Static Haskell Contract Checking

Dan Rosén

Dimitrios Vytiniotis, Koen Claessen, Simon Peyton Jones

Microsoft Research

September 5, 2012

Contracts

```
head :: [a] -> a
head (x:xs) = x
head []     = error "head: empty list!"
```

Some example contracts for `head`:

$$\texttt{head} \in \mathsf{CF} \to \mathsf{CF}$$
$$\texttt{head} \in \{\texttt{xs} \mid \texttt{not (null xs)}\} \to \mathsf{CF}$$
$$\texttt{head} \in \mathsf{CF}\&\{\texttt{xs} \mid \texttt{not (null xs)}\} \to \mathsf{CF}$$

CF stands for Crash-Free

Our Values

$$\text{Haskell values} + \underbrace{\overbrace{\mathtt{BAD}}^{\text{catchable errors}} + \overbrace{\mathtt{UNR}}^{\text{non-termination}}}_{\perp}$$

```
head :: [a] -> a
head (x:xs) = x
head []     = BAD
head BAD    = BAD
head _       = UNR
```

CF means a value recursively does not contain `BAD`
(but it could contain `UNR`)

Translating Data Types to FOL

- Discrimination axioms

$$\begin{array}{ll} \mathsf{cons}(x, xs) \neq \mathsf{nil}, & \mathtt{BAD} \neq \mathtt{UNR} \\ \mathsf{cons}(x, xs) \neq \mathtt{BAD}, & \mathsf{cons}(x, xs) \neq \mathtt{UNR}, \\ \mathsf{nil} \neq \mathtt{BAD}, & \mathsf{nil} \neq \mathtt{UNR} \end{array}$$

- Injectivity axioms

$$\begin{array}{rcl} \mathsf{cons}_0(\mathsf{cons}(x, xs)) & = & x, \\ \mathsf{cons}_1(\mathsf{cons}(x, xs)) & = & xs \end{array}$$

Now $\mathsf{cons}(x, xs) = \mathsf{cons}(y, ys) \to x = y$ (by cons_0)

- Crash-freeness

$$\mathsf{CF}(\mathsf{nil}), \quad \mathsf{CF}(\mathtt{UNR}), \quad \neg\mathsf{CF}(\mathtt{BAD}),$$
$$\mathsf{CF}(\mathsf{cons}(x, xs)) \leftrightarrow (\mathsf{CF}(x) \wedge \mathsf{CF}(xs))$$

Translating Functions to FOL

```
head :: [a] -> a
head (x:xs) = x
head []      = BAD
head BAD     = BAD
head _        = UNR
```

$$
\begin{array}{lcl}
\mathsf{head}(\mathsf{cons}(x, xs)) & = & x, \\
\mathsf{head}(\mathsf{nil}) & = & \mathtt{BAD}, \\
\mathsf{head}(\mathtt{BAD}) & = & \mathtt{BAD}, \\
\mathsf{head}(x) & = & \mathtt{UNR} \\
\quad \vee\ (\exists\, y\ ys . x = \mathsf{cons}(y, ys)) & & \\
\quad \vee\ x = \mathsf{nil} & & \\
\quad \vee\ x = \mathtt{BAD} & &
\end{array}
$$

Translating Functions to FOL

```
head :: [a] -> a
head (x:xs) = x
head []      = BAD
head BAD     = BAD
head _        = UNR
```

$$
\begin{array}{lcl}
\mathsf{head}(\mathsf{cons}(x, xs)) & = & x, \\
\mathsf{head}(\mathsf{nil}) & = & \mathtt{BAD}, \\
\mathsf{head}(\mathtt{BAD}) & = & \mathtt{BAD}, \\
\mathsf{head}(x) & = & \mathtt{UNR} \\
\quad \lor\ x = \mathsf{cons}(\mathsf{cons}_0(x), \mathsf{cons}_1(x)) & & \\
\quad \lor\ x = \mathsf{nil} & & \\
\quad \lor\ x = \mathtt{BAD} & &
\end{array}
$$

Translating Contracts to FOL

$$
\begin{array}{rcl}
\mathcal{C}\{\!\{e \in \{x \mid p\}\}\!\} & = & \mathcal{E}\{\!\{e\}\!\}{=}\mathtt{UNR} \lor \\
 & & \mathcal{E}\{\!\{p\}\!\}[\mathcal{E}\{\!\{e\}\!\}/x]{=}\mathtt{UNR} \lor \\
 & & \mathcal{E}\{\!\{p\}\!\}[\mathcal{E}\{\!\{e\}\!\}/x]{=}\mathtt{True} \\
\mathcal{C}\{\!\{e \in (x{:}C_1) \to C_2\}\!\} & = & \forall x\,.\,\mathcal{C}\{\!\{x \in C_1\}\!\} \to \mathcal{C}\{\!\{e\ x \in C_2\}\!\} \\
\mathcal{C}\{\!\{e \in C_1 \& C_2\}\!\} & = & \mathcal{C}\{\!\{e \in C_1\}\!\} \land \mathcal{C}\{\!\{e \in C_2\}\!\} \\
\mathcal{C}\{\!\{e \in \mathsf{CF}\}\!\} & = & \mathsf{CF}(\mathcal{E}\{\!\{e\}\!\})
\end{array}
$$

```
contract_1 = head ::: Pred (not . null) --> CF
```

$$
(x{=}\mathtt{UNR} \lor \mathtt{not}(\mathtt{null}(x)){=}\mathtt{UNR} \lor \mathtt{not}(\mathtt{null}(x)){=}\mathtt{True}) \to \mathsf{CF}(\mathtt{head}(x))
$$

Querying a Theorem Prover

We ask for the satisfiablitiy of

$$\mathcal{T}_{\text{datatypes}}, \mathcal{T}_{\text{functions}}, \neg\phi_{\text{contract}}$$

If it is unsatisfiable, we know that

$$\mathcal{T}_{\text{datatypes}}, \mathcal{T}_{\text{functions}} \vdash \phi_{\text{contract}}$$

Does this means that the contract hold? What if we have made a bogus, unsound translation? Powered by denotational semantics!

Satisfying a Contract, Denotationally

$$[\![\mathtt{C}]\!]_\rho \subseteq D_\infty$$

$$\begin{array}{rcl}
[\![x \mid e]\!]_\rho & = & \{d \mid d = \bot \;\vee\; [\![e]\!]_{\rho,x\mapsto d} \in \{\mathsf{True}, \bot\}\} \\
[\![(x{:}\mathtt{C}_1) \to \mathtt{C}_2]\!]_\rho & = & \{d \mid \forall d' \in [\![\mathtt{C}_1]\!]_\rho . \mathsf{app}(d, d') \in [\![\mathtt{C}_2]\!]_{\rho,x\mapsto d'}\} \\
[\![\mathtt{C}_1 \& \mathtt{C}_2]\!]_\rho & = & \{d | d \in [\![\mathtt{C}_1]\!]_\rho \wedge d \in [\![\mathtt{C}_2]\!]_\rho\} \\
[\![\mathtt{CF}]\!]_\rho & = & F^\infty_{\mathsf{cf}}
\end{array}$$

where

$$\begin{array}{rcl}
F^\infty_{\mathsf{cf}} & = & \{\bot\} \\
& \cup & \{\ \mathsf{K}(\overline{d}) \mid K^n \in \Sigma,\ d_i \in F^\infty_{\mathsf{cf}}\} \\
& \cup & \{\ \mathsf{Fun}(d) \mid \forall d' \in F^\infty_{\mathsf{cf}} .\ d(d') \in F^\infty_{\mathsf{cf}}\}
\end{array}$$

Soundness Theorem

Theorem

Assume that e and `C` *contain no free term variables. Then the FOL translation of the claim $e \in \mathtt{C}$ holds in the model if and only if the denotation of e is in the semantics of* `C`*. Formally:*

$$\langle D_\infty, \mathcal{I} \rangle \models \mathcal{C}\{\!\{e \in \mathtt{C}\}\!\} \quad \Leftrightarrow \quad [\![e]\!] \in [\![\mathtt{C}]\!]$$

Recursion and Fixed Point Induction

Splitting Goals

`risers` in GHC Core is a bunch of cases...

```
risers = \ xs -> case xs of {
    [] -> []
    y : ys -> case ys of {
        [] -> [[y]]
        z : zs -> case risers (z:zs) of {
            [] -> error "internal error";
            : s ss -> case y <= z of {
                False -> [y] : (s:ss)
                True ->  (y:s) : ss
    } } } }
```

These cases becomes a big chunk of translated formulae, making a big theory. However, we can split every left-hand side of a case alternative a small, separate theory when proving a contract for `risers`. In practice, these smaller theories are much easier for theorem provers to handle.

Printing Counterexamples

We ask for the satisfiablitiy of

$$\mathcal{T}_{\text{datatypes}}, \mathcal{T}_{\text{functions}}, \neg\phi_{\text{contract}}$$

If it is satisfiable, we know that there exists a model M such that

$$M \models \mathcal{T}_{\text{datatypes}}, \mathcal{T}_{\text{functions}}, \neg\phi_{\text{contract}}$$

Happens when:

- the contract does not hold
- assumptions are missing (induction, other contracts)
- the theory is incomplete

Infinite Models

But it seems hard to ever get satisfiable from a theorem prover:

Theorem

First order theories with any function both injective and non-surjective function only admits infinite models.

Recursive and parameterised non-recursive(!) datatypes have this property:

$$\mathsf{just}(x) \neq \mathsf{nothing}, \quad \mathsf{just}_0(\mathsf{just}(x)) = x$$

For the semi-decidable problem to find infinite models no general theorem provers exist.

"Minimisation": Our Trick for Finite Models and Efficiency

- Idea: introduce a new predicate, min, that means a term should be subject to reduction (to weak head normal form).
- Selector axioms:

$$\mathsf{min}(\mathsf{just}(x)) \rightarrow \mathsf{just}_0(\mathsf{just}(x)) = x$$

- The name comes from that we should try to make as few domain elements "min".

Function Translation with Minimisation

```
head :: [a] -> a
head (x:xs) = x
head []     = BAD
head BAD    = BAD
head _       = UNR
```

$$
\begin{array}{lcll}
\mathsf{min}(\mathsf{head}(x)) & \rightarrow & \mathsf{min}(x), & \\
\mathsf{min}(\mathsf{head}(\mathsf{cons}(x, xs))) & \rightarrow & \mathsf{head}(\mathsf{cons}(x, xs)) & = \quad x, \\
\mathsf{min}(\mathsf{head}(\mathsf{nil})) & \rightarrow & \mathsf{head}(\mathsf{nil}) & = \quad \mathtt{BAD}, \\
\mathsf{min}(\mathsf{head}(\mathtt{BAD})) & \rightarrow & \mathsf{head}(\mathtt{BAD}) & = \quad \mathtt{BAD}, \\
\mathsf{min}(\mathsf{head}(x)) & \rightarrow & (\mathsf{head}(x) & = \quad \mathtt{UNR} \\
 & & \quad \vee\ (\exists\ y\ ys. x = \mathsf{cons}(y, ys)) & \\
 & & \quad \vee\ x = \mathsf{nil} & \\
 & & \quad \vee\ x = \mathtt{BAD}) &
\end{array}
$$

Function Translation with Minimisation

```
head :: [a] -> a
head (x:xs) = x
head []     = BAD
head BAD    = BAD
head _       = UNR
```

$$
\begin{array}{lcl}
\mathsf{min}(\mathsf{head}(x)) & \rightarrow & \mathsf{min}(x), \\
\mathsf{min}(\mathsf{head}(\mathsf{cons}(x, xs))) & \rightarrow & \mathsf{head}(\mathsf{cons}(x, xs)) = x, \\
\mathsf{min}(\mathsf{head}(\mathsf{nil})) & \rightarrow & \mathsf{head}(\mathsf{nil}) = \mathtt{BAD}, \\
\mathsf{min}(\mathsf{head}(\mathtt{BAD})) & \rightarrow & \mathsf{head}(\mathtt{BAD}) = \mathtt{BAD}, \\
\mathsf{min}(\mathsf{head}(x)) & \rightarrow & (\mathsf{head}(x) = \mathtt{UNR} \\
& & \quad \vee\ x = \mathsf{cons}(\mathsf{cons}_0(x), \mathsf{cons}_1(x)) \\
& & \quad \vee\ x = \mathsf{nil} \\
& & \quad \vee\ x = \mathtt{BAD})
\end{array}
$$

Contract Translation with Minimisation

Distinguish between assumptions ($e \in C$) and goals ($e \notin C$).

Contracts should only be assumed when they are "min", contracts the prove should always be "min" to drive computation.

$$
\begin{array}{rcl}
\mathcal{C}[\![e \in \{x \mid p\}]\!] & = & \mathsf{min}(\mathcal{E}[\![e]\!]) \wedge \mathsf{min}(\mathcal{E}[\![p]\!][\mathcal{E}[\![e]\!]/x]) \\
 & & (\mathcal{E}[\![e]\!] {=} \mathtt{UNR} \vee \\
 & & \mathcal{E}[\![p]\!][\mathcal{E}[\![e]\!]/x] {=} \mathtt{UNR} \vee \\
 & & \mathcal{E}[\![p]\!][\mathcal{E}[\![e]\!]/x] {=} \mathtt{True})
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{C}[\![e \notin \{x \mid p\}]\!] & = & \mathsf{min}(\mathcal{E}[\![e]\!]) \wedge \mathsf{min}(\mathcal{E}[\![p]\!][\mathcal{E}[\![e]\!]/x]) \\
 & & (\mathcal{E}[\![e]\!] \neq \mathtt{UNR} \vee \\
 & & \mathcal{E}[\![p]\!][\mathcal{E}[\![e]\!]/x] {=} \mathtt{BAD} \vee \\
 & & \mathcal{E}[\![p]\!][\mathcal{E}[\![e]\!]/x] {=} \mathtt{False})
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{C}[\![e \in (x{:}C_1) \to C_2]\!] & = & \forall x \,.\, \mathsf{min}(e\ x) \to \\
 & & \qquad (\mathcal{C}[\![x \notin C_1]\!] \vee \mathcal{C}[\![e\ x \in C_2]\!]) \\
\mathcal{C}[\![e \notin (x{:}C_1) \to C_2]\!] & = & \exists x \,.\, \mathcal{C}[\![x \in C_1]\!] \wedge \mathcal{C}[\![e\ x \notin C_2]\!]
\end{array}
$$

Finite Model Finding

- We use the finite model finder `paradox`, which exhaustively seaches for models with increasing domain size and gives us the smallest possible model.
- Countermodels are typically very few elements (4-6), with many infinite values such as `xs = Nothing : xs`.
- Since constructors now are not injective, we need to do a little work to find out how domain elements really are represented.

Unearthing a Model

```
(-) :: Nat -> Nat -> Nat
x      - Zero   = x
Zero   - _      = error "Negative Nat!"
Succ x - Succ y = x - y
```

$$(\texttt{-}) \in \{\texttt{CF}- > \texttt{CF}- > \texttt{CF}\}$$

paradox gives a countermodel with 5 elements: $\mathbf{D} = \{\mathbf{1}, \mathbf{2}, \cdots, \mathbf{5}\}$

Unearthing a Model

```
(-) :: Nat -> Nat -> Nat
x      - Zero   = x
Zero   - _      = error "Negative Nat!"
Succ x - Succ y = x - y
```

$$(\texttt{-}) \in \{\texttt{CF}- > \texttt{CF}- > \texttt{CF}\}$$

paradox gives a countermodel with 5 elements: $\mathbf{D} = \{\mathbf{1}, \mathbf{2}, \cdots, \mathbf{5}\}$

$$\begin{aligned} \texttt{x} &= \mathbf{3} \\ \texttt{y} &= \mathbf{4} \end{aligned}$$

Figuring out what x and y are

$$\begin{array}{rcl} \mathtt{x} & = & \mathbf{3} \\ \mathtt{y} & = & \mathbf{4} \\ \mathtt{BAD} & = & \mathbf{1} \\ \mathtt{UNR} & = & \mathbf{2} \\ \mathtt{Zero} & = & \mathbf{3} \end{array} \qquad \begin{array}{rcl} \mathtt{Succ}(\mathbf{1}) & = & \mathbf{5} \\ \mathtt{Succ}(\mathbf{2}) & = & \mathbf{2} \\ \mathtt{Succ}(\mathbf{3}) & = & \mathbf{4} \\ \mathtt{Succ}(\mathbf{4}) & = & \mathbf{5} \\ \mathtt{Succ}(\mathbf{5}) & = & \mathbf{5} \end{array} \qquad \begin{array}{rcl} \mathtt{Succ}_0(\mathbf{1}) & = & \mathbf{3} \\ \mathtt{Succ}_0(\mathbf{2}) & = & \mathbf{3} \\ \mathtt{Succ}_0(\mathbf{3}) & = & \mathbf{2} \\ \mathtt{Succ}_0(\mathbf{4}) & = & \mathbf{3} \\ \mathtt{Succ}_0(\mathbf{5}) & = & \mathbf{5} \end{array}$$

x	$\mathtt{Succ}(x)$	$\mathtt{Succ}_0(\mathtt{Succ}(x))$
1	**5**	**5**
2	**2**	**3**
3	**4**	**3**
4	**5**	**5**
5	**5**	**5**

$$\mathtt{y} = \mathtt{Succ\ Zero}, \qquad \mathtt{x} = \mathtt{Zero}$$

Ill-typed Models

In the model above, we have

$$\mathtt{x} = \mathtt{Zero} = \mathtt{True}$$

The reason is that we do not add discrimination axioms for elements of different types - these are never needed in proofs.
Two ways to proceed:

- Do type inference on the model to make sure that it is printed type-correct
- Add discrimination axioms for constructors of different types.

Integer Arithmetic

Project stated of using only pure first order theories, communicating with theorem provers using the TPTP format.
This format (naturally) has no support for built-ins like `Int`.
`z3`, initially being an SMT solver, reads various SMT formats that support `Int`.
However we cannot print countermodels since `z3` is not able to find the finite countermodels as `paradox` can.

Open Questions / Future Work

- Do we need special tehorem provers or SMT theories for (lazy) functional programs?
- Can `z3` be used effictively with triggers (as the `min` predicate)?
- Can `z3` be used to find counter-models?
- How far can automated techniques get us (in comparison with fully or semi interactive tools)?
- Is there a (provably) complete min-axiomatisation with guaranteed finite countermodels?