# Proving Properties of Sorting Programs: A Case Study in Horn Clause Verification

**E. De Angelis**[1], F. Fioravanti[1],
A. Pettorossi[2], and M. Proietti[3]

[1] University of Chieti-Pescara `G. d'Annunzio'
[2] University of Rome `Tor Vergata'
[3] CNR - Istituto di Analisi dei Sistemi ed Informatica

# Overview

**Problem**

Verifying properties of functional programs on recursive datatypes

**Our approach**

- Translating program properties into Constrained Horn Clauses (CHCs) on recursive datatypes
- Transforming CHCs on recursive datatypes into equisatisfiable CHCs on integers and booleans

**Case study**: Verification of

- linear recursive
- non-linear recursive

sorting programs

# Functional programs on recursive datatypes

- **Statically typed, call-by-value, FO functional language (OCaml)**
- **Computing the sum (`listsum`) and the maximum (`listmax`) of the absolute values (`abs`) of the elements of a list:**

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

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

let rec listmax l = match l with
  | Nil -> 0
  | Cons(x, xs) → let m = listmax xs in max (abs x) m;;
```

**Property:** ∀l. `listsum` l `>=` `listmax` l

# Translating Program & Property into CHCs

```
false :- S<M, listsum(L,S), listmax(L,M).                % Query
listsum([],S) :- S=0.
listsum([X|Xs],S) :- S=S1+A, abs(X,A), listsum(Xs,S1).
listmax([],M):- M=0.
listmax([X|Xs],M):- abs(X,A), max(A,M1,M), listmax(Xs,M1).
abs(X,A) :- (X>=0, A=X); (X=<0, A=-X).
max(X,Y,Z) :- (X=<Y-1, Z=Y) ; (X>=Y, Z=X).
```

**The property holds iff the clauses are satisfiable;**
**Indeed these clauses are satisfiable but models are infinite disjunctions:**

```
listsum(L,S) :- (L=[], S=0) ; (L=[X], abs(X,S)) ;
                (L=[X,Y], abs(X,A), abs(Y,B), S=A+B); ...
listmax(L,M) :- (L=[], M=0) ; (L=[A], abs(X,M)) ;
                (L=[X,Y], abs(X,A), abs(Y,B), max(A,B,M)); ...
```

**CHC solvers (Eldarica, Z3) over the quantifier-free Theory of Lists and Linear Integer Arithmetic (LIA) cannot solve them (i.e., construct a model)**

# Solving CHCs on Recursive Datatypes
## (to find a model for the derived CHCs)

- **Approach 1:**

  Extend CHC solving by induction principles:

  **[Reynolds-Kuncak 2015**, **Unno-Torii-Sakamoto 2017].**

- **Approach 2 (this talk):**

  **Transform CHCs on recursive datatypes into equisatisfiable CHCs without recursive datatypes**

  (e.g., on integers and booleans only).

  **[Mordvinov-Fedyukovich 2017**, **De Angelis et al. 2018]**

  **Transformations inspired by techniques for eliminating inductive data structures:**

  Deforestation **[Wadler ‘88]**, Unnecessary Variable Elimination by Unfold/Fold **[PP ‘91]**, Conjunctive Partial Deduction + Redundant Argument Filtering **[DeSchreye et al. ‘99]**

# Recursive Datatype Elimination Algorithm

```
false :- S<M, listsum(L,S), listmax(L,M).     % L  existential list
```

**Define** a new predicate

```
list-sum-max(S,M) :- listsum(L,S), listmax(L,M).
```

variant conjunctions

**Unfold**

```
list-sum-max(S,M) :- S=0, M=0.
list-sum-max(S,M) :- S=S1+A, abs(X,A), max(A,M1,M),
                    listsum(L',S1), listmax(L',M1).
```

**Fold** (eliminate lists)

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

# Solving CHCs on LIA

```
false :- S<M, list-sum-max(S,M).

list-sum-max(S,M) :- S=0, M=0.
list-sum-max(S,M) :- S=S1+A,
  abs(X,A), max(A,M1,M), list-sum-max(S1,M1).
```

- Equisatisfiability guaranteed by fold/unfold rules.
- No infinite models and are needed to show satisfiabilty.
- Solved by Eldarica without induction rules.
  LIA-definable model (CLP syntax used by Eldarica):

```
list-sum-max(S,M) :- S>=M, M>=0.
```

# Insertion Sort & Permutation property

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

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

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 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 and s have the same elements (counting elements of l,s).

∀l,s,x,n1,n2. (`count` x l = n1) ∧ (`iSort` l = s) ∧ (`count` x s = n2) → n1=n2

# Translation into CHCs

## Insertion Sort

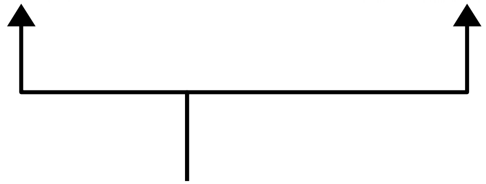
```
false :- N1=\=N2, count(X,L,N1), iSort(L,S), count(X,S,N2).
ins(A,[ ],[A]).
ins(A,[X|Xs],[A,X|Xs]) :- A=<X.
ins(A,[X|Xs],[X|Ys]) :- A>X, ins(A,Xs,Ys).
iSort([ ],[ ]).
iSort([X|Xs],S) :- iSort(Xs,S1), ins(X,S1,S).
count(X,[ ],0).
count(X,[H|T],N) :- X=H, N=M+1, count(X,T,M).
count(X,[H|T],N) :- X=\=H,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

**CHC solvers (Eldarica, Z3) over the quantifier-free theory of lists and Linear Integer Arithmetic (LIA) cannot solve these clauses.**

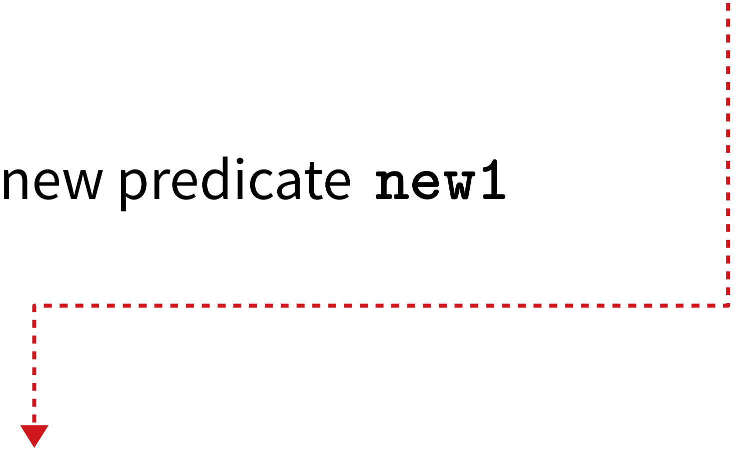
# Recursive Datatype Elimination Algorithm

## Insertion Sort

```
false :- N1=\=N2, count(X,L,N1), iSort(L,S), count(X,S,N2).
```



existential lists



**Define** a new predicate `new1`

**Do NOT occur in the head of the new definition**

```
new1(X,N1,N2) :- count(X,L,N1),iSort(L,S),count(X,S,N2).
```

# Insertion Sort: I’d like to remove those lists, but I can’t due to a DIFFERENCE in ...

Definition of `new1`

```
new1(X',N1',N2'):-     count(X',L',N1'),iSort(L',S'),count(X',S',N2').
```

variant atoms

**NOT** variant conjunction

Unfolding of `new1`

```
new1(X,0,0).
new1(X,N1,M):- N1=N+1,count(X,L,N),iSort(L,S),ins(X,S,T),count(X,T,M).
new1(X,N1,M):- X=\=Y,count(X,L,N1),iSort(L,S),ins(Y,S,T),count(X,T,M).
```

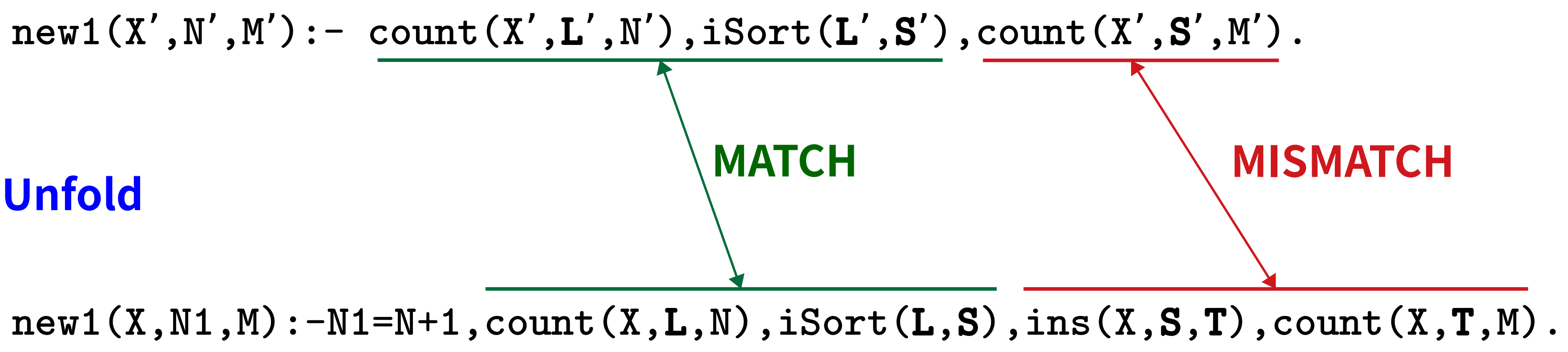
**Folding is not possible** (The Elimination Algorithm **does not terminate**)

**New idea of this work:**

**Match; Introduce “DIFFERENCE” predicates; Replace; Fold**

# Insertion Sort: Match

## Match clause to be folded against definition



**Define**

new1(X′,N′,M′):- count(X′,**L′**,N′),iSort(**L′**,**S′**),count(X′,**S′**,M′).

MATCH

MISMATCH

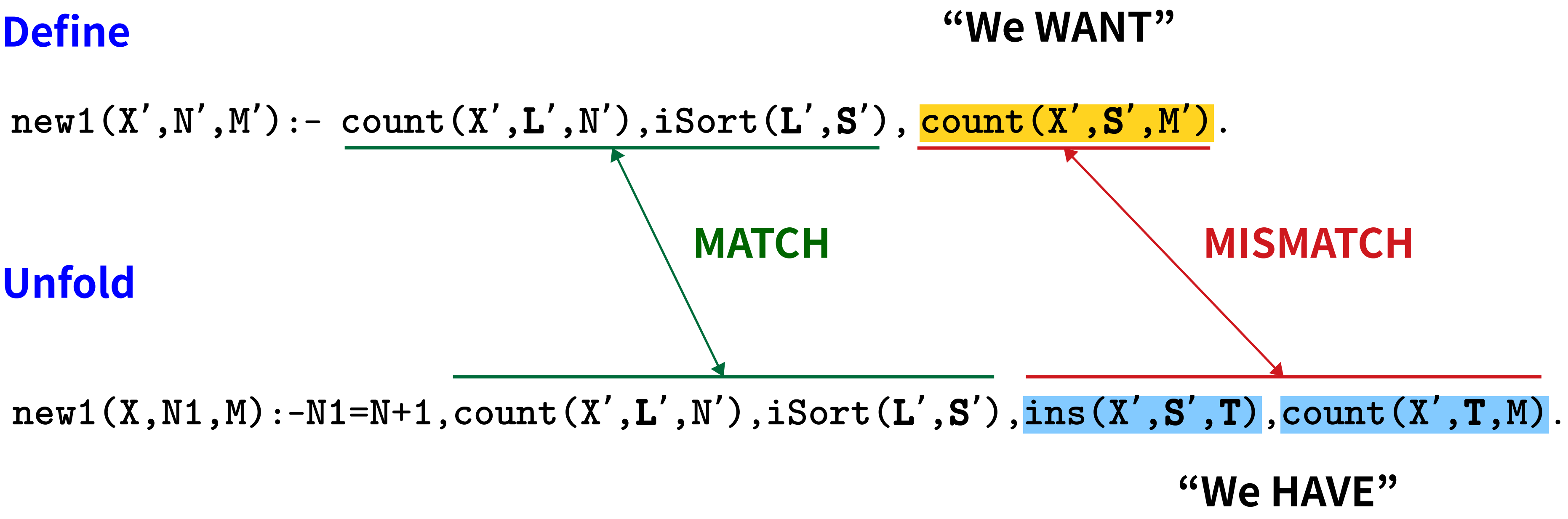
**Unfold**

new1(X,N1,M):-N1=N+1,count(X,**L**,N),iSort(**L**,**S**),ins(X,**S**,**T**),count(X,**T**,M).

**Substitution:** {L/L′, S/S′, X/X′, N/N′}

# Insertion Sort: Match

**Match clause to be folded against definition**



**Define**

**“We WANT”**

```
new1(X',N',M'):- count(X',L',N'),iSort(L',S'), count(X',S',M').
```

**MATCH**

**MISMATCH**

**Unfold**

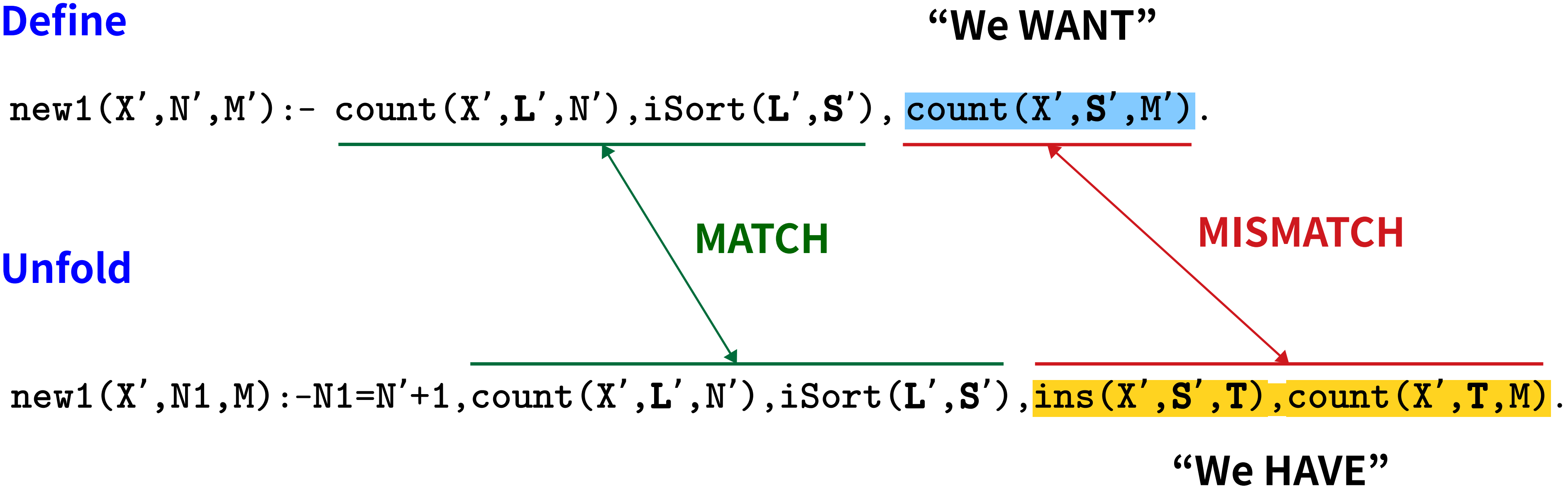
```
new1(X,N1,M):-N1=N+1,count(X',L',N'),iSort(L',S'),ins(X',S',T),count(X',T,M).
```

**“We HAVE”**

# Insertion Sort: DIFFERENCE predicate

## Introduce DIFFERENCE predicate

**Difference between clause to be folded (“We HAVE”) and the definition (“We WANT”)**



**Define**

**“We WANT”**

```
new1(X′,N′,M′):- count(X′,L′,N′),iSort(L′,S′), count(X′,S′,M′).
```

**Unfold**

```
new1(X′,N1,M):-N1=N′+1,count(X′,L′,N′),iSort(L′,S′),ins(X′,S′,T),count(X′,T,M).
```

**“We HAVE”**

**Define**

```
diff1(X′,M,M′):- ins(X′,S′,T), count(X′,T,M), count(X′,S′,M′).
```

# Insertion Sort: Replace

**Replace** ins(X′,S′,T), count(X′,T,M) (“We HAVE”)

**by** count(X′,S′,M′), diff1(X′,M′,M) (“We WANT”)

**Define**

“We WANT”

new1(X′,N′,M′):- count(X′,**L**′,N′),iSort(**L**′,**S**′),count(X′,**S**′,M′).

**Unfold**

new1(X′,N1,M):-N1=N′+1,count(X′,**L**′,N′),iSort(**L**′,**S**′),ins(X′,**S**′,**T**),count(X′,**T**,M).

“We HAVE”

**Replace**

new1(X′,M′,M):-N1=N+1,count(X′,**L**′,N′),iSort(**L**′,**S**′), count(X′,**S**′,M′),
diff1(X′,M′,M).

# Correctness of Replacement

**Replace** `ins(X',S',T), count(X',T,M)` **("We HAVE")**

**by** `count(X',S',M'), diff1(X',M',M)` **("We WANT")**

$$\begin{tikzcd}
(X',S') \arrow[r, "id_1 \times \mathtt{ins}"] \arrow[d, "id_1 \times \mathtt{count}"'] & (X',T) \arrow[r, "\mathtt{count}"] & M \\
(X',M') \arrow[urr, "\mathtt{diff1}"'] & &
\end{tikzcd}$$

**Suppose Cls U {C} → Cls U {D} by replacement.**

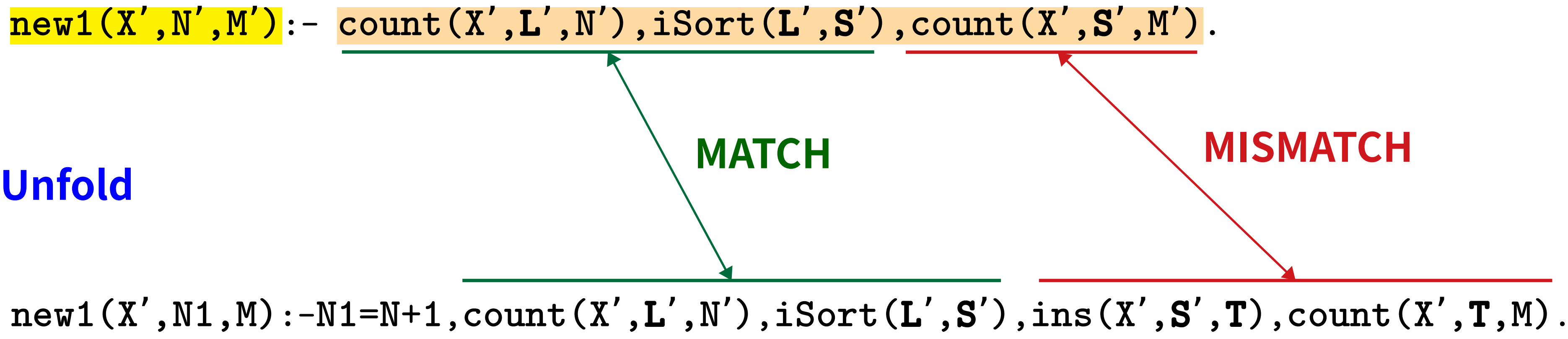
**If `diff1` is a function then Cls U {C} is SAT IFF Cls U {D} is SAT.**

**Otherwise, Cls U {C} is SAT IF Cls U {D} is SAT.**

# Insertion Sort: Fold

## Define

“We WANT”



```
new1(X′,N′,M′):- count(X′,L′,N′),iSort(L′,S′),count(X′,S′,M′).
```

MATCH

MISMATCH

## Unfold

```
new1(X′,N1,M):-N1=N+1,count(X′,L′,N′),iSort(L′,S′),ins(X′,S′,T),count(X′,T,M).
```

“We HAVE”

## Replace

```
new1(X′,M′,M):-N1=N+1,count(X′,L′,N′),iSort(L′,S′), count(X′,S′,M′),
                                                     diff1(X′,M′,M).
```

## Fold

```
new1(X′,M′,M):-N1=N+1, new1(X′,N′,M′), diff1(X′,M′,M).
```

# Insertion Sort: final set of CHCs w/o lists

```
false :- N1=\=N2, new1(X,N1,N2).
new1(X,0,0).
new1(X,N1,N2) :- N1=N1'+1, new1(X',N1',N2'), diff1(X',N2',N2).
new1(X,N1,N2) :- X=\=Y, new1(Y,N1,N2b), diff2(X,Y,N2b,N2).
diff1(X,0,1).
diff1(X,N1,N2) :- N2=M2+1, N1=M1+1, new3(X,M2,M1).
diff1(X,N1,N2) :- X=<Y, N2=N+1, X=\=Y, new4(X,Y,N,N1).
diff2(X,Y,0,0) :- Y=\=X.
diff2(X,Y,M,N) :- X=<Y, Y=\=X, M=K+1, new3(Y,N,K).
diff2(X,Y,M,N) :- X=<Z, Y=\=X, Y=\=Z, N=M, new5(Y,N).
diff2(X,Y,M,N) :- X>Y, N=H+1, M=K+1, diff2(X,Y,K,H).
new3(X,N1,N)  :- N1=N+1, new5(X,N).
new4(X,Y,N,N) :- X=<Y, X=\=Y, new5(X,N).
new5(X,0).
new5(X,N1) :- N1=N+1, new5(X,N).
```

**`diff2` is a difference predicate:**

```
diff2(X,Y,N,M):-X=\=Y,ins(X,S1,S),count(Y,S,M),count(Y,S1,N).
```

# Insertion Sort: Computation of Model

**Eldarica proves satisfiability by computing a LIA-definable model (rewritten for legibility):**

```
false :- N1=\=N2, N1=N2, N2>=0.

new1(A,B,C) :- B=C,  B>=0.
new2(A,B) :- B = 0.

diff1(A,B,C) :- B>=0, C=B+1.          % diff1 is a function

diff2(A,B,C,D) :- C>=0, D=C.          % diff2 is a function

new3(A,B,C) :- C=B-1, B>=1.
new4(A,B,C,D) :- D=C, C>=0, B>=A+1.
new5(A,B) :- B>=0.
```

**Difference predicates are functions.**

**Thus, the initial and transformed clauses are equisatisfiable.**

# Difference Predicates and Lemma Discovery

**Eldarica model of difference predicates**

```
diff1(X’,N2’,N2) :- N2=N2’+1, N2’>=0.
diff2(X,Y,N2’,N2) :- N2=N2’, N2’>=0.
```

**Difference predicates correspond to lemmata in a proof by structural induction**

```
diff1(X’,N2’,N2):- ins(X’,S’,S),count(X’,S,N2),count(X’,S’,N2’).
diff2(X,Y,N2’,N2):- X=\=Y,ins(X,S1,S),count(Y,S,N2),count(Y,S1,N2’).
```

**can be rewritten as**

∀((count X’ (ins X’ S’) = N2) ∧ (count X’ S’ = N2’) → N2=N2’+1 ∧ N2’>=0)

∀(X=\=Y ∧ (count Y (ins X S1) = N2) ∧ (count Y S1 = N2’) → N2=N2’ ∧ N2’>=0)

# More Sorting Programs and Properties

- **Transformations done using the MAP interactive system**
- **Satisfiability proof an model computation done by Eldarica**

| | Permutation | Ordered | Length | Sum |
|---|---|---|---|---|
| **InsertionSort** | ✓ | ✓ | ✓ | ✓ |
| **SelectionSort** | ✓ | ✓ | ✓ | |
| **QuickSort** | ✓ | ✗ | | ✓ |
| **MergeSort** | | | | ✓ |

✓ **Transformation & Model Computation succeeded**

✗ **We stopped the transformation**

**Blank** **We did not try**

# Conclusions

- CHC transformations aid verification of programs that manipulate recursive datatypes
- In the sorting examples, the Elimination Algorithm + Difference Predicate Intro transforms non-solvable (by CHC solvers) CHCs into equisatisfiable solvable CHCs
- CHC solving < (Transformation; CHC solving) ~ (Induction + CHC solving)
- Advantage of the transformation-based approach: separation of inductive reasoning (by transformation) from CHC solving
- Ongoing work:
  - Automation (some work done)
  - Benchmarking: compare with Inductive Theorem Provers (e.g., ACL2, Clam, Leon, Isabelle)