Failure Tabling Constraint Logic Programming by Interpolation

Jorge A. Navas
with Graeme Gange, Peter J. Stuckey, Harald Søndergaard, and Peter Schachte

The University of Melbourne
Department of Computing and Information Systems

ICLP'13, August 27 2013

Well-Known Limitations of CLP

- Standard CLP execution model is based on *depth-first search* with *left-to-right* clause selection
- Derivation tree (DT) may be huge even if many subtrees are redundant
- Even worse, DT may contain infinite derivations

Tabled CLP (TCLP)

Basic Idea:

Record calls and their answers for reuse in future calls

- Similar idea to LP tabling but TCLP makes explicit the need of the tabling execution on the constraint domain
- To check whether a call can be reused it requires *constraint entailment*
- To determine the calling constraint for a tabled call it needs *constraint projection*
- It has been successfully implemented by e.g. Chico et.al. (FLOPS'12) but there is a catch:
  - Many constraint domains have no constraint projection
  - Even if the domain has projection it is often too expensive

Failure Tabled Constraint Logic Programming (FTCLP)

- **Our contribution**: we propose a new concept of tabling by learning only from failed derivations
- **What it does**: it can prune redundant derivations and can also avoid non-terminating executions
- **How it works**: no use of constraint projection whatsoever!

FTCLP: High-Level Idea

Execute the CLP program top-down while labelling nodes in the DT

1. **reuse condition** that if implied by a goal's constraint store then it is guaranteed that no new answers can be generated
2. **set of answers** for a goal which is the constraint stores resulting from the successful derivations for the goal

Key technical contribution:

We will generate reuse conditions using *Craig interpolation*

Craig Interpolation

Given two FOL A and B such that $A \wedge B = \bot$, an **interpolant** I is another FOL s.t.

Craig Interpolation

Given two FOL A and B such that $A \wedge B = \bot$, an **interpolant** I is another FOL s.t.

1. $A \models I$

Craig Interpolation

Given two FOL A and B such that $A \wedge B = \bot$, an **interpolant** I is another FOL s.t.

1. $A \models I$
2. $I \wedge B = \bot$

Craig Interpolation

Given two FOL A and B such that $A \wedge B = \bot$, an **interpolant** I is another FOL s.t.

1. $A \models I$
2. $I \wedge B = \bot$
3. $vars(I) \subseteq vars(A) \cap vars(B)$

Craig Interpolation

Given two FOL A and B such that $A \wedge B = \bot$, an **interpolant** I is another FOL s.t.

1. $A \models I$
2. $I \wedge B = \bot$
3. $vars(I) \subseteq vars(A) \cap vars(B)$

- Example with $LA(\mathbb{Q})$:

  $A \equiv x = 2 \wedge y = x + 3 \wedge z = y + 4$ and $B \equiv z \geq 20$

  Examples of interpolants: $I_1 \equiv z \leq 9$ and $I_2 \equiv z < 20$
- Interpolants are not unique!
- I is a form of **weaker projection** that still maintains the unsatisfiability of $A \wedge B$
- Efficient algorithms for QF fragments of EUF, DL, LA, Arrays, and BV where interpolants can be extracted from the refutation proof in linear time on the size of the proof!

Toy Program

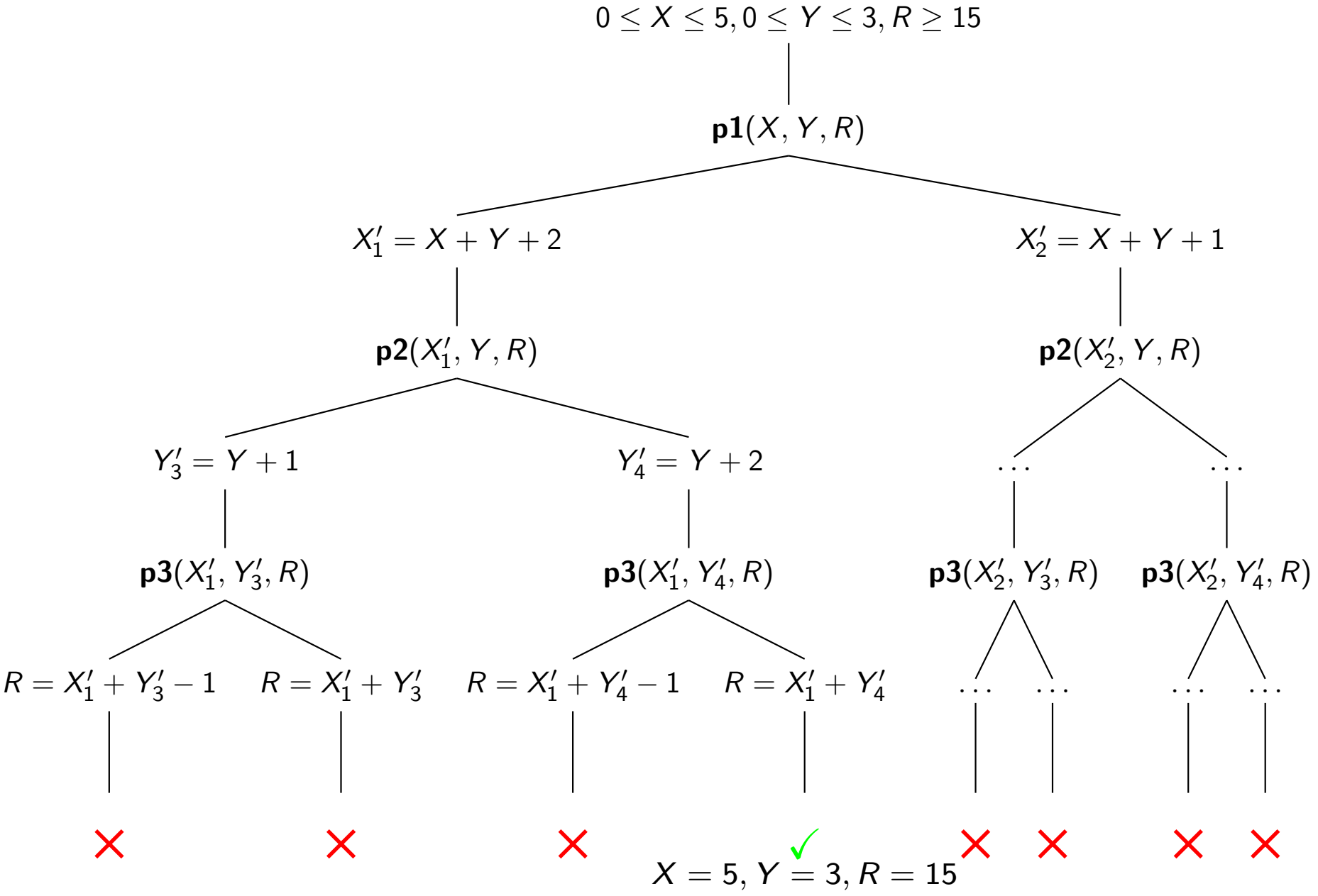
```
?- 0 <= X, X <= 5, 0 <= Y, Y <= 3,
   R >= 15, p1(X,Y,R).

p1(X1,Y1,R1) :- X1' = X1 + Y1 + 2, p2(X1',Y1,R1).
p1(X2,Y2,R2) :- X2' = X2 + Y2 + 1, p2(X2',Y2,R2).

p2(X3,Y3,R3) :- Y3' = Y3 + 1, p3(X3,Y3',R3).
p2(X4,Y4,R4) :- Y4' = Y4 + 2, p3(X4,Y4',R4).

p3(X5,Y5,R5) :- R5  = X5 + Y5 - 1.
p3(X6,Y6,R6) :- R6  = X6 + Y6.
```

CLP Execution of the Program



Generate Interpolants from Failed Derivations (I)

- Consider all CLP constraints along the leftmost derivation:

$$(0 \leq X \leq 5) \wedge (0 \leq Y \leq 3) \wedge (R \geq 15) \wedge$$
$$(X_1' = X + Y + 2) \wedge (Y_3' = Y + 1) \wedge (R = X_1' + Y_3' - 1)$$

- We can partition in several ways:

$$\overbrace{(0 \leq X \leq 5) \wedge \ldots \wedge (X_1' = X + Y + 2) \wedge (Y_3' = Y + 1) \wedge}^{A} \quad \overbrace{(R = X_1' + Y_3' - 1)}^{B}$$

$$\overbrace{(0 \leq X \leq 5) \wedge \ldots \wedge (X_1' = X + Y + 2) \wedge}^{A} \quad \overbrace{(Y_3' = Y + 1) \wedge (R = X_1' + Y_3' - 1)}^{B}$$

$$\overbrace{(0 \leq X \leq 5) \wedge \ldots \wedge}^{A} \quad \overbrace{(X_1' = X + Y + 2) \wedge (Y_3' = Y + 1) \wedge (R = X_1' + Y_3' - 1)}^{B}$$

- Let us focus on the second partition:

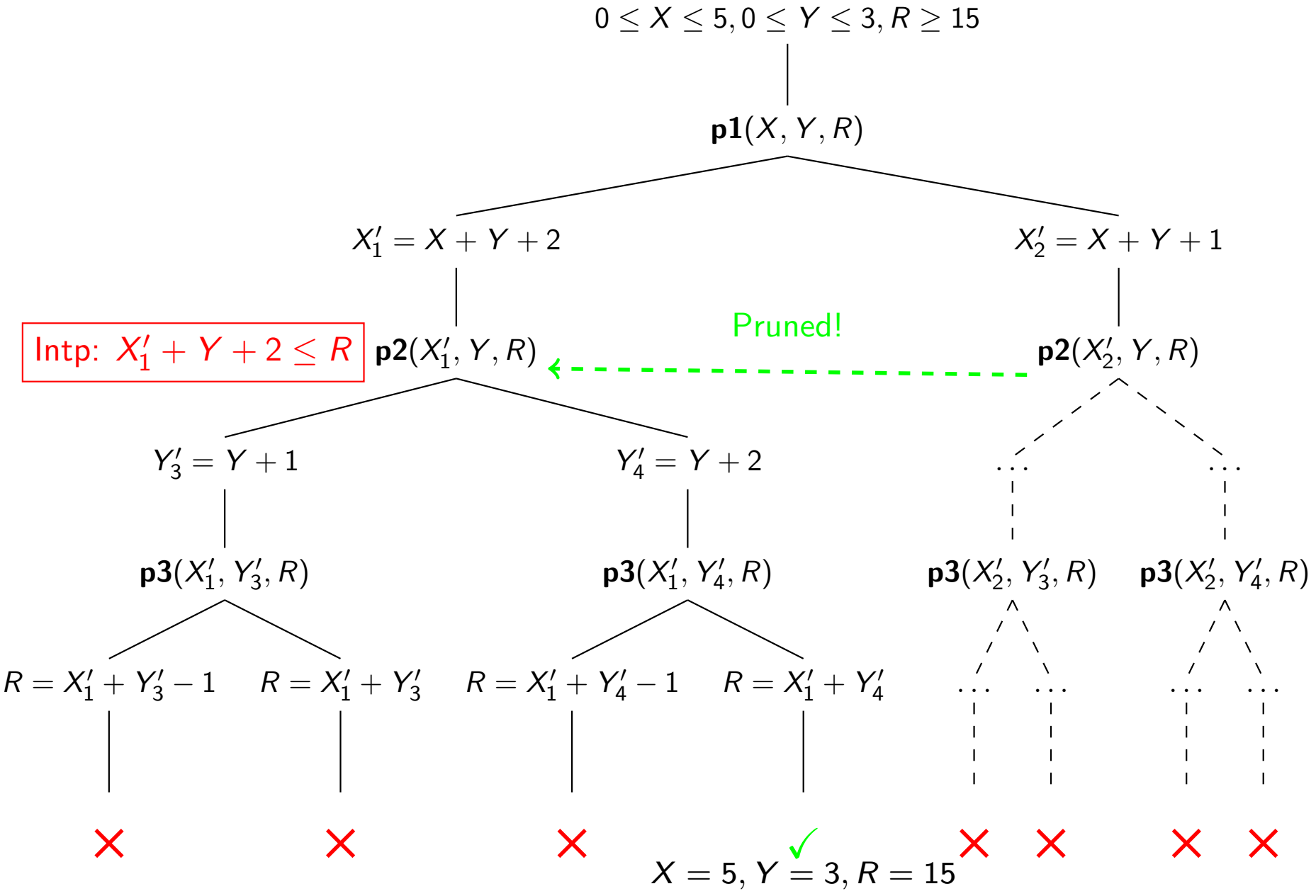
$$\overbrace{(0 \leq X \leq 5) \wedge (0 \leq Y \leq 3) \wedge (R \geq 15) \wedge (X_1' = X + Y + 2) \wedge}^{A}$$
$$\overbrace{(Y_3' = Y + 1) \wedge (R = X_1' + Y_3' - 1)}^{B}$$

Generate Interpolants from Failed Derivations (II)

$$\overbrace{(0 \leq X \leq 5) \wedge (0 \leq Y \leq 3) \wedge (R \geq 15) \wedge (X'_1 = X + Y + 2) \wedge}^{A}$$
$$\overbrace{(Y'_3 = Y + 1) \wedge (R = X'_1 + Y'_3 - 1)}^{B}$$

- A contains all the constraints up to $\mathbf{p2}(X'_1, Y, R)$ while B is the rest of constraints up to the end of the failed derivation
- We can compute an interpolant for the partition (A, B). E.g., $I_{\mathbf{p2}} \equiv X'_1 + Y + 2 \leq R$
- Note that the only common variables for (A, B) are X'_1, Y, and R which are the body atom variables!
- $I_{\mathbf{p2}}$ is a *generalization* of the constraint store when $\mathbf{p2}(X'_1, Y, R)$ is called but preserving the fact that the derivation failed!

Using Interpolants for Pruning Subtrees



A Snippet from a Verification Problem

```
int x,y,i,j;
x = i; y = j;
while (x != 0){
  x = x - 1;
  y = y - 1;
}
if (i == j)
  assert(y <= 0);
```

```
?- X = I, Y = J, prove(X,Y,I,J).
prove(X1,Y1,I1,J1):-
    X1 <> 0,
    X1'= X1 - 1, Y1' = Y1 - 1,
    prove(X1',Y1',I1,J1).

prove(X2,Y2,I2,J2):-
    X2 = 0,
    error(X2,Y2,I2,J2).

error(_,Y3,I3,J3) :-
    I3 = J3, Y3 > 0.
```

Pruning Infinite Derivations

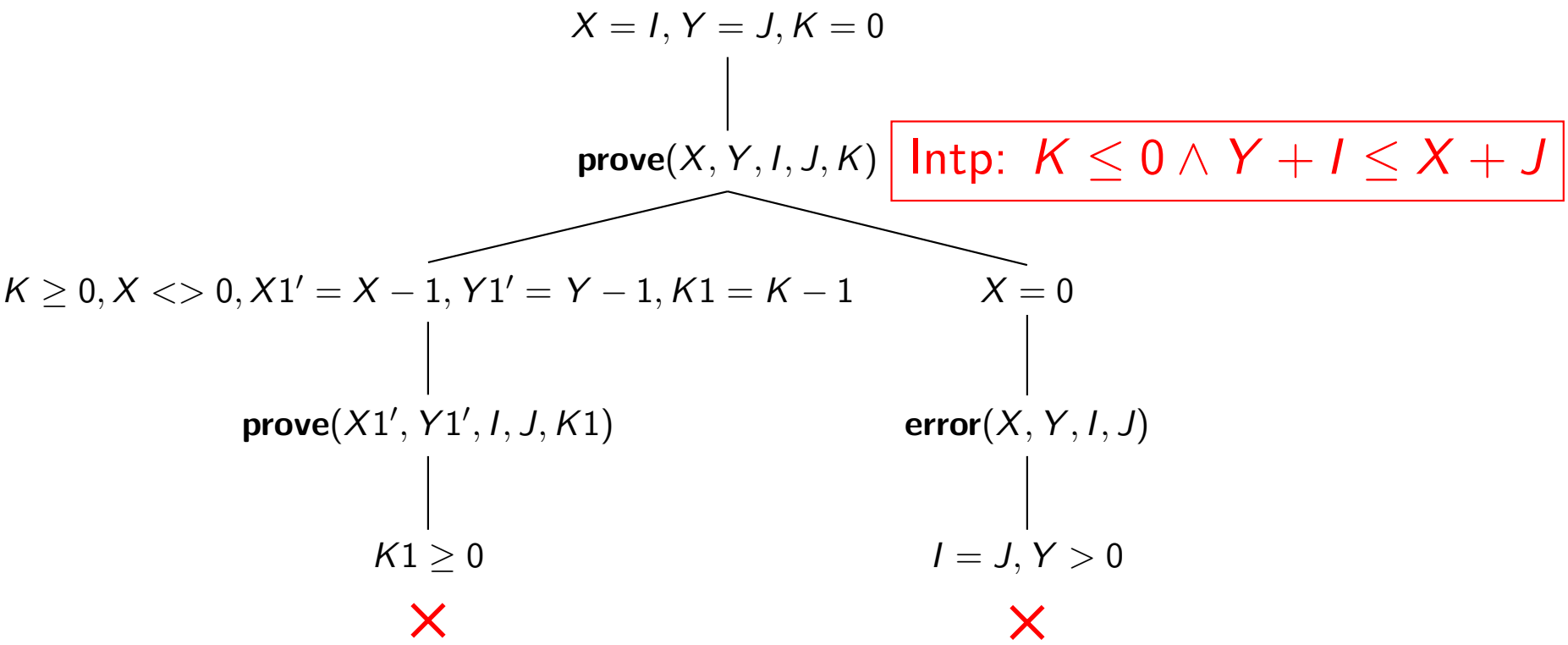
Main idea

1. Counter instrumentation-based transformation to ensure a finite DT
2. Compute interpolants and use them as a heuristics to subsume all infinite child derivations
3. If yes, we are done. Otherwise, we will unroll the DT

```
prove(X1,Y1,I1,J1,K1):-
    K1 >= 0,
    X1 <> 0,
    X1'= X1 - 1, Y1' = Y1 - 1,
    K1'= K1 - 1,
    prove(X1',Y1',I1,J1,K1').
prove(X2,Y2,I2,J2,_):-
    X2 = 0, error(X2,Y2,I2,J2).

error(_,Y3,I3,J3)   :- I3 = J3, Y3 > 0.
```

Finite Derivation Tree with Fake Interpolants



Building a Finite DT by Removing Fake Interpolants

- Trick: we eliminate the dependency on K doing $Intp' \equiv Intp \wedge K = 0$
- Check whether $Intp'$ is inductive invariant trough all recursive clauses:

$$
\begin{array}{ll}
(Y_1 + I_1 \leq X_1 + J_1) & \wedge \\
X_1 \neq 0, X_1' = X_1 - 1, Y_1' = Y_1 - 1 & \models \\
(Y_1' + I_1 \leq X_1' + J_1) &
\end{array}
$$

Building a Finite DT by Removing Fake Interpolants

- Trick: we eliminate the dependency on K doing $Intp' \equiv Intp \wedge K = 0$
- Check whether $Intp'$ is inductive invariant trough all recursive clauses:

$$\begin{array}{ll} (Y_1 + I_1 \leq X_1 + J_1) & \wedge \\ X_1 \neq 0, X_1' = X_1 - 1, Y_1' = Y_1 - 1 & \models \\ (Y_1' + I_1 \leq X_1' + J_1) & \checkmark \end{array}$$

- A projection-based method would possibly generate $Y = J \wedge I = X$ which is not invariant!

Observation #1

The inductive invariant check can be expensive so we rely on SMT

Observation #2

If the check does not hold then we increment K and try again. This process may not terminate!

Implementation

- *Proof-of-concept* CLP meta-interpreter written in Ciao and SMT solver MathSAT for checking satisfiability and generation of interpolants
- Implemented some important optimizations to reduce the number of interpolants (e.g., use of inductive sequence interpolants)
- Still some important limitations:
  - It is not integrated within the tabling Ciao implementation
  - MathSAT behaves poorly in an incremental setting. E.g., the clpr package from Ciao (by C.Holzbaur) is at least one order of magnitude faster than MathSAT

Some Preliminary Experiments: Finite Derivations

Execution of a CLP program that implements the RCSP problem with data from the OR-library. All answers with cost $\leq$ C are generated

		CLP			FTCLP			FTCLP+opt
C	Answers	Time(s)	States	Failure	Time(s)	States	Failure	Time(s)
150	2	1.7	5112	1056	4.3	4269	862	3.8
200	16	14.4	40342	8302	22.5	16870	3026	18.8
225	58	44.1	116684	23961	47.3	28624	4636	37.5
250	164	138	323154	65865	88.2	43796	6327	67.0
275	451	450	827770	167969	148.6	61276	7529	111.5

Some Preliminary Experiments: Infinite Derivations

Comparing FTCLP with several verifiers for some verification problems ERR indicates an error, ∞ timeout after 5 min, and UNSAFE is false positive

Program	FTCLP	BLAST	HSF	TRACER	TRACER SP
t1	∞	∞	0.3s	∞	∞
t1-a	0.1s	ERR	0.2s	∞	∞
t2	0.1s	0.1s	UNSAFE	0.1s	∞
t3	0.1s	0.1s	0.25s	0.1s	∞
t4	0.1s	0.6s	0.3s	0.2s	∞
t5	0.2s	0.2s	0.3s	0.1s	0.1s

Conclusions

- We have presented a new concept of tabling called **Failure Tabled CLP**
- The main novelty is that constraint projection is not needed
- Instead, we use interpolants which are a form of **weak projection**. They are available for many useful theories and cheap to compute
- This is a *partially* successful story of bringing ideas from another area (software verification) to LP/CLP community
- Future work: integration into a real tabling system, more incremental solvers, and more experiments
- Available at `http://code.google.com/p/ftclp`

Questions?

Some Backup Slides

Interpolants for Conjunctions of $LA(\mathbb{Q})$-Literals

- We cover only the fragment of equalities and weak inequalities in the form $0 \leq c$ where c is a negative numerical constant
- Equalities can be split into a pair of inequalities. E.g., $c = 0$ is translated into $0 \leq c$ and $0 \leq -c$
- We can generate refutations and their interpolants following the proof rules:

$$\frac{}{(A, B) \vdash 0 \leq x \quad [x]} \; (0 \leq x) \in A$$

$$\frac{}{(A, B) \vdash 0 \leq x \quad [0]} \; (0 \leq x) \in B$$

$$\frac{(A, B) \vdash 0 \leq x \quad [x'] \quad (A, B) \vdash 0 \leq y \quad [y']}{(A, B) \vdash 0 \leq c_1 x + c_2 y \quad [c_1 x' + c_2 y']} \; c1, c2 > 0$$

- We derive the interpolant for a linear combination of inequalities by taking the same linear combination of the contributions of A

Example: Interpolants for Conjunctions of $LA(\mathbb{Q})$-Literals

$$A = \{(0 \leq x - 2), (0 \leq -x + 2), (0 \leq y - x - 3), (0 \leq x - y + 3), \\ (0 \leq z - y - 4), (0 \leq y - z + 4)\}$$
$$B = \{(0 \leq z - 20)\}$$

A $LA(\mathbb{Q})$ proof of unsatisfiability:

$$\dfrac{\dfrac{\dfrac{1 * (0 \leq -x + 2) \qquad 1 * (0 \leq x - y + 3)}{1 * (0 \leq -y + 5)} \qquad 1 * (0 \leq y - z + 4)}{1 * (0 \leq -z + 9)} \qquad 1 * (0 \leq z - 20)}{(0 \leq -11)}$$

and its corresponding inequality interpolants:

$$\dfrac{\dfrac{\dfrac{[-x + 2] \qquad [x - y + 3]}{[-y + 5]} \qquad [y - z + 4]}{[-z + 9]} \qquad [0]}{[-z + 9]}$$

Thus, the interpolant obtained is $0 \leq -z + 9 \equiv z \leq 9$

Verification Programs (I)

t1.c (from Angelis et.al.)	t1.pl
`int x=1;` `int y=0;` `while(*){` `  x=x+y, y++;` `}` `//if (y >= 0)` `assert(x >= y);`	Safe inductive invariant: $I(X, Y) \Longrightarrow Y \geq 0 \wedge X \geq Y$ `t1 :-` `  {X .=. 1, Y .=. 0},` `  l(X,Y).` `l(X,Y):-` `  {X1 .=. X+Y, Y1 .=. Y+1},` `  l(X1,Y1).` `l(X,Y):- error(X,Y).` `error(X,Y):- {Y .>. X}.`

Verification Programs (II)

t3.c (from Ball et.al.)	t3.pl
`int lock,old,new;` `old=0; lock=0;` `new=old+1;` `while(new!=old){` `  lock=1;` `  old=new;` `  if (*) {` `  lock=0;` `  new++;` `  }` `}` `assert(lock != 0);`	Safe ind. inv.: $I(Lock, Old, New) \Longrightarrow Old + 1 \leq New$ `?- {Lock .=. 0,` `  Old .=. 0,` `  New .=. Old + 1},` `  l(Lock,Old,New).` `l(Lock,Old,New):-` `  {New .<>. Old},` `  l_body(Lock,Old,New,Lock1,Old1,New1),` `  l(Lock1,Old1,New1).` `l(Lock,Old,New):-` `  { New .=. Old},` `  error(Lock).` `l_body(_,Old,New,Lock1,Old1,New1):-` `  {Lock1 .=. 0,`

Verification Programs (III)

t4.c (from Beyer et.al.)

```
int i,a,b,n;
i=0; a=0; b=0;
assume(n > 0);
while (i < n){
   if (*){
     a=a+1;
     b=b+2;
   }
   else{
     a=a+2;
     b=b+1;
   }
    i++;
}
assert(a+b == 3*n);
```

t4.pl

Safe inductive invariant: $I(I, A, B, N) \implies A + B \leq 3 * I$

```
?-  {N .>. 0, I .=. 0,
     A .=. 0, B .=. 0 }, l(I,A,B,N).
l(I,A,B,N):-
    {I .<. N},
    l_body(A,B,A1,B1),
    {I1 .=. I+1},
    l(I1,A1,B1,N).
l(I,A,B,N):- {I .>=.N}, error(A,B,N).
l_body(A0,B0,A1,B1):-
    {A1 .=. A0+1, B1 .=. B0+2}.
l_body(A0,B0,A1,B1):-
    {A1 .=. A0+2, B1 .=. B0+1}.
error(A,B,N):-  {A + B .<>. 3 * N}.
```

Verification Programs (IV)

t5.c (SSH)	t5.pl
`` int e=0; int s=2; while (*) { if (s == 2){ if (e == 0) e=1; s=3; } else if (s == 3){ if (e == 1) e=2; s=4; } else if (s == 4){ if (e == 3) error(); s=5; } } ```	Safe inductive invariant: $I(E,S) \Longrightarrow E \leq 2 \wedge S \leq 4$ ``` ?- {E .=. 0, S .=. 2}, l(E,S). l(E0,S0):- l_body(E0,S0,E1,S1), l(E1,S1). l(E,S) :- error(E,S). error(E,S):- {E .=. 3, S .=. 4}. l_body(E0,S0,E1,S1):- {S0.=.2}, l_body_1(E0,S0,E1,S1). l_body(E0,S0,E1,S1):- {S0.=.3}, l_body_2(E0,S0,E1,S1). l_body(E0,S0,E1,S1):- {S0.=.4}, l_body_3(E0,S0,E1,S1). l_body(E0,S0,E1,S1):- {S0.>.4,E1.=.E0,S1.=.S0}. l_body_1(E0,_S0,E2,S2):- l_body_1_1(E0,E1), {S2 .=. 3, E2 .=. E1}. l_body_2(E0,_S0,E2,S2):- l_body_2_1(E0,E1), {S2 .=. 4, E2 .=. E1}. l_body_3(E0,S0,E1,S1):- l_body_3_1(E0,E1,S0,S1). l_body_1_1(E0,E1):- {E0 .=. 0, E1 .=. 1}. l_body_1_1(E0,E1):- {E0 .<>. 0, E1 .=. E0}. l_body_2_1(E0,E1):- {E0 .=. 1, E1 .=. 2}. l_body_2_1(E0,E1):- {E0 .<>. 1, E1 .=. E0}. l_body_3_1(E0,E1,S0,S1):- {S0.=.4,E0.=.2,E1.=.E0,S1.=.S0}. l_body_3_1(E0,E1,S0,S1):- {S0.<>.4,S1.=.5,E1.=.E0}. l_body_3_1(E0,E1,S0,S1):- {E0.<>.2,S1.=.5,E1.=.E0}. ```