Constrained Horn Clauses

Sumanth Prabhu S

May 29, 2021
Motivation: Program Safety Verification

Program and an assert: all the states that reach assert satisfy the condition in assert?
Motivation: Program Safety Verification

Program and an assert: all the states that reach assert satisfy the condition in assert?
How different areas of computation are related?
AI vs Databases vs Programming ??

Logic was well understood
Propositional logic $<$ FOL $<$ Higher-order logic etc.

Can logic unify different areas of computation?
Horn Clauses

In Propositional Logic: A clause with at most one positive literal

\[ A_1 \land \cdots \land A_n \implies B \quad (1) \]
\[ A_1 \cdots A_n \quad (2) \]
Exercise 1

\[
P, Q \ldots \text{ are propositional variables}
\]

\[
P \implies Q
\]

\[
L \land M \implies P
\]

\[
B \land L \implies M
\]

\[
A \land P \implies L
\]

\[
A \land B \implies L
\]

\[
A
\]

\[
B
\]

Can we conclude Q?

Reference: Figure 7.16, Artificial Intelligence A Modern Approach
Exercise 1

$P, Q \ldots$ are propositional variables

\[
P \implies Q \\
L \land M \implies P \\
B \land L \implies M \\
A \land P \implies L \\
A \land B \implies L \\
A \\
B \\
\text{Forward: } A, B \rightarrow L; L, B \rightarrow M; L, M \rightarrow P; P \rightarrow Q
\]
Horn Clauses

\[ \forall \vec{x}_0 . \, true \implies r_0(\vec{x}_0) \]  \hspace{1cm} (3)

\[ \forall \vec{x}_1 \ldots \vec{x}_{n+1} . \, \bigwedge_{1 \leq i \leq n} r_i(\vec{x}_i) \implies r_{n+1}(\vec{x}_{n+1}) \]  \hspace{1cm} (4)

\[ \forall \vec{x}_1 \ldots \vec{x}_{n+1} . \, \bigwedge_{1 \leq i \leq n} r_i(\vec{x}_i) \implies false \]  \hspace{1cm} (5)
Applications

- Database: Specify dependencies
  Employees of same department should have same manager: \( \forall e_1, d, m_1, e_2, d, m_2. \) Row\((e_1, d, m_1) \land Row(e_2, d, m_e) \implies Equal(m_1, m_2) \)

- Specification of Data-structures
  \( append(X, Y, Z) \)
  \( append(X, Y, Z) \implies append(cons(U, X), Y, cons(U, Z)) \)

- Artificial Intelligence
  - Given a knowledge base as Horn Clauses, can we a sentence?

More details refer: LOGIC PROGRAMMING, Robert Kowalski
Exercise 2

```c
int x = 0, y = 0;
while (*) {
    x = x + 1;
    y = y + x;
}
assert (y ≥ 0);
```

Listing 1: A Program from Understanding IC3

'*' denotes non-deterministic loop (i.e. the loop can run any number of iterations)
'int' is mathematical integer (i.e. no overflow)

Is the assert in program safe?
Inductive Invariants

int x = y = 0
while (*) {
    x = x + 1
    y = y + x
} 
assert(y >= 0)

Program Reference: Understanding IC3
Inductive Invariants

```
int x = y = 0
while (*) {
    x = x + 1
    y = y + x
}
assert(y >= 0)
```

Safe Inductive Invariants:

\[(x \geq 0 \land y \geq 0), \quad (x \geq 0 \land y - x \geq 0)\]

Program Reference: Understanding IC3
Inductive Invariants

\[ \text{Post(Inv) } \subseteq \text{Inv} \]
Given: $\langle V \cup V', \text{Init}, \text{Tr} \rangle$ and Prop

\[ \{x, y, x', y'\}, x = 0 \land y = 0, x' = x + 1 \land y' = y + x' \]
and $(y \geq 0)$

Find a relation inv such that:

- **Initiation:** $\forall V . \text{Init}(V) \Rightarrow \text{inv}(V)$

- **Consecution:** $\forall V, V' . \text{inv}(V) \land \text{Tr}(V, V') \Rightarrow \text{inv}(V')$

- **Safety:** $\forall V . \text{inv}(V) \Rightarrow \text{Prop}(V)$

How to specify this problem and how to solve it?

**Constrained Horn Clauses**
Syntax of Constrained Horn Clauses

A CHC over a set of uninterpreted relation symbols \( \mathcal{R} \) has the form of one of the following three implications:

1. \( \forall \vec{x}_1 . \varphi(\vec{x}_1) \implies r_1(\vec{x}_1) \quad (6) \)
2. \( \forall \vec{x}_0 \ldots \vec{x}_{n+1} . \bigwedge_{0 \leq i \leq n} r_i(\vec{x}_i) \land \varphi(\vec{x}_0, \ldots, \vec{x}_{n+1}) \implies r_{n+1}(\vec{x}_{n+1}) \quad (7) \)
3. \( \forall \vec{x}_0 \ldots \vec{x}_n . \bigwedge_{0 \leq i \leq n} r_i(\vec{x}_i) \land \varphi(\vec{x}_0, \ldots, \vec{x}_n) \implies false \quad (8) \)

where:

- \( r_i \in \mathcal{R}, \vec{x}_i \) is a vector of variables of length \( \text{arity}(r_i) \);
- for some \( i \) and \( j \), such that \( i \neq j \), it could be (though not necessary) that \( r_i = r_j \);
- \( \varphi \) is a satisfiable quantifier-free formula in a theory \( T \) that does not contain any uninterpreted symbols.
Syntax of Constrained Horn Clauses

A CHC over a set of uninterpreted relation symbols $\mathcal{R}$ has the form of one of the following three implications:

$$\forall \vec{x}_1 \cdot \varphi(\vec{x}_1) \Rightarrow r_1(\vec{x}_1) \quad (6)$$

$$\forall \vec{x}_0 \ldots \vec{x}_{n+1} \cdot \mathop{\bigwedge}_{0 \leq i \leq n} r_i(\vec{x}_i) \land \varphi(\vec{x}_0, \ldots, \vec{x}_{n+1}) \Rightarrow r_{n+1}(\vec{x}_{n+1}) \quad (7)$$

$$\forall \vec{x}_0 \ldots \vec{x}_n \cdot \mathop{\bigwedge}_{0 \leq i \leq n} r_i(\vec{x}_i) \land \varphi(\vec{x}_0, \ldots, \vec{x}_n) \Rightarrow false \quad (8)$$

where:

- $body(C)$ and $head(C)$ denotes the left and right side of the implication in $C$, resp.;
- A CHC of type (6) is called fact, of type (7) inductive, and type (8) query
- $C$ is linear if $|rel(body(C))| \leq 1$; otherwise it is non-linear.
Semantics

- **interpretation** for $r \in \mathcal{R}$ is a map
  $\lambda x_1 \ldots \lambda x_{a_r} . \varphi(x_1, \ldots, x_{a_r})$, where $\varphi$ is a quantifier-free formula without any symbols from $\mathcal{R}$.

- Extended to $\mathcal{R}$ by interpreting each symbol $r \in \mathcal{R}$.

- $\varphi[\mathcal{M}/\mathcal{R}]$ is the formula obtained by replacing each occurrence of a term of the form $r(x_1, \ldots, x_{a_r})$ by $\mathcal{M}(r)(x_1, \ldots, x_{a_r})$.

- A system $S$ of CHCs over $\mathcal{R}$ is said to be **satisfiable** if there exists an interpretation $\mathcal{M}$ for $\mathcal{R}$ which makes all implications in $S$ valid, i.e., for all $C \in S$, it holds that $\text{body}(C)[\mathcal{M}/\mathcal{R}] \implies \text{head}(C)[\mathcal{M}/\mathcal{R}]$. 


CHCs - An Example

\[ \forall x, y . \ x = 0 \land y = 0 \implies \text{inv}(x, y) \]

\[ \forall x_0, y_0, x_1, y_1 . \ \text{inv}(x_0, y_0) \land x_1 = x_0 + 1 \land y_1 = y_0 + x_1 \implies \text{inv}(x_1, y_1) \]

\[ \forall x, y . \ \text{inv}(x, y) \land \neg (y \geq 0) \implies \text{false} \]
CHCs - Inductive Invariants

- Given $\langle V \cup V', \text{Init}, \text{Tr} \rangle$ and $\text{Prop}$
  - $\{x, y, x', y\}, x = 0 \land y = 0, x' = x + 1 \land y' = y + x'$
    and $(y \geq 0)$

- Given $M := \{\text{inv} \mapsto \lambda x, y . x \geq 0 \land y \geq 0\}$

- Initiation: $\forall V . \text{Init}(V) \Rightarrow \text{inv}(V)$
  - $\forall x, y . (x = 0 \land y = 0) \Rightarrow (x \geq 0 \land y \geq 0)$

- Consecution: $\forall V, V'. \text{inv}(V) \land \text{Tr}(V, V') \Rightarrow \text{inv}(V')$
  - $\forall x, y, x', y' . (x \geq 0 \land y \geq 0) \land x' = x + 1 \land y' = y + x' \Rightarrow (x' \geq 0 \land y' \geq 0)$

- Safety: $\forall V . \text{inv}(V) \Rightarrow \text{Prop}(V)$
  - $\forall x, y . (x \geq 0 \land y \geq 0) \Rightarrow (y \geq 0)$
Universally Quantified! How to check satisfiability? Negate the formulas and check satisfiability Is any of these is SAT?

$$(x = 0 \land y = 0) \land \neg(x \geq 0 \land y \geq 0)$$

$$(x \geq 0 \land y \geq 0) \land (x' = x + 1 \land y' = y + x') \land (x' \geq 0 \land y' \geq 0)$$

$$(x \geq 0 \land y \geq 0) \land \neg(y \geq 0)$$
CHCs - Refutation

- Given \( \langle V \cup V', Init, Tr \rangle \) and \( Prop \)
  \[ \{x, y, x', y'\}, x = 0 \land y = 0, x' = x + 1 \land y' = y + x' \text{ and } (y = x) \]
- Initiation: \( \forall V . Init(V) \Rightarrow inv(V) \)
- Consecution: \( \forall V, V' . inv(V) \land Tr(V, V') \Rightarrow inv(V') \)
- Safety: \( \forall V . inv(V) \Rightarrow Prop(V) \)

\[ \Rightarrow x = 0 \land y = 0 \]
\[ \Rightarrow inv(0, 0), inv(0, 0) \land x' = x + 1 \land y' = y + x' \]
\[ \Rightarrow inv(1, 1) \]
\[ \Rightarrow inv(1, 1), inv(0, 0) \land x' = x + 1 \land y' = y + x' \]
\[ \Rightarrow inv(2, 3) \]
Exercise 3

Encode the following program as CHCs

```c
int x = 0, y = 0
int m = n = *
assume(m ≥ 0)
while (n ≠ 0) {
    n--;
    if(*) x++;
    else y++;
}
while (x ≠ 0){m--;x--;}
while(y ≠ 0) {m--;y--;}
assert(m==0);
```
Exercise 3 - Solution

\[ x = 0 \land y = 0 \land m \geq 0 \implies \text{inv}_1(x, y, m, n) \]

\[ \text{inv}_1(x, y, m, n) \land \neg(n = 0) \land n' = n-1 \land (x' = x+1 \lor y' = y+1) \implies \text{inv}_1(x', y', m, n) \]

\[ \text{inv}_1(x, y, m, n) \land n = 0 \implies \text{inv}_2(x, y, m, n) \]

\[ \text{inv}_2(x, y, m, n) \land \neg(x = 0) \land m' = m - 1 \land x' = x - 1 \implies \text{inv}_2(x', y, m', n) \]
When $T$ is LIA, LRA finding a solution is *undecidable*

Reference: The Universal Fragment of Presburger Arithmetic with Unary Uninterpreted Predicates is Undecidable, Horbach et al.
References for CHCs in Program Verification

Horn Clause Solvers for Program Verification

Nikolaj Bjørner, Arie Gurfinkel, Ken McMillan and Andrey Rybalchenko

Microsoft Research, Software Engineering Institute

Abstract. Automatic program verification and symbolic model checking tools interface with theorem proving technologies that check satisfiability of formulas. A theme pursued in the past years by the authors of this paper has been to encode symbolic model problems directly as Horn clauses and develop dedicated solvers for Horn clauses. Our solvers are called Duality, HSF, Seaform, and μZ and we have devoted considerable attention in recent papers to algorithms for solving Horn clauses. This paper complements these strides as we summarize main useful properties of Horn clauses, illustrate encodings of procedural program verification into Horn clauses and then highlight a number of useful simplification strategies at the level of Horn clauses. Solving Horn clauses amounts to establishing Existential positive Fixed-point Logic formulas, a perspective that was promoted by Blass and Gurevich.

1 Introduction

We make the overall claim that Constrained Horn Clauses provide a suitable basis for automatic program verification, that is, symbolic model checking. To sub-

Position paper

Synthesizing Software Verifiers from Proof Rules

Sergey Grebenshchikov  
Technische Universität München  
grebenshchikov@cs.tum.de

Nuno P. Lopes  
INESC-ID / IST - TU Lisbon  
nuno.lopes@ist.ist.utl.pt

Comelio Popescu  
Technische Universität München  
popescu@model.in.tum.de

Andrey Rybalchenko  
Technische Universität München  
rybalchenko@tum.de

Encoding rules for various program constructs
So far:
  - Horn Clauses
  - Constrained Horn Clauses
  - Invariant Inference as CHCs

Next:
  - How to solve CHCs?
Solving Constrained Horn Clauses using Syntax and Semantics
Reference: Fedyukovich, Prabhu, Madhukar, and Gupta, FMCAD 2018
Guess and Check Framework

Iterative learning: \( \text{inv} \iff l_0 \land l_1 \land \cdots \land l_n \)

A learner using Syntax Guided Synthesis
Syntax Guided Synthesis

```c
int x = y = 0
int m = n = *;
assume(m >= 0);

while (n != 0) {
    n--;
    if (*) then x++;
    else y++;
}

while (x != 0) {
    m--; x--;
}
while (y != 0) {
    m--; y--;
}
assert(m == 0);
```

Invariants needed:

for first loop:

\[(x + y + n = m)\]

for second loop:

\[(x + y + n = m) \land (n = 0)\]

for third loop:

\[(x + y + n = m) \land (n = 0) \land (x = 0)\]
int x = y = 0
int m = n = *;
assume(m >= 0);

while (n != 0) {
    n--;
    if (*) then x++;
    else y++;
}

while (x != 0) {
    m--; x--;
}
while (y != 0) {
    m--; y--;
}
assert(m == 0);

\begin{align*}
\text{assert}(m & = 0) \\
x = 0 & \rightarrow x \geq 0, -x \geq 0 \\
y = 0 & \rightarrow y \geq 0, -y \geq 0 \\
m \geq 0 & \rightarrow m \geq 0 \\
m = n & \rightarrow m \geq n, -m \geq n \\
n \neq 0 & \rightarrow -n > 0 \lor n > 0
\end{align*}
Syntax Guided Synthesis

```c
int x = y = 0
int m = n = *;
assume(m >= 0);

while (n != 0) {
    n--;
    if (*) then x++;
    else y++;
}
while (x != 0) {
    m--; x--;
}
while (y != 0) {
    m--; y--;
}
assert(m == 0);

{x ≥ 0, −x ≥ 0, y ≥ 0, −y ≥ 0,
 m ≥ 0, m ≥ n, −m ≥ n,
 − n > 0 ∨ n > 0}

k ::= 1 | −1
v ::= x | y | m | n
e ::= k · v | k · v + k · v
cand ::= e ≥ c | e > c ∨ e > c
```
Syntax Guided Synthesis

\[
\begin{align*}
\text{int } x &= y = 0 \\
\text{int } m &= n = *; \\
\text{assume}(m \geq 0); \\
\text{while } (n \neq 0) \{ \\
& \quad n--; \\
& \quad \text{if } (*) \text{ then } x++; \\
& \quad \text{else } y++; \\
\} \\
\text{while } (x \neq 0) \{ \\
& \quad m--; x--; \\
\} \\
\text{while } (y \neq 0) \{ \\
& \quad m--; y--; \\
\} \\
\text{assert}(m == 0);
\end{align*}
\]

\[
\{ n \geq 0, -n \geq 0, -x > 0 \lor x > 0 \}
\]
Syntax Guided Synthesis

```plaintext
int x = y = 0
int m = n = *;
assume(m >= 0);

while (n != 0) {
    n--;
    if (*) then x++;
    else y++;
}

while (x != 0) {
    m--; x--;
}
while (y != 0) {
    m--; y--;
}
assert(m == 0);

\{ n \geq 0, -n \geq 0, -x > 0 \lor x > 0 \}

c ::= 0
k ::= 1 | -1
v ::= x \mid n
e ::= k \cdot v
cand ::= e \geq c \mid e > c \lor e > c
```
Syntax Guided Synthesis

```c
int x = y = 0
int m = n = *;
assume(m >= 0);

while (n != 0) {
    n--;
    if (*) then x++;
    else y++;
}

while (x != 0) {
    m--; x--;
}

while (y != 0) {
    m--; y--;
}
assert(m == 0);

\{ x \geq 0, -x \geq 0, -y > 0 \lor y > 0,
    y \geq 0, -y \geq 0, m \geq 0, -m \geq 0 \}\n```
Syntax Guided Synthesis

```plaintext
int x = y = 0
int m = n = *;
assume(m >= 0);

while (n != 0) {
    n--;
    if (*) then x++;
    else y++;
}

while (x != 0) {
    m--; x--;
}
while (y != 0) {
    m--; y--;
}
assert(m == 0);

{x \geq 0, -x \geq 0, -y > 0 \lor y > 0, y \geq 0, -y \geq 0, m \geq 0, -m \geq 0}

\{c \ ::= \ 0, k \ ::= \ 1 \mid -1, v \ ::= \ x \mid y \mid m, e \ ::= \ k \cdot v, cand \ ::= \ e \geq c \mid e > c \lor e > c\}
```
Insufficiency of the grammars

\[ c ::= 0 \]
\[ k ::= 1 \mid -1 \]
\[ v ::= x \mid y \mid m \mid n \]
\[ e ::= k \cdot v \mid k \cdot v + k \cdot v \]
\[ \text{cand} ::= e \geq c \mid e > \]
\[ c \lor e > c \]

\((x + y + n = m)\)

\[ c ::= 0 \]
\[ k ::= 1 \mid -1 \]
\[ v ::= x \mid y \mid m \mid n \]
\[ e ::= k \cdot v \]
\[ \text{cand} ::= e \geq c \mid e > \]
\[ c \lor e > c \]

\((x + y + n = m) \land (n = 0) \land (x = 0)\)
Data Candidates

\[ a \cdot x + b \cdot y + c \cdot m + d \cdot n + e = 0 \]

\[
\begin{pmatrix}
  x & y & m & n \\
  0 & 0 & 5 & 5 \\
  1 & 0 & 5 & 4 \\
  2 & 0 & 5 & 3 \\
  2 & 1 & 5 & 2 \\
  2 & 2 & 5 & 1 \\
\end{pmatrix}
\]

\[ x + y + n - m = 0 \]
Propagation

\[ \text{inv}_1(x, y, m, n) \land n=0 \land \]
\[ x_1=x \land y_1=y \land m_1=m \land n_1=n \Rightarrow \]
\[ \text{inv}_2(x_1, y_1, m_1, n_1) \]

No change in variables so candidates of \text{inv}_1
are likely to be candidates of \text{inv}_2.
Propogation

\[ \text{inv}_i(X) \land \phi(X, X') \Rightarrow \text{inv}_j(X') \]

**Forward:**
\[ \text{Cand}_j = \exists X \ \text{Cand}_i(X) \land \phi(X, X') \]

**Backward:**
\[ \text{Cand}_i = \exists X' \ \text{Cand}_j(X') \land \phi(X, X') \]
The Science, Art and Magic of Constrained Horn Clauses
Arie, Gurfinkel and Nikolaj Bjørner

Horn Clause Solvers for Program Verification, Bjørner, et al.

Synthesizing Software Verifier from Proof Rules, Grebenschikov, et al.