Constrained Horn Clauses as a Basis of Automatic Program Verification:
The Higher-order Case

Luke Ong
(Joint with Toby Cathcart Burn and Steven Ramsay)

University of Oxford

IFIP WG2.2 Meeting, 18-20 September 2017, Bordeaux
“Constrained Horn clauses are a suitable basis for automatic program verification, i.e., symbolic model checking.” [Bjørner et al. 2012]

Constrained means truth of formula is relative to a decidable 1st-order background theory (e.g. ZLA).

**Example: safety verification**

Recursive predicate:
\[
\forall x. (Initial(x) \implies Reach(x))
\]
\[
\forall x. (Reach(x) \land Trans(x, x') \implies Reach(x'))
\]

Query:
\[
\forall x. (Reach(x) \implies Safe(x))
\]

Solve for (unknown) predicate \( Reach \), which defines an inductive invariant.

Many algorithmic solutions. Examples: CLP (Jaffar et al.); IC3 algorithms (Bradley); lazy annotation (Jaffar, McMillan, etc.).
Desirable features of Horn clauses

Horn clauses originated from theorem proving in 1st-order logic.

1. **Syntactic simplicity** eases presentation of proof procedure.
   E.g. 1st-order resolution: resolvent of two Horn clauses is a Horn clause.

2. **Solving satisfiability of Horn clause fragments is simpler**

<table>
<thead>
<tr>
<th>Logic</th>
<th>Horn</th>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional</td>
<td>P</td>
<td>NP</td>
</tr>
<tr>
<td>Bernays-Schönfinkel ($\exists^* \forall^*$)</td>
<td>DEXPTIME</td>
<td>NEXPTIME</td>
</tr>
</tbody>
</table>

3. Horn clauses enjoy **least model property**:  
   - useful for **model building**: as symbolic representation of partial models (even for non-Horn theories).
**Why constrained Horn clauses (rather than model checking)?**

1. **Expressivity**: Horn constraints can express standard verification proof rules, and encode safety, liveness, CTL+FO, and game solving. [Rybalchenko et al. PLDI12, POPL14]

2. **Adoption of standards** (i.e. SMT formats *and* Horn constraints) promotes
   - exchange of software model checking benchmarks
   - separation of concerns: let verification-condition generators worry about specificities of programming languages, whilst “model checking” is kept purely logical, and hence generic.

3. **Extensibility** and **retargetability** of verification tool (chain).

**Why higher-order constrained Horn clauses?**

The reasons above are just as applicable to higher-order computation! ... More on this anon.
1. Higher-order constrained Horn clauses (HoCHC): satisfiability and safety problems

2. Standard semantics of higher-order logic

3. Monotone semantics satisfies least model property

4. Algorithmic solutions of HoCHC safety problem: 1. via refinement types

5. Automation via prototype tool Horus
Outline

1. Higher-order constrained Horn clauses (HoCHC): satisfiability and safety problems

2. Standard semantics of higher-order logic

3. Monotone semantics satisfies least model property

4. Algorithmic solutions of HoCHC safety problem: 1. via refinement types

5. Automation via prototype tool Horus
Higher-order constrained Horn clauses arise naturally as definitions of inductive invariants of higher-order programs.

Example: safety verification

```plaintext
let add x y = x + y
letrec iter f s n = if n ≤ 0 then s else f n (iter f s (n − 1))
in λn. assert (n ≤ (iter add 0 n))
```

- \((\text{iter } f \ s \ n)\) computes \(f n (f (n − 1) (f (n − 2) (\cdots (f 1 s) \cdots ))))\).
- Thus \((\text{iter } \text{add } 0 \ n) = n + (n − 1) + \cdots + 1 + 0\).

Say the program is safe if assertion is never violated.
An **inductive invariant** of a defined function is a relation overapproximating its input-output graph.

The system below describes the class of all invariants sufficiently strong to guarantee the assertion:

\[
\begin{align*}
\forall x y z. & \quad (z = x + y \Rightarrow \text{Add} \ x \ y \ z) \\
\forall f s n m. & \quad (n \leq 0 \land m = s \Rightarrow \text{Iter} \ f \ s \ n \ m) \\
\forall f s n m. & \quad (n > 0 \land (\exists p. \text{Iter} \ f \ s \ (n - 1) \ p \land f n p m) \Rightarrow \text{Iter} \ f \ s \ n \ m) \\
\forall n m. & \quad (\text{Iter} \ \text{Add} \ 0 \ n \ m \Rightarrow n \leq m)
\end{align*}
\]
Some features of HoCHC

\[ \forall x y z. (z = x + y \Rightarrow \text{Add } x y z) \]
\[ \forall f s n m. (n \leq 0 \land m = s \Rightarrow \text{Iter } f s n m) \]
\[ \forall f s n m. \\
\quad (n > 0 \land (\exists p. \text{Iter } f s (n - 1) p \land f n p m) \Rightarrow \text{Iter } f s n m) \]
\[ \forall n m. (\text{Iter Add } 0 n m \Rightarrow n \leq m) \]

- Higher-order “unknown” relation:
  \( \text{Iter} : (\text{int} \rightarrow \text{int} \rightarrow \text{int} \rightarrow \text{bool}) \rightarrow \text{int} \rightarrow \text{int} \rightarrow \text{int} \rightarrow \text{bool} \)

- Quantification at higher sort: \( \text{int} \rightarrow \text{int} \rightarrow \text{int} \rightarrow \text{bool} \)

- Literals headed by variables: \( f n p m \)

Every model of the system is an invariant witnessing safety of the program.
Relational sorts: $\sigma ::= \text{int} \rightarrow \text{bool} \mid \text{int} \rightarrow \sigma \mid \sigma \rightarrow \sigma'$

Fix a sorting $\Delta$ of higher-order relational variables ("unknowns")

**goal** $G ::= A \mid \varphi \mid G \land G \mid G \lor G \mid \exists x: \sigma. \ G$

**definite** $D ::= \text{true} \mid \forall x: \sigma. \ D \mid D \land D \mid G \Rightarrow X \ x_1 \ldots \ x_n$

- $A$ ranges over atoms e.g. $\text{Iter} \ f \ m \ (n - 1) \ p$, $f \ n \ p \ r$
- $\varphi$ ranges over constraints e.g. $x > 3$
- $X$ ranges over $\Delta$ e.g. $\text{Iter}$

**Satisfiability Problem:** $\langle \Delta, D \rangle$ is solvable if for all models $\mathcal{A}$ of background theory $Th$, there is valuation $\alpha$ of $\Delta$ s.t. $\mathcal{A}, \alpha \models D$.

**Safety Problem:** $\langle \Delta, D, G \rangle$ is solvable if for all models $\mathcal{A}$ of $Th$, there is valuation $\alpha$ of $\Delta$ s.t. $\mathcal{A}, \alpha \models D$, yet $\mathcal{A}, \alpha \not\models G$. 
Example: an instance of HoCHC safety problem $\langle \Delta, D, G \rangle$

(1) $\forall x y z. (z = x + y \Rightarrow \text{Add } x y z)$
(2) $\forall f s n m. (n \leq 0 \land m = s \Rightarrow \text{Iter } f s n m)$
(3) $\forall f s n m.$
   $\quad (n > 0 \land (\exists p. \text{Iter } f s (n - 1) p \land f n p m) \Rightarrow \text{Iter } f s n m)$
(4) $\forall n m. (\text{Iter Add } 0 n m \Rightarrow n \leq m)$

- Sorting $\Delta$ of relational variables:
  \[
  \begin{cases}
    \text{Add} : \text{int} \to \text{int} \to \text{int} \to \text{bool} \\
    \text{Iter} : (\text{int} \to \text{int} \to \text{int} \to \text{bool}) \to \text{int} \to \text{int} \to \text{int} \to \text{int} \to \text{bool}
  \end{cases}
  \]
- Definite formula $D = (1) \land (2) \land (3)$.
- Goal formula $G = \neg (4) = \exists n m. ((\text{Iter Add } 0 n m) \land m < n)$.

Safety problem $\langle \Delta, D, G \rangle$ is solvable. I.e. w.r.t. the unique model of ZLA (∵ complete theory), there is a valuation satisfying $D$ but refuting $G$. 
Systems of definite clauses can be presented (equivalently) in program form.

\[ Add = \lambda x \ y \ z. (z = x + y) \]

\[ Iter = \lambda f \ s \ n \ m. \left( \begin{array}{c} (n \leq 0 \land m = s) \\ \lor \ \exists p. \ 0 < n \land Iter f s (n - 1) p \land f n p m \end{array} \right) \]
1. Higher-order constrained Horn clauses (HoCHC): satisfiability and safety problems

2. Standard semantics of higher-order logic

3. Monotone semantics satisfies least model property

4. Algorithmic solutions of HoCHC safety problem: 1. via refinement types

5. Automation via prototype tool Horus
Standard semantics of higher-order logic

**Sorts:** \( \sigma ::= \text{one} \mid \text{bool} \mid \text{int} \mid \sigma_1 \rightarrow \sigma_2 \)

\[
S[\text{one}] := \{\star\} \\
S[\text{bool}] := \{0, 1\} \\
S[\text{int}] := \mathbb{Z} \\
S[\sigma_1 \rightarrow \sigma_2] := S[\sigma_1] \Rightarrow S[\sigma_2] \quad (\text{all functions})
\]

**Syntax:** Standard presentation as a *simply-typed \(\lambda\)-calculus* with logical constants: \(\neg, \land, \lor, \forall_\sigma, \exists_\sigma\), etc.

\[
\neg : \text{bool} \rightarrow \text{bool} \quad \forall_\sigma, \exists_\sigma : (\sigma \rightarrow \text{bool}) \rightarrow \text{bool}
\]

We write \(\exists_\sigma (\lambda x: \sigma. M)\) as \(\exists x: \sigma. M : \text{bool}\).

**Semantics:** completely standard.

**Example:** \(\mathcal{A} \models_S \exists x : (\text{int} \rightarrow \text{bool}) \rightarrow \text{bool} . G\)

“There is some predicate \(x\) on sets of integers that makes \(G\) true in \(\mathcal{A}\).”
Failure of least model property in standard semantics!

Counterexample:

\[
\begin{cases}
  P : ((\text{one} \rightarrow \text{bool}) \rightarrow \text{bool}) \rightarrow \text{bool} \\
  Q : \text{one} \rightarrow \text{bool}
\end{cases}
\]

\[\forall x : (\text{one} \rightarrow \text{bool}) \rightarrow \text{bool} . (x \ Q \Rightarrow P \ x)\]

Theorem

*Satisfiable systems of higher-order constrained Horn clauses do not necessarily possess (unique) least models.*

(Least with respect to inclusion of relations.)
$$\forall x. (x \ Q \implies P \ x)$$ has two minimal models (=valuations) \(\alpha\) & \(\beta\)

\[P : ((\text{one} \to \text{bool}) \to \text{bool}) \to \text{bool}, \quad Q : \text{one} \to \text{bool}\]

\[S[\text{one}] := \{\star\} \]

\[S[\text{one} \to \text{bool}] := \\{\{\star \mapsto 0\}, \{\star \mapsto 1\}\}\]

\[S[(\text{one} \to \text{bool}) \to \text{bool}] := \]

\[
\begin{align*}
\{ & - \mapsto 0, \\
\{ & + \mapsto 1 \}
\end{align*}
\]

\[
\begin{align*}
\{ & - \mapsto 0, \\
\{ & + \mapsto 0 \}
\end{align*}
\]

\[
\begin{align*}
\{ & - \mapsto 1, \\
\{ & + \mapsto 1 \}
\end{align*}
\]

\[
\begin{align*}
\{ & - \mapsto 1, \\
\{ & + \mapsto 0 \}
\end{align*}
\]

\[\alpha(Q) = - \quad \beta(Q) = + \]

\[\alpha(P)(\text{id}) = 0 \quad \beta(P)(\text{id}) = 1 \]

\[\alpha(P)(\text{cst0}) = 0 \quad \beta(P)(\text{cst0}) = 0 \]

\[\alpha(P)(\text{cst1}) = 1 \quad \beta(P)(\text{cst1}) = 1 \]

\[\alpha(P)(\text{neg}) = 1 \quad \beta(P)(\text{neg}) = 0 \]
1 Higher-order constrained Horn clauses (HoCHC): satisfiability and safety problems

2 Standard semantics of higher-order logic

3 Monotone semantics satisfies least model property

4 Algorithmic solutions of HoCHC safety problem: 1. via refinement types

5 Automation via prototype tool Horus
Interpret $\rightarrow$ as the monotone function space.

\[ M[\text{int}] := \mathbb{Z} \text{ (ordered discretely)} \]
\[ M[\text{bool}] := \text{lattice } \{0, 1\} \text{ (or } \{f, t\} \text{) with } 0 \sqsubseteq 1 \]
\[ M[\sigma_1 \rightarrow \sigma_2] := M[\sigma_1] \Rightarrow_m M[\sigma_2] \text{ (monotone fns)} \]

**Example:** $A \models_M \exists x : (\text{int } \rightarrow \text{ bool}) \rightarrow \text{ bool } . G$

“There is some monotone predicate $x$ on sets of integers that makes $G$ true in $A$.”

In monotone semantics, satisfiable Horn clauses have least models (because “immediate consequence operator” is monotone) and constructible by Knaster-Tarski.
Examples

\[ \mathcal{M}[\text{int} \to \text{bool}] \quad \text{All sets of integers} \]

\[ \mathcal{M}[\text{(int} \to \text{bool}) \to \text{bool}] \quad \text{All upward-closed (w.r.t.} \subseteq \text{) sets of sets of integers} \]

\[ \mathcal{M}[\text{(int} \to \text{bool}) \to \text{bool}) \to \text{bool}] \quad \text{All upward-closed sets of upward-closed sets of sets of integers} \]

Counter-intuitive (?) Take \( x : (\text{int} \to \text{bool}) \to \text{bool} \).

\[ x \mapsto \{\{1\}\} \not\subseteq \exists y : (\text{int} \to \text{bool}) . \exists z : \text{int} . (x \ y \land y \ z) \]

(\because \text{valuation is invalid:} \{\{1\}\} \notin \mathcal{M}[(\text{int} \to \text{bool}) \to \text{bool}])
Each is good for something

Standard Semantics

😊 Completely standard satisfiability problem (modulo background theory) in higher-order logic.

😊 No least model.

Monotone Semantics

😊 Bespoke satisfiability problem with a restricted class of models.

😊 Least model arising in the usual way.

Can we have the best of both worlds?
I.e. can we specify problems in standard semantics, but solve/compute in monotone semantics?
Standard and monotone semantics are equivalent for the HoCHC Satisfiability Problem

We *can* have the best of both worlds!

**Theorem (Model correspondence)**

*Given a clause (set) $H$, $H$ is satisfiable in the standard semantics iff $H$ is satisfiable in the monotone semantics.*
Proof idea

For each sort of relations $\rho$, monotone and standard semantics are locked in two-sided Galois connections:

\[
\begin{array}{c}
S[\rho] & \overset{I_\rho}{\leftrightarrow} & M[\rho] & \overset{U_\rho}{\leftrightarrow} & S[\rho] \\
\text{Standard} & \text{Monotone} & \text{Standard}
\end{array}
\]

Define, by recursion over sorts:

\[
\begin{align*}
I_{\text{bool}}(b) & := b \\
I_{\text{int} \to \rho}(r) & := I_\rho \circ r \\
I_{\rho_1 \to \rho_2}(r) & := I_{\rho_2} \circ r \circ L_{\rho_1}
\end{align*}
\]

\[
\begin{align*}
J_{\text{bool}}(b) & := b \\
J_{\text{int} \to \rho}(r) & := J_\rho \circ r \\
J_{\rho_1 \to \rho_2}(r) & := J_{\rho_2} \circ r \circ U_{\rho_1}
\end{align*}
\]

where

- $U_\rho$ is the right adjoint of $J_\rho$, i.e., uniquely determined by: for all $a, b$

\[
J_\rho a \subseteq b \iff a \subseteq U_\rho b
\]

- $L_\rho$ is the left adjoint of $I_\rho$
Standard and monotone semantics are equivalent also for HoCHC Safety Problem

**Theorem (Equivalence / Inter-reducibility)**

For all $\Delta, D$ and $G$, T.F.A.E.

(i) HoCHC Safety Problem $\langle \Delta, D, G \rangle$ in **standard semantics** is solvable

(ii) HoCHC Safety Problem $\langle \Delta, D, G \rangle$ in **monotone semantics** is solvable

(iii) In all models of the background theory, the **least valuation** $M[D]$ invalidates $G$ (i.e. $M[G](M[D]) = 0$).

Thus: we can specify problems using the standard semantics, and then solve in the monotone semantics.
1. Higher-order constrained Horn clauses (HoCHC): satisfiability and safety problems

2. Standard semantics of higher-order logic

3. Monotone semantics satisfies least model property

4. Algorithmic solutions of HoCHC safety problem: 1. via refinement types

5. Automation via prototype tool Horus
Refinement types as higher-order invariants

Solving HoCHC problems is about finding (higher-order) symbolic models. Models are valuations.

\[ R : ((\text{int} \to \text{bool}) \to \text{bool}) \to \text{bool} \]

**Symbolic model:**

\[ R \mapsto \lambda f. (\forall g. f \ g \Rightarrow (\forall x. g \ x \Rightarrow \phi) \Rightarrow \psi) \Rightarrow \chi \]

**Dependent refinement type:**

\[ R : f : (g : (x : \text{int} \to \text{bool}^{\langle \phi \rangle}) \to \text{bool}^{\langle \psi \rangle}) \to \text{bool}^{\langle \chi \rangle} \]

Dependency – “\( f : x : T_1 \to T_2 \)” means: for each \( a : T_1 \), the value of \( f \ a \) has type \( T_2[a/x] \).

Refinement – “\( b : \text{bool}^{\langle \varphi \rangle} \)” means: \( b \Rightarrow \varphi \)
Dependent refinement types: syntax and semantics

\[
T ::= \text{bool}(\langle \varphi \rangle) \mid x : \text{int} \rightarrow T \mid T_1 \rightarrow T_2
\]

Refinement at bool: \( \varphi \) is a 1st-order formula of constraint language

Dependence at int: \( x \) can occur freely in \( T \)

Order-ideal semantics: Given a valuation \( \alpha \) of int-sorted vars:

\[
\begin{align*}
\llbracket \text{int} \rrbracket(\alpha) & := \mathbb{Z} \\
\llbracket \text{bool}(\langle \varphi \rangle) \rrbracket(\alpha) & := \{f, \llbracket \varphi \rrbracket(\alpha)\} \\
\llbracket x : \text{int} \rightarrow T \rrbracket(\alpha) & := \prod_{d \in \mathbb{Z}} \llbracket T \rrbracket(\alpha[x \mapsto d])
\end{align*}
\]
Examples of refinement type

Idea: \([\text{bool}\langle \varphi \rangle](\alpha)\) is downward closure of value of \(\varphi\).

1. \([\text{bool}\langle x \leq y \rangle](\{x \mapsto 1, \ y \mapsto 2\}) = \{f, t\}\)

Fact: \(b \in [\text{bool}\langle \varphi \rangle](\alpha) \iff \alpha \models b \Rightarrow \varphi\)

2. A function type.

\[
\begin{align*}
[\text{x:int } \to \text{ bool}\langle \varphi \rangle] & = \prod n \in \mathbb{Z} \cdot [\text{bool}\langle \varphi[n/x] \rangle] \\
& = \{ f \mid \forall n \in \mathbb{Z} \cdot f \ n \in [\text{bool}\langle \varphi[n/x] \rangle] \} \\
& = \{ f \mid \forall n \in \mathbb{Z} \cdot (f \ n \Rightarrow \varphi[n/x]) \} \\
& = \{ f \mid \forall x : \text{int}. (f \ x \Rightarrow \varphi) \}
\end{align*}
\]
Decidable judgement: \( \Gamma \vdash G : T \)

- Type environment \( \Gamma \): finite map from variables to refinement types
- Goal term \( G \): subterm of body of Horn clause
- \( T \): refinement type

Intuition:

\[
\Gamma \vdash G : \text{bool} \langle \varphi \rangle
\]

In symbolic model \( \Gamma \) (i.e. models satisfying \( \Gamma \)), truth of \( G \) is bounded above by constraint \( \varphi \) (or “\( G \) implies \( \varphi \)”).

Thus \( \varphi \) is an over-approximation of \( G \), which may have higher-order subterms.
Some proof rules of typing judgements

(TConstraint) \[ \frac{}{\Gamma \vdash \varphi : \text{bool} \langle \varphi \rangle} \quad \varphi \in Fm \]

(TExists) \[ \frac{\Gamma, x : \iota \vdash G : \text{bool} \langle \varphi \rangle}{\Gamma \vdash \exists x : \iota. G : \text{bool} \langle \psi \rangle} \quad Th \models \varphi \Rightarrow \psi \]

(TAbsl) \[ \frac{\Gamma, x : \text{int} \vdash G : T}{\Gamma \vdash \lambda x : \text{int}. G : x : \text{int} \rightarrow T} \]

(TAppl) \[ \frac{\Gamma \vdash G : x : \text{int} \rightarrow T \quad \Gamma \vdash N : \text{int}}{\Gamma \vdash G \, N : T[N/x]} \]

(TSub) \[ \frac{\Gamma \vdash G : T_1 \quad \vdash T_1 \subseteq T_2}{\Gamma \vdash G : T_2} \]

(Nothing surprising here.)
Subtyping judgement: \( \vdash T_1 \sqsubseteq T_2 \)

Subtyping, \( \sqsubseteq \), captures implication in the background theory \( Th \).

\[
\vdash \text{bool}(\varphi) \sqsubseteq \text{bool}(\psi) \quad (Th \models \varphi \Rightarrow \psi)
\]

\[
\vdash T_1 \sqsubseteq T_2 \quad \vdash x : \text{int} \rightarrow T_1 \sqsubseteq x : \text{int} \rightarrow T_2
\]

\[
\vdash T'_1 \sqsubseteq T_1 \quad \vdash T_2 \sqsubseteq T'_2
\]

\[
\vdash T_1 \rightarrow T_2 \sqsubseteq T'_1 \rightarrow T'_2
\]
Reducing HoCHC to 1st-order Horn constraints

**Theorem (Soundness)**

If \( \Gamma \vdash G : T \) then \( \Gamma \models G : T \).

**A sound approach to solving HoCHC**

Given HoCHC safety problem \( \langle \Delta, D, G \rangle \):

- If there is a type environment \( \Gamma \) (that refines \( \Delta \)) such that \( \vdash D : \Gamma \) and \( \Gamma \vdash G : \text{bool}\langle\text{false}\rangle \), then for each model \( \mathcal{A} \) of background theory, \( \mathcal{M}[\Gamma] \) is a valuation that satisfies \( D \) but refutes \( G \).

- Typability of clauses, \( \Gamma \vdash G : T \), is reducible to 1st-order constrained Horn clause solving. This is more or less standard.

The method is incomplete.
Working example revisited

From safety verification problem:

let \text{add} \ x \ y = x + y
letrec \text{iter} \ f \ s \ n = \text{if} \ n \leq 0 \text{ then} \ s \text{ else} \ f \ n (\text{iter} \ f \ s \ (n - 1))
in \lambda n. \text{assert} \ (n \leq (\text{iter} \ \text{add} \ 0 \ n))

obtain HoCHC safety problem:

\forall x \ y \ z. (z = x + y \Rightarrow \text{Add} \ x \ y \ z)
\forall f \ s \ n \ m. (n \leq 0 \land m = s \Rightarrow \text{Iter} \ f \ s \ n \ m)
\forall f \ s \ n \ m.
\quad (n > 0 \land (\exists p. \text{Iter} \ f \ s \ (n - 1) \ p \land f \ n \ p \ m) \Rightarrow \text{Iter} \ f \ s \ n \ m)
\forall n \ m. (\text{Iter} \ \text{Add} \ 0 \ n \ m \Rightarrow n \leq m)

Goal clause: \( G = \exists m \ n. (\text{Iter} \ \text{Add} \ 0 \ n \ m) \land n > m \)

Task (type checking): Find \( \Gamma \) s.t. \( \vdash D : \Gamma \) and \( \Gamma \vdash G : \text{bool}(\text{false}) \).
Model = valuation, here expressed as refinement type assignment

\[
\begin{align*}
Add & \mapsto (x: \text{int} \to y: \text{int} \to z: \text{int} \to \text{bool}\langle z = x + y \rangle) \\
Iter & \mapsto (x: \text{int} \to y: \text{int} \to z: \text{int} \to \text{bool}\langle 0 < x \Rightarrow y < z \rangle) \to \\
& \quad s: \text{int} \to n: \text{int} \to m: \text{int} \to \text{bool}\langle 0 \leq s \Rightarrow n \leq m \rangle
\end{align*}
\]
For example...

\[ r : \text{int} \]
\[ n : \text{int} \]

\[ \text{Add} : (x : \text{int} \rightarrow \cdots \rightarrow \text{bool}(z = x + y)) \]

\[ \text{Iter} : (x : \text{int} \rightarrow \cdots \rightarrow \text{bool}(0 < x \Rightarrow y < z)) \rightarrow \cdots \rightarrow \text{bool}(0 \leq m \Rightarrow n \leq r) \]
\[
\Gamma \vdash > : (x : \text{int} \to y : \text{int} \to \text{bool}(x > y)) \quad \Gamma \vdash n : \text{int} \quad \Gamma \vdash r : \text{int}
\]

\[
\Gamma \vdash > n : (y : \text{int} \to \text{bool}(n > y)) \quad \Gamma \vdash r : \text{int}
\]

\[
\Gamma \vdash n > r : \text{bool}(n > r)
\]
\[
T_1 = (x: \text{int} \rightarrow \cdots \rightarrow \text{bool}(0 < x \Rightarrow y < z)) \rightarrow m: \text{int} \rightarrow \cdots \rightarrow \text{bool}(0 \leq m \Rightarrow n \leq r)
\]

\[
\Gamma \vdash Add : (x: \text{int} \rightarrow \cdots \rightarrow \text{bool}(z = x + y))
\]

\[
\Gamma \vdash Iter : T_1 \quad \Gamma \vdash Add : (x: \text{int} \rightarrow \cdots \rightarrow \text{bool}(0 < x \Rightarrow y < z))
\]

\[
\Gamma \vdash Iter Add : x: \text{int} \rightarrow \cdots \rightarrow \text{bool}(0 \leq x \Rightarrow y \leq z)
\]

\[
\Gamma \vdash Iter Add \ 0 \ n \ r : \text{bool}(n \leq r)
\]
\( \land : \forall X Y. \text{bool}(X) \rightarrow \text{bool}(Y) \rightarrow \text{bool}(X \land Y) \)

\[ \Gamma \vdash \land : \text{bool}(n \leq r) \rightarrow \text{bool}(n > r) \rightarrow \text{bool}(n \leq r \land n > r) \]

\[ \vdots \]

\[ \Gamma \vdash (\text{Iter Add 0 n r}) \land (n > r) : \text{bool}(n \leq r \land n > r) \]

\[ \Gamma \vdash (\text{Iter Add 0 n r}) \land (n > r) : \text{bool}(\text{false}) \]
Type inference: how to grow a symbolic model

1. From higher-order relational vars: \( \Gamma = \{ \\
\begin{align*}
R : (x : int \rightarrow \text{bool}(Z_1 x)) & \rightarrow \text{bool}(Z_2) \\
S : y : int & \rightarrow (x : int \rightarrow \text{bool}(Z_3 x y))
\end{align*} \)

2. Create refinement template:

3. Check that type environment \( \Gamma \) is a model.

4. Except, whenever forced to check the validity of an implication:

\[
\begin{align*}
Th \models Z_3 n z \Rightarrow Z_1 z & \quad \text{(Sub-Bool)} \\
\text{bool}(Z_3 n z) \sqsubseteq \text{bool}(Z_1 z)
\end{align*}
\]

add clause ‘\( Z_3 n z \Rightarrow Z_1 z \)’ to the (1st-order) Horn constraint system.
1 Higher-order constrained Horn clauses (HoCHC): satisfiability and safety problems

2 Standard semantics of higher-order logic

3 Monotone semantics satisfies least model property

4 Algorithmic solutions of HoCHC safety problem: 1. via refinement types

5 Automation via prototype tool Horus
Prototype tool for solving HoCHC safety problem: Horus

Web interface to Horus: http://mjolnir.cs.ox.ac.uk/horus

Tests

Verification problems taken from MoCHi test suite (Kobayashi et al. PLDI’11) but reexpressed as HoCHC safety problems.

In all the examples (without local assertions), except neg:

- Horus takes around 0.01s to transform the system of clauses and
- Z3 takes around 0.02s to solve the transformed 1st-order system.

Example. In Problem mc91, we verify: $M(n) = 91$ for all $n \leq 101$.

https://github.com/penteract/HigherOrderHornRefinement
Further directions

Related work: see paper on arXiv.

1. Other approaches to reduce HoCHC problems to 1st-order problems (e.g. via Reynolds’ defunctionalisation)
2. Adequacy of HoCHC for safety verification of higher-order programs in general (cf. Blass & Gurevich)