inv stp}) \]
\[ \Rightarrow (x:\!a \rightarrow b:\!b \rightarrow \{ b':b | \text{stp} \, x\, b\, b' \}) \]
\[ \rightarrow \{ b:b | \text{inv} \,[] \, b \} \rightarrow \text{xs} : [a] \]
\[ \rightarrow \{ v:b | \text{inv} \, xs\, v \} \]
Applications

Higher-Order Functions

Database Schema

Floyd-Hoare Logic

Capability Safe IO Monad
Applications

Higher-Order Functions

Database Schema and Queries

Floyd-Hoare Logic

Capability Safe IO Monad
Database Schema and Queries

Modular Specifications
select, project, join, ...

Abstract Refinements
Describe generic key-value relationships

Bounds
Describe schema disjointness, union,...
Floyd-Hoare Logic (in ST)

Modular Specifications
sequence, if, while, for, ...

Abstract Refinements
Describe generic state assertions

Bounds
Describe Floyd-Hoare Constraints
Applications

Higher-Order Functions

Database Schema

Floyd-Hoare Logic (in ST)

Capability Safe IO Monad
Goal

Automatic Verification of Modular Specifications

Key Idea

Constrained Abstract Refinements

Applications

HOFs, Databases, Floyd-Hoare, Capabilities.
Goal
Automatic Verification of Modular Specifications

Key Idea
Constrained Abstract Refinements

Applications
HOFs, Databases, Floyd-Hoare, Capabilities.

Implementation
Implementation

Liquid Haskell
Liquid Types for Haskell

CUFP Tutorial on Thu!
Implementation

Abstract Refinements desugar to Ghost variables

Bounds desugar to Ghost functions
(à la TypeClass Dictionaries)
Goal
Automatic Verification of Modular Specifications

Key Idea
Constrained Abstract Refinements

Applications
HOFs, Databases, Floyd-Hoare, Capabilities.

Implementation
Ghost variables (à la TypeClass Dictionaries)
Goal
Automatic Verification of Modular Specifications

Key Idea
Constrained Abstract Refinements

Applications
HOFs, Databases, Floyd-Hoare, Capabilities.

Implementation
Ghost variables (à la TypeClass Dictionaries)

Details/FAQ
Details/FAQ

1. How do we prove soundness?
2. How fast is (bounded) type checking?
3. How easy is it so come up with bounds?
4. What are the limitations?
5. What’s so great about decidability?

Questions?/Thank you!
END
$O(n^k)$ calls where $k$ is the number of parameters in the bound and $n$ the number of variables in scope