Removing Algebraic Data Types from Constrained Horn Clauses Using Difference Predicates*

Emanuele De Angelis, CNR-IASI, Rome, Italy
Fabio Fioravanti, Univ. Chieti-Pescara, Italy
Alberto Pettorossi, Univ. Rome Tor Vergata, Italy
Maurizio Proietti, CNR-IASI, Rome, Italy

HCVS - Luxembourg (virtually), March 28, 2021

Paper presented at IJCAR 2020

Overview

1. Constrained Horn clauses for verifying programs computing on algebraic data types (ADTs);

2. ADT Removal: Transforming CHCs on ADTs into CHCs on basic types (e.g., integers and booleans);

3. ADT removal with difference predicates (related to lemmas in proofs by induction);

4. Experimental evaluation and comparison with induction-based methods.

1

Constrained Horn clauses for program verification

Constrained Horn clauses

- Constrained Horn clauses are a fragment of FOL

  (body) $\forall(A_1 \wedge \ldots \wedge A_n \wedge c \Rightarrow A_0)$ *(head)*

  (1) $A_1, \ldots, A_n$, $n \geq 0$, are *atoms*, (2) c is a *constraint* in a first order theory T, (3) A_0 is an *atom or false*.

- Prolog syntax: A_0 :- c, A_1, … , A_n.

- Satisfiability: Given a set S of CHCs, has $S \cup T$ a model?

- Solution of S: A model of $S \cup T$, expressed in T *(if sat)*; the existence of a solution implies satisfiability, not vice versa.

- Solvers compute solutions (if any) for CHCs over Linear Integer/Real Arithmetic, Booleans, Arrays, Bit-vectors,…

Program verification with CHCs

- Summing the first *n* non-negative integers

Hoare triple
{n≥0} x=0; y=0; while (x<n) { x=x+1; y=x+y} {y≥x}

Translation

Constrained Horn Clauses (Prolog syntax)

```
p(X, Y, N) :- N≥0, X=0, Y=0.                         %Init
p(X1, Y1, N) :- X<N, X1=X+1, Y1=X1+Y, p(X, Y, N).   %Loop
false :- Y<X, X≥N, p(X, Y, N).                       %Exit
```

- Hoare triple valid iff CHCs satisfiable

- Solution of the CHCs: p(X, Y, N) ≡ (X≥0, Y≥X, N≥0)   % loop invariant

Programs on ADTs

- Statically typed, call-by-value, first order, functional language.
- Computing the sum and the maximum of the absolute values of the elements of a list:

```
type list = Nil | Cons of int * list;;

let rec asum l = match l with
  | Nil →  0
  | Cons(x, xs) → (abs x) + asum xs;;

let rec listmax l = match l with
  | Nil → 0
  | Cons(x, xs) → max x (listmax xs);;

 % Property: ∀l. asum l >= listmax l
```

Translation into CHCs

- The program and the property are translated into CHCs:

```
asum([],S) :- S=0.
asum([X|Xs],S) :- S=S1+A, abs(X,A), asum(Xs,S1).
listmax([],M) :- M=0.
listmax([X|Xs],M) :- max(A,M1,M), listmax(Xs,M1).
false :- S<M, asum(L,S), listmax(L,M).          % Property
```

f(x,y)
"f x evaluates to y"

- The property holds iff the set of clauses is satisfiable;

- CHC solvers cannot compute a solution because the set of clauses has no model expressible in the quantifier-free Theory of Lists and Linear Integer Arithmetic (LIA).

Solving CHCs on ADTs

- Approach 1 [Reynolds-Kuncak 2015, Unno-Torii-Sakamoto 2017]: Extend CHC/SMT solvers with induction rules;
- Approach 2:
  Transform CHCs S on ADTs into CHCs S':
  - S' on basic types only (e.g., integers or booleans)
  - The transformation is sound: S' satisfiable $\Rightarrow$ S satisfiable;
- Advantage of Approach 2: No need of extending CHC solvers.
- Related to techniques for eliminating data structures in FP and LP:
  - Deforestation [Wadler '88],
  - Existential Variable Elimination by Unfold/Fold [PP '91].

2

Transforming constrained Horn clauses

Unfold/Fold transformations of CHCs

- Unfold. (Linear Resolution)

  replace H :- c, A, G. where: A_1 :- d_1,G_1. ... A_m :- d_m,G_m.

  by (H :- c, d_1,G_1, G.)ϑ_1 ... (H :- c, d_m,G_m, G.)ϑ_m

  where ϑ_i is the most general unifier of A and Ai.

- Fold. (inverse Linear Resolution)

  replace H :- d, Bϑ, G. where: K :- c, B. and $\boldsymbol{T} \vDash d \rightarrow c\vartheta$

  by H :- d, Kϑ, G.

… Unfold/Fold transformations of CHCs

- Other rules: Delete clauses with unsat body, Apply functionality of predicates.

- Under suitable conditions, unfold/fold transformations are sound.

ADT Removal by Unfold/Fold

- Property:
  false :- S<M, asum(**L**,S), listmax(**L**,M). **L**: list S,M: int

ADT Removal by Unfold/Fold

- Property:
  false :- S<M, asum(L,S), listmax(L,M). L: list S,M: int
- Define a new predicate (on integers):
  p(S,M) :- asum(L,S), listmax(L,M).

ADT Removal by Unfold/Fold

- Property:
  false :- S<M, asum(**L**,S), listmax(**L**,M).   **L**: list   S,M: int
- Define a new predicate (on integers):
  p(S,M) :- asum(**L**,S), listmax(**L**,M).
- Unfold wrt asum and listmax:
  p(S,M) :- S=0, M=0.
  p(S,M) :- S=S1+A, abs(X,A), max(X,M1,M), asum(**Xs**,S1), listmax(**Xs**,M1).

ADT Removal by Unfold/Fold

- Property:
  false :- S<M, asum(L,S), listmax(L,M).

  L: list S,M: int

- Define a new predicate (on integers):
  p(S,M) :- asum(L,S), listmax(L,M).

  Variant of

- Unfold wrt asum and listmax:
  p(S,M) :- S=0, M=0.
  p(S,M) :- S=S1+A, abs(X,A), max(X,M1,M), asum(Xs,S1), listmax(Xs,M1).

- Fold:
  p(S,M) :- S=0, M=0.
  p(S,M) :- S=S1+A, abs(X,A), max(X,M1,M), **p**(S1,M1).
  false :- S<M, **p**(S,M).

  Eliminate all lists

Solving the transformed CHCs on LIA

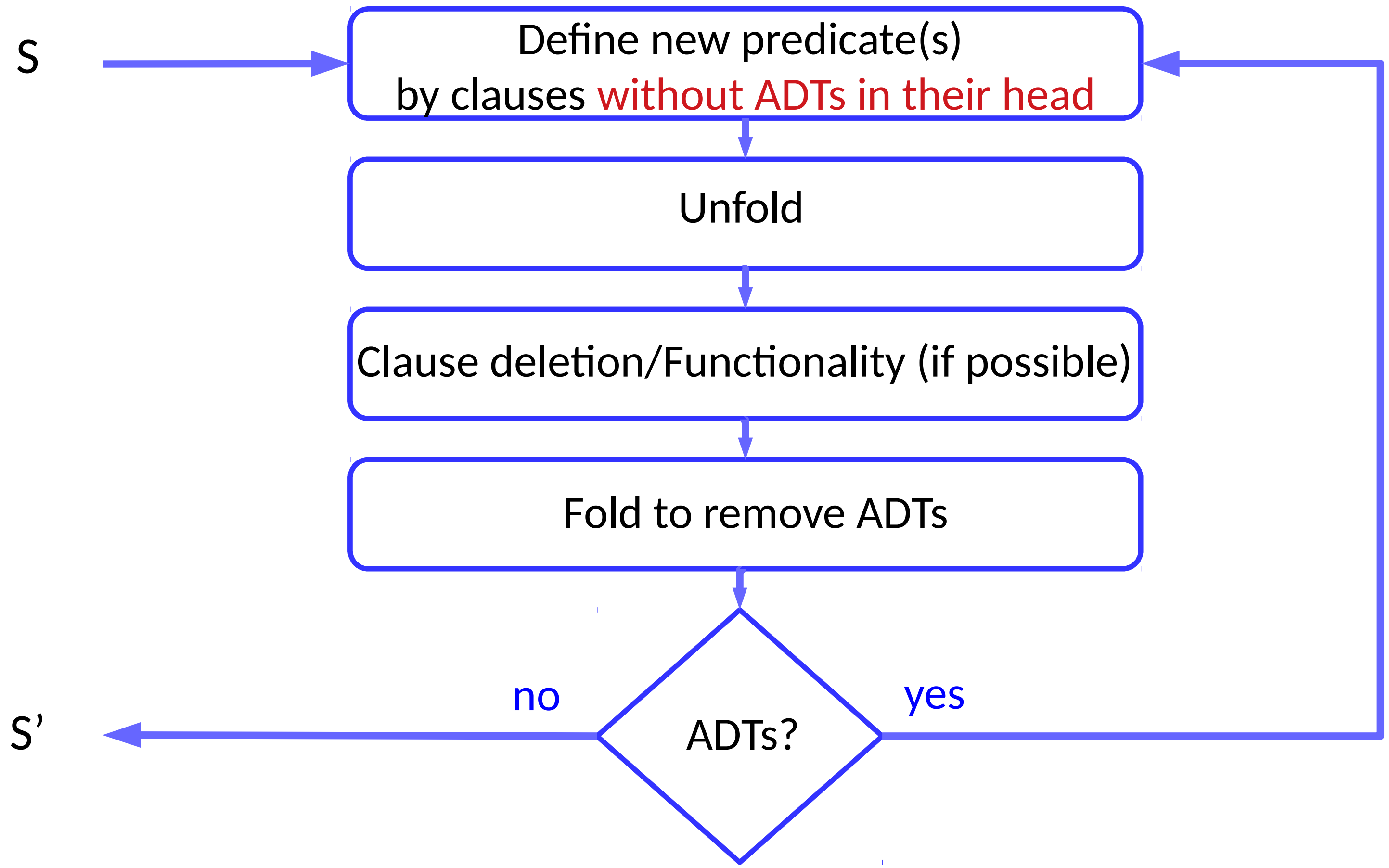
```
p(S,M) :- S=0, M=0.
p(S,M) :- S=S1+A, abs(X,A), max(X,M1,M), p(S1,M1).
false :- S<M, p(S,M).
```

- Solved by Eldarica (and Spacer/Z3) without induction rules.
- Eldarica computes the following model in LIA:

  p(S,M) $\equiv$ (S>=M, M>=0)

- Soundness guaranteed by unfold/fold rules
  $\Rightarrow$ ∀l. asum l >= listmax l holds

ADT Removal Algorithm (Basic version)



Limitation of the basic ADT removal algorithm

- The algorithm does not support lemma generation, and will not terminate when lemmas are needed.
- An extra transformation rule for lemma generation: introducing difference predicates.

ADT Removal with Difference Predicates

Insertion Sort

```
type list = Nil | Cons of int * list;;

let rec ins x l = match l with
  | Nil -> Cons(x,Nil)
  | Cons(y,ys) -> if x<=y then Cons(x,Cons(y,ys)) else Cons(y,ins x ys);;

let rec sort l = match l with
  | Nil -> Nil
  | Cons(x,xs) -> ins x (sort xs);;

let rec count x l = match l with
  | Nil -> 0
  | Cons(y,ys) -> if x=y then 1 + count x ys else count x ys;;

% Property:  ∀l.∀x. (count x l) = (count x (sort l))
```

Insertion Sort: Translation into CHCs

```
ins(A,[ ],[A]).
ins(A,[X|Xs],[A,X|Xs]) :- A=<X.
ins(A,[X|Xs],[X|Ys]) :- A>X, ins(A,Xs,Ys).
sort([ ],[ ]).
sort([X|Xs],S) :- sort(Xs,S1), ins(X,S1,S).
count(X,[ ],0).
count(X,[H|T],N) :- X=H,  N=M+1, count(X,T,M).
count(X,[H|T],N) :- X=\=H, count(X,T,N).
false :- N1=\=N2, count(X,L,N1), sort(L,S), count(X,S,N2).    % Property
```

- State-of-the-art CHC solvers cannot solve these clauses

Insertion Sort: ADT removal

false :- N1=\=N2, count(X,L,N1), sort(L,S), count(X,S,N2).   L,S: list, X,N1,N2: int

Insertion Sort: ADT removal

false :- N1=\=N2, count(X,L,N1), sort(L,S), count(X,S,N2).   L,S: list, X,N1,N2: int

Define a new predicate (on integers):
p1(X,N1,N2) :- count(X,L,N1), sort(L,S), count(X,S,N2).

Insertion Sort: ADT removal

false :- N1=\=N2, count(X,L,N1), sort(L,S), count(X,S,N2).   L,S: list, X,N1,N2: int

Define a new predicate (on integers):
p1(X,N1,N2) :- count(X,L,N1), sort(L,S), count(X,S,N2).

Unfold (and Rename Variables):
p1(X,0,0).
p1(X,M,K) :- M=N1+1, count(X,L,N1), sort(L,S), ins(X,S,S2), count(X,S2,K).
p1(X,N1,N2) :- X=\=Y, count(X,L,N1), sort(L,S), ins(Y,S,S2), count(X,S2,N2).

Insertion Sort: ADT removal

false :- N1=\=N2, count(X,L,N1), sort(L,S), count(X,S,N2).   L,S: list, X,N1,N2: int

Define a new predicate (on integers):
p1(X,N1,N2) :- count(X,L,N1), sort(L,S), count(X,S,N2).

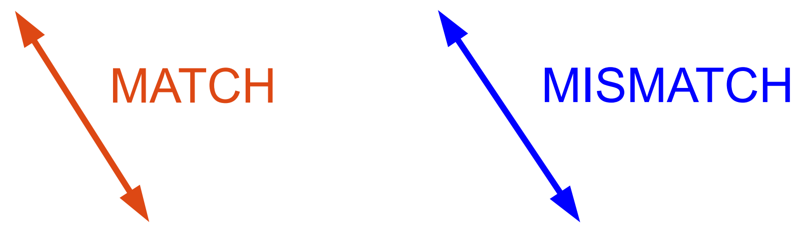
MATCH   MISMATCH

Unfold (and Rename Variables):
p1(X,0,0).
p1(X,M,K) :- M=N1+1, count(X,L,N1), sort(L,S), ins(X,S,S2), count(X,S2,K).
p1(X,N1,N2) :- X=\=Y, count(X,L,N1), sort(L,S), ins(Y,S,S2), count(X,S2,N2).

Fold impossible (ADT removal introduces new predicates and does not terminate)

Insertion Sort: Difference predicate

p1(X,N1,N2) :- count(X,L,N1), sort(L,S), count(X,S,N2).



p1(X,M,K) :- M=N1+1, count(X,L,N1), sort(L,S), ins(X,S,S2), count(X,S2,K).

Difference predicate (on integers):

diff1(X,N2,K) :- count(X,S,N2), ins(X,S,S2), count(X,S2,K).

Insertion Sort: Differential Replacement

p1(X,N1,N2) :- count(X,L,N1), sort(L,S), count(X,S,N2).

MATCH MATCH

p1(X,M,K) :- M=N1+1, count(X,L,N1), sort(L,S), count(X,S,N2), diff1(X,N2,K).

~~ins(X,S,S2), count(X,S2,K)~~

Difference predicate (on integers):

diff1(X,N2,K) :- count(X,S,N2), ins(X,S,S2), count(X,S2,K).

Insertion Sort: Fold

```
p1(X,N1,N2) :- count(X,L,N1), sort(L,S), count(X,S,N2).
```

Fold

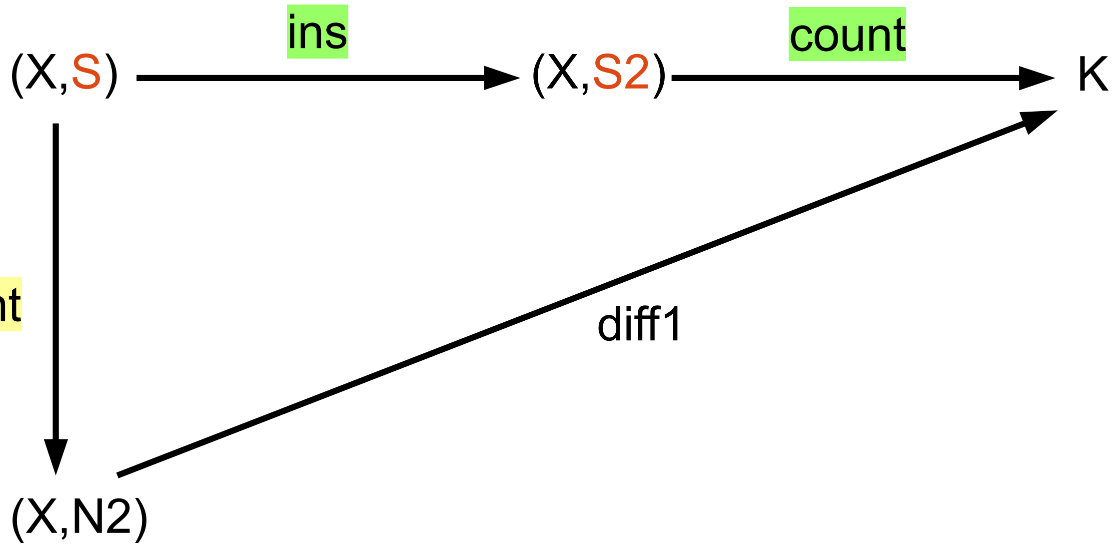
```
p1(X,M,K) :- M=N1+1, p1(X,N1,N2), diff1(X,N2,K).        % No lists
```

Difference predicate (on integers):

```
diff1(X,N2,K) :- count(X,S,N2), ins(X,S,S2), count(X,S2,K).
```

Soundness of Differential Replacement

Replace ins(X,S,S2), count(X,S2,K) by count(X,S,N2), diff1(X,N2,K).
"Difference"



Soundness: Suppose Cls U {C} → Cls U {D} by differential replacement, where count is a total function and its output variable N2 does not occur in C. If Cls U {D} is SAT then Cls U {C} is SAT.

Insertion Sort: Final set of clauses without lists

```
false :- N1=\=N2, p1(X,N1,N2).
p1(X,0,0).
p1(X,M,K) :- M=N1+1, p1(X,N1,N2), diff1(X,N2,K).
p1(X,N1,N2) :- X=\=Y, p1(Y,N1,N2b), diff2(X,Y,N2b,N2).
diff1(X,0,N2)  :- N2=N1+1, p2(X,N1).
diff1(X,N1,N2) :- N2=M2+1, N1=M1+1, p3(X,M2,M1).
diff1(X,N1,N2) :- X=<Y, N2=N+1, X=\=Y, p4(X,Y,N,N1).
diff2(X,Y,0,0) :- Y=\=X.
diff2(X,Y,M,N) :- X=<Y, Y=\=X, M=K+1, p3(Y,N,K).
diff2(X,Y,M,N) :- X=<Z, Y=\=X, Y=\=Z, N=M, p5(Y,N).
diff2(X,Y,M,N) :- X>Y, N=H+1, M=K+1, diff2(X,Y,K,H).
p2(X,0).
p3(X,N1,N)  :- N1=N+1, p5(X,N).
p4(X,Y,N,N) :- X=<Y, X=\=Y, p5(X,N).
p5(X,0).
p5(X,N1) :- N1=N+1, p5(X,N).
```

% No lists

- New predicates introduced by ADT removal: **diff2,** p2, p3, p4, p5.

Insertion Sort: Satisfiability

- Eldarica computes a model in LIA:
- p1(A,B,C) ≡ (B=C, B>=0)

  diff1(A,B,C) ≡ (C=B+1, B>=0)
  diff2(A,B,C,D) ≡ (D=C, C>=0)

  p2(A,B) ≡ (B = 0)
  p3(A,B,C) ≡ (C=B-1, B>=1)
  p4(A,B,C,D) ≡ (D=C, C>=0, B>=A+1)
  p5(A,B) ≡ (B>=0)
- Property ∀l.∀x. (count x l) = (count x (sort l)) holds.

Difference Predicates and Lemma Discovery

- Eldarica model of difference predicate (renamed variables):

  diff1(X,N,K) ≡ (K=N+1, N>=0)

  where diff1 is defined as:

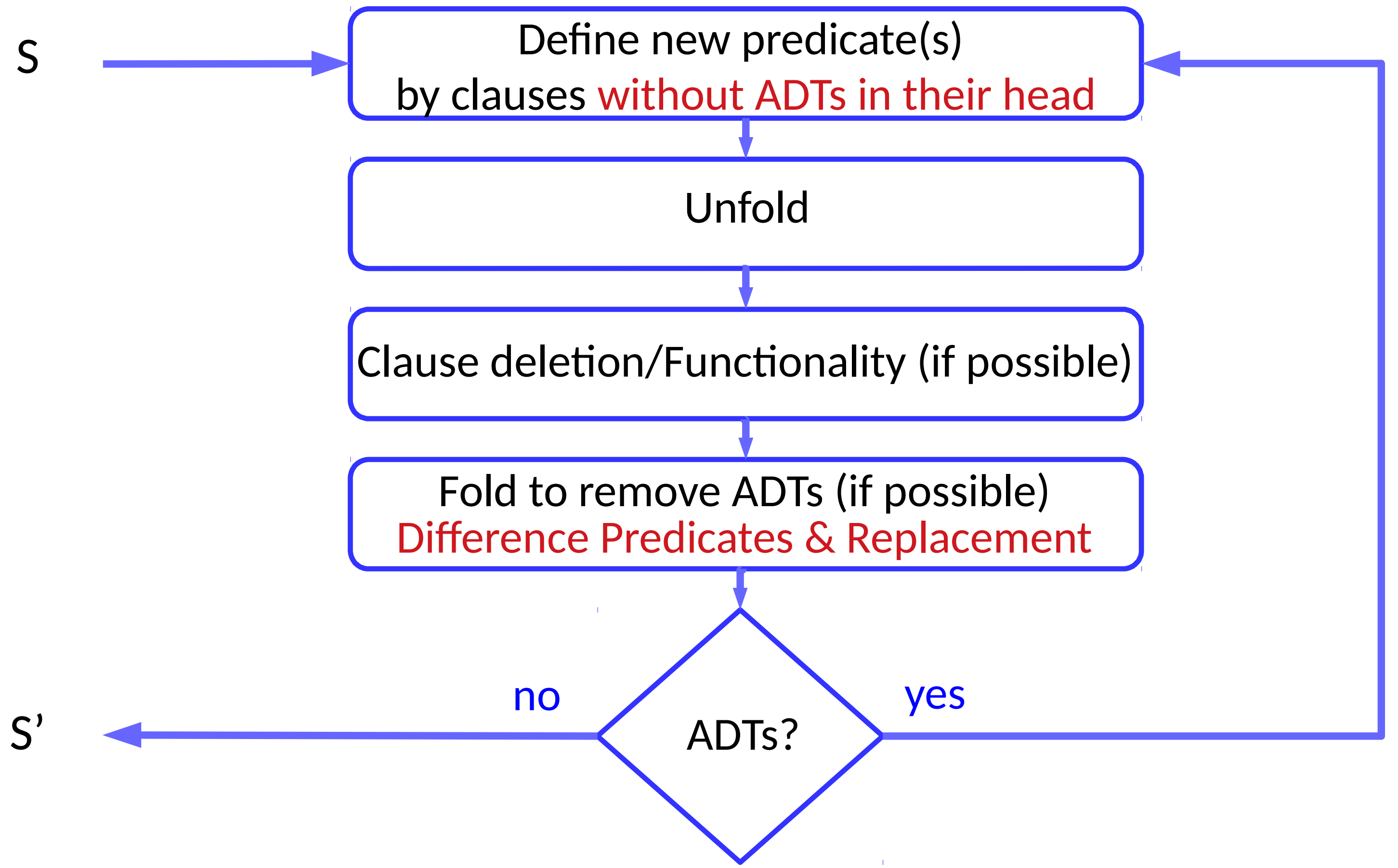
  diff1(X,N,K) :- count(X,S,N), ins(X,S,S1), count(X,S1,K).

- In functional notation can be rewritten as:

  $\forall((\text{count } x\ s) = n \wedge (\text{count } x\ (\text{ins } x\ s)) = k \rightarrow (k=n+1 \wedge n>=0))$

- Corresponds to a lemma in a proof by structural induction of the property.

ADT Removal with Difference Predicates



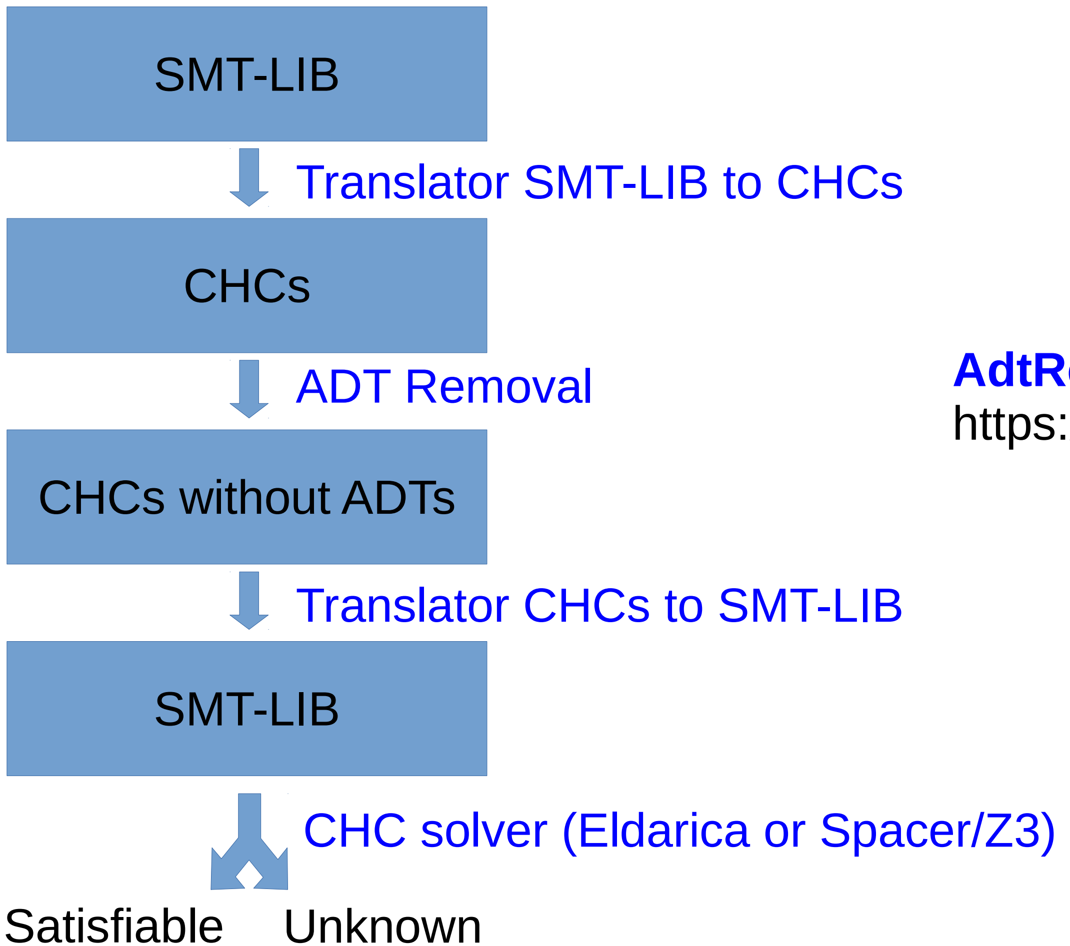
Soundness of ADT Removal with Difference Predicates

If the ADT removal algorithm terminates,

- S’ has no predicates on ADTs
- by the soundness of the transformation rules, if S’ is satisfiable then S is satisfiable

Experimental Evaluation

Implementation



AdtRem
https://fmlab.unich.it/adtrem/

Comparison with CVC4+Induction

- Benchmark: 169 satisfiability problems on ADTs (in SMT-LIB format).

	CLAM	HipSpec	IsaPlanner	Leon	Total
number of problems	53	11	63	42	169
Eldarica	0	2	4	9	15
Z3	6	0	2	10	18
$\mathcal{R}$ **ADT Removal**	(18) 36	(2) 4	(56) 59	(18) 30	(94) 129
Eldarica$_{\text{noADT}}$	(18) 32	(2) 4	(56) 57	(18) 29	(94) 122
Z3$_{\text{noADT}}$	(18) 29	(2) 3	(55) 56	(18) 26	(93) 114

(N) = number of problem solved without difference predicates.

- CVC4+Ind: SMT solver CVC4 extended with induction [Reynolds-Kuncak 15]

CVC4+Ind (dtt)	17	5	37	15	74

(dtt)= encoding of natural numbers as built-in SMT-LIB type *Int* (same as AdtRem encoding).

CVC4+Ind (dtt) with user-provided auxiliary lemmas 100
CVC4+Ind (dti) with double encoding of Nat and auxiliary lemmas 134

Comparison with CVC4+Ind by Examples

Problem	*Property proved by* ADTREM *and not by* CVC4
CLAM goal6	$\forall x, y.\ len(rev(append(x, y))) = len(x) + len(y)$
CLAM goal49	$\forall x.\ mem(x, sort(y)) \Rightarrow mem(x, y)$
IsaPlanner goal52	$\forall n, l.\ count(n, l) = count(n, rev(l))$
IsaPlanner goal80	$\forall l.\ sorted(sort(l))$
Leon heap-goal13	$\forall x, l.\ len(qheapsorta(x, l)) = hsize(x) + len(l)$

Problem	*Property proved by* CVC4 *and not by* ADTREM
CLAM goal18	$\forall x, y.\ rev(append(rev(x), y)) = append(rev(y), x)$
HipSpec rev-equiv-goal4	$\forall x, y.\ qreva(qreva(x, y), nil) = qreva(y, x)$
HipSpec rev-equiv-goal6	$\forall x, y, z.\ append(qreva(x, y), z) = qreva(x, append(y, z))$

Conclusions

- CHC transformations aid verification of programs that compute on ADTs;
- *ADT-removal; Solving*:
  - much more effective than *Solving CHCs on ADTs*;
  - competitive wrt *Solving extended with Induction*;
- Advantage of the transformation-based approach: Separation of inductive reasoning from CHC solving;
- Future work: Find sufficient conditions for the termination of the transformation (for classes of CHCs).

Thanks!

Questions?

AdtRem system and benchmarks:

https://fmlab.unich.it/adtrem/