
A Coinduction Rule for Entailment of Recursively-Defined Properties

Joxan Jaffar, Andrew E. Santosa, Răzvan Voicu
{joxan, andrews, razvan}@comp.nus.edu.sg

School of Computing
National University of Singapore

Problem and Approach

We want to prove *recursive assertions* $G \models H$, e.g., $p(X) \models q(X)$, where p and q are *recursive predicates*

Typical computer-assisted inductive proof:

- Requires user-specified *schema*
- Requires well-foundedness
e.g., *structural induction, computational induction*

We present search-based algorithmic inductive proof:

- Not requiring schemas
- Not requiring well-foundedness argument
- Based on *coinduction: search* for *finite counterexample* using *circular argument*

Problem and Approach

We want to prove *recursive assertions* $G \models H$, e.g., $p(X) \models q(X)$, where p and q are *recursive predicates*

Typical computer-assisted inductive proof:

- Requires user-specified *schema*
- Requires well-foundedness
e.g., *structural induction, computational induction*

We present search-based algorithmic inductive proof:

- Not requiring schemas
- Not requiring well-foundedness argument
- Based on *coinduction: search* for *finite counterexample* using *circular argument* → Automatable!

Outline

1. CLP and Recursive Assertions
2. Proof example
3. Proof rules and algorithm
4. Data structure verification
5. Related Work and Conclusion

Outline

1. CLP and Recursive Assertions
2. Proof example
3. Proof rules and algorithm
4. Data structure verification
5. Related Work and Conclusion

Constraint Logic Programs

A conjunction of implications (*Horn clauses*)

$$\text{even}(X) \text{ :- } X = 0.$$
$$\text{even}(X) \text{ :- } \text{even}(Y), X = Y + 2.$$

A clause's antecedent is called *body*,
its consequent is called *head*.

Head contains *single atom*.

Body contains *atoms* and *constraints*.

Executable via *resolution*.

Least model = set of true ground atoms.

E.g., $\{\text{even}(0), \text{even}(2), \text{even}(4), \dots\}$.

Recursive Assertions

CLP Program:

$even(0).$

$even(X + 2) \text{ :- } even(X).$

$m4(0).$

$m4(X + 4) \text{ :- } m4(X).$

We may want to prove:

$$m4(X) \models even(X)$$

Typically requires manual induction

Assertions about Program Heap

For data structure verification

h: **Heap** **Linked list** h[p] =

| |
|-------------|
| <i>data</i> |
| <i>next</i> |

array **node:** h[p+1] =

```
{h = h0, p = p0 > 0}
while (p > 0) do
  h[p] := 0
  p := h[p+1]
end
{∃y. allz(h0, h, p0, y), h[y+1] = 0}
```

***allz*(H₁, H₂, X, Y)** : The heap H₁ and H₂ are the same, but for the linked list segment from cell pointed to by X to the cell pointed to by Y, all data values in H₂ are set to 0.

Assertions about Program Heap

$allz(H_1, H_2, X, Y)$: The heap H_1 and H_2 are the same, but for the linked list segment from cell pointed to by X to the cell pointed to by Y , all data values in H_2 are set to 0.

Assertion Predicate:

$allz(H, \langle H, L, 0 \rangle, L, L) :- L > 0.$

$allz(H_1, \langle H_2, L, 0 \rangle, L, R) :-$
 $L > 0, allz(H_1, H_2, H_1[L + 1], R).$

$\langle H, X, Y \rangle$ heap H' where $H'[X] = Y$ and
 $H'[Z] = H[Z]$ for all $Z \neq X$

Assertions about Program Heap

$allz(H_1, H_2, X, Y)$: The heap H_1 and H_2 are the same, but for the linked list segment from cell pointed to by X to the cell pointed to by Y , all data values in H_2 are set to 0.

Assertion Predicate:

$allz(H, \langle H, L, 0 \rangle, L, L) \quad :- \quad L > 0.$

$allz(H_1, \langle H_2, L, 0 \rangle, L, R) \quad :-$
 $L > 0, allz(H_1, H_2, H_1[L + 1], R).$

$\langle H, X, Y \rangle$ heap H' where $H'[X] = Y$ and
 $H'[Z] = H[Z]$ for all $Z \neq X$

“ $allz(h_0, h, p_0, p)$ is a loop invariant”

$allz(H_0, H, P_0, P), H[P + 1] > 0 \models$
 $allz(H_0, \langle H, H[P + 1], 0 \rangle, P_0, H[P + 1])$

Outline

1. CLP and Recursive Assertions
- 2. Proof example**
3. Proof rules and algorithm
4. Data structure verification
5. Related Work and Conclusion

Proof Example

$$A: m4(X) \models \text{even}(X)$$

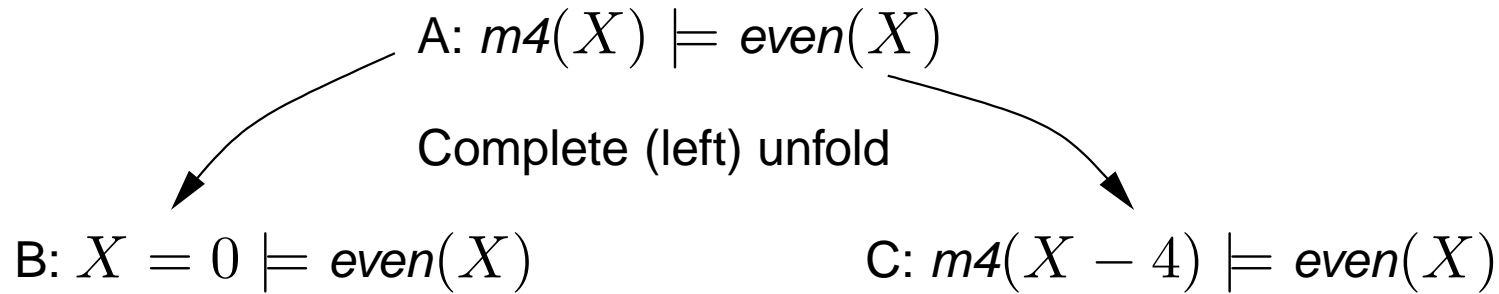
$m4(0).$

$m4(X + 4) :- m4(X).$

$\text{even}(0).$

$\text{even}(X + 2) :- \text{even}(X).$

Proof Example



$m4(0).$

$m4(X + 4) :- m4(X).$

$even(0).$

$even(X + 2) :- even(X).$

Proof Example

$$A: m4(X) \models even(X)$$

Complete (left) unfold

$$B: X = 0 \models even(X)$$

$$C: m4(X - 4) \models even(X)$$

Right unfold

$$D: X = 0 \models X = 0$$

$$m4(0).$$

$$m4(X + 4) :- m4(X).$$

$$even(0).$$

$$even(X + 2) :- even(X).$$

Proof Example

A: $m4(X) \models even(X)$

Complete (left) unfold

B: $X = 0 \models even(X)$

C: $m4(X - 4) \models even(X)$

Right unfold

D: $X = 0 \models X = 0$

Constraint proof

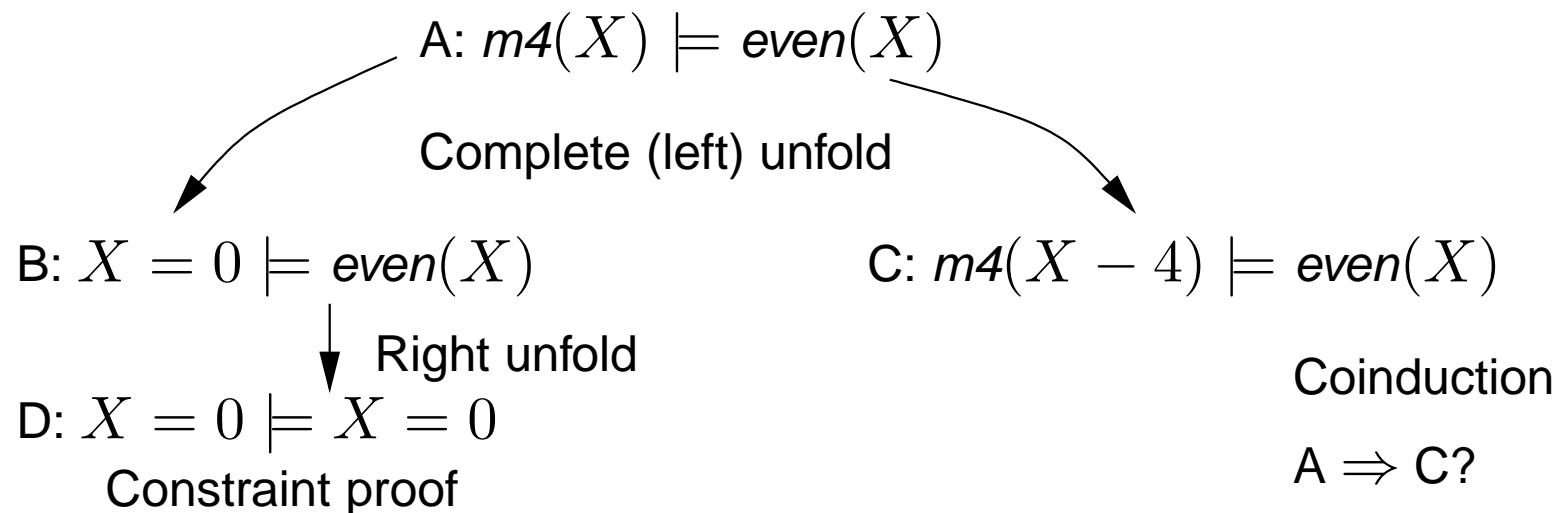
$m4(0).$

$m4(X + 4) :- m4(X).$

$even(0).$

$even(X + 2) :- even(X).$

Proof Example



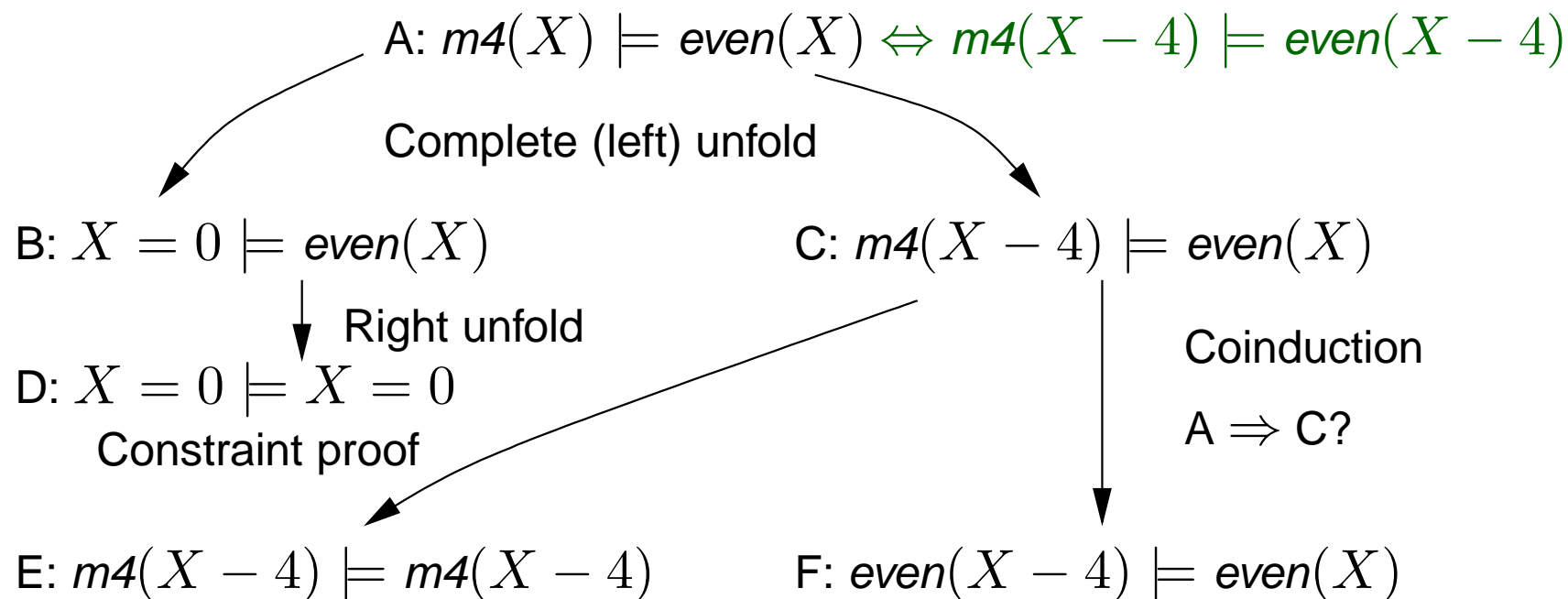
$m4(0).$

$m4(X + 4) :- m4(X).$

$even(0).$

$even(X + 2) :- even(X).$

Proof Example



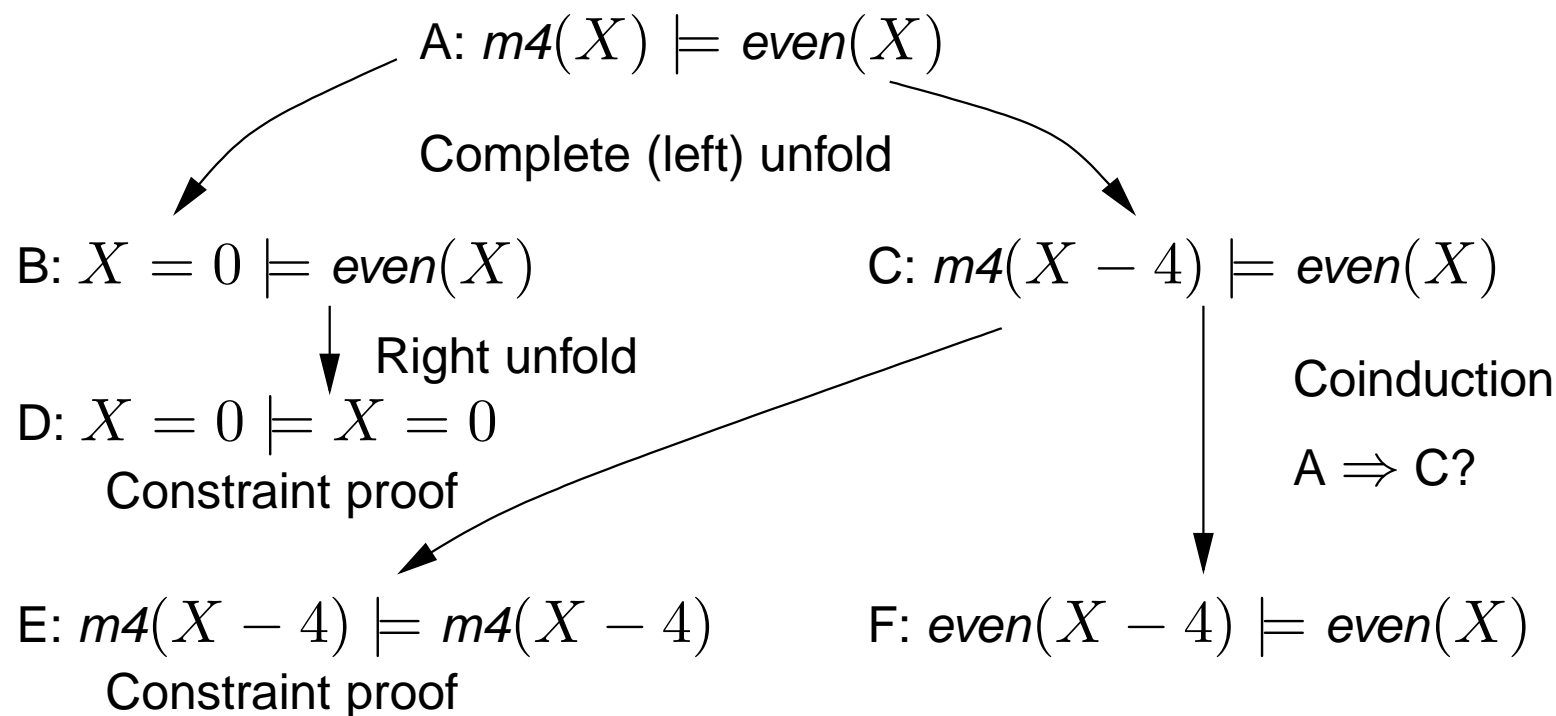
$m4(0).$

$m4(X + 4) :- m4(X).$

$\text{even}(0).$

$\text{even}(X + 2) :- \text{even}(X).$

Proof Example



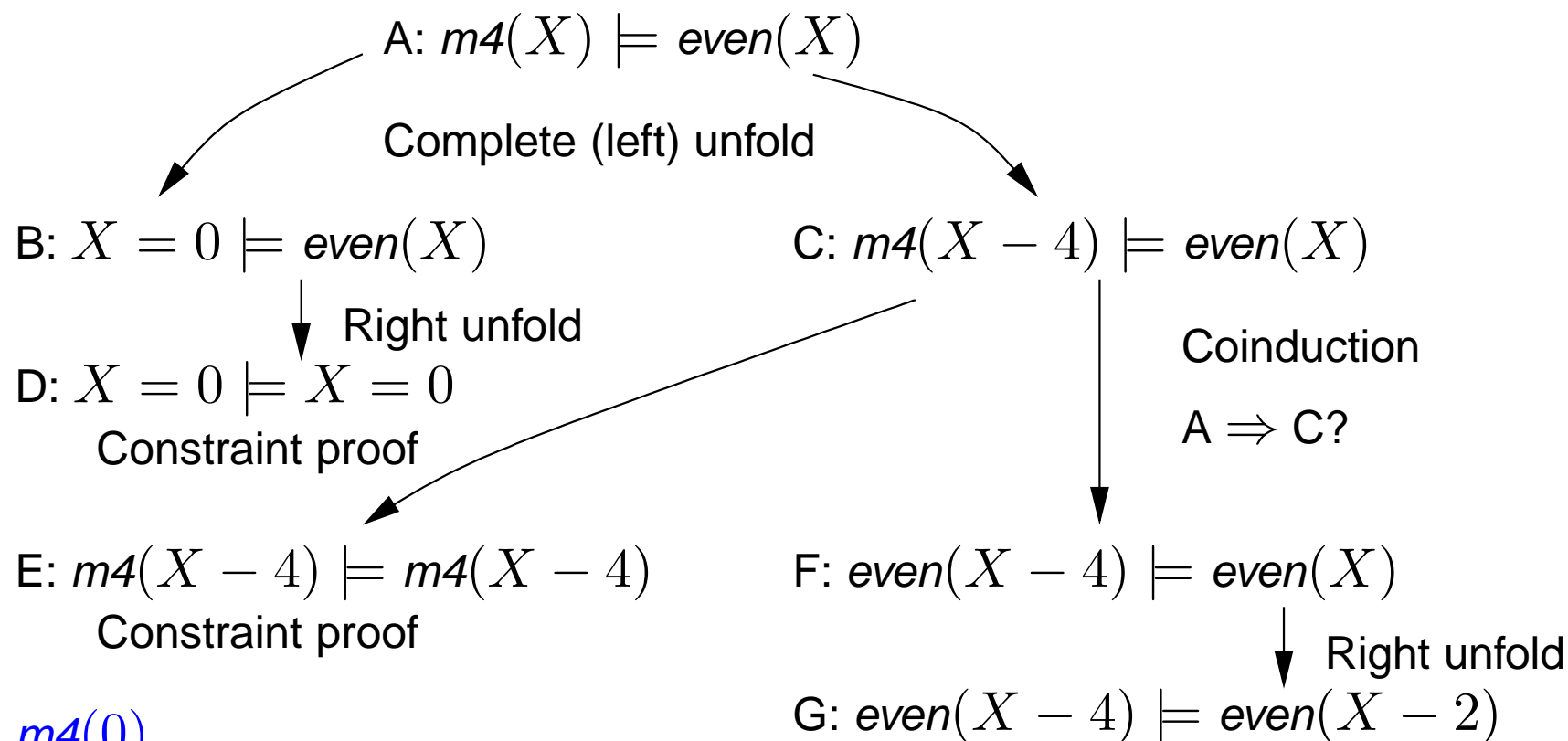
$m4(0).$

$m4(X + 4) :- m4(X).$

$even(0).$

$even(X + 2) :- even(X).$

Proof Example



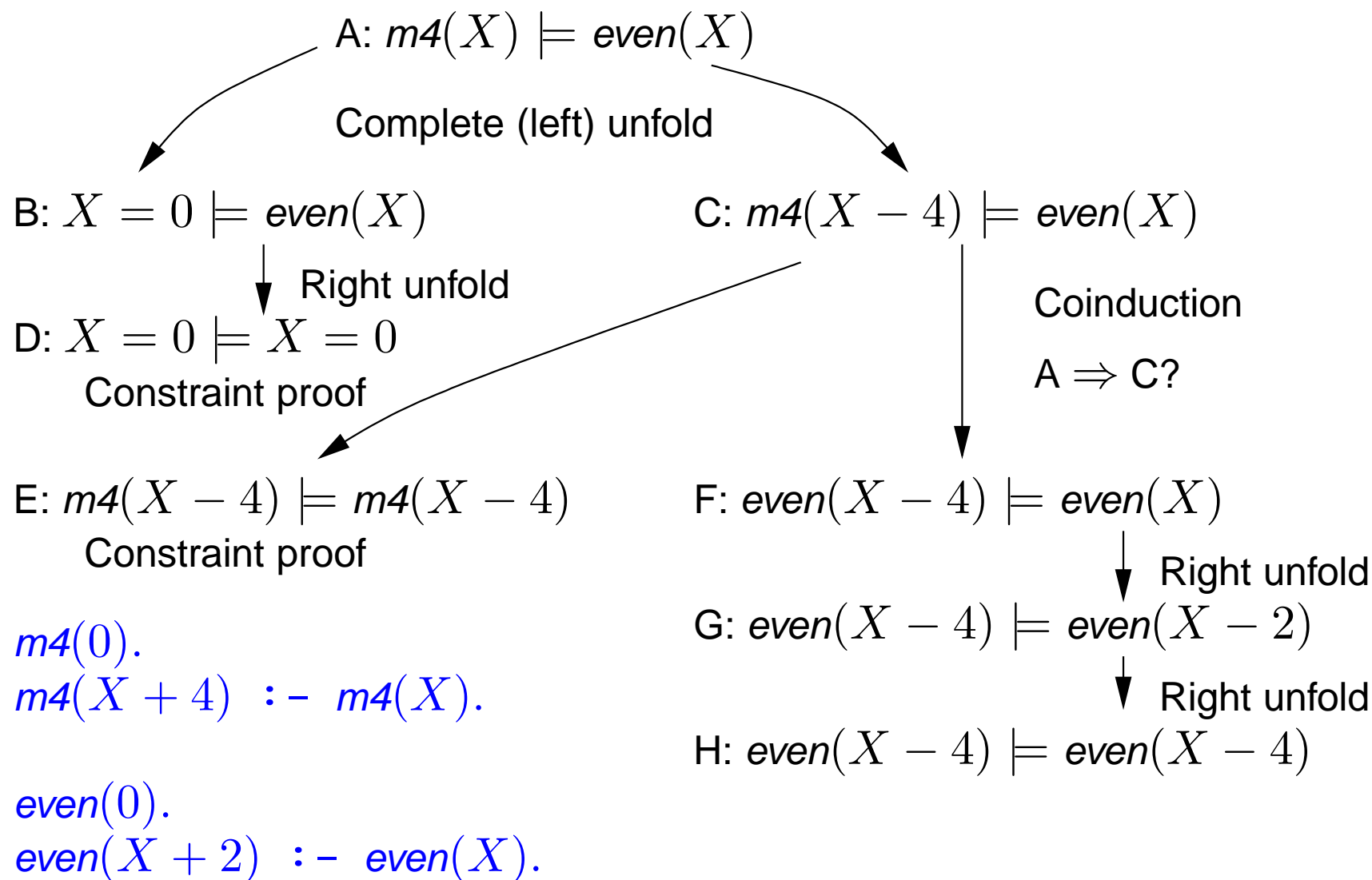
$m4(0).$

$m4(X + 4) :- m4(X).$

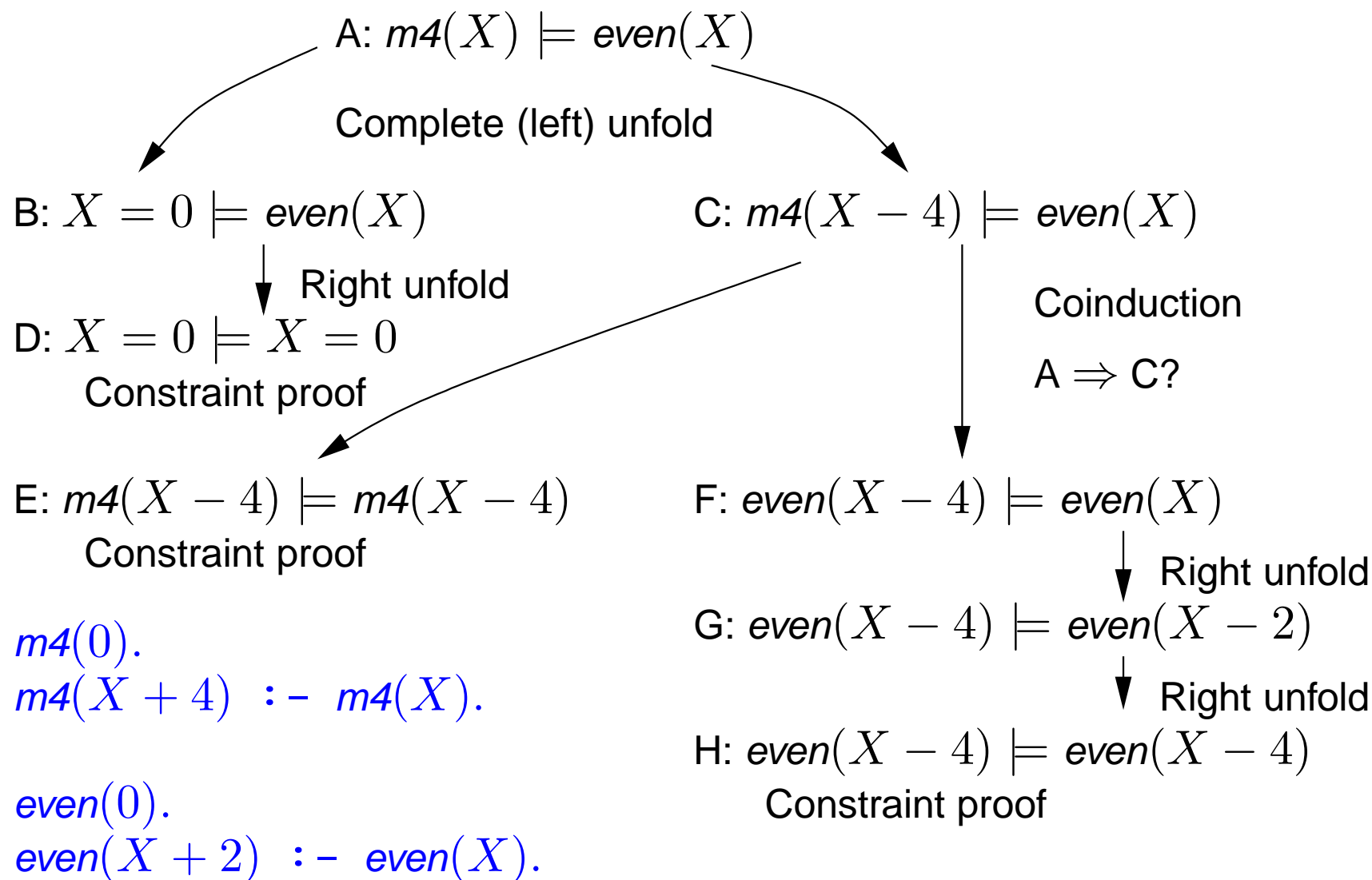
$even(0).$

$even(X + 2) :- even(X).$

Proof Example



Proof Example



Outline

1. CLP and Recursive Assertions
2. Proof example
- 3. Proof rules and algorithm**
4. Data structure verification
5. Related Work and Conclusion

Proof Rules

$$\begin{array}{l} \text{(LU+I)} \quad \frac{\Pi \uplus \{\tilde{A} \vdash G \models H\}}{\Pi \cup \bigcup_{i=1}^n \{\tilde{A} \cup \{G \models H\} \vdash G_i \models H\}} \quad \text{unfold}(G) = \{G_1, \dots, G_n\} \\ \\ \text{(RU)} \quad \frac{\Pi \uplus \{\tilde{A} \vdash G \models H\}}{\Pi \cup \{\tilde{A} \vdash G \models H'\}} \quad H' \in \text{unfold}(H) \\ \\ \text{(CO)} \quad \frac{\Pi \uplus \{\tilde{A} \vdash G \models H\}}{\Pi \cup \{\emptyset \vdash H'\theta \models H\}} \quad G' \models H' \in \tilde{A} \text{ and there exists a substitution } \theta \text{ s.t. } G \models G'\theta. \\ \\ \text{(CP)} \quad \frac{\Pi \uplus \{\tilde{A} \vdash G \wedge p(\tilde{x}) \models H \wedge p(\tilde{y})\}}{\Pi \cup \{\tilde{A} \vdash G \models H \wedge \tilde{x} = \tilde{y}\}} \\ \\ \text{(DP)} \quad \frac{\Pi \uplus \{\tilde{A} \vdash G \models H\}}{\Pi} \quad G \models H \text{ holds by constraint solving} \end{array}$$

Theorem

A proof obligation $G \models H$ holds, that is, least model of $P \rightarrow (G \models H)$ for the given program P , if, starting with the proof obligation $\emptyset \vdash G \models H$, there exists a sequence of applications of proof rules that results in proof obligations $\tilde{A} \vdash G' \models H'$ such that (a) H' contains only constraints, and (b) $G' \models H'$ can be discharged by the constraint solver.

On Coinduction

- No “base case”

On Coinduction

- No “base case”
- “Dynamic” hypothesis → not requiring schema

On Coinduction

- No “base case”
- “Dynamic” hypothesis → not requiring schema

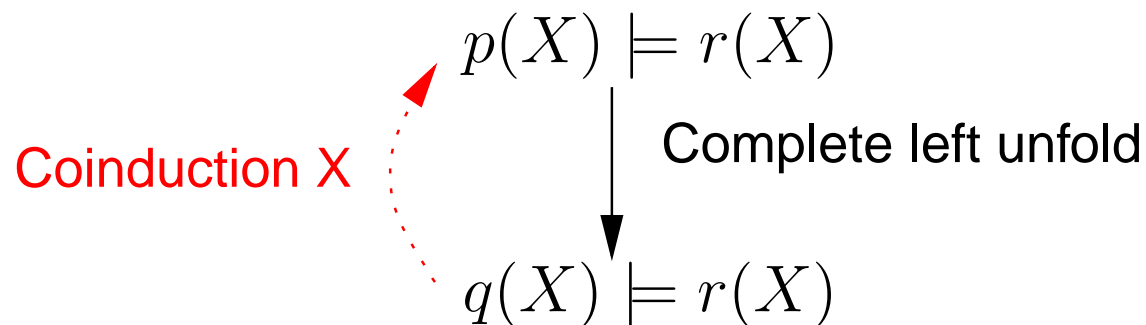
$$p(X) \models r(X)$$

Program:

$p(X) \quad :- \quad q(X).$
 $q(X) \quad :- \quad q(X).$
 $r(X).$

On Coinduction

- No “base case”
- “Dynamic” hypothesis → not requiring schema



Program:

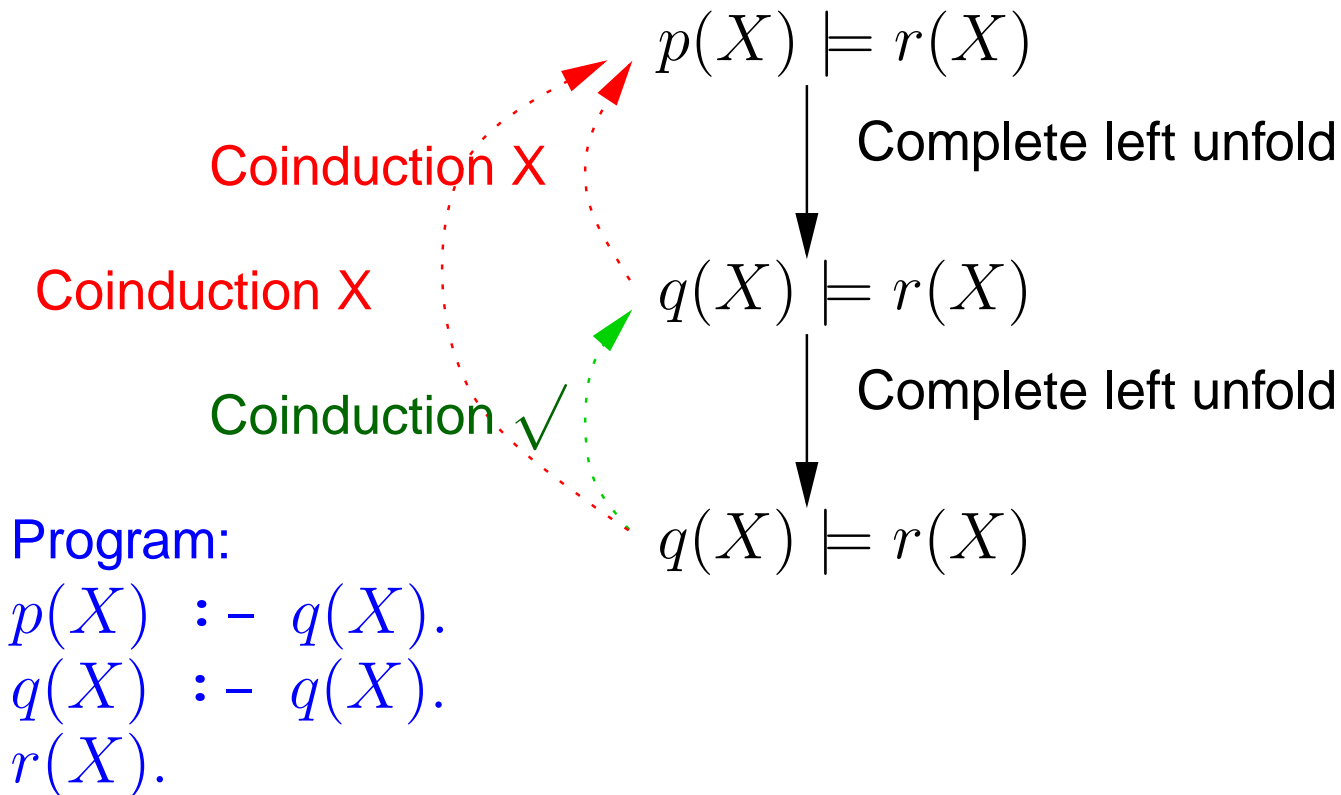
$p(X) :- q(X).$

$q(X) :- q(X).$

$r(X).$

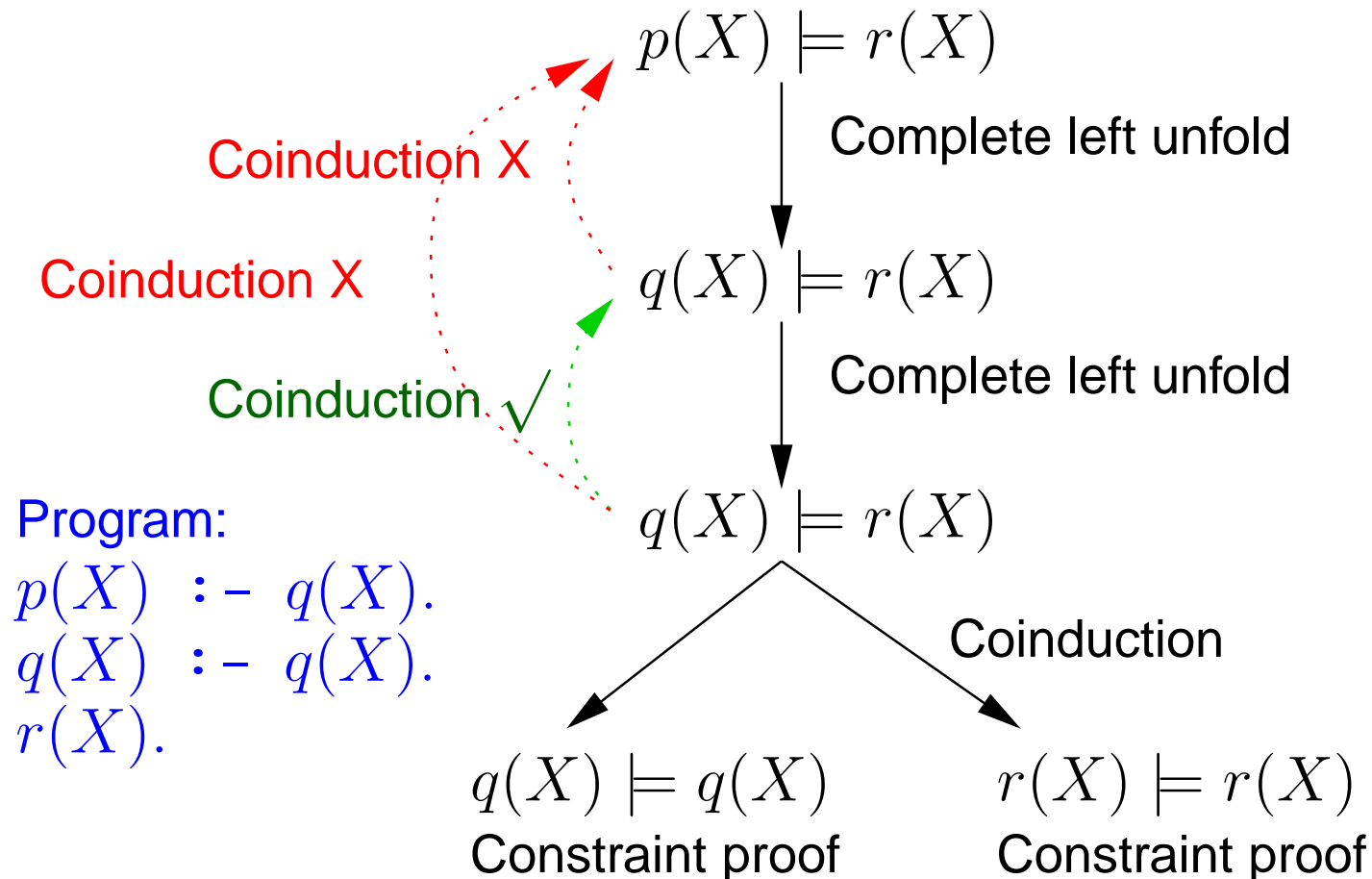
On Coinduction

- No “base case”
- “Dynamic” hypothesis \rightarrow not requiring schema



On Coinduction

- No “base case”
- “Dynamic” hypothesis \rightarrow not requiring schema



The Algorithm

REDUCE($G \models H$) returns boolean

- Constraint Proof: (CP) + Constraint Solving (DP)
Apply a constraint proof to $G \models H$
If successful, return *true* otherwise return *false*
- Coinduction: (CO)
if there is an assumption $G' \models H'$ such that
REDUCE($G \models G'\theta$) = *true* and REDUCE($H'\theta \models H$) = *true*
then return *true* otherwise return *false*
- Unfold: *(next page ...)*

The Algorithm

- **Unfold: choose left or right**
 - case: Left: (LU+I)**
 - memoize** $(G \models H)$ as an assumption
 - choose** an atom A in G to reduce
 - for all** reducts G_L of G using A :
 - if** $\text{REDUCE}(G_L \models H) = \text{false}$ **return** *false*
 - return** *true*
 - case: Right: (RU)**
 - choose** an atom A in H to reduce, obtaining G_R
 - return** $\text{REDUCE}(G \models G_R)$

Outline

1. CLP and Recursive Assertions
2. Proof example
3. Proof rules and algorithm
- 4. Data structure verification**
5. Related Work and Conclusion

Data Structure Verification

$$\text{allz}(H_0, H, P_0, P) \models \text{allz}(H_0, \langle H, H[P + 1], 0 \rangle, P_0, H[P + 1])$$

Constraint proof X

Coinduction X

(Left) complete unfold ✓

... \models ...

$$\begin{aligned} &\text{allz}(H_0, H_1, H_0[P_0 + 1], P), P_0 > 0, H_1[P + 1] > 0 \models \\ &\text{allz}(H_0, \langle \langle H_1, P_0, 0 \rangle, H_1[P + 1], 0 \rangle, P_0, H_1[P + 1]) \end{aligned}$$

Data Structure Verification

$$\text{allz}(H_0, H, P_0, P) \models \text{allz}(H_0, \langle H, H[P + 1], 0 \rangle, P_0, H[P + 1])$$

Constraint proof X

Coinduction X

(Left) complete unfold ✓

$$\dots \models \dots$$

$$\begin{aligned} &\text{allz}(H_0, H_1, H_0[P_0 + 1], P), P_0 > 0, H_1[P + 1] > 0 \models \\ &\text{allz}(H_0, \langle \langle H_1, P_0, 0 \rangle, H_1[P + 1], 0 \rangle, P_0, H_1[P + 1]) \end{aligned}$$

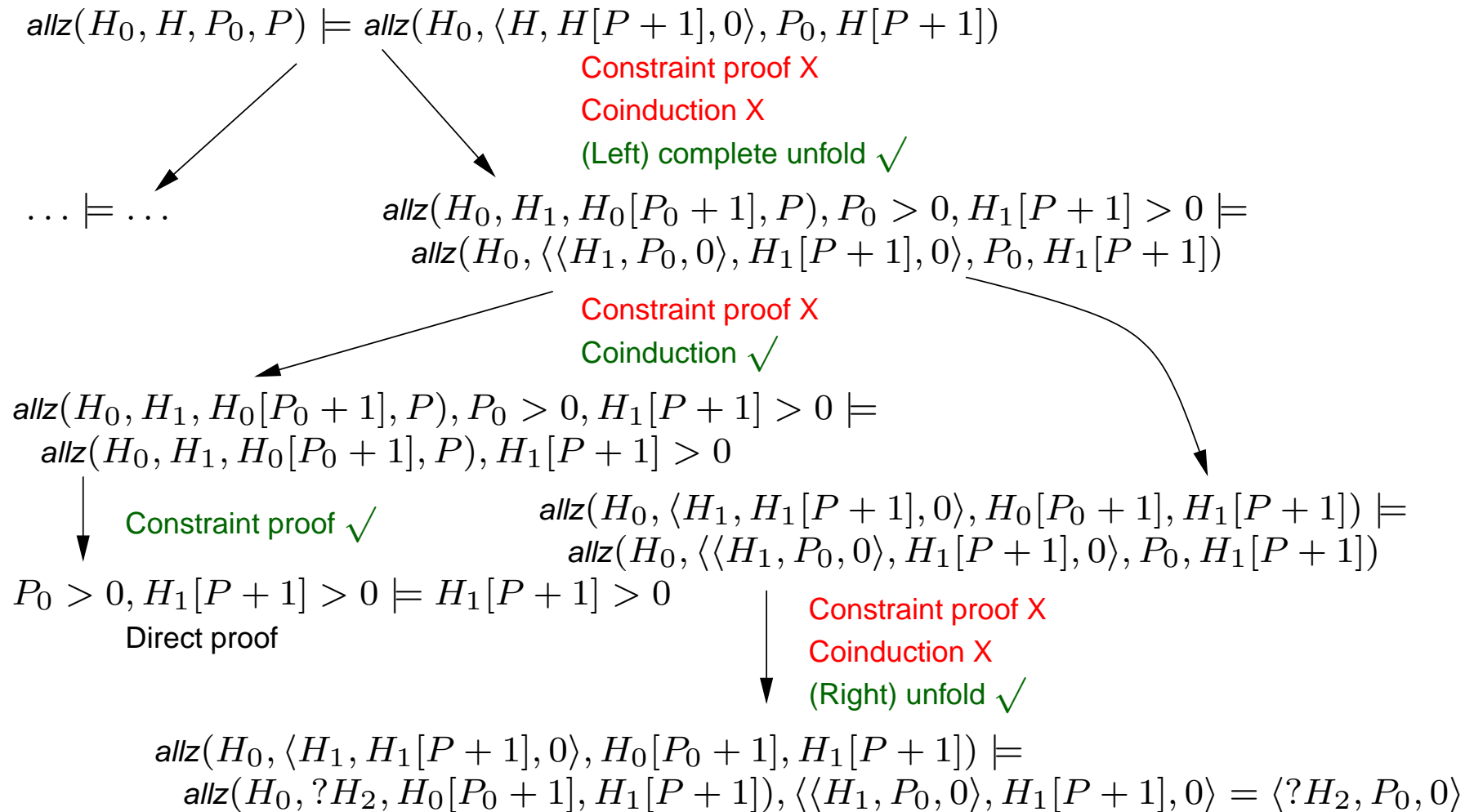
Constraint proof X

Coinduction ✓

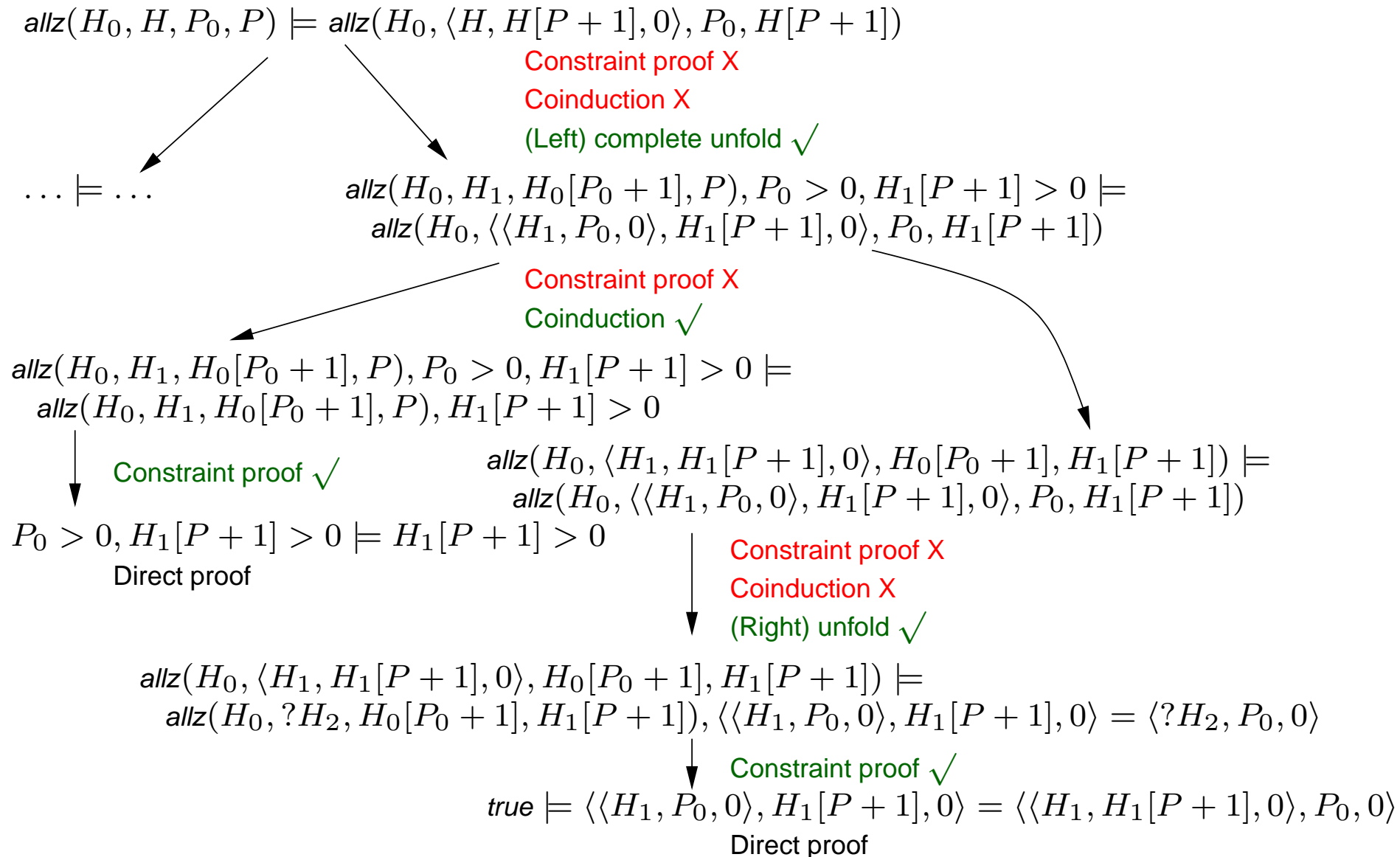
$$\begin{aligned} &\text{allz}(H_0, H_1, H_0[P_0 + 1], P), P_0 > 0, H_1[P + 1] > 0 \models \\ &\text{allz}(H_0, H_1, H_0[P_0 + 1], P), H_1[P + 1] > 0 \end{aligned}$$

$$\begin{aligned} &\text{allz}(H_0, \langle H_1, H_1[P + 1], 0 \rangle, H_0[P_0 + 1], H_1[P + 1]) \models \\ &\text{allz}(H_0, \langle \langle H_1, P_0, 0 \rangle, H_1[P + 1], 0 \rangle, P_0, H_1[P + 1]) \end{aligned}$$

Data Structure Verification



Data Structure Verification



Outline

1. CLP and Recursive Assertions
2. Proof example
3. Proof rules and algorithm
4. Data structure verification
5. Related Work and Conclusion

Related Work

- Tabled Logic Programming (XSB) (Sagonas et al. 2003): Uses SLG resolution: *Returns answers* in the least model of the logic program.
- Coinductive Logic Programming (Co-LP) (Gupta et al. 2007): *Returns answers* in the greatest fixpoint of the logic program.

Our notion of “coinduction”: *No answers*

- We compute a set S of (possibly) circularly dependent premises of $G \models H$: gfp of a function producing premises of $G \models H$.
- But, we reason about the least model of the program: the least model implies S , hence $G \models H$

Related Work

- Specialized version appeared in RTSS '04, VMCAI '06
- Boyer-Moore prover (1975), inspiring many LP-based proof systems
- Inductive proofs of (constraint) logic programs: Kanamori & Fujita (1986), Hsiang & Srivas (1985, 1987), Mesnard et al. (1996), Craciunescu (2002), etc.: typically uses schema
- Unfold/fold systems: Pettorossi & Proietti (1999), Roychoudhury et al. (2004), Nguyen et al. (2007)
- PTP: Stickel (1992): uses memoing
- Checking for already-visited states in model checking

Conclusion

- We presented search-based algorithmic inductive proof:
 - Not requiring schemas
 - Not requiring well-foundedness argument
 - Automatable, based on *search* to find *finite counterexample* of $G \models H$ using *circular argument*

- Future work:
 - Implementation
 - Automatic verification of programs with pointer data structures