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
$$\begin{cases} \forall \overline{x} \,.\, \big(\mathit{Initial}(\overline{x}) \;\Rightarrow\; \mathit{Reach}(\overline{x})\big) \\ \forall \overline{x} \,.\, \big(\mathit{Reach}(\overline{x}) \;\wedge\; \mathit{Trans}(\overline{x}, \overline{x'}) \;\Rightarrow\; \mathit{Reach}(\overline{x'})\big) \end{cases}$$

Query: $\forall \overline{x} \,.\, \big(\mathit{Reach}(\overline{x}) \Rightarrow \mathit{Safe}(\overline{x})\big)$

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

Logic	Horn	General
Propositional	P	NP
Bernays-Schönfinkel ($\exists^* \forall^*$)	DEXPTIME	NEXPTIME

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.

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

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

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

- $(iter\ f\ s\ n)$ computes $f\,n\,\big(f\,(n-1)\,(f\,(n-2)\,(\cdots(f\,1\,s)\cdots))\big)$.
- Thus $(iter\ add\ 0\ n) = n + (n-1) + \cdots + 1 + 0$.

Say the program is safe if assertion is never violated.

Example: safety verification

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

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:

$$\forall x\, y\, z\, .\, \big(z = x + y \Rightarrow Add\ x\ y\ z\big)$$
$$\forall f\, s\, n\, m\, .\, \big(n \leq 0 \wedge m = s \Rightarrow Iter\ f\ s\ n\ m\big)$$
$$\forall f\, s\, n\, m\, .$$
$$\big(n > 0 \wedge (\exists p.\ Iter\ f\ s\ (n-1)\ p \wedge f\ n\ p\ m) \Rightarrow Iter\ f\ s\ n\ m\big)$$
$$\forall n\, m\, .\, \big(Iter\ Add\ 0\ n\ m \Rightarrow n \leq m\big)$$

Some features of HoCHC

$$\forall x\, y\, z\,.\, \big(z = x + y \Rightarrow \mathit{Add}\ x\ y\ z\big)$$
$$\forall f\, s\, n\, m\,.\, \big(n \leq 0 \wedge m = s \Rightarrow \mathit{Iter}\ f\ s\ n\ m\big)$$
$$\forall f\, s\, n\, m\,.$$
$$\big(n > 0 \wedge (\exists p.\ \mathit{Iter}\ f\ s\ (n - \mathit{1})\ p \wedge f\ n\ p\ m) \Rightarrow \mathit{Iter}\ f\ s\ n\ m\big)$$
$$\forall n\, m\,.\, \big(\mathit{Iter}\ \mathit{Add}\ \mathit{0}\ n\ m \Rightarrow n \leq m\big)$$

- Higher-order "unknown" relation:
  $\mathit{Iter} : (\mathsf{int} \to \mathsf{int} \to \mathsf{int} \to \mathsf{bool}) \to \mathsf{int} \to \mathsf{int} \to \mathsf{int} \to \mathsf{bool}$
- Quantification at higher sort: $\mathsf{int} \to \mathsf{int} \to \mathsf{int} \to \mathsf{bool}$
- Literals headed by variables: $f\ n\ p\ m$

Every model of the system *is* an invariant witnessing safety of the program.

Higher-order constrained Horn clauses (HoCHC): definitions

Relational sorts: $\sigma ::= \mathsf{int} \to \mathsf{bool} \mid \mathsf{int} \to \sigma \mid \sigma \to \sigma'$

Fix a sorting Δ of higher-order relational variables ("unknowns")

$$\begin{array}{rcl} \text{goal } G & ::= & A \mid \varphi \mid G \wedge G \mid G \vee G \mid \exists x{:}\sigma.\, G \\ \text{definite } D & ::= & \mathsf{true} \mid \forall x{:}\sigma.\, D \mid D \wedge D \mid G \Rightarrow X\, x_1 \ldots x_n \end{array}$$

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

- Satisfiability Problem: $\langle \Delta, D \rangle$ is solvable if for all models $\mathcal{A}$ of background theory Th, there is valuation α of Δ 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 α of Δ s.t. $\mathcal{A}, \alpha \models D$, yet $\mathcal{A}, \alpha \not\models G$.

Example: an instance of HoCHC safety problem $\langle \Delta, D, G \rangle$

$$
\begin{array}{ll}
(1) & \forall x\, y\, z\, .\, \big(z = x + y \Rightarrow \mathit{Add}\ x\ y\ z\big) \\
(2) & \forall f\, s\, n\, m\, .\, \big(n \leq 0 \wedge m = s \Rightarrow \mathit{Iter}\ f\ s\ n\ m\big) \\
(3) & \forall f\, s\, n\, m\, . \\
& \quad \big(n > 0 \wedge (\exists p.\, \mathit{Iter}\ f\ s\ (n - \mathit{1})\ p \wedge f\ n\ p\ m) \Rightarrow \mathit{Iter}\ f\ s\ n\ m\big) \\
(4) & \forall n\, m\, .\, \big(\mathit{Iter}\ \mathit{Add}\ \mathit{0}\ n\ m \Rightarrow n \leq m\big)
\end{array}
$$

- Sorting Δ of relational variables:

$$
\begin{cases}
\mathit{Add} & : \quad \mathsf{int} \to \mathsf{int} \to \mathsf{int} \to \mathsf{bool} \\
\mathit{Iter} & : \quad (\mathsf{int} \to \mathsf{int} \to \mathsf{int} \to \mathsf{bool}) \to \mathsf{int} \to \mathsf{int} \to \mathsf{int} \to \mathsf{bool}
\end{cases}
$$

- Definite formula $D = (1) \wedge (2) \wedge (3)$.
- Goal formula $G = \neg(4) = \exists n\, m\, .\, \big((\mathit{Iter}\ \mathit{Add}\ \mathit{0}\ n\ m) \wedge m < n\big)$.

Safety problem $\langle \Delta, D, G \rangle$ is solvable. I.e. w.r.t. the unique model of ZLA ($\because$ complete theory), there is a valuation satisfying D but refuting G.

Definite clauses as logic programs

Systems of definite clauses can be presented (equivalently) in program form.

$$Add = \lambda x\, y\, z.\, (z = x + y)$$

$$Iter = \lambda f\, s\, n\, m\, .\left(\begin{array}{rl} & (n \leq 0 \;\wedge\; m = s) \\ \vee & \exists p\, .\, 0 < n \;\wedge\; Iter\, f\; s\, (n - 1)\, p \;\wedge\; f\; n\; p\; m \end{array} \right)$$

Outline

Standard semantics of higher-order logic

Sorts: $\sigma \ ::= \ \mathsf{one} \mid \mathsf{bool} \mid \mathsf{int} \mid \sigma_1 \to \sigma_2$

$$\begin{array}{rcl} \mathcal{S}[\![\mathsf{one}]\!] & \coloneqq & \{\star\} \\ \mathcal{S}[\![\mathsf{bool}]\!] & \coloneqq & \{0,1\} \\ \mathcal{S}[\![\mathsf{int}]\!] & \coloneqq & \mathbb{Z} \\ \mathcal{S}[\![\sigma_1 \to \sigma_2]\!] & \coloneqq & \mathcal{S}[\![\sigma_1]\!] \Rightarrow \mathcal{S}[\![\sigma_2]\!] \quad (\textit{all} \text{ functions}) \end{array}$$

Syntax: Standard presentation as a simply-typed λ-calculus with logical constants: $\neg, \wedge, \vee, \forall_\sigma, \exists_\sigma$, etc.

$$\neg : \mathsf{bool} \to \mathsf{bool} \qquad \forall_\sigma, \exists_\sigma : (\sigma \to \mathsf{bool}) \to \mathsf{bool}$$

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

Semantics: completely standard.

Example: $\mathcal{A} \ \models_{\mathcal{S}} \ \exists x : (\mathsf{int} \to \mathsf{bool}) \to \mathsf{bool}\, .\, G$

"There is some predicate x on sets of integers that makes G true in $\mathcal{A}$."

Counterexample:

$$\begin{cases} P & : & ((\mathsf{one} \to \mathsf{bool}) \to \mathsf{bool}) \to \mathsf{bool} \\ Q & : & \mathsf{one} \to \mathsf{bool} \end{cases}$$

$$\forall x : (\mathsf{one} \to \mathsf{bool}) \to \mathsf{bool} \,.\, \big(x \; Q \Rightarrow P \; x \big)$$

Theorem

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

(Least with respect to inclusion of relations.)

$\forall x. (x\ Q \Rightarrow P\ x)$ **has two minimal models (=valuations)** α **&** β

$P : ((\mathsf{one} \to \mathsf{bool}) \to \mathsf{bool}) \to \mathsf{bool} \qquad Q : \mathsf{one} \to \mathsf{bool}$

$$\mathcal{S}[\![\mathsf{one}]\!] := \{\star\}$$

$$\mathcal{S}[\![\mathsf{one} \to \mathsf{bool}]\!] := \{\, \underbrace{\{\star \mapsto 0\}}_{-}, \quad \underbrace{\{\star \mapsto 1\}}_{+} \,\}$$

$\mathcal{S}[\![(\mathsf{one} \to \mathsf{bool}) \to \mathsf{bool}]\!] :=$

$$\left\{ \underbrace{\left\{ \begin{array}{c} - \mapsto 0 \\ + \mapsto 1 \end{array} \right\}}_{\mathrm{id}}, \underbrace{\left\{ \begin{array}{c} - \mapsto 0 \\ + \mapsto 0 \end{array} \right\}}_{\mathrm{cst0}}, \underbrace{\left\{ \begin{array}{c} - \mapsto 1 \\ + \mapsto 1 \end{array} \right\}}_{\mathrm{cst1}}, \underbrace{\left\{ \begin{array}{c} - \mapsto 1 \\ + \mapsto 0 \end{array} \right\}}_{\mathrm{neg}} \right\}$$

$$\begin{array}{rcl} \alpha(Q) & = & - \\ \alpha(P)(\mathrm{id}) & = & 0 \\ \alpha(P)(\mathrm{cst0}) & = & 0 \\ \alpha(P)(\mathrm{cst1}) & = & 1 \\ \alpha(P)(\mathrm{neg}) & = & 1 \end{array} \qquad \begin{array}{rcl} \beta(Q) & = & + \\ \beta(P)(\mathrm{id}) & = & 1 \\ \beta(P)(\mathrm{cst0}) & = & 0 \\ \beta(P)(\mathrm{cst1}) & = & 1 \\ \beta(P)(\mathrm{neg}) & = & 0 \end{array}$$

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

Monotone semantics of higher-order logic

Interpret $\rightarrow$ as the monotone function space.

$$
\begin{array}{rcl}
\mathcal{M}[\![\mathsf{int}]\!] & := & \mathbb{Z} \quad \text{(ordered discretely)} \\
\mathcal{M}[\![\mathsf{bool}]\!] & := & \text{lattice } \{0,1\} \text{ (or } \{\mathsf{f},\mathsf{t}\}) \text{ with } 0 \sqsubseteq 1 \\
\mathcal{M}[\![\sigma_1 \rightarrow \sigma_2]\!] & := & \mathcal{M}[\![\sigma_1]\!] \Rightarrow_m \mathcal{M}[\![\sigma_2]\!] \quad \text{(monotone fns)}
\end{array}
$$

Example: $\mathcal{A} \vDash_{\mathcal{M}} \exists x : (\mathsf{int} \rightarrow \mathsf{bool}) \rightarrow \mathsf{bool} \,.\, G$

> "There is some monotone predicate x on sets of integers that makes G true in $\mathcal{A}$."

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

Examples

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

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

$$x \mapsto \{\{1\}\} \quad \not\models \quad \exists y : (\mathsf{int} \to \mathsf{bool}) \, . \, \exists z : \mathsf{int} \, . \, \big(x\, y \wedge y\, z\big)$$

($\because$ valuation is invalid: $\{\{1\}\} \notin \mathcal{M}[\![(\mathsf{int} \to \mathsf{bool}) \to \mathsf{bool}]\!]$)

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 ρ, monotone and standard semantics are locked in **two-sided Galois connections**:

$$\mathcal{S}[\![\rho]\!] \quad \underset{\mathsf{L}_\rho}{\overset{\mathsf{I}_\rho}{\leftrightarrows}} \quad \mathcal{M}[\![\rho]\!] \quad \underset{\mathsf{J}_\rho}{\overset{\mathsf{U}_\rho}{\leftrightarrows}} \quad \mathcal{S}[\![\rho]\!]$$

Standard — Monotone — Standard

Define, by recursion over sorts:

$$\begin{aligned} \mathsf{I}_{\mathsf{bool}}(b) &:= b \\ \mathsf{I}_{\mathsf{int}\to\rho}(r) &:= \mathsf{I}_\rho \circ r \\ \mathsf{I}_{\rho_1\to\rho_2}(r) &:= \mathsf{I}_{\rho_2} \circ r \circ \mathsf{L}_{\rho_1} \end{aligned} \qquad \begin{aligned} \mathsf{J}_{\mathsf{bool}}(b) &:= b \\ \mathsf{J}_{\mathsf{int}\to\rho}(r) &:= \mathsf{J}_\rho \circ r \\ \mathsf{J}_{\rho_1\to\rho_2}(r) &:= \mathsf{J}_{\rho_2} \circ r \circ \mathsf{U}_{\rho_1} \end{aligned}$$

where

- U_ρ is the right adjoint of J_ρ, i.e., uniquely determined by: for all a, b

$$J_\rho\, a \subseteq b \;\Leftrightarrow\; a \subseteq U_\rho\, b$$

- L_ρ is the left adjoint of I_ρ

Standard and monotone semantics are equivalent also for HoCHC Safety Problem

Theorem (Equivalence / Inter-reducibility)

For all Δ, 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 $\mathcal{M}[\![D]\!]$ invalidates G (i.e. $\mathcal{M}[\![G]\!](\mathcal{M}[\![D]\!]) = 0$).*

Thus: we can specify problems using the standard semantics, and then solve in the monotone semantics.

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

Refinement types as higher-order invariants

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

$$R : ((\mathsf{int} \to \mathsf{bool}) \to \mathsf{bool}) \to \mathsf{bool}$$

Symbolic model:

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

Dependent refinement type:

$$R : f\!: (g\!: (x\!: \mathsf{int} \to \mathsf{bool}\langle\phi\rangle) \to \mathsf{bool}\langle\psi\rangle) \to \mathsf{bool}\langle\chi\rangle$$

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 – "$b : \mathsf{bool}\langle\varphi\rangle$" means: $b \Rightarrow \varphi$

Dependent refinement types: syntax and semantics

$$T \;\; \coloneqq \;\; \mathsf{bool}\langle\varphi\rangle \mid x{:}\mathsf{int} \to T \mid T_1 \to T_2$$

Refinement at bool: φ is a 1st-order formula of constraint language
Dependence at int: x can occur freely in T

Order-ideal semantics: Given a valuation α of int-sorted vars:

$$\begin{aligned} [\![\mathsf{int}]\!](\alpha) &\coloneqq \mathbb{Z} \\ [\![\mathsf{bool}\langle\varphi\rangle]\!](\alpha) &\coloneqq \{\mathsf{f}, [\![\varphi]\!](\alpha)\} \\ [\![x{:}\mathsf{int} \to T]\!](\alpha) &\coloneqq \prod d \in \mathbb{Z} \,.\, [\![T]\!](\alpha[x \mapsto d]) \end{aligned}$$

Examples of refinement type

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

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

Fact: $b \in [\![\mathsf{bool}\langle\varphi\rangle]\!](\alpha) \ \Leftrightarrow\ \alpha \vDash b \Rightarrow \varphi$

2. A function type.

$$
\begin{array}{rl}
 & [\![x{:}\mathsf{int} \to \mathsf{bool}\langle\varphi\rangle]\!] \\
= & \prod n \in \mathbb{Z} \,.\, [\![\mathsf{bool}\langle\varphi[n \,/\, x]\rangle]\!] \\
= & \{f \mid \forall n \in \mathbb{Z} \,.\, f\, n \in [\![\mathsf{bool}\langle\varphi[n \,/\, x]\rangle]\!]\} \\
= & \{f \mid \forall n \in \mathbb{Z} \,.\, (f\, n \Rightarrow \varphi[n/x])\} \\
= & \{f \mid \forall x : \mathsf{int} \,.\, (f\, x \Rightarrow \varphi)\}
\end{array}
$$

Decidable judgement: $\Gamma \vdash G : T$

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

Intuition:

$$\Gamma \vdash G : \mathsf{bool}\langle\varphi\rangle$$

> In symbolic model Γ (i.e. models satisfying Γ), truth of G is bounded above by constraint φ (or "G implies φ").

Thus φ is an over-approximation of G, which may have higher-order subterms.

Some proof rules of typing judgements

$$\text{(TConstraint)} \quad \frac{}{\Gamma \vdash \varphi : \mathsf{bool}\langle\varphi\rangle} \; \varphi \in Fm$$

$$\text{(TExists)} \quad \frac{\Gamma, x : \iota \vdash G : \mathsf{bool}\langle\varphi\rangle}{\Gamma \vdash \exists x{:}\iota.\, G : \mathsf{bool}\langle\psi\rangle} \; Th \vDash \varphi \Rightarrow \psi$$

$$\text{(TAbsI)} \quad \frac{\Gamma, x : \mathsf{int} \vdash G : T}{\Gamma \vdash \lambda x{:}\mathsf{int}.\, G : x{:}\mathsf{int} \to T}$$

$$\text{(TAppI)} \quad \frac{\Gamma \vdash G : x{:}\mathsf{int} \to T \qquad \Gamma \vdash N : \mathsf{int}}{\Gamma \vdash G\, N : T[N/x]}$$

$$\text{(TSub)} \quad \frac{\Gamma \vdash G : T_1 \qquad \vdash T_1 \sqsubseteq T_2}{\Gamma \vdash G : T_2}$$

(Nothing surprising here.)

Subtyping judgement: $\vdash T_1 \sqsubseteq T_2$

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

$$\frac{}{\vdash \mathsf{bool}\langle\varphi\rangle \sqsubseteq \mathsf{bool}\langle\psi\rangle}\ (Th \vDash \varphi \Rightarrow \psi)$$

$$\frac{\vdash T_1 \sqsubseteq T_2}{\vdash x{:}\mathsf{int} \to T_1 \sqsubseteq x{:}\mathsf{int} \to T_2}$$

$$\frac{\vdash T_1' \sqsubseteq T_1 \qquad \vdash T_2 \sqsubseteq T_2'}{\vdash T_1 \to T_2 \sqsubseteq T_1' \to T_2'}$$

Reducing HoCHC to 1st-order Horn constraints

Theorem (Soundness)

If $\Gamma \vdash G : T$ *then* $\Gamma \vDash G : T$*.*

A sound approach to solving HoCHC

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

- If there is a type environment Γ (that refines Δ) such that $\vdash D : \Gamma$ and $\Gamma \vdash G : \mathrm{bool}\langle \mathrm{false} \rangle$, 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:

$$\begin{array}{l}\mathsf{let}\ add\ x\ y = x + y\\ \mathsf{letrec}\ iter\ f\ s\ n = \mathsf{if}\ n \le 0\ \mathsf{then}\ s\ \mathsf{else}\ f\ n\ (iter\ f\ s\ (n - 1))\\ \qquad \mathsf{in}\ \lambda n.\,\mathsf{assert}\ \big(n \le (iter\ add\ 0\ n)\big)\end{array}$$

obtain HoCHC safety problem:

$$\begin{array}{l}\forall x\, y\, z\,.\,\big(z = x + y \Rightarrow Add\ x\ y\ z\big)\\ \forall f\, s\, n\, m\,.\,\big(n \le 0 \wedge m = s \Rightarrow Iter\ f\ s\ n\ m\big)\\ \forall f\, s\, n\, m\,.\\ \qquad \big(n > 0 \wedge (\exists p.\, Iter\ f\ s\ (n - 1)\ p \wedge f\ n\ p\ m) \Rightarrow Iter\ f\ s\ n\ m\big)\\ \forall n\, m\,.\,\big(Iter\ Add\ 0\ n\ m \Rightarrow n \le m\big)\end{array}$$

Goal clause: $G = \exists m\, n\,.\,(Iter\ Add\ 0\ n\ m) \wedge n > m$

Task (type checking): Find Γ s.t. $\vdash D : \Gamma$ and $\Gamma \vdash G : \mathsf{bool}\langle \mathsf{false}\rangle$.

Working example problem: a solution

Model = valuation, here expressed as refinement type assignment

$$\begin{cases} \mathit{Add} \;\mapsto\; x{:}\mathsf{int} \to y{:}\mathsf{int} \to z{:}\mathsf{int} \to \mathsf{bool}\langle z = x + y\rangle \\ \mathit{Iter} \;\mapsto\; \big(x{:}\mathsf{int} \to y{:}\mathsf{int} \to z{:}\mathsf{int} \to \mathsf{bool}\langle \mathit{0} < x \Rightarrow y < z\rangle\big) \to \\ \qquad\qquad s{:}\mathsf{int} \to n{:}\mathsf{int} \to m{:}\mathsf{int} \to \mathsf{bool}\langle \mathit{0} \le s \Rightarrow n \le m\rangle \end{cases}$$

For example...

$$\underbrace{\begin{array}{r l} r : & \mathsf{int} \\ n : & \mathsf{int} \\ Add : & (x\!: \mathsf{int} \to \cdots \to \mathsf{bool}\langle z = x + y \rangle) \\ Iter : & (x\!: \mathsf{int} \to \cdots \to \mathsf{bool}\langle 0 < x \Rightarrow y < z \rangle) \to \cdots \to \mathsf{bool}\langle 0 \le m \Rightarrow n \le r \rangle \end{array}}_{\Gamma}$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash > : (x{:}\,\mathsf{int} \to y{:}\,\mathsf{int} \to \mathsf{bool}\langle x > y\rangle)} \qquad \dfrac{}{\Gamma \vdash n : \mathsf{int}}}{\Gamma \vdash > n : (y{:}\,\mathsf{int} \to \mathsf{bool}\langle n > y\rangle)} \qquad \dfrac{}{\Gamma \vdash r : \mathsf{int}}}{\Gamma \vdash n > r : \mathsf{bool}\langle n > r\rangle}$$

$$T_1 = (x\colon \mathrm{int} \to \cdots \to \mathrm{bool}\langle 0 < x \Rightarrow y < z\rangle) \to m\colon \mathrm{int} \to \cdots \to \mathrm{bool}\langle 0 \le m \Rightarrow n \le r\rangle$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \mathit{Iter} : T_1} \qquad \dfrac{\dfrac{}{\Gamma \vdash \mathit{Add} : (x\colon \mathrm{int} \to \cdots \to \mathrm{bool}\langle z = x + y\rangle)}}{\Gamma \vdash \mathit{Add} : (x\colon \mathrm{int} \to \cdots \to \mathrm{bool}\langle 0 < x \Rightarrow y < z\rangle)}}{\Gamma \vdash \mathit{Iter}\ \mathit{Add} : x\colon \mathit{int} \to \cdots \to \mathrm{bool}\langle 0 \le x \Rightarrow y \le z\rangle}}{\vdots}$$

$$\dfrac{}{\Gamma \vdash \mathit{Iter}\ \mathit{Add}\ 0\ n\ r : \mathrm{bool}\langle n \le r\rangle}$$

$$\cfrac{\cfrac{\cfrac{\wedge : \forall XY.\, \mathsf{bool}\langle X\rangle \to \mathsf{bool}\langle Y\rangle \to \mathsf{bool}\langle X \wedge Y\rangle}{\Gamma \vdash \wedge : \mathsf{bool}\langle n \le r\rangle \to \mathsf{bool}\langle n > r\rangle \to \mathsf{bool}\langle n \le r \wedge n > r\rangle}}{\vdots}}{\cfrac{\Gamma \vdash (\mathit{Iter}\ \mathit{Add}\ 0\ n\ r) \wedge (n > r) : \mathsf{bool}\langle n \le r \wedge n > r\rangle}{\Gamma \vdash (\mathit{Iter}\ \mathit{Add}\ 0\ n\ r) \wedge (n > r) : \mathsf{bool}\langle \mathsf{false}\rangle}}$$

Type inference: how to grow a symbolic model

1. From higher-order relational vars: $\begin{cases} R : (\mathsf{int} \to \mathsf{bool}) \to \mathsf{bool} \\ S : \mathsf{int} \to (\mathsf{int} \to \mathsf{bool}) \end{cases}$

2. Create refinement template:

$$\Gamma = \begin{cases} R : (x{:}\mathsf{int} \to \mathsf{bool}\langle Z_1\ x\rangle) \to \mathsf{bool}\langle Z_2\rangle \\ S : y{:}\mathsf{int} \to (x{:}\mathsf{int} \to \mathsf{bool}\langle Z_3\ x\ y\rangle) \end{cases}$$

3. Check that type environment Γ is a model.

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

$$\frac{Th \vDash Z_3\, n\, z \Rightarrow Z_1\, z}{\mathsf{bool}\langle Z_3\ n\ z\rangle \sqsubseteq \mathsf{bool}\langle Z_1\ z\rangle}\ (\text{Sub-Bool})$$

add clause '$Z_3\, n\, z \Rightarrow Z_1\, z$' to the (1st-order) Horn constraint system.

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

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 rexpressed as HoCHC safety problems.

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

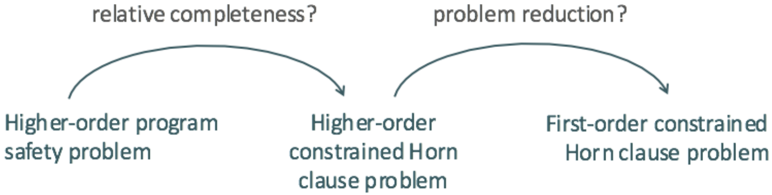
- Horus takes around 0.01s to transform the system of clauses and
- Z3 takes around 0.02s to solve the transfromed 1st-order system.

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)