Type Reconstruction for General Refinement Types

Kenn Knowles
University of California, Santa Cruz

joint work with:
Cormac Flanagan (University of California, Santa Cruz)

March 25, 2007

Assertions / Refinement Types

The role of assertions in axiomatic semantics

- $\texttt{assert}(x > 0)$

Is the role of "general" refinement types in type systems.

- $x \ : \ \{n{:}\texttt{Int} \mid n > 0\}$

Predicates are written in the language itself, and are thus also executable contracts on base types.

[Freeman-Pfenning 91, Hayashi 93, Denney 98, Davies-Pfenning 00, Dunfield 02, Mandelbaum et al 03, Ou et al 04]

Checking Assertions / Refinement Types

$\texttt{assert}(x > 0)$	$\lambda x : \{n\!:\!\texttt{Int} \mid n > 0\}$
$x := x + 1$	
$\texttt{assert}(x > 1)$	$\texttt{let } x_1 : \{n\!:\!\texttt{Int} \mid n > 1\} = x + 1 \texttt{ in}$
$x := x + 1$	
$\texttt{assert}(x > 2)$	$\texttt{let } x_2 : \{n\!:\!\texttt{Int} \mid n > 2\} = x_1 + 1 \texttt{ in } x_2$

Check that $x > 0$ implies $x + 1 > 1$ implies $x + 2 > 2$

Subtyping

$3 \;:\; \{n\!:\!\texttt{Int} \mid n > 0\}$?

$\Updownarrow$

$\{n\!:\!\texttt{Int} \mid n = 3\} <: \{n\!:\!\texttt{Int} \mid n > 0\}$?

$\Updownarrow$

$n : \texttt{Int} \vdash (n = 3) \Rightarrow (n > 0)$?

$\Updownarrow$

$\forall n\!:\!\texttt{Int}.$ if $n = 3 \longrightarrow^* \texttt{true}$ then $n > 0 \longrightarrow^* \texttt{true}$

Undecidable in general.

Dependent types

To express pre- and postconditions

$$//@\texttt{requires}(x \geq 0)$$
$$//@\texttt{ensures}(\texttt{retval} \geq 0 \wedge \texttt{retval} * \texttt{retval} - x \leq 0.001)$$
$$\texttt{double sqrt}(\texttt{double } x)$$

We use dependent function types.

$$\texttt{sqrt} : x : \{r : \texttt{double} \mid r \geq 0\} \rightarrow \{y : \texttt{double} \mid y \geq 0 \wedge y * y - x \leq 0.001\}$$

These are also executable as run-time checks.

[Augustsson 98, Xi-Pfenning 99, Findler-Felleisen 02, Ou et al 04]

Hybrid Type Checking (HTC)

A strategy for enforcing undecidable type systems.

$$e \ : \ T? \quad \longrightarrow \boxed{\text{HTC}} \longrightarrow \begin{cases} \times & \text{(no)} \\ e & \text{(yes)} \\ \langle T \rangle \ e & \text{(maybe)} \end{cases}$$

Crucial idea: *operational* definition of implication links types with run-time checks.

[Flanagan POPL'06, Gronski et al SFP'06, Flanagan et al FOOL'06]

Breakdown

	First-Order Imperative	**Higher-order Functional**
Specs	Assertions Loop Invariants, Pre/Postconditions	Refinement Types Dependent Types
Dynamic	Contracts	Type Casts
Static	Hoare Logic + Theorem Proving	(Hybrid) Type Checking + Theorem Proving
Automatic Inference	Weakest Precond. Strongest Postcond.	??

Type Reconstruction: Defined

Given a program with type variables,

- *Type reconstruction:* Find a replacement of type variables such that the resulting program is well-typed.
- *i.e.* determine if that program is *typeable*.
- Typeability generalizes type checking, so it is definitely undecidable!

Type Reconstruction: Redefined

Given a program with type variables,

- *Type reconstruction (take 2):* Find a replacement of those type variables such that the resulting program is typeable if and only if the original program is.

- Usefully generalizes the traditional definition when type checking is undecidable.

- Equivalent to traditional definition when type checking is decidable.

Basic Reconstruction

Some parts are easy:

- Reconstruction of the underlying simple types unaffected by predicates.

$$\begin{array}{l}\texttt{let}\ id\ \ x \\ \quad\ = x\ \texttt{in} \\ id\ 3\end{array}$$

- Ideas from type reconstruction with subtypes apply.

[Mitchell 83, Fuh-Mishra 88, Aiken-Wimmers 93, Eifrig-Smith 95, Heintze 92, Fähndrich-Aiken 96, Flanagan-Felleisen 97]

Basic Reconstruction

Some parts are easy:

- Reconstruction of the underlying simple types unaffected by predicates.

$$\begin{array}{l}\texttt{let}\ id\ (x : \{n\!:\!\texttt{Int} \mid \gamma_1\}) \\ \quad = x\ \texttt{in} \\ id\ 3 \\ \text{where}\ (n = 3) \Rightarrow \gamma_1\end{array}$$

- Ideas from type reconstruction with subtypes apply.

[Mitchell 83, Fuh-Mishra 88, Aiken-Wimmers 93, Eifrig-Smith 95, Heintze 92, Fähndrich-Aiken 96, Flanagan-Felleisen 97]

Basic Reconstruction

Some parts are easy:

- Reconstruction of the underlying simple types unaffected by predicates.

$$\begin{array}{l}\texttt{let}\ id\ (x : \{n\!:\!\texttt{Int} \mid \gamma_1\}) : \{n\!:\!\texttt{Int} \mid \gamma_2\} \\ \quad = x\ \texttt{in} \\ id\ 3 \\ \ \text{where}\ (n = 3) \Rightarrow \gamma_1 \\ \ \text{and}\ \gamma_1 \Rightarrow \gamma_2\end{array}$$

- Ideas from type reconstruction with subtypes apply.

[Mitchell 83, Fuh-Mishra 88, Aiken-Wimmers 93, Eifrig-Smith 95, Heintze 92, Fähndrich-Aiken 96, Flanagan-Felleisen 97]

Predicate Reconstruction

Some parts are harder:

- Lexical scope - naïve propagation according to dataflow paths will let variables escape their scope.
- Dependent types - an implication constraint may be contingent upon particular values of program variables.
- (Mutual) Recursion - causes cycles in the dataflow paths.
- Our "logic" is the operational semantics of programs. Can it express all solutions?

Example

$$\texttt{let } fact\ \ x$$
$$\quad = \texttt{ if } (x \leq 1) \texttt{ then } 1 \texttt{ else } (fact\ (x-1)) * x$$
$$\texttt{in}$$
$$fact\ 3$$

Example

$$\texttt{let}\ fact\ (x : \{n : \texttt{Int} \mid \gamma_1\})$$
$$\quad = \ \texttt{if}\ (x \leq 1)\ \texttt{then}\ 1\ \texttt{else}\ (fact\ (x-1)) * x$$
$$\texttt{in}$$
$$fact\ 3$$

Example

$$\texttt{let}\ fact\ (x : \{n : \texttt{Int} \mid \gamma_1\})\ :\ \{n : \texttt{Int} \mid \gamma_2\}$$
$$=\ \texttt{if}\ (x \leq 1)\ \texttt{then}\ 1\ \texttt{else}\ (fact\ (x-1)) * x$$
$$\texttt{in}$$
$$fact\ 3$$

Example

$$\texttt{let } fact\ (x : \{n\!:\!\texttt{Int} \mid \gamma_1\}) \ : \ \{n\!:\!\texttt{Int} \mid \gamma_2\}$$
$$= \ \texttt{if } (x \leq 1) \texttt{ then } 1 \texttt{ else } (fact\ (x-1)) * x$$
$$\texttt{in}$$
$$fact\ 3$$

$$n : \texttt{Int} \ \vdash \ (n = 3) \ \Rightarrow \ \gamma_1$$

$$x : \{n\!:\!\texttt{Int} \mid \gamma_1\}, n : \texttt{Int} \ \vdash \ (n = x - 1) \ \Rightarrow \ \gamma_1$$

Example

$$\texttt{let } fact\ (x : \{n\!:\!\texttt{Int} \mid \gamma_1\})\ :\ \{n\!:\!\texttt{Int} \mid \gamma_2\}$$
$$\quad = \texttt{ if } (x \leq 1) \texttt{ then } 1 \texttt{ else } (fact\ (x-1)) * x$$
$$\texttt{in}$$
$$fact\ 3$$

$$n : \texttt{Int} \ \vdash\ (n = 3)\ \Rightarrow\ \gamma_1$$

$$x : \{n\!:\!\texttt{Int} \mid \gamma_1\}, n : \texttt{Int} \ \vdash\ (n = x - 1)\ \Rightarrow\ \gamma_1$$

$$\gamma_1\ :=\ (n = 3) \vee (n = x - 1)\ ?$$

Example

$$\texttt{let } fact\ (x : \{n\!:\!\texttt{Int} \mid \gamma_1\}) \ : \ \{n\!:\!\texttt{Int} \mid \gamma_2\}$$
$$\quad = \ \texttt{if } (x \leq 1) \texttt{ then } 1 \texttt{ else } (fact\ (x-1)) * x$$
$$\texttt{in}$$
$$fact\ 3$$

$$n : \texttt{Int} \ \vdash \ (n = 3) \ \Rightarrow \ \gamma_1$$

$$x : \{n\!:\!\texttt{Int} \mid \gamma_1\}, n : \texttt{Int} \ \vdash \ (n = x - 1) \ \Rightarrow \ \gamma_1$$

$$\Updownarrow$$

$$n : \texttt{Int} \ \vdash \ (\hat{\exists} x : \{n\!:\!\texttt{Int} \mid \gamma_1\}.\ n = x - 1) \ \Rightarrow \gamma_1$$

Nondeterministic Existentials

Remember, our logic is operational semantics.

$$\hat{\exists} x : T.\ s \longrightarrow [x := t]\, s \qquad \text{whenever} \qquad \vdash t : T$$

With this definition, the classic tautology still holds:

$$
\begin{array}{l}
x : T \vdash p \Rightarrow q \\
\qquad \Updownarrow \\
\forall t : T.\ \ \text{if } [x := t]\, p \longrightarrow^* \texttt{true} \text{ then } q \longrightarrow^* \texttt{true} \qquad (x \text{ is not free in } q) \\
\qquad \Updownarrow \\
\ \text{if } \hat{\exists} x : T.\ p \longrightarrow^* \texttt{true} \text{ then } q \longrightarrow^* \texttt{true} \\
\qquad \Updownarrow \\
\vdash (\hat{\exists} x : T.\ p) \Rightarrow q
\end{array}
$$

Example

$$\texttt{let } fact\ (x : \{n\!:\!\texttt{Int} \mid \gamma_1\}) \ : \ \{n\!:\!\texttt{Int} \mid \gamma_2\}$$
$$\quad = \ \texttt{if } (x \leq 1) \texttt{ then } 1 \texttt{ else } (fact\ (x-1)) * x$$
$$\texttt{in}$$
$$fact\ 3$$

$$n : \texttt{Int} \ \vdash \ (n = 3) \ \Rightarrow \ \gamma_1$$

$$x : \{n\!:\!\texttt{Int} \mid \gamma_1\}, n : \texttt{Int} \ \vdash \ (n = x - 1) \ \Rightarrow \ \gamma_1$$

$$\Updownarrow$$

$$n : \texttt{Int} \ \vdash \ (\hat{\exists} x : \{n\!:\!\texttt{Int} \mid \gamma_1\}.\ n = x - 1) \ \Rightarrow \gamma_1$$

Example

$$\texttt{let}\ fact\ (x : \{n : \texttt{Int} \mid \gamma_1\})\ :\ \{n : \texttt{Int} \mid \gamma_2\}$$
$$=\ \texttt{if}\ (x \leq 1)\ \texttt{then}\ 1\ \texttt{else}\ (fact\ (x-1)) * x$$
$$\texttt{in}$$
$$fact\ 3$$

$$n : \texttt{Int} \vdash (n = 3) \Rightarrow \gamma_1$$

$$n : \texttt{Int} \vdash (\hat{\exists} x : \{n : \texttt{Int} \mid \gamma_1\}.\ n = x - 1) \Rightarrow \gamma_1$$

Example

$$\texttt{let } fact\ (x : \{n : \texttt{Int} \mid \gamma_1\})\ :\ \{n : \texttt{Int} \mid \gamma_2\}$$
$$\quad =\ \texttt{if } (x \leq 1) \texttt{ then } 1 \texttt{ else } (fact\ (x-1)) * x$$
$$\texttt{in}$$
$$fact\ 3$$

$$n : \texttt{Int} \ \vdash\ (n = 3)\ \Rightarrow\ \gamma_1$$

$$n : \texttt{Int} \ \vdash\ (\hat{\exists} x : \{n : \texttt{Int} \mid \gamma_1\}.\ n = x - 1) \Rightarrow \gamma_1$$

$$\gamma_1 :=\ (n = 3) \vee (\hat{\exists} x : \{n : \texttt{Int} \mid \gamma_1\}.\ n = x - 1)\ ?$$

Predicate functions

$$\gamma_1 = (n = 3) \vee (\hat{\exists} x : \{n : \texttt{Int} \mid \gamma_1\}.\ n = x - 1)$$

- We desire a solution to the above equation.
- Interpret γ_1 as a function $\mathcal{F}_{\gamma_1}$ over $fv(\gamma_1) = \{n : \texttt{Int}\}$
- $\mathcal{F}_{\gamma_1} : \texttt{Int} \rightarrow \texttt{Bool}$
- In the case of cycles, $\mathcal{F}_{\gamma_1}$ can be recursively defined.

Predicate functions

$$\gamma_1 = (n = 3) \vee (\hat{\exists} x : \{n : \texttt{Int} \mid \gamma_1\}.\ n = x - 1)$$

- The solution:
  $$\mathcal{F}_{\gamma_1}\ n \quad = \quad (n = 3)\ \vee\ (\hat{\exists} x : \{n : \texttt{Int} \mid \mathcal{F}_{\gamma_1}\ n\}.\ n = x - 1)$$
- Entirely a syntactic transformation.
- Because our logic is very expressive!
- As with weakest preconditions/strongest postconditions.

Predicate functions

$$\gamma_1 = (n = 3) \vee (\hat{\exists} x : \{n : \texttt{Int} \mid \gamma_1\}.\ n = x - 1)$$

- The solution:

$$\begin{aligned}
\mathcal{F}_{\gamma_1}\ n &= (n = 3) \vee (\hat{\exists} x : \{n : \texttt{Int} \mid \mathcal{F}_{\gamma_1}\ n\}.\ n = x - 1) \\
&= (n = 3) \vee (n = 2) \vee (\hat{\exists} x.\ \cdots) \\
&= (n = 3) \vee (n = 2) \vee (n = 1) \vee \cdots \\
&= (n = 3) \vee (n = 2) \vee (n = 1) \vee (n = 0) \vee \cdots \\
&= n \leq 3
\end{aligned}$$

Example

$$\texttt{let } fact \quad (x : \{n{:}\texttt{Int} \mid n \leq 3\}) \quad : \{n{:}\texttt{Int} \mid \gamma_2\}$$
$$= \texttt{ if } (x \leq 1) \texttt{ then } 1 \texttt{ else } (fact\ (x-1)) * x$$
$$\texttt{in}$$
$$fact\ 3$$

- We always infer the strongest predicate
- In *this* program, $fact$ is applied to 3, 2, ...
- hence "$n \leq 3$"

Example

$$\texttt{let } fact \quad (x : \{n{:}\texttt{Int} \mid n \leq 3\}) \quad : \{n{:}\texttt{Int} \mid \gamma_2\}$$
$$= \texttt{ if } (x \leq 1) \texttt{ then } 1 \texttt{ else } (fact\ (x-1)) * x$$
$$\texttt{in}$$
$$fact\ 3$$

Path-insensitive:

$$\cdots \vdash (n = 1) \Rightarrow \gamma_2$$

$$\cdots \vdash (n = (fact(x-1)) * x) \Rightarrow \gamma_2$$

Example

$$\begin{array}{l}\texttt{let } fact \quad (x : \{n{:}\texttt{Int} \mid n \leq 3\}) \quad : \{n{:}\texttt{Int} \mid \gamma_2\} \\ \quad = \texttt{ if } (x \leq 1) \texttt{ then } 1 \texttt{ else } (fact\ (x-1)) * x \\ \texttt{in} \\ fact\ 3\end{array}$$

Path-insensitive:

$$\cdots \vdash (n = 1) \Rightarrow \gamma_2$$

$$\cdots \vdash (n = (fact(x-1)) * x) \Rightarrow \gamma_2$$

$$\begin{array}{lll}\mathcal{F}_{\gamma_2}\ x\ n & = & (n = 1) \vee \\ & & (\hat{\exists} fact : (x' : \{n{:}\texttt{Int} \mid n \leq 3\} \rightarrow \{y{:}\texttt{Int} \mid \mathcal{F}_{\gamma_2}\ x'\ y\}). \\ & & \quad n = (fact(x-1)) * x)\end{array}$$

Path Insensitive Example

$$
\begin{aligned}
\mathcal{F}_{\gamma_2}\ x\ n \quad &= \quad (n = 1) \vee \\
&\quad (\hat{\exists} fact : (x' : \{n : \texttt{Int} \mid n \leq 3\} \rightarrow \{y : \texttt{Int} \mid \mathcal{F}_{\gamma_2}\ x'\ y\}). \\
&\qquad n = (fact(x-1)) * x) \\
&= \quad (n = 1) \vee (n = 2) \vee (n = 3) \vee (n = 0) \vee \cdots (n = -\infty) \vee \\
&\quad (\hat{\exists} fact : (x' : \{n : \texttt{Int} \mid n \leq 3\} \rightarrow \{y : \texttt{Int} \mid \mathcal{F}_{\gamma_2}\ x'\ y\}). \\
&\qquad n = (fact(x-1)) * x) \\
&= \quad (n = -\infty) \vee \cdots \vee (n = \infty) \vee \\
&\quad (\hat{\exists} fact : (x' : \{n : \texttt{Int} \mid n \leq 3\} \rightarrow \{y : \texttt{Int} \mid \mathcal{F}_{\gamma_2}\ x'\ y\}). \\
&\qquad n = (fact(x-1)) * x) \\
&= \quad (n = -\infty) \vee \cdots \vee (n = \infty)
\end{aligned}
$$

Example

$$\texttt{let } fact \quad (x : \{n{:}\texttt{Int} \mid n \leq 3\}) \quad : \{n{:}\texttt{Int} \mid \gamma_2\}$$
$$= \texttt{ if } (x \leq 1) \texttt{ then } 1 \texttt{ else } (fact\ (x-1)) * x$$
$$\texttt{in}$$
$$fact\ 3$$

Example

$$\begin{array}{l}\texttt{let } fact \quad (x : \{n{:}\texttt{Int} \mid 1 \leq n \leq 3\}) \quad : \{n{:}\texttt{Int} \mid \gamma_2\} \\ \quad = \texttt{ if } (x \leq 1) \texttt{ then } 1 \texttt{ else } (fact\ (x-1)) * x \\ \texttt{in} \\ fact\ 3\end{array}$$

Path-sensitive:

$$\cdots \vdash (x \leq 1) \wedge (n = 1) \Rightarrow \gamma_2$$

$$\cdots \vdash (x > 1) \wedge (n = (fact(x-1)) * x) \Rightarrow \gamma_2$$

$$\begin{array}{rcl}\mathcal{F}_{\gamma_2}\ x\ n & = & [(x \leq 1) \wedge (n = 1)] \vee \\ & & [\hat{\exists} fact : (x' : \{n{:}\texttt{Int} \mid 1 \leq n \leq 3\} \rightarrow \{y{:}\texttt{Int} \mid \mathcal{F}_{\gamma_2}\ x'\ y\}). \\ & & \quad (x > 1) \wedge (n = (fact(x-1)) * x)]\end{array}$$

Path Sensitive Example

$$
\begin{aligned}
\mathcal{F}_{\gamma_2}\ x\ n \quad = \quad & [(x \leq 1) \wedge (n = 1)] \vee \\
& [\hat{\exists} fact : (x' : \{n : \texttt{Int} \mid 1 \leq n \leq 3\} \rightarrow \{y : \texttt{Int} \mid \mathcal{F}_{\gamma_2}\ x'\ y\}). \\
& \qquad (x > 1) \wedge (n = (fact(x-1)) * x)] \\
= \quad & [(x \leq 1) \wedge (n = 1)] \vee [(x > 1) \wedge (x-1 \leq 1) \wedge (n = 2)] \vee \\
& [\hat{\exists} fact : (x' : \{n : \texttt{Int} \mid 1 \leq n \leq 3\} \rightarrow \{y : \texttt{Int} \mid \mathcal{F}_{\gamma_2}\ x'\ y\}). \\
& \qquad (x > 1) \wedge (n = (fact(x-1)) * x)] \\
& \qquad \vdots \\
= \quad & [(x \leq 1) \wedge (n = 1)] \vee \\
& [(x = 2) \wedge (n = 2)] \vee \\
& [(x = 3) \wedge (n = 6)]
\end{aligned}
$$

Example

$$\texttt{let } fact \quad (x : \{n{:}\texttt{Int} \mid 1 \leq n \leq 3\}) \quad : \{n{:}\texttt{Int} \mid \cdots\}$$
$$= \texttt{ if } (x \leq 1) \texttt{ then } 1 \texttt{ else } (fact\ (x-1)) * x$$
$$\texttt{in}$$
$$fact\ 3$$

Properties of our algorithm

Nice properties:

- The output program is well-typed if and only if the input program is typeable.
- We infer the most precise predicate for each refinement.
- And they are correct by construction – they need never be inserted as runtime checks.

Remaining work:

- Automated simplification of inferred refinements.
- Weakest predicates for contravariant positions.
- More complex language features.

Breakdown

	First-Order Imperative	**Higher-order Functional**
Specs	Assertions Loop Invariants, Pre/Postconditions	Refinement Types Dependent Types
Dynamic	Contracts	Type Casts
Static	Hoare Logic + Theorem Proving	(Hybrid) Type Checking + Theorem Proving
Automatic Inference	Weakest Precond. Strongest Postcond.	Strongest predicate reconstruction

High-level Contributions

- Type reconstruction (re)defined in a way that is more useful for undecidable type systems.
- Type reconstruction solved for a basic calculus with types refined by arbitrary program terms.
- Our algorithm achieves for higher-order functional programs what strongest postconditions achieves for first-order imperative programs.