Solving non-linear Horn clauses using a linear solver

Bishoksan Kafle

Roskilde University
Joint work with John Gallagher and Pierre Ganty (IMDEA Software institute)

Roskilde University, 04-11-2015

Example: Fibonacci numbers or Fibonacci sequence

Is a sequence: $1, 1, 2, 3, 5, 8, 13, 21...$

Mathematically,

- $fib(n) = 1,\ n \geq 0$ and $n \leq 1$.
- $fib(n) = fib(n-1) + fib(n-2),\ n > 1.$

As Constrained Horn clauses,

```
fib(A, B):- A>=0,  A=<1, B=1.
fib(A, B) :- A > 1, A2 = A-2, fib(A2, B2),
             A1 = A-1, fib(A1, B1), B = B1+B2.
```

Encoding property as Horn clauses

```
Property :  A>5, fib(A,B), B>=A.

c1. fib(A, B):- A>=0,  A=<1, B=1. (linear)
c2. fib(A, B) :- A > 1, A2 = A-2,  fib(A2, B2),
             A1 = A-1, fib(A1, B1),  B = B1+B2. (non-linear)
c3. false:- A>5, fib(A,B), B<A.     (linear)
```

Horn clauses : $p(X) \leftarrow \mathcal{C}, p_1(X_1), \ldots, p_k(X_k)$ $(k \geq 0)$

Solving Horn clauses

Given,

```
c1. fib(A, B):- A>=0,  A=<1, B=1.
c2. fib(A, B) :- A > 1, A2 = A-2,  fib(A2, B2),
            A1 = A-1, fib(A1, B1),  B = B1+B2.
c3. false:- A>7, fib(A,B), B<A.
```

Finding an interpretation for the predicates in the above program (fib(A,B) and false, where false is always interpreted as false) such that each clause is satisfied (finding a model).

```
fib(A,B) :- [A>=0,B>=1,B>=A].
```

Linear Horn clause solver

A solver which can only deal with linear clauses e.g., c1 and c3 but not c2.

```
c1. fib(A, B):- A>=0,  A=<1, B=1. (linear)
c2. fib(A, B) :- A > 1, A2 = A-2,  fib(A2, B2),
           A1 = A-1, fib(A1, B1),  B = B1+B2. (non-linear)
c3. false:- A>7, fib(A,B), B<A.    (linear)
```

So we cannot solve the above clauses just like this using a linear solver!

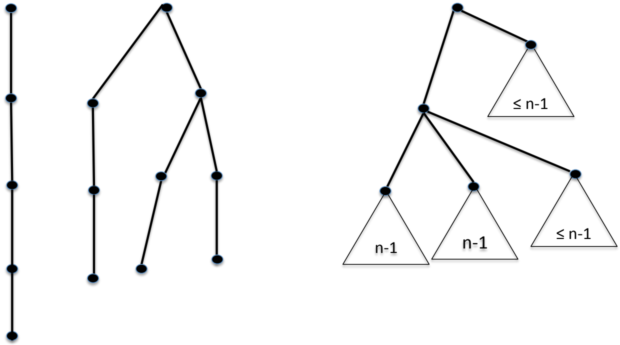
Can we solve these clauses using a linear solver?

Yes, we can!

idea: interleave program transformation and linear solving

- program transformation is based on the idea of tree dimension of Horn clause derivations

What is the dimension of a tree?

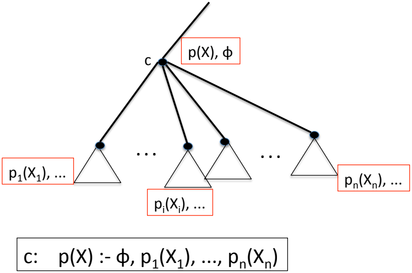


dimension 0 | dimension 1 | . . . | dimension n

The dimension of tree is a measure of its (non)-linearity

Horn clauses derivation trees

A derivation tree for a set of clauses P is a labelled tree, where each node is labelled by the id of a clause in P, the head, and constraint of the corresponding clause.

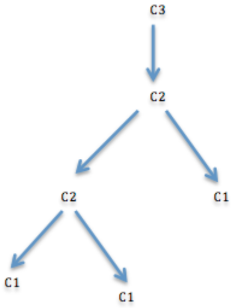


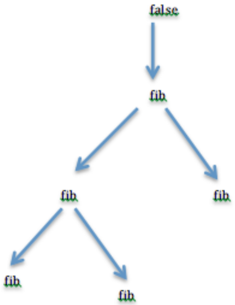
A trace tree is a derivation tree containing only the clause id labels.

false has a derivation $\Rightarrow$ there exists a derivation tree whose root is labelled by false.

Derivation tree - Example

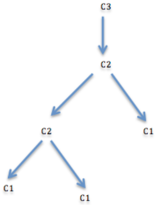
```
c1. fib(A, A):- A>=0,  A=<1.
c2. fib(A, B) :- A > 1, A2 = A-2, fib(A2, B2),
             A1 = A-1, fib(A1, B1), B = B1+B2.
c3. false:- A>5, fib(A,B), B<A.
```

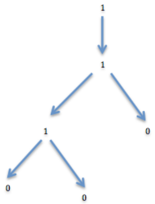




Trace tree dimension - Example

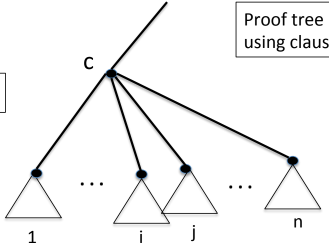
```
c1. fib(A, A):- A>=0,  A=<1.
c2. fib(A, B) :- A > 1, A2 = A-2,
            fib(A2, B2),
            A1 = A-1, fib(A1, B1),
            B = B1+B2.
c3. false:-
           A>5, fib(A,B), B<A.
```





Program transformation - exactly-k predicate

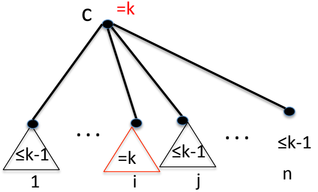
Proof tree using clause c

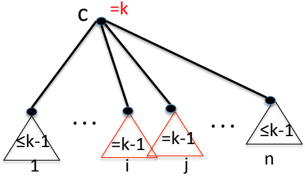


c: $p(X) \text{ :- } \phi, p_1(X_1), \ldots, p_n(X_n)$

c: $p^{=k}(X) \text{ :- } \phi, p_1^{\leq k-1}(X_1), \ldots, p_i^{=k-1}(X_i), \ldots, p_j^{=k-1}(X_j), \ldots, p_n^{\leq k-1}(X_n)$

c: $p^{=k}(X) \text{ :- } \phi, p_1^{\leq k-1}(X_1), \ldots, p_i^{=k}(X_i), \ldots, p_j^{\leq k-1}(X_j), \ldots, p_n^{\leq k-1}(X_n)$





Generation of at-most-k dimension clauses $P^{\leq k}$

For each predicate p in a set of CHCs P, and a given dimension k, generate clauses for:

- predicates $p^{=0}, p^{=1}, \ldots, p^{=k}$;
- predicates $p^{\leq 0}, p^{\leq 1}, \ldots, p^{\leq k}$;

$p^{\leq k}$ defined by the clauses

$$p^{\leq k} \leftarrow p^{=0}$$
$$\vdots$$
$$p^{\leq k} \leftarrow p^{=k}$$

The resulting set of clauses is called $P^{\leq k}$.

Example: at-most-0 dimension clauses.

```
c1. fib(A, A):- A>=0,  A=<1.
c2. fib(A, B) :- A > 1, A2 = A-2,
           fib(A2, B2),
           A1 = A-1, fib(A1, B1),
           B = B1+B2.
c3. false:-
          A>5, fib(A,B), B<A.
```

```
fib(0)(A,B) :- A>=0, A=<1, B=1.
false(0) :- A>5, B<A, fib(0)(A,B).

false[0] :- false(0).
fib[0](A,B) :- fib(0)(A,B).
```

Algorithm for solving non-linear clauses

Given a set of Horn clauses P (linear and non-linear)

1. initialize $k = 0$,
2. generate linear clauses (k-dim program),
3. solve linear clauses (use a linear solver),
4. if not solvable then return P is not solvable,
5. if solvable get solution S,
6. check if S is a solution of P, if so S is a solution of P, return S,
7. if not, transform P to (k+1)-linear ((k+1)-dim program) clauses, say P_1,
8. plug S in P_1 giving rise to linear clauses, set $k = k + 1$ and go to 3.

Example I

Linear clauses (0-dim)

```
'fib(0)'(A,B) :-
   A>=0,
   A=<1,
   B=1.
'false(0)' :-
   A>5,
   B<A,
   'fib(0)'(A,B).
'false[0]' :-
   'false(0)'.
'fib[0]'(A,B) :-
   'fib(0)'(A,B).
```

Solution:

```
'fib(0)'(A,B) :- [-A>= -1,A>=0,B=1].
'fib[0]'(A,B) :- [-A>= -1,A>=0,B=1].
```

Inductive check

Solution of linear clauses:

```
'fib(0)'(A,B) :- [-A>= -1,A>=0,B=1].
'fib[0]'(A,B) :- [-A>= -1,A>=0,B=1].
```

use the following to check if the solution is inductive wrt the original program (mapping)

```
fib(A,B) :- [-A>= -1,A>=0,B=1].
fib(A,B) :- [-A>= -1,A>=0,B=1].
```

Example II

Part of 1-Linear clauses (1-dim)

```
'fib(0)'(A,B) :- A>=0, A=<1, B=1.
'false(1)' :- A>5, B<A,  'fib(1)'(A,B).
'false[1]' :- 'false(0)'.
'false(0)' :- A>5, B<A,  'fib(0)'(A,B).
'fib(1)'(A,B) :- A>1, C=A-2, E=A-1,
   B=F+D,  'fib(1)'(C,D),  'fib[0]'(E,F).
...
```

Linear solution (0-dim):

```
'fib(0)'(A,B) :- [-A>= -1,A>=0,B=1].
'fib[0]'(A,B) :- [-A>= -1,A>=0,B=1].
```

Example III

Linearisation:

```
'false(1)' :-
   1*A>5,
   1*A+ -1*B>0,
   'fib(1)'(A,B).
'fib(1)'(A,B) :-
   -1*A>= -2,
   1*A>1,
   1*A+ -1*C=2,
   1*B+ -1*D=1,
   'fib(1)'(C,D).
   ...
```

Inductive soulution when $k = 2$

solution for a 2-dim clauses

```
'fib(0)'(A,B) :- [-A>= -1,A>=0,B=1].
'fib[0]'(A,B) :- [-A>= -1,A>=0,B=1].
'fib(1)'(A,B) :- [A>=2,A+ -B=0].
'fib[1]'(A,B) :- [A+ -B>= -1,B>=1,-A+B>=0].
'fib(2)'(A,B) :- [A>=4,-2*A+B>= -3].
'fib[2]'(A,B) :- [A>=0,B>=1,-A+B>=0].
```

inductive solution to the original set of clauses

```
fib(A,B) :- [-A>= -1,A>=0,B=1].
fib(A,B) :- [-A>= -1,A>=0,B=1].
fib(A,B) :- [A>=2,A+ -B=0].
fib(A,B) :- [A+ -B>= -1,B>=1,-A+B>=0].
fib(A,B) :- [A>=4,-2*A+B>= -3].
fib(A,B) :- [A>=0,B>=1,-A+B>=0].
```

Experiments

Program	Result	Time(s)	dim(k)
addition	safe	4	1
bfprt	safe	2	2
binarysearch	safe	2	1
countZero	safe	2	1
identity	safe	2	1
merge	safe	2	1
palindrome	safe	1	1
fib	safe	2	2
revlen	safe	1	1
avg. time(s)		2	

Conclusions and Future Work

- solve non-linear clauses using only a linear solver.
- The approach seems feasible for solving solvable non-linear clauses.

Other possible directions:

- generation of counterexamples if the clauses cannot be solved.
- refinement.

Thank you