UNBOUNDED VERIFICATION WITH HORN CLAUSES

AWS ALBARGHOUTHI

Abstract. These notes describe the process of encoding program executions as a formula over *constrained Horn clauses*, a class of first-order logic formulas. Horn clauses allow us to encode the search for a Hoare-logic annotation of the program that proves its correctness.

1. Motivating Example

We begin with an example illustrating constrained Horn clauses and how they can encode Hoare-logic proofs.

A Hoare-style Proof Consider the following program and the associated Hoare triple:

$$\begin{aligned}&\{x > 0\}\\&y \leftarrow x;\\&z \leftarrow x + y\\&\{z > 0\}\end{aligned}$$

To prove that the above Hoare triple is valid, we follow the composition rule of Hoare-logic:

$$\frac{\{\phi\}P_1\{\psi\} \qquad \{\psi\}P_2\{\chi\}}{\{\phi\}P_1; P_2\{\chi\}}\ \textsc{Composition}$$

Reading the rule upwards, to prove the above Hoare triple, we need to construct two valid Hoare triples of the following form:

$$\{x > 0\}\ y \leftarrow x\ \{r(x, y, z)\}$$

$$\{r(x, y, z)\}\ z \leftarrow x + y\ \{z > 0\}$$

where $r(x, y, z)$ is some formula over x, y, z that we need to discover. One solution for $r(x, y, z)$ is $x > 0 \wedge y > 0$. This gives us the following two valid Hoare triples:

$$\{x > 0\}\ y \leftarrow x\ \{x > 0 \wedge y > 0\}$$

$$\{x > 0 \wedge y > 0\}\ z \leftarrow x + y\ \{z > 0\}$$

which following the composition rule, imply that our original Hoare triple is valid.

Encoding Above, we showed how to pose the search for a Hoare-logic annotation as a search for a relation $r(x, y, z)$ over program variables. We can view $r(x, y, z)$ as a relation in first-order logic, and generate a set of constraints whose models give solutions of r.

Consider the following formulas, C_1 and C_2, which encode the two Hoare triples above:

$$C_1 \triangleq \forall V, V'.\, (x > 0 \wedge \mathsf{enc}(y = x)) \implies r(x', y', z')$$
$$C_2 \triangleq \forall V, V'.\, (r(x, y, z) \wedge \mathsf{enc}(z = x + y)) \implies r(x', y', z')$$

Both of these formulas are over the theory of linear integer arithmetic (LIA), where we also have a relation symbol r, a ternary relation over integers, i.e., in $\mathbb{Z}^3$. Since there are no free variables in C_1, C_2, a model for $m \models C_1 \wedge C_2$ will only give an interpretation for $r(x, y, z)$.

The two formulas C_1 and C_2 look very much like our encodings for checking Hoare triples from the previous lecture. By construction, any model m of C_1 must be such that for any initial state where $x > 0$, if we execute $y = x$, we end in a state in $r(x', y', z')$; note that we apply the relation r to the final-state variables. So any interpretation of r must result in a valid Hoare triple $\{x > 0\}\ y \leftarrow x\ \{r(x, y, z)\}$.

Recall that a model m maps r to a susbet of $\mathbb{Z}^3$, we will only consider subsets of $\mathbb{Z}^3$ that we can write as formulas in LIA; therefore, we cannot have a model, for instance, that contains only prime numbers, a relation that is not representable in our simple theory of LIA.

One possible m is the one that sets $r(x, y, z)$ to the formula $x > 0 \wedge y > 0$. If we plug in this formula for occurrences of r in C_1, C_2, we get the following formulas:

$$m(C_1) \triangleq \forall V, V'.\, (x > 0 \wedge \mathsf{enc}(y = x)) \implies x' > 0 \wedge y' > 0$$
$$m(C_2) \triangleq \forall V, V'.\, (x > 0 \wedge y > 0 \wedge \mathsf{enc}(z = x + y)) \implies x' > 0 \wedge y' > 0$$

Both formulas are valid.

2. Constrained Horn Clauses

A *constrained Horn clause* C, or Horn clause for short, is a first-order logic formula of the form

$$r_1(\vec{v}_1) \wedge r_2(\vec{v}_2) \wedge \ldots \wedge r_{n-1}(\vec{v}_{n-1}) \wedge \varphi \implies H_C$$

where:

- each relation $r_i \in R$ is of arity equal to the length of the vector of variables $\vec{v}_i$;
- φ is a conjunction of atoms over the first-order theory, which contain non relation symbols outside those interpreted by the theory (e.g., ϕ could be $x > 0 \wedge y > 0$);
- the left-hand side of the implication ($\implies$) is called the *body* of C; and
- H_C, the *head* of C, is either a relation application $r_n(\vec{v}_n)$ or an interpreted formula φ'.
- All free variables are assumed to be universally quantified, e.g., $x + y > 0 \implies r(x, y)$ means $\forall x, y.\, x + y > 0 \implies r(x, y)$.

Semantics. We will write $\mathcal{C}$ for a set of clauses $\{C_1, \ldots, C_n\}$. Let G be a graph over relation symbols such that there is an edge (r_1, r_2) iff r_1 appears in the body of some clause C_i and r_2 appears in its head. We say that $\mathcal{C}$ is *recursive* iff G has a cycle. The set $\mathcal{C}$ is *satisfiable* if there exists an interpretation m of relation symbols r_i such that every clause $C \in \mathcal{C}$ is valid. We say that m satisfies $\mathcal{C}$ (denoted $m \models \bigwedge_{C_i \in \mathcal{C}} C_i$) iff for all $C \in \mathcal{C}$, mC is valid (i.e., equivalent to *true*), where mC is

C with every relation application $r(\vec{v})$ replaced by its interepretation in m, which we denote as $m(r(\vec{v}))$.

3. Constructing Horn Clauses from Programs

We now consider programs P of the form:

$$P_{pre}; \textsf{while } b \textsf{ do } P_{body}$$

We would like to show that a Hoare triple $\{\phi\}P\{\psi\}$ is valid. To do so, we will generate a number of Horn clauses whose solution is an inductive loop invariant:

$$\begin{aligned}
C_1 &\triangleq (\phi \wedge \mathsf{enc}(P_{pre})) \Longrightarrow inv(V') && \text{initiation}\\
C_2 &\triangleq (inv(V) \wedge b \wedge \mathsf{enc}(P_{body})) \Longrightarrow inv(V') && \text{consecution}\\
C_3 &\triangleq inv(V) \wedge \neg b \Longrightarrow \psi && \text{safety}
\end{aligned}$$

A model $m \models C_1 \wedge C_2 \wedge C_3$ gives an interpretation of *inv* that is an inductive loop invariant.

The encoding for our program model is sound and complete.

Theorem 3.1 (Soundness)**.** *Let $m \models C_1 \wedge C_2 \wedge C_3$. The predicate $m(inv)$ is an inductive loop invariant.*

Theorem 3.2 (Completeness)**.** *If $C_1 \wedge C_2 \wedge C_3$ is unsatisfiable, then $\{\phi\}$ P $\{\psi\}$ is not a valid Hoare triple.*

While the above theorems assure that our encoding is correct; in practice, checking satisfiability of constrained Horn clauses in LIA is undecidable.

Example 3.3. Consider the following program from your assignment:

$$\begin{array}{l}
\{x \geqslant 0 \wedge y > 0\}\\
r \leftarrow x;\\
q \leftarrow 0;\\
\textsf{while } r \geqslant y \textsf{ do}\\
\quad r \leftarrow r - y;\\
\quad q \leftarrow q + 1;\\
\{x = y * q + r \wedge 0 \leqslant r < y\}
\end{array}$$

We show the encoding below:

$$\begin{aligned}
C_1 &\triangleq x \geqslant 0 \wedge y > 0 \wedge \mathsf{enc}(P_{pre}) \Longrightarrow inv(x', y', r', q') && \text{initiation}\\
C_2 &\triangleq inv(x, y, r, q) \wedge r \geqslant y \wedge \mathsf{enc}(P_{body}) \Longrightarrow inv(x', y', r', q') && \text{consecution}\\
C_3 &\triangleq inv(x, y, r, q) \wedge r < y \Longrightarrow x = y * q + r \wedge 0 \leqslant r < y && \text{safety}
\end{aligned}$$

where $\mathsf{enc}(P_{pre})$ and $\mathsf{enc}(P_{body})$ are as described in the previous lectures.

4. Constructing Horn Clauses from Arbitrary Programs

Loops In the previous section, we saw how to encode the verification problem for programs with a single loop. We now consider programs with an arbitrary number of loops.

To do so, we assume each statement P in the program has a unique line number, denoted by $\ell(P)$, and a child or two, $\ell_1(P)$ and $\ell_2(P)$, denoting the true and false branches of a while loop or if statement. The following definition of $\mathsf{encHorn}$ demonstrates how to take a program statement and encode it as a set of Horn clauses. Note that if and while statements require two Horn clauses, one for each possible branch taken.

$$\begin{aligned}
\mathsf{encHorn}(x \leftarrow a) &\triangleq inv_i(V) \wedge \mathsf{enc}(x \leftarrow a) \Longrightarrow inv_j(V') \\
\mathsf{encHorn}(\mathsf{if}\ b\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2) &\triangleq \{inv_i(V) \wedge b \Longrightarrow inv_j(V), \\
&\qquad inv_i(V) \wedge \neg b \Longrightarrow inv_k(V)\} \\
\mathsf{encHorn}(\mathsf{while}\ b\ \mathsf{do}\ P_1) &\triangleq \{inv_i(V) \wedge b \Longrightarrow inv_j(V), \\
&\qquad inv_i(V) \wedge \neg b \Longrightarrow inv_k(V)\}
\end{aligned}$$

where above $\ell(P) = i$, $\ell_1(P) = j$, and $\ell_2(P) = k$, for every case of P considered.

Now, given a program P, we apply $\mathsf{encHorn}$ to every statement in P, i.e., to every while statement, if statement, and assignment statement, and collect all Horn clauses in a set. Let $\mathcal{C}$ be the set of Horn clauses collected for P. Assume that P's first statement is labeled *en* (entry) and last statement is labeled *ex* (exit). Then, to prove a Hoare triple $\{\phi\}\ P\ \{\psi\}$, we solve the following Horn clauses:

$$\mathcal{C} \cup \{\phi \Longrightarrow inv_{en}(V),\ inv_{ex}(V) \Longrightarrow \psi\}$$

The two additional Horn clauses signify that the set of initial states must be in the invariant at statement *en*, and the invariant at the exit statement must be in ψ.

Recursion We now consider the problem of encoding programs with procedures. Given a procedure f that takes one input and returns one output, we encode the procedure as a transition relation $f(x, y)$, where x denotes the input to the function and the y the output. This transition relation is called a *function summary*, as it captures the effect of the input–output behavior of a function while hiding the internal computation.

For simplicity, we show how to perform the encoding for a single recursive function, f, by redefining the encoding function $\mathsf{enc}(f)$ as follows:

$$\begin{aligned}
\mathsf{enc}(x \leftarrow a) &\triangleq x' = a \wedge \bigwedge_{y \neq x, y \in V} y' = y \\
\mathsf{enc}(\mathsf{if}\ b\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2) &\triangleq (b \Longrightarrow \mathsf{enc}(P_1)) \wedge (\neg b \Longrightarrow \mathsf{enc}(P_2)) \\
\mathsf{enc}(P_1; P_2) &\triangleq \exists V''.\ trans_1(V, V'') \wedge trans_2(V'', V') \\
&\qquad \text{where } trans_1(V, V') \equiv \mathsf{enc}(P_1) \\
&\qquad\qquad trans_2(V, V') \equiv \mathsf{enc}(P_2) \\
\mathsf{enc}(y \leftarrow f(x)) &\triangleq f(x, y') \wedge \bigwedge_{z \neq y, z \in V} z' = z
\end{aligned}$$

The following Horn clause encodes the transition relation $f(x, y)$:

Now, suppose we want to show the that following Hoare triple is valid:

$$\{\phi\}\ y \leftarrow f(x)\ \{\psi\}$$

We encode the following Horn clauses:

$$\mathsf{enc}(f) \implies f(x, y)$$
$$\phi \wedge f(x, y) \implies \psi$$

The first Horn clause encodes the relation $f(x, y)$; the second encodes the Hoare triple, using the transition relation for f.

Example 4.1. Consider the following popular recursive function, called McCarthy 91:

$$
\begin{array}{l}
mc(p): \\
\mathsf{if}\ p > 100 \\
\quad r \leftarrow p - 10 \\
\mathsf{else} \\
\quad p1 \leftarrow p + 11 \\
\quad p2 \leftarrow mc(p1) \\
\quad r \leftarrow mc(p2)
\end{array}
$$

where r is the return value.

We want to show the following Hoare triple:

$$\{true\}\ r \leftarrow mc(p)\ \{r \geqslant 91\}$$

Using the above encoding, we get:

$$
\begin{aligned}
p > 100 \wedge r = p - 10 &\implies mc(p, r) \\
p \leqslant 100 \wedge p1 \leftarrow p + 11 \wedge mc(p1, p2) \wedge mc(p2, r) &\implies mc(p, r) \\
true \wedge mc(p, r) &\implies r \geqslant 91
\end{aligned}
$$

One solution for the above is the interpretation that sets $mc(p, r)$ to the following relation:

$$mc(p, r) \equiv r \geqslant 91$$

In other words, in order to prove that that return value of mc is always greater than or equal to 91, all we have to do is to assume the summary that it always returns a value greater than or equal to 91, regardless of the output.

5. Solving Constrained Horn Clauses

We now discuss how to solve a set of Horn clauses $\mathcal{C}$. First, we begin by showing how to compute the *least fixpoint*—i.e., the smallest possible solution for the predicate symbols. This process may not terminate, and therefore we will then apply some approximation.

(1) Initially, the solution for any predicate r is set to *false*; we denote this by $sol(r) = false$.

(2) Pick any clause in $\mathcal{C}$ of the following form

$$r_1(\vec{v}_1) \land r_2(\vec{v}_2) \land \ldots \land r_{n-1}(\vec{v}_{n-1}) \land \varphi \Longrightarrow r_n(\vec{v}_n)$$

that is, the head of the clause is a predicate application. Then, set

$$sol(r_n) = sol(r_n) \lor \exists V.\, sol(r_1(\vec{v}_1)) \land sol(r_2(\vec{v}_2)) \land \ldots \land sol(r_{n-1}(\vec{v}_{n-1})) \land \varphi$$

where V is the set of all variables that are not in $\vec{v}_n$.

(3) Repeat 2 until $sol(r_i)$ reach a fixpoint.

Once we've computed solutions for all r_i, we can check if the solution validates clauses whose head is an interpreted formula. If that's the case, then the clauses $\mathcal{C}$ are satisfiable; otherwise, they are not.

Predicate abstraction The above process terminates assuming we're working with propositional logic; however, in general it may not terminate. The reason is that there may be infinitely many logically incomparable formulas added to the solution, never arriving at a fixpoint. To work around this, we can over-approximate the least fixpoint by fixing a finite language of possible solutions. We do this with predicate abstraction.

We assume we are given a finite set of predicates *Preds*—atomic formulas of the form, e.g., $x > 0$. Given any formula φ, we can compute the strongest formula over *Preds* that subsumes φ. There are two possibilities:

- **Cartesian Abstraction** computes the strongest formula without disjunctions. It is defined as follows:

  $$\alpha_C(\varphi) \triangleq \bigwedge\{p \mid \varphi \Rightarrow p, p \in \mathit{Preds}\} \land \bigwedge\{\neg p \mid \varphi \Rightarrow \neg p, p \in \mathit{Preds}\}$$

- **Boolean Abstraction** computes the strongest formula. Let X be the set of all formulas of the form $(\neg)p_1 \land \ldots \land (\neg)p_n$, where $\mathit{Preds} = \{p_1, \ldots, p_n\}$.

  $$\alpha_B(\varphi) \triangleq \bigvee\{\phi \mid \varphi \land \phi \text{ is SAT}\}$$

Both forms of predicate abstraction can result in finitely many formulas. Now we can replace step (2) in the fixpoint algorithm with the following:

$$sol(r_n) = sol(r_n) \lor \alpha_B(\exists V.\, sol(r_1(\vec{v}_1)) \land sol(r_2(\vec{v}_2)) \land \ldots \land sol(r_{n-1}(\vec{v}_{n-1})) \land \varphi)$$

This ensures that we reach a fixpoint in finitely many steps, as in every step we weaken $sol(r_i)$ using one of finitely many formulas. However, if we find a solution that does not satisfy the clauses, that does not mean that the clauses are unsatisfiable.

University of Wisconsin–Madison
E-mail address: aws@cs.wisc.edu