Constrained Horn Clause Solving

Ray Luo

Constrained Horn Clauses

CHC: a *First Order Logic* formula of the following form:

Unknown predicate symbols (relation)

$$\forall \mathcal{V} \cdot (\varphi \wedge p_1(X_1) \wedge \cdots \wedge p_k(X_k) \rightarrow h(X)), \text{ for } k \geq 1$$

For all variables ("Constrained") Conjunction Implication p->q means ~p or q

$$\forall \mathcal{V} \cdot \underbrace{(\varphi \wedge p_1(X_1) \wedge \cdots \wedge p_k(X_k)}_{\text{Body}} \rightarrow \underbrace{h(X)}_{\text{Head}}), \text{ for } k \geq 1$$

- CHC system: a set of CHCs containing rules and queries
- Rule: *head* is *not* P-free (has at least one unknown predicate)
- Query: *head* is P-free
- Fact: a rule whose *body* is P-free (does not have an unknown predicate)

Constrained Horn Clauses

Relational Post-fixed Point Problem (RPFP)

(\forall x,y)

$$x = 1 \wedge y = 0 \rightarrow p(x, y) \tag{1}$$

$$p(x, y) \wedge x' = x + y \wedge y' = y + 1 \rightarrow p(x', y') \tag{2}$$

$$p(x, y) \wedge x' = x + y \wedge y' = y + 1 \rightarrow x' >= y' \tag{3}$$

Fact (Rule)
Rule
Query

- A CHC system is SAT <=>

There is an interpretation $\mathbb{I}$ such that under $\mathbb{I}$ **all clauses** are valid

- How about UNSAT? No such interpretation. How can we prove it?

Constrained Horn Clauses

$$x = y \implies P(x, y) \quad (7)$$

$$P(x, y) \land z = y + 1 \implies P(x, z) \quad (8)$$

$$P(x, y) \land P(y, z) \implies Q(x, z) \quad (9)$$

$$Q(x, z) \implies x + 2 \leq z \quad (10)$$

Ex:
x=y=z=0

- By finding a *ground refutation* of False! ("unwinding" the CHC)
- Namely, find an assignment that for any interpretation that the CHC system can never be SAT (one of the CHC is UNSAT is enough)
- Rules can always be SAT as we can set the relations to True
- Then we can simplify the problem by finding an assignment that make all rules SAT but make one of the queries UNSAT!

- CHC solving is one of the basis of program/hardware verification
- CHCs can encode programs and VCs
  - For example: (Safety) Inductive Invariant learning problem could be reduced to CHC solving problem (SAT, correct; UNSAT, buggy)
- Program -> SSA (IR), CFG -> CHC
  - Relation (p): vertex in CFG (functionality of a basic block)
  - Clause: edge in CFG (transition of basic blocks)

A logical formula such that its validity means some aspect of program correctness.

(\forall x,y)

$$x = 1 \land y = 0 \to p(x, y) \quad (1)$$

$$p(x, y) \land x' = x + y \land y' = y + 1 \to p(x', y') \quad (2)$$

$$p(x, y) \land x' = x + y \land y' = y + 1 \to x' >= y' \quad (3)$$

$$x = 1 \land y = 0 \to x >= y \quad (4)$$

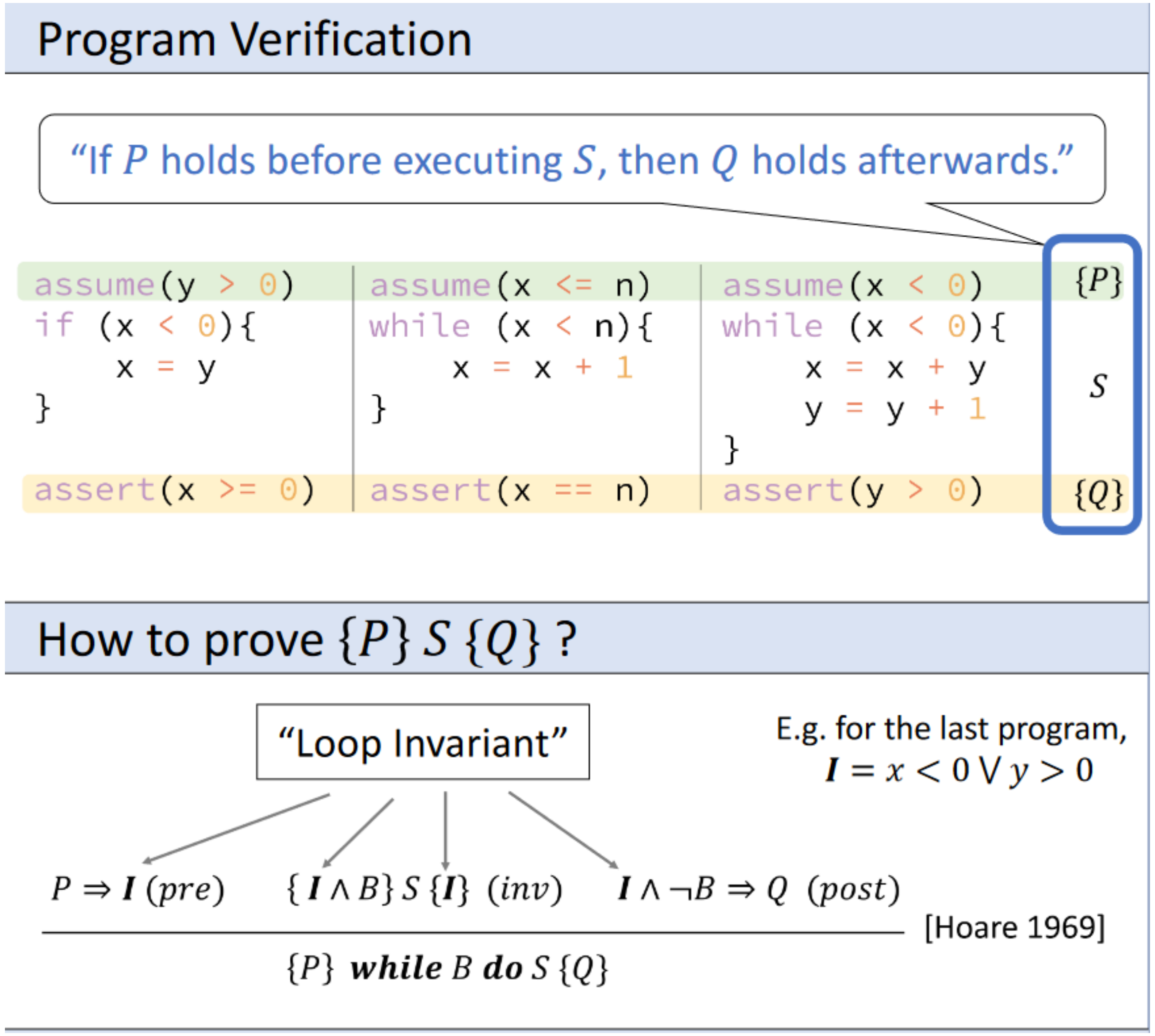
```
main(){
  int x,y;
  x=1; y=0;
  while(*){
   x=x+y;
   y++;}
  assert(x>=y);}
```

Some nondeterministic expression of x, y

Constrained Horn Clauses

CHC AND PROGRAMS

Loop Invariant Learning



Loop Invariant Illustration*

- Learning loop (inductive) invariants is an important task in program verification
- Hoare logic
- A loop invariant encodes some basic functionality of the loop body
- Precondition: Fact
- Postcondition: Query

* https://www.cs.mcgill.ca/~xsi/data/code2inv_poster.pdf

Data-driven CHC solving

- CEGAR: counter-example guided abstraction refinement
- The **paradigm of learning** is one prevailing technique (guess and check)
  - Teacher: an oracle, verify the learner's hypothesis, e.g., Z3

  And provide feedback (counterexamples)
  - Learner: propose a possible invariant, i.e., hypothesis,

  can apply ML techniques
  - The process of learning and teaching is done iteratively

  and alternatively
    - a Counterexample for a CHC is an assignment that refute current hypothesis
    - Generate positive and negative samples from counterexamples

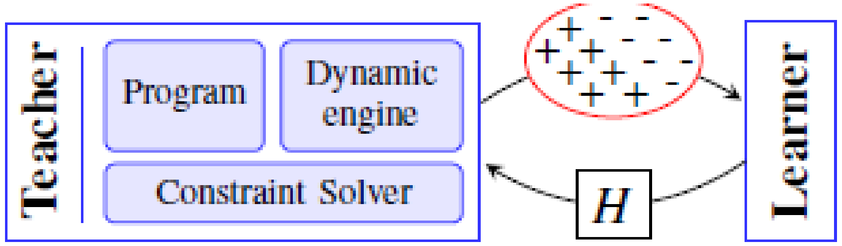


Figure from: ICE: A Robust Framework for Learning

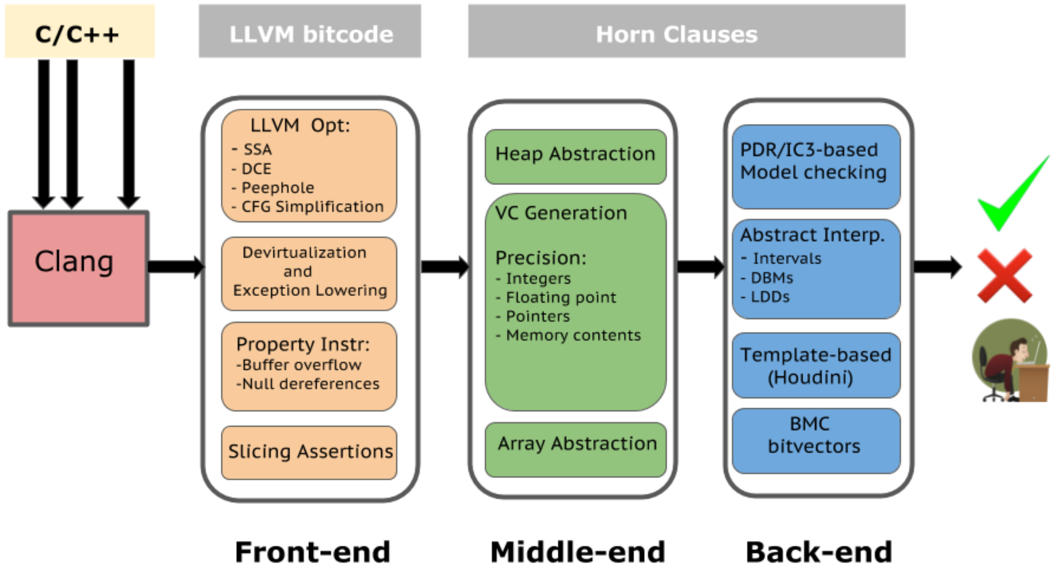
Data Driven CHC Solving

- Learning by counterexamples
  - Program configuration: the *values of all variables* (when entering a basic block)
  - Positive samples: A reachable program configuration
    - (i, j, p) = (0, 0, 25) (i=0, j=0, p=25 is reachable)
  - Negative samples: A program configuration that can never be reached
    - (i, j, p) = (100, 0, 25)
  - Then the learner could be a classifier that can correctly classify the examples and counterexamples!

```
#include <vcc.h>
int foo(int a[], int p)
_(requires (p>=25 && p<75))
_(requires a[p]==1)
_(requires \thread_local_array
                      (a, 100))
{
  int i=0, j=0;
  while (i<100)
  _(invariant (i>p ==> j==1))
  {
    if (a[i]==1)
       j = 1;
    i = i+1;
  }
  _(assert j==1);
}
```

Seahorn: C program to SMT-LIB2 CHC

- Seahorn
  - Seahorn is an automated analysis framework for LLVM-based languages
  - <u>Front end</u>: Takes an LLVM based program (e.g., C, C++) as input and generates LLVM IR bitcode
  - <u>Middle end</u>: Input the optimized LLVM bitcode and emits **VCs** as CHCs
  - Back end: Takes CHCs as input and outputs the result of the analysis



```
D: > Program Files > Docker Toolbox > chc-study > test > c > pie > ICE > benchmarks > C sum01.c > ...
extern void __VERIFIER_error() __attribute__ ((__noreturn__));
extern int __VERIFIER_nondet_int();
void __VERIFIER_assert(int cond) {
  if (!(cond)) {
    ERROR: __VERIFIER_error();;
  }
  return;
}
#define a (1)
int main() {
  int i, n=__VERIFIER_nondet_int(), sn=0;
  for(i=1; i<=n; i++) {
    sn = sn + a;
  }
  __VERIFIER_assert(sn==n*a || sn == 0);
}
```

```
(rule (let ((a!1 (and (main@.lr.ph main@%_1_0 main@%sn.0.i2_0 main@%i.0.i1_0)
                true
                (= main@%_3_0 (+ main@%sn.0.i2_0 1))
                (= main@%_4_0 (+ main@%i.0.i1_0 1))
                (= main@%_5_0 (< main@%i.0.i1_0 main@%_1_0))
                (=> main@orig.main.exit.loopexit_0
                    (and main@orig.main.exit.loopexit_0 main@.lr.ph_0))
                (=> (and main@orig.main.exit.loopexit_0 main@.lr.ph_0)
                    (not main@%_5_0))
                (=> (and main@orig.main.exit.loopexit_0 main@.lr.ph_0)
                    (= main@%.lcssa_0 main@%_3_0))
                (=> (and main@orig.main.exit.loopexit_0 main@.lr.ph_0)
                    (= main@%.lcssa_1 main@%.lcssa_0))
                (=> main@orig.main.exit_0
                    (and main@orig.main.exit_0 main@orig.main.exit.loopexit_0))
                (=> (and main@orig.main.exit_0 main@orig.main.exit.loopexit_0)
                    (= main@%sn.0.i.lcssa_0 main@%.lcssa_1))
                (=> (and main@orig.main.exit_0 main@orig.main.exit.loopexit_0)
                    (= main@%sn.0.i.lcssa_1 main@%sn.0.i.lcssa_0))
                (=> main@orig.main.exit_0
                    (= main@%_6_0 (= main@%sn.0.i.lcssa_1 main@%_1_0)))
                (=> main@orig.main.exit_0
                    (= main@%_7_0 (= main@%sn.0.i.lcssa_1 0)))
                (=> main@orig.main.exit_0
                    (= main@%..i_0 (or main@%_6_0 main@%_7_0)))
                (=> main@orig.main.exit_0 (not main@%_8_0))
                (=> main@orig.main.exit_0 (= main@%_9_0 (xor main@%_8_0 true)))
                (=> main@orig.main.exit_0 (= main@%_10_0 (= main@%..i_0 false)))
                (=> main@orig.main.exit_0 main@%_10_0)
                (=> main@orig.main.exit.split_0
                    (and main@orig.main.exit.split_0 main@orig.main.exit_0))
                main@orig.main.exit.split_0)))
  (=> a!1 main@orig.main.exit.split)))
(query main@orig.main.exit.split)
```

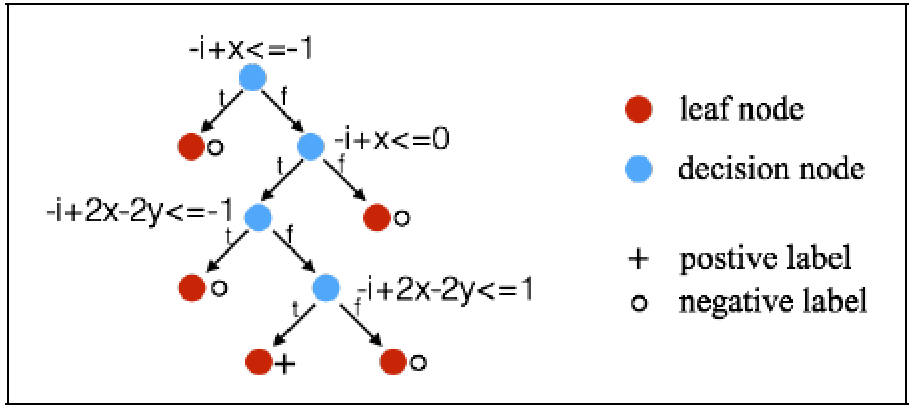
```
main@.lr.ph:
  (!(((1*main@%_1)+(-1*main@%i.0.i1))<=-1))
  (((1*main@%sn.0.i2)+(-1*main@%i.0.i1))<=-1)
  (!(((1*main@%sn.0.i2)+(-1*main@%i.0.i1))<=-2))
```

sum01.c | LLVM IR bitcode | CHC (*.smt file for Z3-Spacer) | Result: Inductive Invariant

Example on how the framework works

Guess: LinearArbitrary

$$\rho \equiv \left\{ \begin{array}{c} (-10i - x + 5y + 6n + 7 \geq 0 \wedge \\ -i + x \geq 0 \wedge i - x \geq 0 \wedge -i + 2x - 2y \geq 0) \vee \\ 2i + 3x + 4y + 2n - 34 \geq 0 \end{array} \right\}$$



Using SVMs,
we generate those hyperplanes (candidates)

Then the C50 DT recombines those candidates (standard: higher information gain)
Conjunction over nodes in path then disjunction over all positive paths

Great fit to our lightweight machine learning scene

*: A Data-Driven CHC Solver, He et al., PLDI'18

Check: Z3 (Spacer)

- Negate a CHC (for all -> there exist) and call Z3 to get counterexample (SAT, and an assignment)
- Collect positive samples by *implicit "unwinding"* the CHCs like running a program (positive states are derived from other positive states)
- Accumulate positive samples to get more accurate interpretation guess (refinement)

Algorithm 3: CHCSolve $(\mathcal{H})$

$\mathcal{A} = \lambda p : true$;
$\forall\, p \in \mathcal{P}(\mathcal{H}).\ s^+(p) = s^-(p) = \emptyset$;
while
  $\exists\, (C \equiv \phi \wedge p_1[\overline{T_1}] \wedge p_2[\overline{T_2}] \wedge \cdots \wedge p_k[\overline{T_k}] \rightarrow h[\overline{T}]) \in \mathcal{H}$
  s.t. not (*Z3Check* $(C[\mathcal{A}])$) **do**
    do
      $s =$ *Z3Model* $(C[\mathcal{A}])$;
      $\forall\, i.\ 1 \leq i \leq k.\ s_i = \{\text{Z3Eval}\ (t_{ij}, s) \mid t_{ij} \in$
          $\overline{T_i} \equiv \{t_{i1}, \cdots, t_{in}\}\}$;
      $s_h = \{\text{Z3Eval}\ (t_j, s) \mid t_j \in \overline{T} \equiv \{t_1, \cdots, t_n\}\}$;
      if $(\forall\, i.\ 1 \leq i \leq k.\ s_i \in s^+(p_i))$ **then**
        if $h \in \mathcal{P}$ **then**
          $s^+(h) = s^+(h) \cup \{s_h\}$;
          $s^-(h) = \emptyset$;
          $\mathcal{A}(h) = true$;
        else
          return $\mathcal{H}$ *is unsat with counterex* s_h
      else
        for $i \leftarrow 1$ **to** k **do**
          if $s_i \notin s^+(p_i)$ **then**
            $s^-(p_i) = s^-(p_i) \cup \{s_i\}$;
            $\mathcal{A}(p_i) = \text{Learn}\ (s^+(p_i), s^-(p_i))$;
        end
    while *not* (*Z3Check* $(C[\mathcal{A}])$);
end
return $\mathcal{H}$ *is sat with interpretation* $\mathcal{A}$

McGill UNIVERSITY

McGill School of Computer Science

Thank you
for your careful listening!