

Recursive Abstractions for Parameterized Systems

Joxan Jaffar and Andrew Santosa

{joxan, andrews}@comp.nus.edu.sg.

School of Computing
National University of Singapore

Problem and Approach

- Parameterized system: concurrent program with unspecified N identical processes
- Reasoning using properties with process ids quantified from 1 to N

Problem and Approach

- Parameterized system: concurrent program with unspecified N identical processes
- Reasoning using properties with process ids quantified from 1 to N
- *Problem: Automation*

Problem and Approach

- Parameterized system: concurrent program with unspecified N identical processes
- Reasoning using properties with process ids quantified from 1 to N
- *Problem: Automation*
- Our solution:
 - We model such properties using arrays and recursive predicates on arrays
 - We present an algorithm to prove assertions of recursive predicates on arrays

Key: Automatic Induction via Search

Counting Ones

Concurrent program of N processes:

$\langle 0 \rangle \quad \mathbf{x}=\mathbf{x}+1; \quad \langle 1 \rangle \quad || \quad \dots \quad || \quad \langle 0 \rangle \quad \mathbf{x}=\mathbf{x}+1; \quad \langle 1 \rangle$

$\mathcal{G}_0 : ((0, 0, \dots, 0), 0)$



$\mathcal{G}_1 : ((0, \dots, 0, 1, 0, \dots, 0), 1)$



$\mathcal{G}_2 : ((0, \dots, 0, 1, 0, \dots, 0, 1, 0, \dots, 0), 2)$

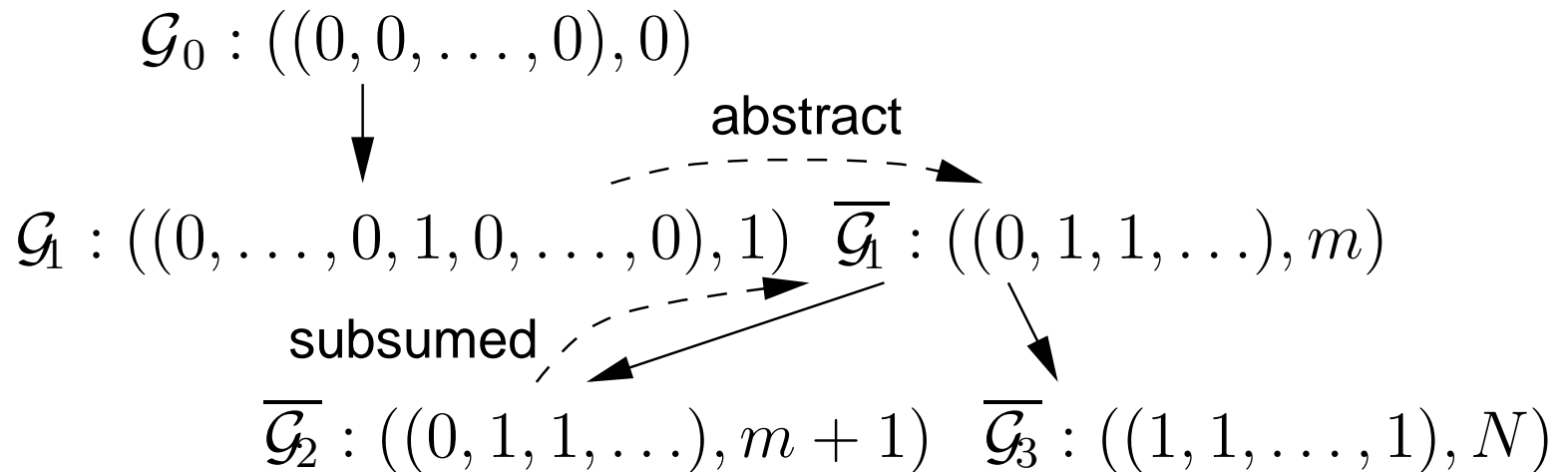


Program State: (array of program points, value of \mathbf{x})

Counting Ones

Concurrent program of N processes:

$\langle 0 \rangle \quad \mathbf{x}=\mathbf{x}+1; \quad \langle 1 \rangle \quad || \quad \dots \quad || \quad \langle 0 \rangle \quad \mathbf{x}=\mathbf{x}+1; \quad \langle 1 \rangle$

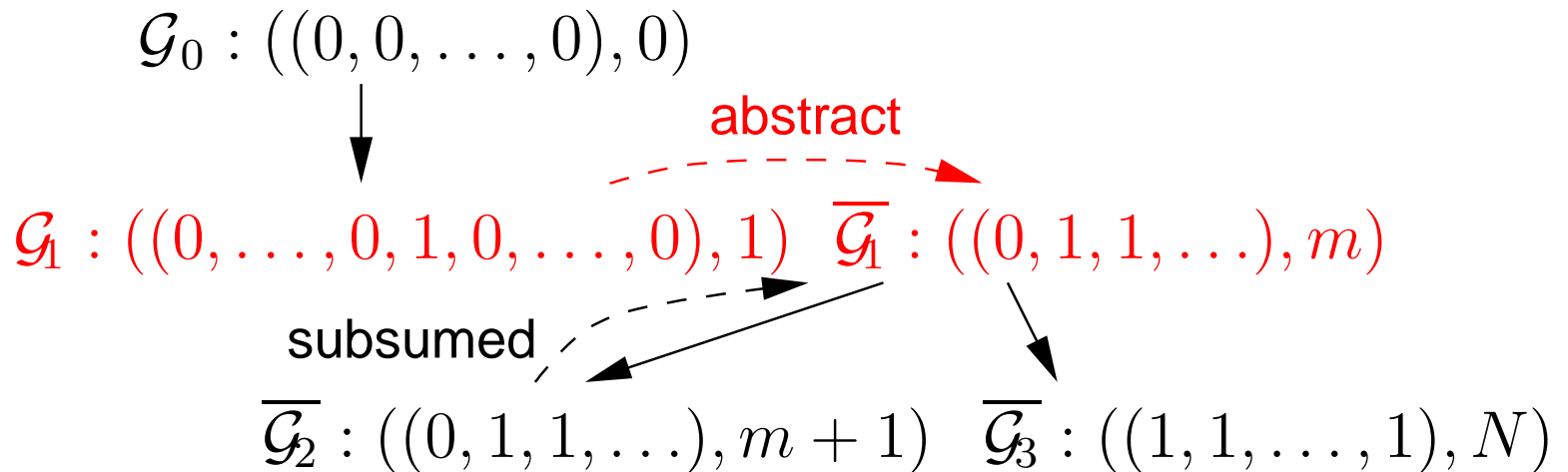


Program State: (array of program points, value of \mathbf{x})

Counting Ones

Concurrent program of N processes:

$\langle 0 \rangle \quad \mathbf{x}=\mathbf{x}+1; \quad \langle 1 \rangle \quad || \quad \dots \quad || \quad \langle 0 \rangle \quad \mathbf{x}=\mathbf{x}+1; \quad \langle 1 \rangle$



Program State: (array of program points, value of \mathbf{x})

Outline

1. Representing properties
2. Proving properties

Outline

1. Representing properties

2. Proving properties

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*.

Least fixpoint semantics:

$$\{\text{even}(0), \text{even}(2), \dots\}$$

Arrays

- A map from integer to integer $A[I] = X$
- Special expression $\langle A, I, X \rangle$:

Semantics based on McCarthy's Axioms:

- $\langle A, I, X \rangle[J] = A[J]$ if $I \neq J$
- $\langle A, I, X \rangle[J] = X$ if $I = J$

Recursive Predicates with Arrays

allzeroes(0, *K*, 0).

allzeroes(*Id*, $\langle K, Id, 0 \rangle$, 0) :- *Id* > 0, *allzeroes*(*Id* - 1, *K*, 0).

allones(0, *K*, 0).

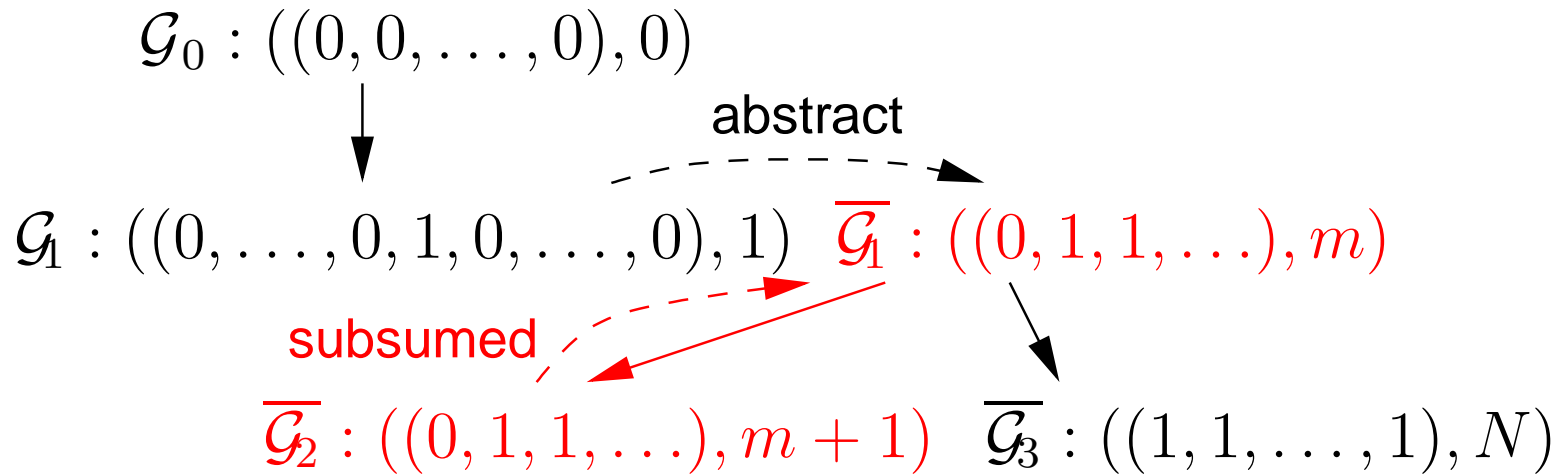
allones(*Id*, $\langle K, Id, 1 \rangle$, *X* + 1) :- *Id* > 0, *allones*(*Id* - 1, *K*, *X*).

bit(0). *bit*(1).

abs(0, *K*, 0).

abs(*Id*, $\langle K, Id, B \rangle$, *X* + *B*) :-
Id > 0, *bit*(*B*), *abs*(*Id* - 1, *K*, *X*).

Counting Ones



1 : $abs(N, K', X'), \quad (\overline{\mathcal{G}}_1)$
 $1 \leq Id_2 \leq N, K'[Id_2] = 0, \quad \text{transition}$
 $K'' = \langle K', Id_2, 1 \rangle, X'' = X' + 1 \quad \text{relation}$
 $\models abs(N, K'', X'')$

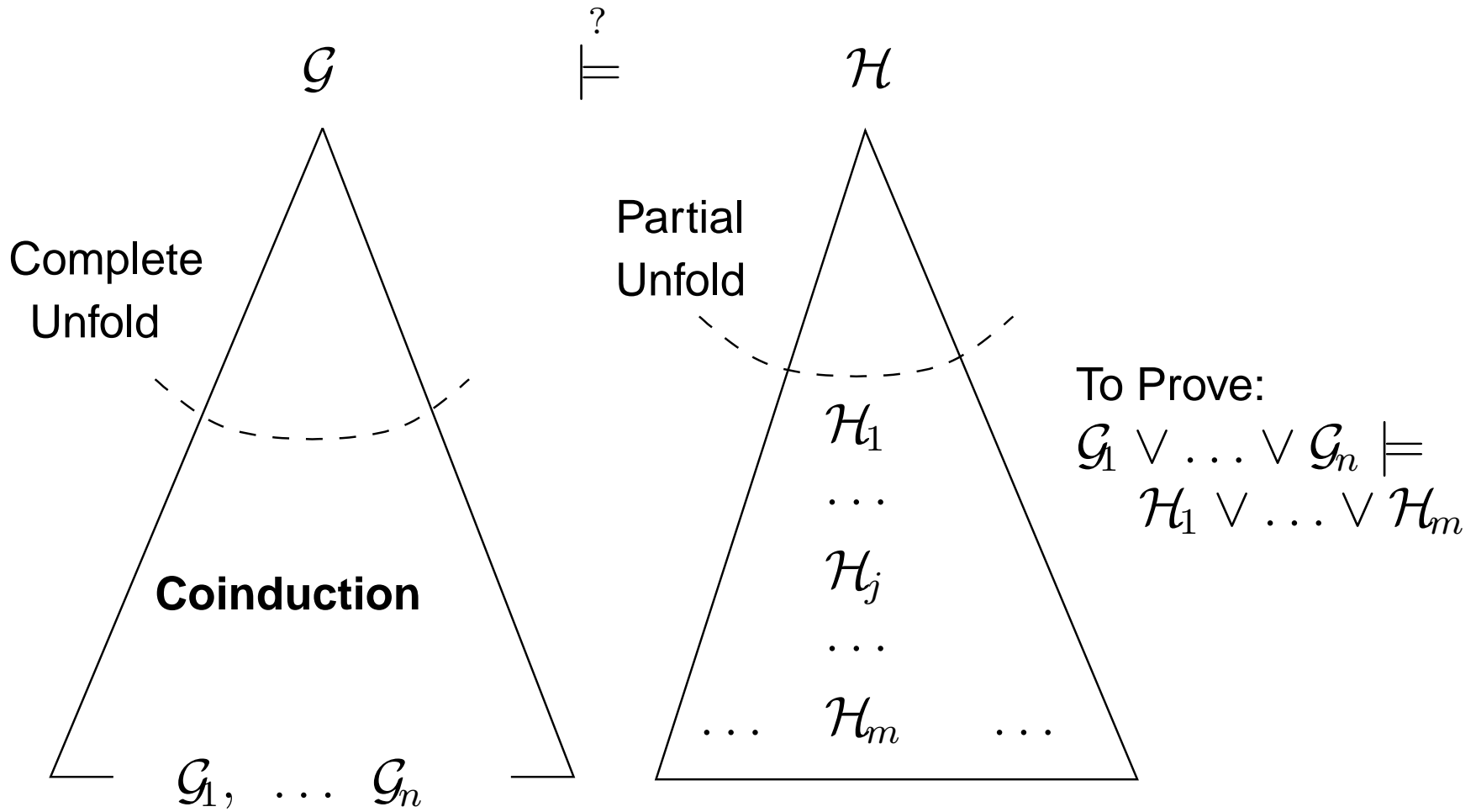
Assertion automatically generated by symbolic execution

Outline

1. Representing properties

2. Proving properties

Proof Method



Proof Rules

$$(LU+I) \quad \frac{\Pi \uplus \{\tilde{A} \vdash \mathcal{G} \models \mathcal{H}\}}{\Pi \cup \bigcup_{i=1}^n \{\tilde{A} \cup \{\mathcal{G}\} \vdash \mathcal{G}_i \models \mathcal{H}\}} \quad \text{unfold}(\mathcal{G}) = \{\mathcal{G}_1, \dots, \mathcal{G}_n\}$$

$$(RU) \quad \frac{\Pi \uplus \{\tilde{A} \vdash \mathcal{G} \models \mathcal{H}\}}{\Pi \cup \{\tilde{A} \vdash \mathcal{G} \models \mathcal{H}'\}} \quad \mathcal{H}' \in \text{unfold}(\mathcal{H})$$

$$(CO) \quad \frac{\Pi \uplus \{\tilde{A} \vdash \mathcal{G} \models \mathcal{H}\}}{\Pi \cup \{\tilde{A} \vdash \mathcal{H}'\theta \models \mathcal{H}\}} \quad \begin{array}{l} \mathcal{G}' \models \mathcal{H}' \in \tilde{A} \text{ and there} \\ \text{exists a substitution } \theta \text{ s.t.} \\ \mathcal{G} \models \mathcal{G}'\theta \end{array}$$

Proof Rules

$$(CP) \quad \frac{\Pi \uplus \{\tilde{A} \vdash \mathcal{G} \wedge p(\tilde{x}) \models \mathcal{H} \wedge p(\tilde{y})\}}{\Pi \uplus \{\tilde{A} \vdash \mathcal{G} \models \mathcal{H} \wedge \tilde{x} = \tilde{y}\}}$$

$$(SPL) \quad \frac{\Pi \uplus \{\tilde{A} \vdash \mathcal{G} \models \mathcal{H}\}}{\Pi \cup \bigcup_{i=1}^k \{\tilde{A} \vdash \mathcal{G} \wedge \psi_i \models \mathcal{H}\}} \quad \psi_1 \vee \dots \vee \psi_k \text{ is valid}$$

$$(EXR) \quad \frac{\Pi \uplus \{\tilde{A} \vdash \mathcal{G} \models \mathcal{H}(z)\}}{\Pi \uplus \{\tilde{A} \vdash \mathcal{G} \wedge z = e \models \mathcal{H}(z)\}} \quad z \text{ is existential}$$

Simple Example

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

$m4(0).$

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

$\text{even}(0).$

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

Simple Example

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

Complete (left) unfold

B: $X = 0 \models \text{even}(X)$

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

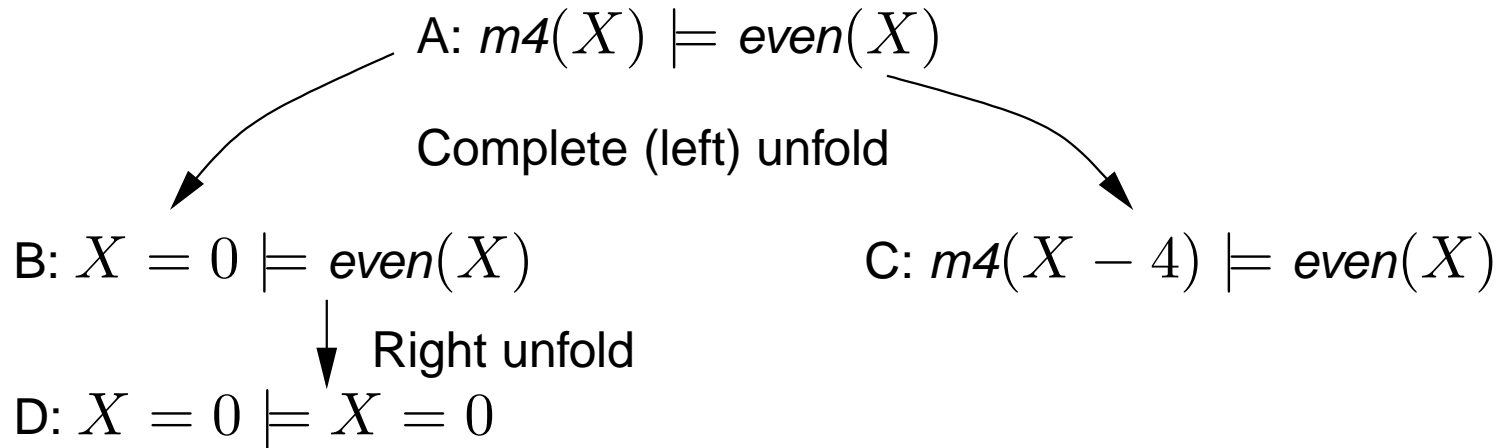
$m4(0).$

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

$\text{even}(0).$

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

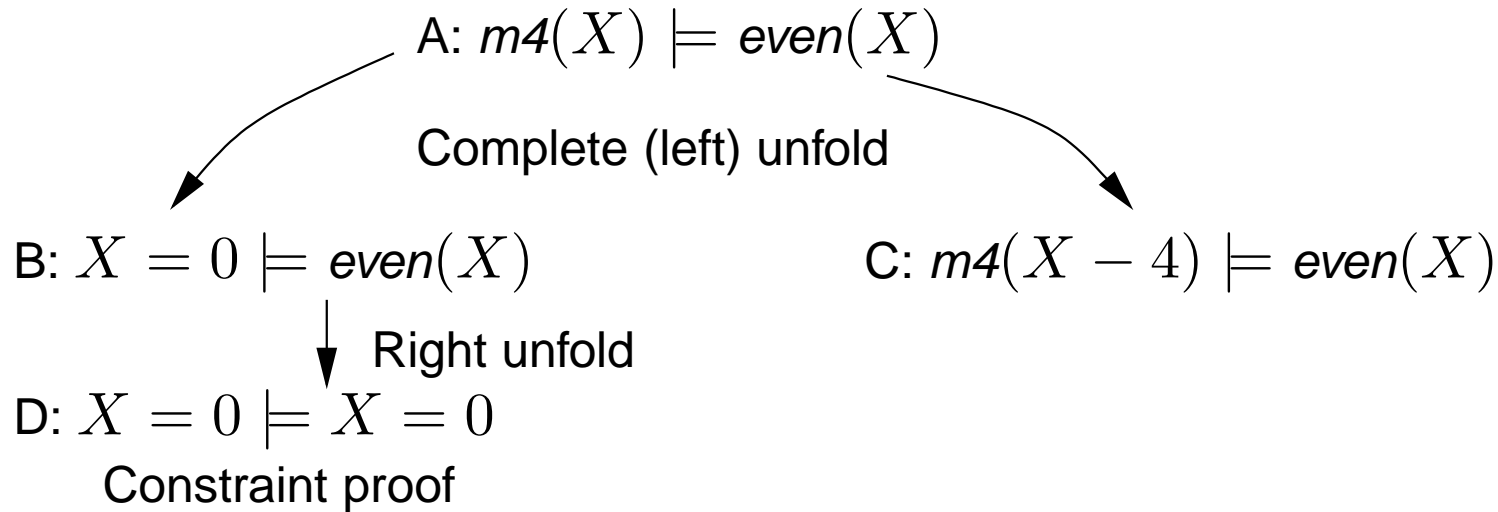
Simple Example



$m4(0).$
 $m4(X + 4) :- m4(X).$

$even(0).$
 $even(X + 2) :- even(X).$

Simple Example



$m4(0).$
 $m4(X + 4) :- m4(X).$

$even(0).$
 $even(X + 2) :- even(X).$

Simple Example

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

Complete (left) unfold

B: $X = 0 \models \text{even}(X)$

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

↓ Right unfold

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

Constraint proof

Coinduction

$A \Rightarrow C?$

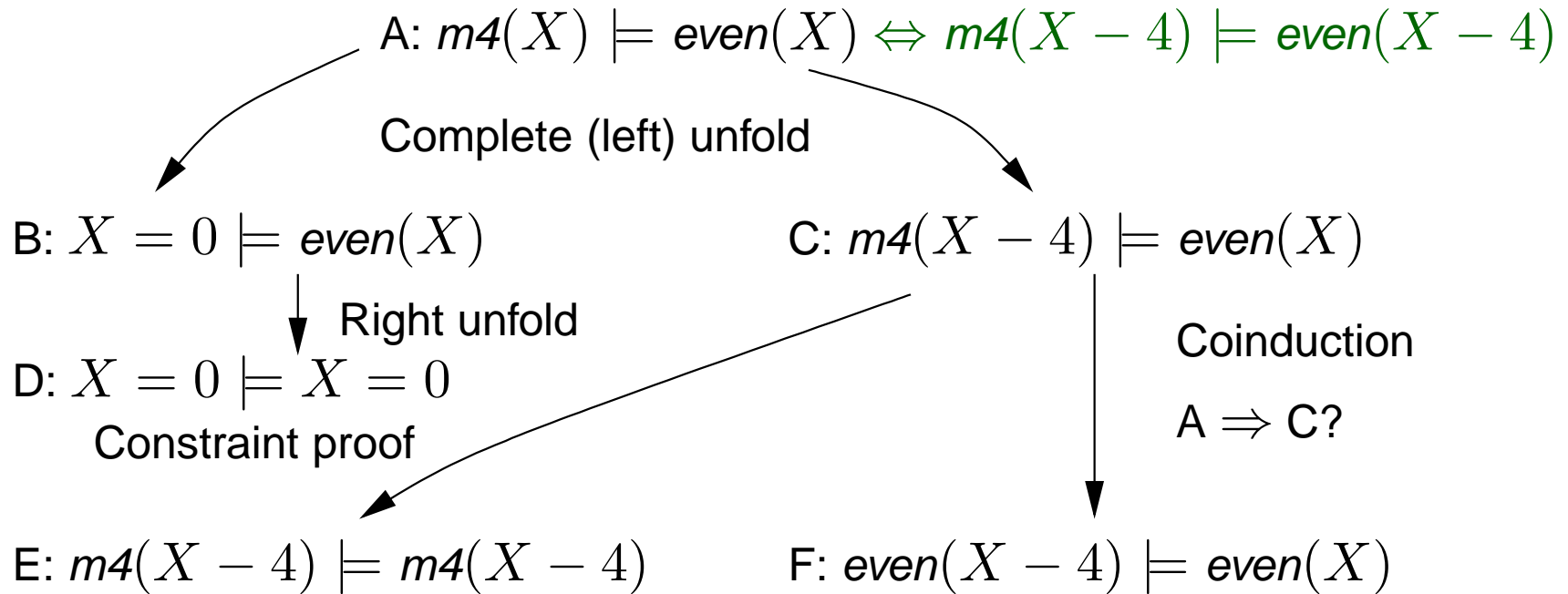
$m4(0).$

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

$\text{even}(0).$

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

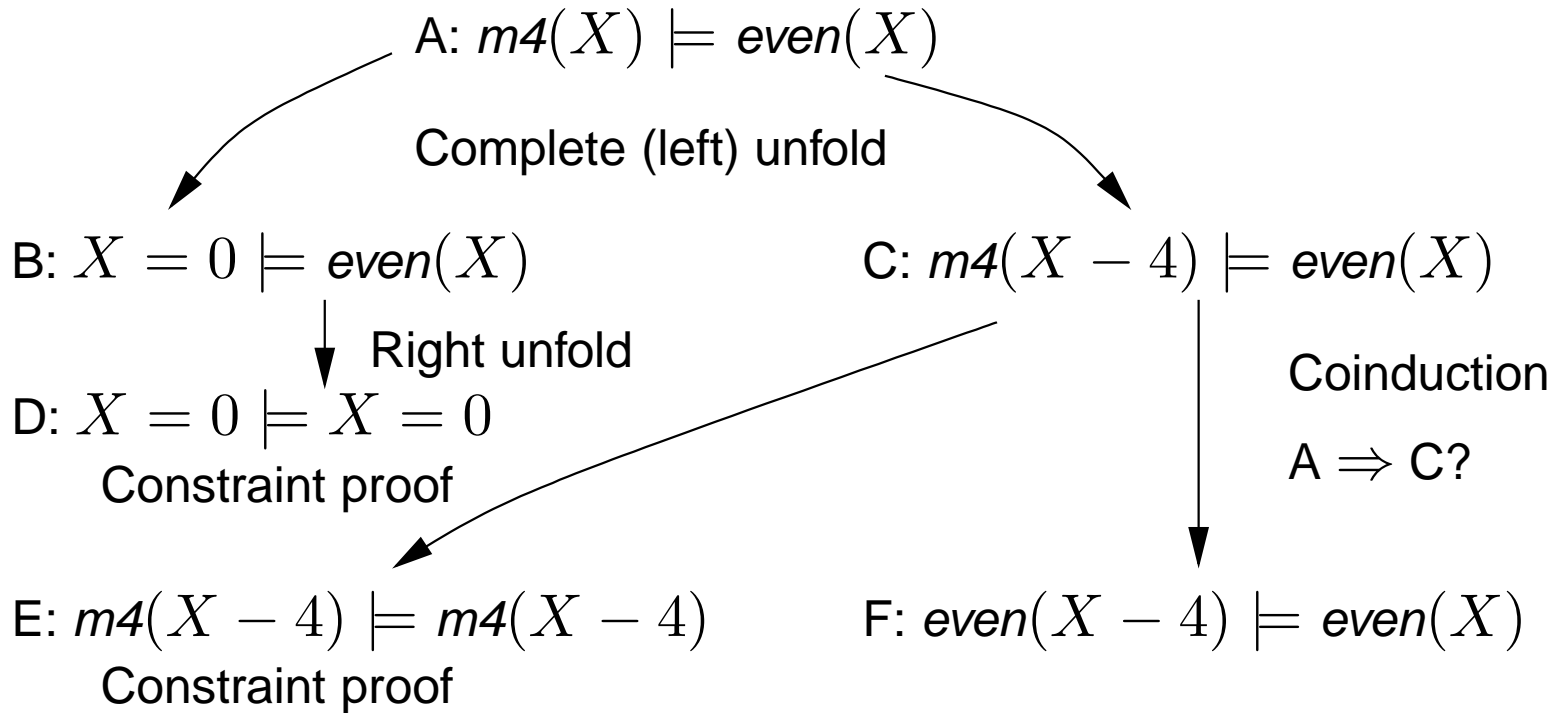
Simple Example



$m4(0).$
 $m4(X + 4) :- m4(X).$

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

Simple Example



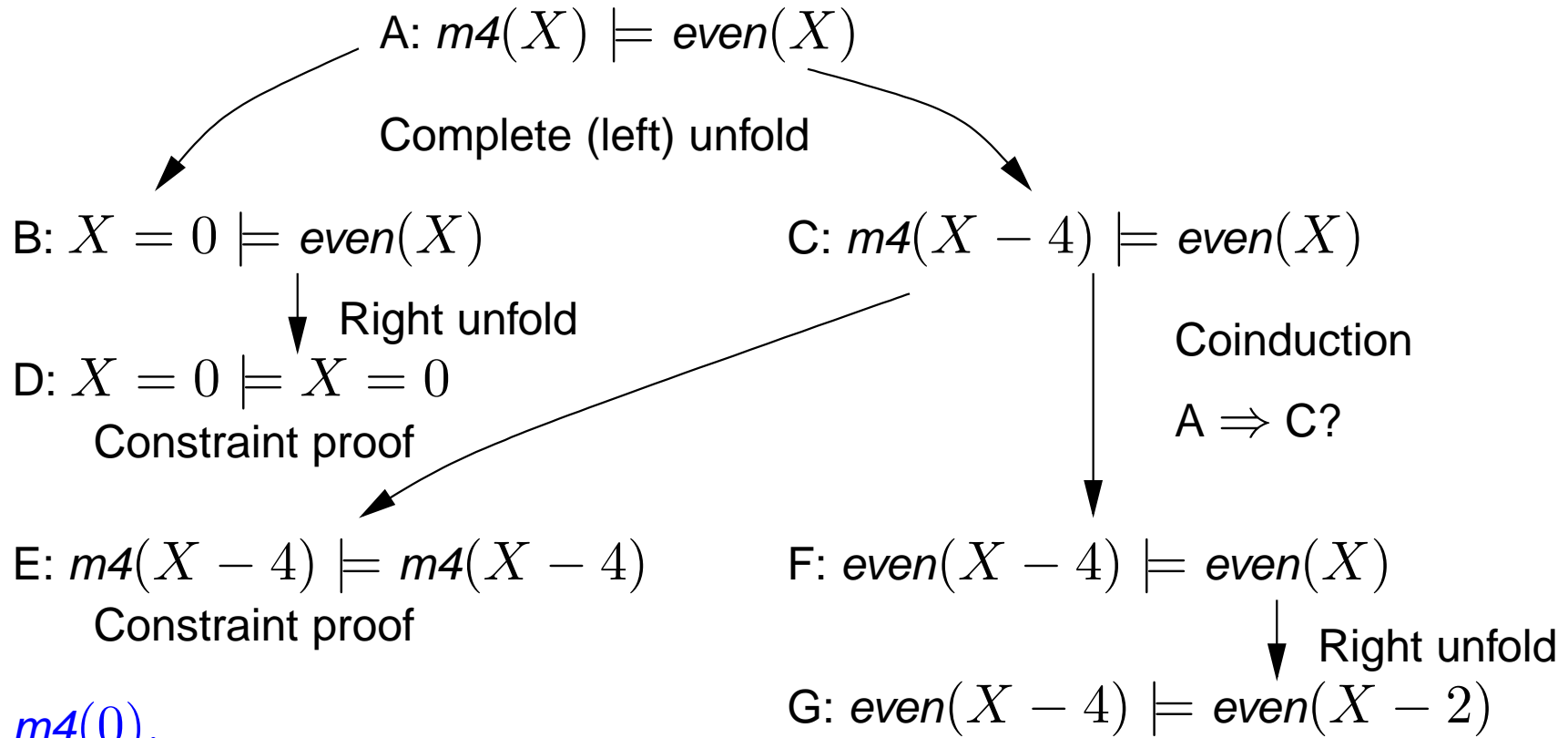
$m4(0).$

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

$even(0).$

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

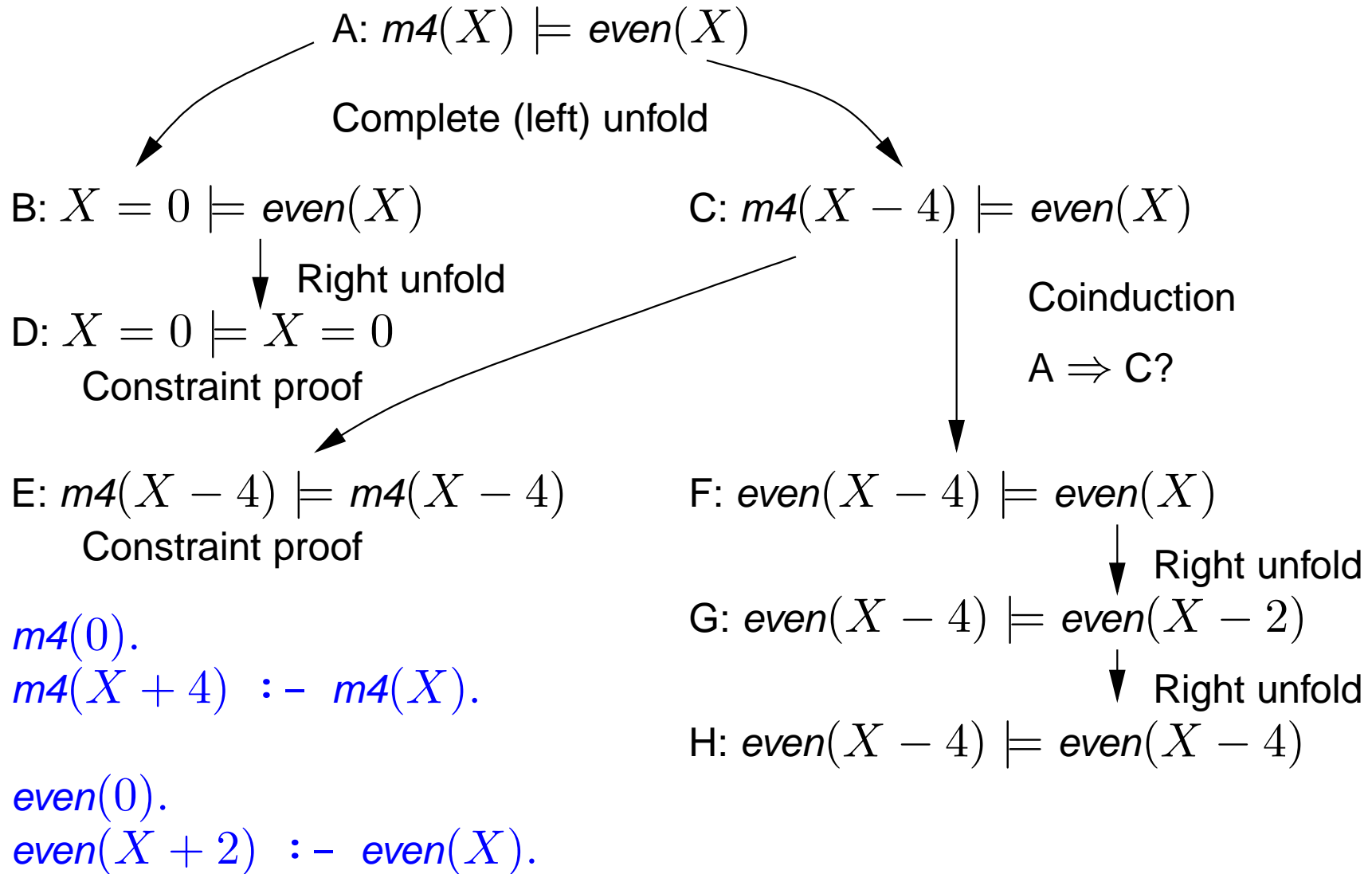
Simple Example



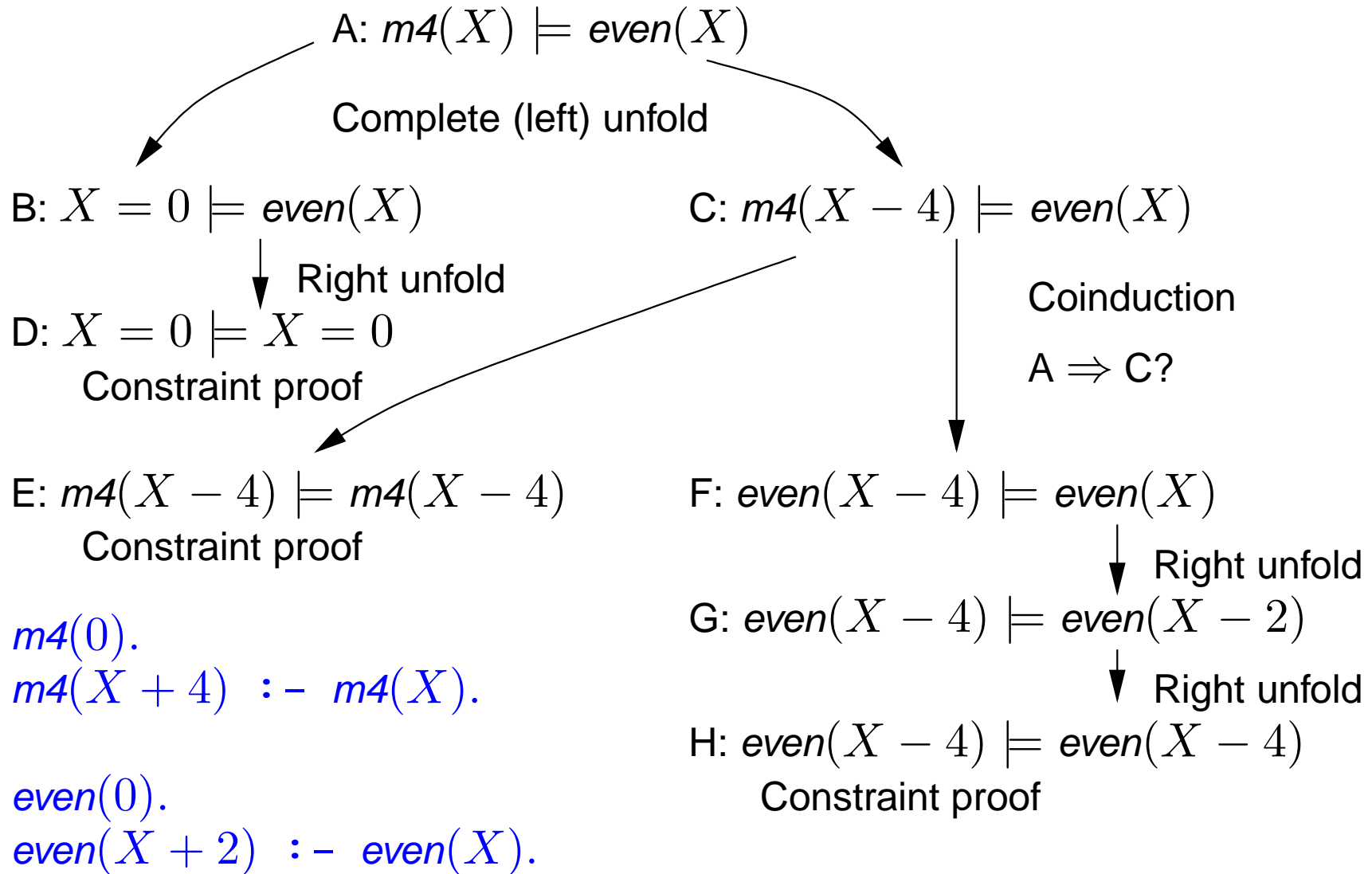
$m4(0).$
 $m4(X + 4) :- m4(X).$

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

Simple Example



Simple Example



On Coinduction

- “Dynamic” hypothesis → not requiring schema

On Coinduction

- “Dynamic” hypothesis \rightarrow not requiring schema

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

Program:

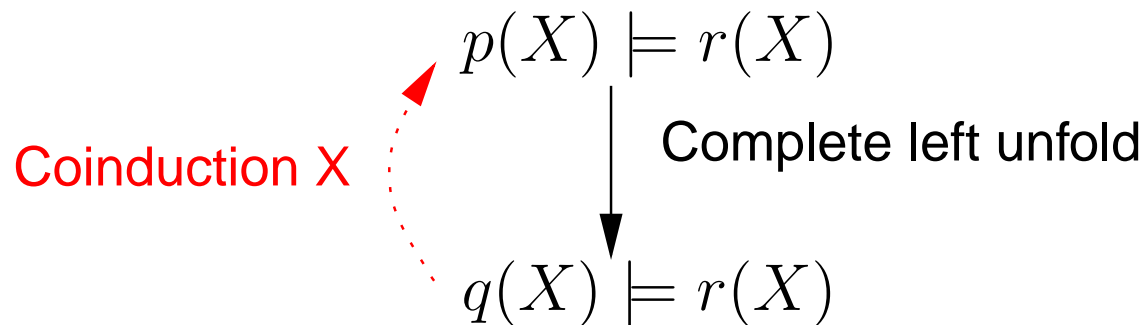
$p(X) :- q(X).$

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

$r(X).$

On Coinduction

- “Dynamic” hypothesis → not requiring schema

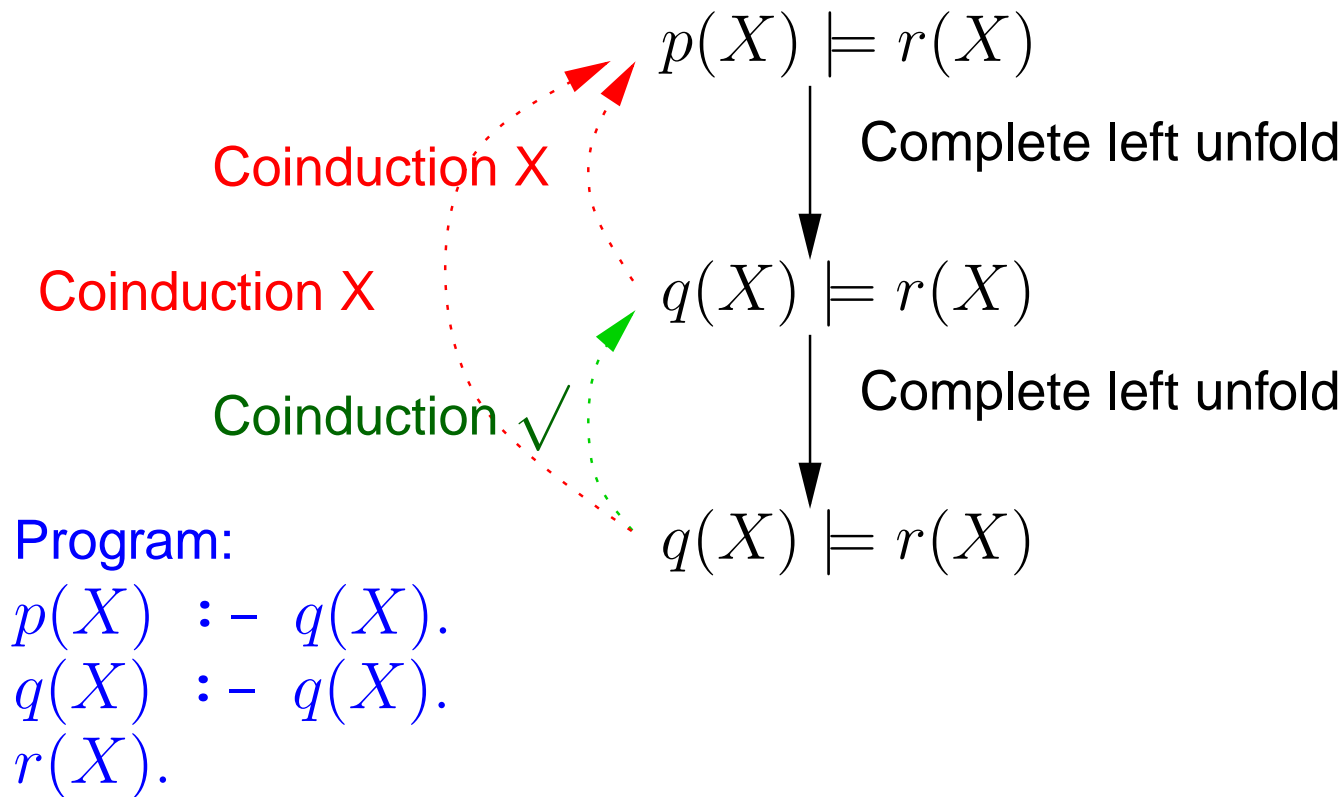


Program:

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

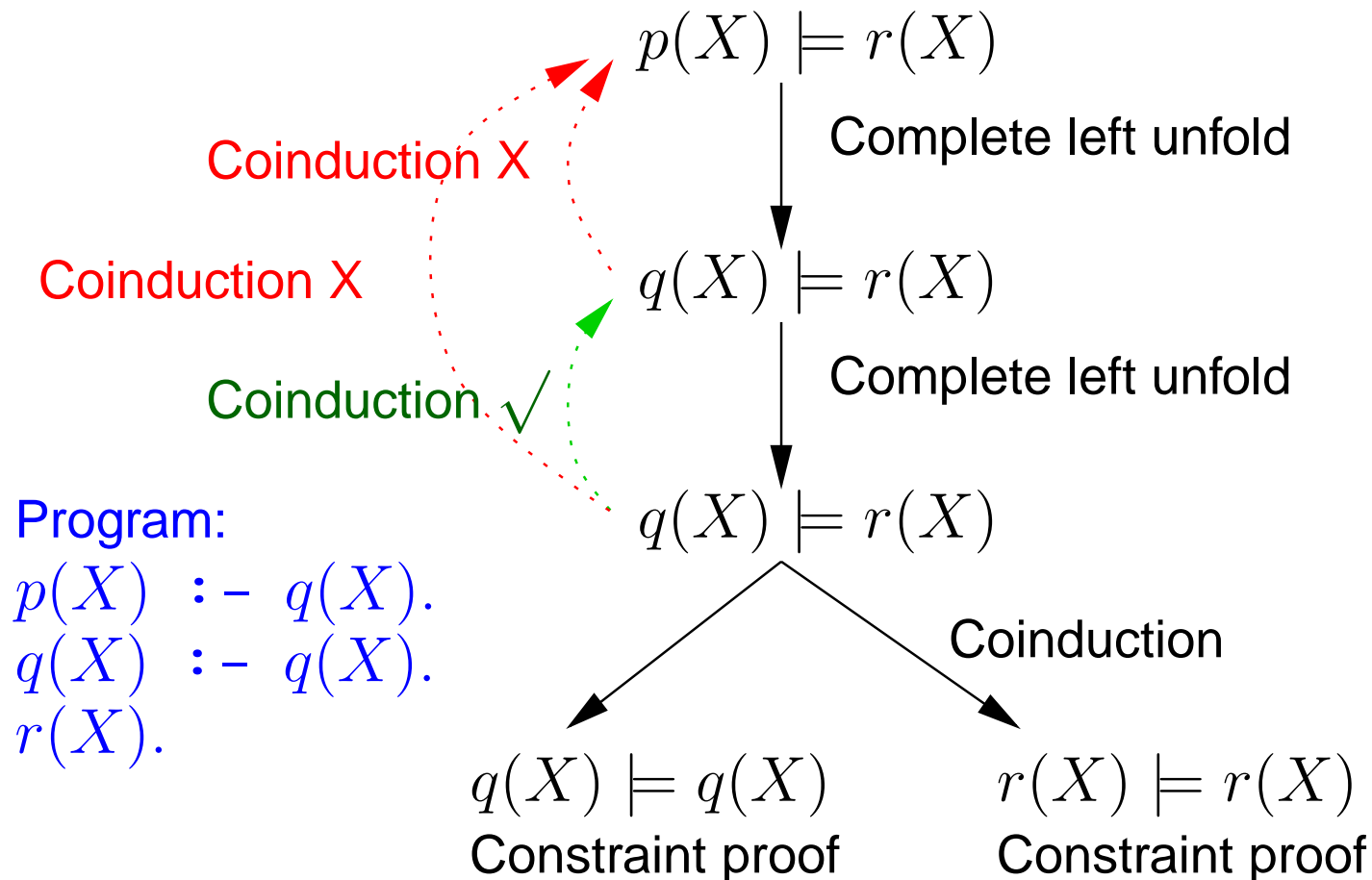
On Coinduction

- “Dynamic” hypothesis \rightarrow not requiring schema



On Coinduction

- “Dynamic” hypothesis \rightarrow not requiring schema



Algorithm

$\text{REDUCE}(\mathcal{G} \models \mathcal{H})$ returns boolean

choose one of the following:

- **Constraint Proof: (CP) + Constraint Solving**

Apply a constraint proof to $\mathcal{G} \models \mathcal{H}$.

If successful, return *true*, otherwise

return *false*

- **Coinduction: (CO)**

if there is an assumption $\mathcal{G}' \models \mathcal{H}'$ such that

$\text{REDUCE}(\mathcal{G} \models \mathcal{G}'\theta) = \text{true} \wedge$

$\text{REDUCE}(\mathcal{H}'\theta \models \mathcal{H}) = \text{true}$

then return *true*.

Algorithm

- **Unfold:**

choose left or right

case: Left: (LU+I)

Memoize $(\mathcal{G} \models \mathcal{H})$ as an assumption

choose an atom A in \mathcal{G} to reduce

for all reducts \mathcal{G}_L of \mathcal{G} using A :

if $\text{REDUCE}(\mathcal{G}_L \models \mathcal{H}) = \textit{false}$ return *false*

return *true*

case: Right: (RU)

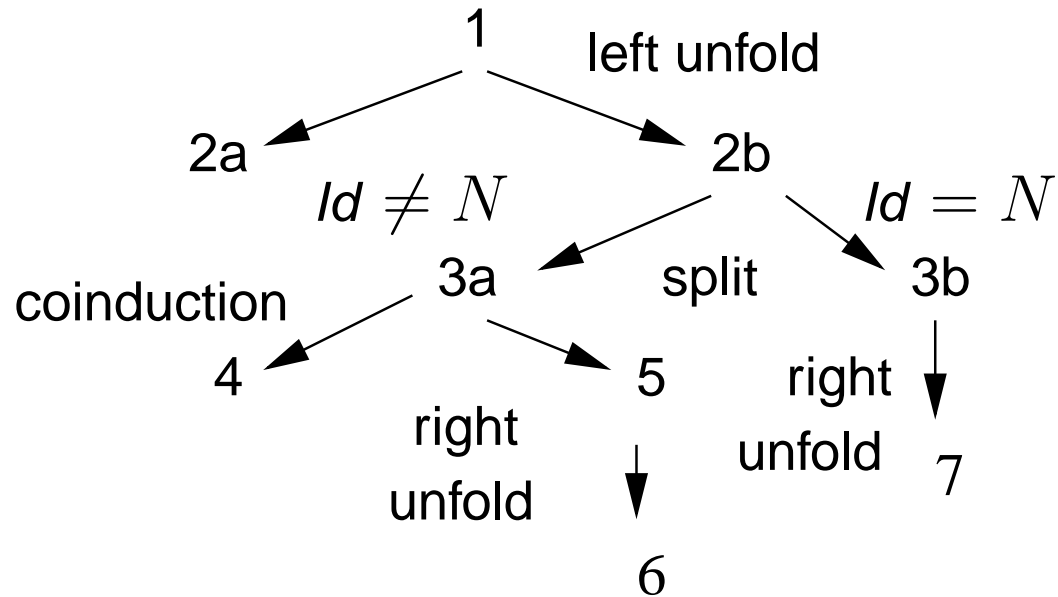
choose an atom A in \mathcal{H} to reduce, obtaining \mathcal{G}_R

return $\text{REDUCE}(\mathcal{G} \models \mathcal{G}_R)$

Algorithm

- **Split:**
Find an index variable Id and a parameter variable N and apply the split rule using $Id \neq N \vee Id = N$ to split \mathcal{G} into \mathcal{G}_1 and \mathcal{G}_2 .
return $\text{REDUCE}(\mathcal{G}_1 \models \mathcal{H}) \wedge \text{REDUCE}(\mathcal{G}_2 \models \mathcal{H})$
- **Existential Variable Removal:**
If an existential array variable z appears in the form $z = \langle x, i, e \rangle$, then $\mathcal{H}' = \mathcal{H}[\langle x, i, e \rangle / z]$
If however z appears in the form $x = \langle z, i, e \rangle$ where x is not existential, then find an expression in \mathcal{G} of the form $x = \langle x', i, e \rangle$ and $\mathcal{H}' = \mathcal{H}[x' / z]$
return $\text{REDUCE}(\mathcal{G} \models \mathcal{H}')$

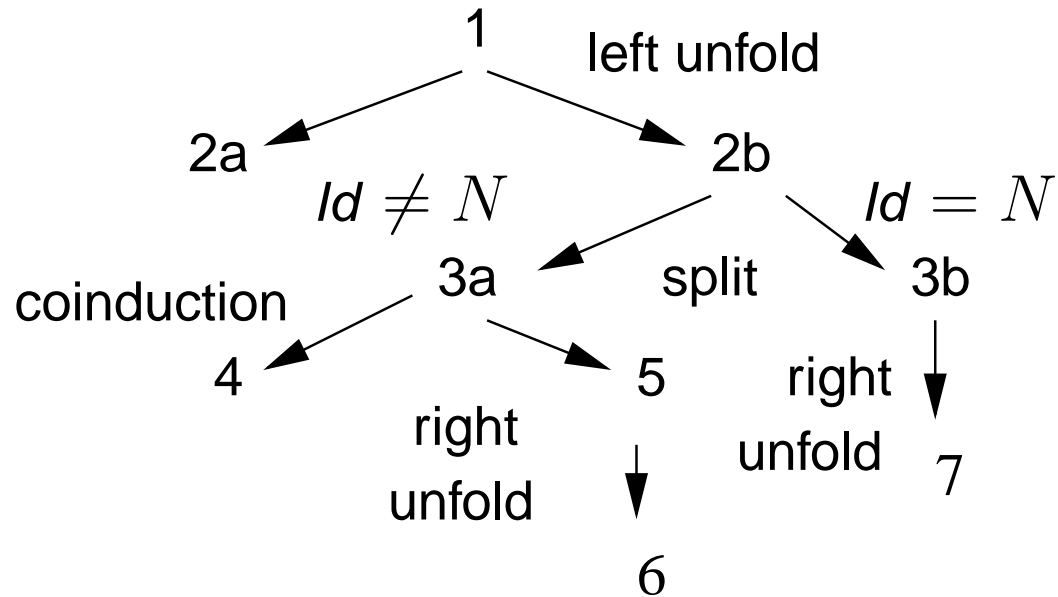
Counting Ones



1 : $abs(N, K', X'), 1 \leq ld_2 \leq N, K'[ld_2] = 0,$
 $\models abs(N, \langle K', ld_2, 1 \rangle, X' + 1)$

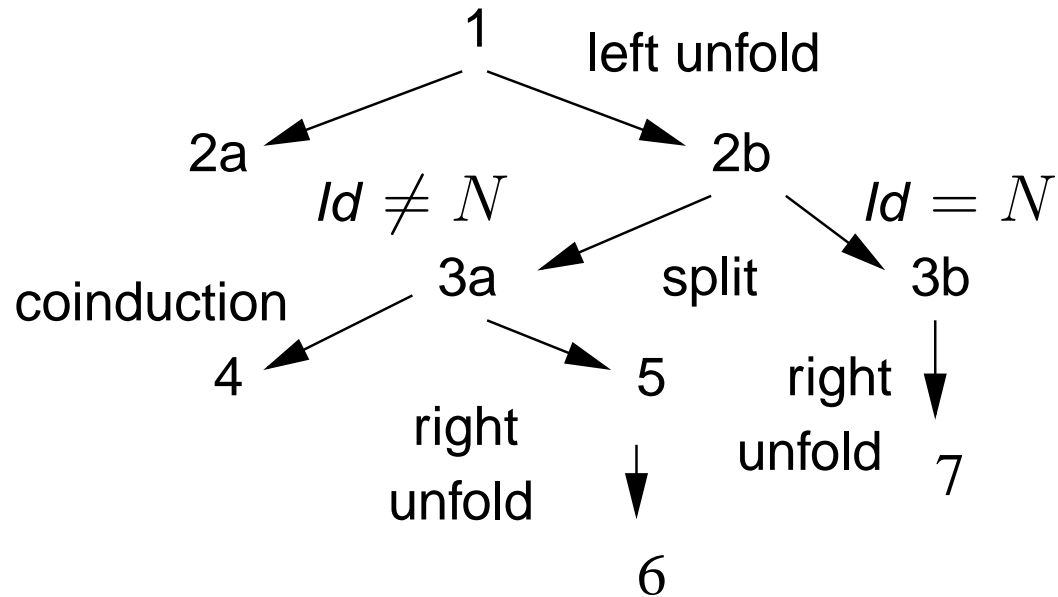
3a: $abs(N - 1, K'', X' - B), bit(B), K''[ld_2] = 0, 1 \leq ld_2 < N$
 $\models abs(N, \langle \langle K'', N, B \rangle, ld_2, 1 \rangle, X' + 1)$

Counting Ones



5: $abs(N - 1, \langle K'', Id_2, 1 \rangle, X' - B + 1), bit(B),$
 $K''[Id_2] = 0, 1 \leq Id_2 < N$
 $\models abs(N, \langle \langle K'', N, B \rangle, Id_2, 1 \rangle, X' + 1)$

Counting Ones



6: $abs(N - 1, \langle K'', Id_2, 1 \rangle, X' - B + 1), bit(B),$
 $K''[Id_2] = 0, 1 \leq Id_2 < N$
 $\models abs(N - 1, ?K''', X' + 1 - ?B'), bit(?B'),$
 $\langle \langle K'', N, B \rangle, Id_2, 1 \rangle = \langle ?K''', N, ?B' \rangle$

Bakery Algorithm

```
process(id) {  
  ⟨0⟩ t[id] = max(t[1], ..., t[N]) + 1;  
  ⟨1⟩ await(forall j≠id : t[j]==0 ∨ t[id]<t[j]);  
  ⟨2⟩ t[id] = 0; goto ⟨0⟩ }
```

$allz(K, T, 1) :- K[1] = 0, T[1] = 0.$

$allz(K, T, Id) :- Id > 1, K[Id] = 0, T[Id] = 0,$
 $allz(K, T, Id - 1).$

$max(T, 1, X) :- X \geq T[1].$

$max(T, Id, X) :- Id > 1, X \geq T[Id], max(T, Id - 1, X).$

$abs(K, T, 1) :- (K[1] = 0, T[1] = 0) \vee (K[1] = 1, T[1] > 0).$

$abs(K, T, Id) :- Id > 1, ((K[Id] = 0, T[Id] = 0) \vee$
 $(K[Id] = 1, T[Id] > 0)), abs(K, T, Id - 1).$

Bakery Algorithm

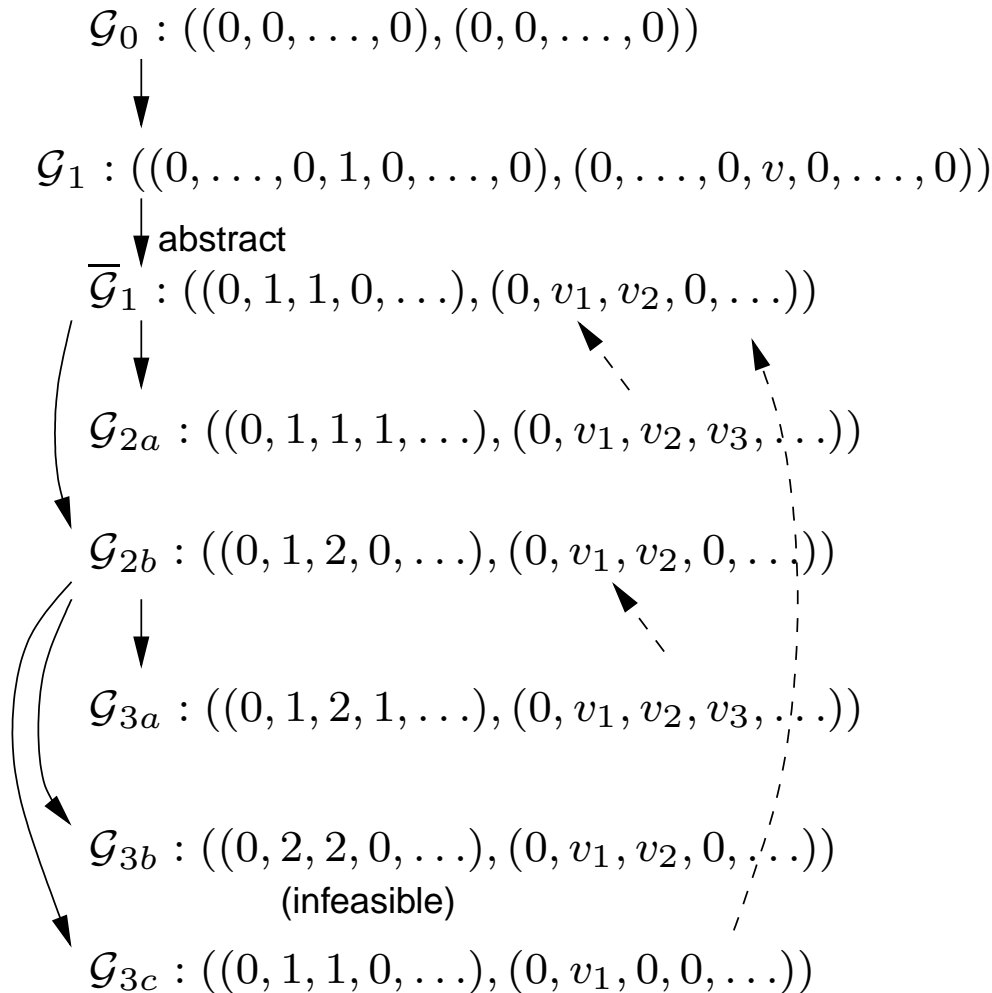
```
process(id) {  
  ⟨0⟩ t[id] = max(t[1], ..., t[N]) + 1;  
  ⟨1⟩ await(forall j≠id : t[j]==0 ∨ t[id]<t[j]);  
  ⟨2⟩ t[id] = 0; goto ⟨0⟩ }
```

allz(K, T, N) : $T[id] = 0$ for all id from 1 to N

max(T, N, X) : $X \geq T[id]$ for all id from 1 to N

abs(K, T, N) : $T[id] = 0$ when $K[id] = 0$ and $T[id] > 0$
when $K[id] = 1$ for all id from 1 to N

Bakery Algorithm



$allz(K, T, N),$
 $max(T, N, X),$
 $1 \leq Id \leq N, K[Id] = 0,$
 $K' = \langle K, Id, 1 \rangle,$
 $T' = \langle T, Id, X + 1 \rangle$
 $\models abs(K', T', N)$

Related Work

- Coinduction rule for entailment of recursively-defined properties (Jaffar et al. 2008)
- Unfold/fold proof method (Rouychoudhury et al. 2001)
- Environment predicates (Clarke et al. 2006, 2008)
- Indexed predicates (Lahiri & Bryant 2004)
- Gap-order constraints (Abdulla et al. 2007)
- Counter abstraction (Pnueli et al. 2002, Sun et al. 2009)
- Invisible invariants (Pnueli et al. 2001)
- Ordered-set abstraction and induction (McMillan 2000)

Conclusion

- Modeling of parameterized systems properties using arrays and recursive predicates
- Algorithm to prove recursive assertions:
Automatic: Inductive proof using **Search:** We **automatically** constructs induction hypothesis.
- In this paper, and in other experiments, we have shown that many typical program verification lemmas can be proved.
- Implementation Status: Automatic implementation in CLP(R) can prove recursive assertions for data structure verification