

A Framework for Program Reasoning Based on Constraint Traces

Andrew E. Santosa

`andrews@comp.nus.edu.sg`

School of Computing
National University of Singapore

- Transition systems and deduction rules are naturally modeled as CLP clauses.

- Transition systems and deduction rules are naturally modeled as CLP clauses.
- Advancing the limits of program reasoning via CLP technology.

Our Approach

1. Modeling of programs in CLP: sequential, concurrent, multiprocedure, hardware constraints, arrays, pointer data structures, timed automata, and Statecharts

Our Approach

1. Modeling of programs in CLP: sequential, concurrent, multiprocedure, hardware constraints, arrays, pointer data structures, timed automata, and Statecharts
2. A proof method for general CLP:
 - a. Proving assertions $G \models \psi$, expressing *safety*.
 - b. *Abstract intermittently*.
 - c. Main algorithm for program verification and analysis + *dynamic summarization*.

Our Approach

1. Modeling of programs in CLP: sequential, concurrent, multiprocedure, hardware constraints, arrays, pointer data structures, timed automata, and Statecharts
2. A proof method for general CLP:
 - a. Proving assertions $G \models \psi$, expressing *safety*.
 - b. *Abstract intermittently*.
 - c. Main algorithm for program verification and analysis + *dynamic summarization*.
3. Proving *recursive assertions* $G \models H$ where H contains CLP atoms.
 - a. Verification of *data structure properties*
 - b. Proving *non-behavioral properties* (e.g. *new classes of symmetries*) for *reduction*

Our Approach

1. Modeling of programs in CLP: sequential, concurrent, multiprocedure, hardware constraints, arrays, pointer data structures, timed automata, and Statecharts
2. A proof method for general CLP:
 - a. Proving assertions $G \models \psi$, expressing *safety*.
 - b. *Abstract intermittently*.
 - c. Main algorithm for program verification and analysis + *dynamic summarization*.
3. Proving *recursive assertions* $G \models H$ where H contains CLP atoms.
 - a. Verification of *data structure properties*
 - b. Proving *non-behavioral properties* (e.g. *new classes of symmetries*) for *reduction*
4. Implementation & experiments

Our Approach

1. Modeling of programs in CLP: sequential, concurrent, multiprocedure, hardware constraints, arrays, pointer data structures, timed automata, and Statecharts
2. A proof method for general CLP:
 - a. Proving assertions $G \models \psi$, expressing *safety*.
 - b. Abstract *intermittently*.
 - c. Main algorithm for program verification and analysis + dynamic summarization.
3. Proving *recursive assertions* $G \models H$ where H contains CLP atoms.
 - a. Verification of *data structure properties*
 - b. Proving *non-behavioral properties* (e.g. *new classes of symmetries*) for *reduction*
4. Implementation & experiments

What is a Constraint Logic Program?

A conjunction of implications (*Horn clauses*)

$$p(X) \text{ :- } X = 0.$$

$$p(X) \text{ :- } p(X'), X = X' + 2.$$

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., $\{p(0), p(2), p(4), \dots\}$.

Modeling Programs Top-Down

Initially $x = 0$ and $y = 0$.

Process 1

loop forever

$\langle 0 \rangle \quad x := y + 1$

$\langle 1 \rangle \quad \mathbf{await} (x < y \vee y = 0)$

$\langle 2 \rangle \quad x := 0$

end loop

Process 2

loop forever

$\langle 0 \rangle \quad y := x + 1$

$\langle 1 \rangle \quad \mathbf{await} (y < x \vee x = 0)$

$\langle 2 \rangle \quad y := 0$

end loop

Top-Down CLP Model

$p(0, 0, 0, 0)$.

$p(1, L_2, X', Y) :- X' = Y + 1, p(0, L_2, X, Y)$.

$p(L_1, 1, X, Y') :- Y' = X + 1, p(L_1, 0, X, Y)$.

CLP program least model = Collecting Semantics

Modeling Programs Bottom-Up

Initially $x = s = 0, n \geq 0$.

$\langle 0 \rangle$ **while** $(x < n)$ **do**

$\langle 1 \rangle$ $s := s + x$

$\langle 2 \rangle$ $x := x + 1$

end do

Bottom-Up CLP Model

$p(0, X, S, N, S_f) :- X < N, p(1, X, S, N, S_f).$

$p(0, X, S, N, S_f) :- X \geq N, p(\Omega, X, S, N, S_f).$

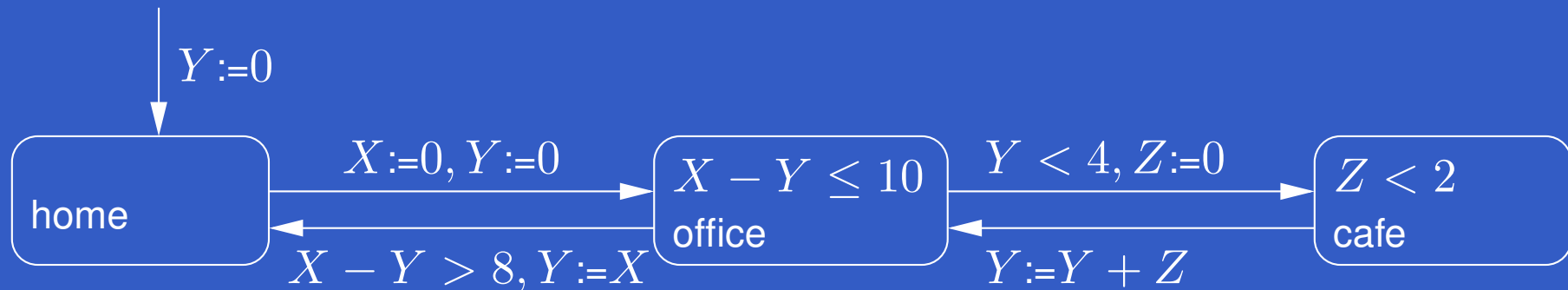
$p(1, X, S, N, S_f) :- p(2, X, S + X, N, S_f).$

$p(2, X, S, N, S_f) :- p(0, X + 1, S, N, S_f).$

$p(\Omega, X, S, N, S).$

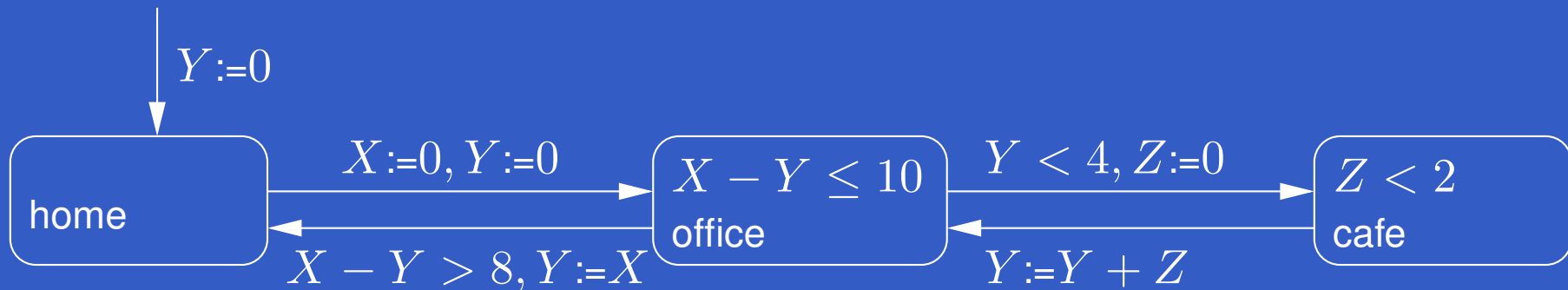
Least model = States reaching “point of interest”

Timed Automata



- X, Z are clock variables, Y is normal variable
- Clocks and normal variables can be mixed
- Constraints with more than 2 clocks are possible

Timed Automata



$$p(\mathit{home}, X, Y, Z) \quad :- \quad E \geq 0, X = E, Y = 0, Z = E.$$

$$p(\mathit{office}, X', Y', Z') \quad :-$$

$$E \geq 0, X' = E, Y' = 0, Z' = Z + E, X' - Y \leq 10,$$

$$p(\mathit{home}, _, Y, Z).$$

$$p(\mathit{cafe}, X', Y', Z') \quad :-$$

$$E \geq 0, X' = X + E, Y' = Y, Z' = E, Y < 4, X - Y \leq 10, Z' < 2,$$

$$p(\mathit{office}, X, Y, _).$$

No need to consider *regions* [Gupta 1997]

Our Approach

1. Modeling of programs in CLP: sequential, concurrent, multiprocedure, hardware constraints, arrays, pointer data structures, timed automata, and Statecharts
2. A proof method for general CLP:
 - a. Proving assertions $G \models \psi$, expressing *safety*.
 - b. *Abstract intermittently*.
 - c. Main algorithm for program verification and analysis + *dynamic summarization*.
3. Proving *recursive assertions* $G \models H$ where H contains CLP atoms.
 - a. Verification of *data structure properties*
 - b. Proving *non-behavioral properties* (e.g. *new classes of symmetries*) for *reduction*
4. Implementation & experiments

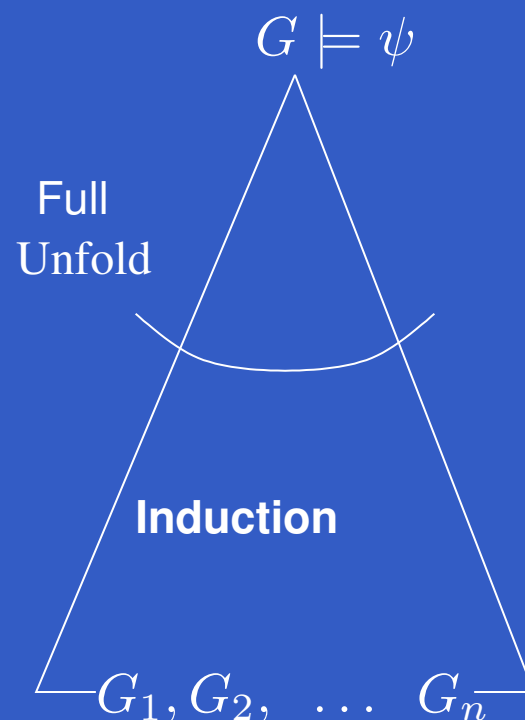
Traditional Safety (Invariance)

$G \models \psi$, where G is a goal and ψ is a constraint

Examples:

- Bakery: $p(2, 2, X, Y) \models \square$
- Sum: $p(0, 0, 0, N, S_f), N \geq 0 \models S_f = (N^2 - N)/2$
- Timed Automaton: $p(\text{home}, X, Y, Z) \models Y \leq 20$

Summary of Proof Technique



Proof holds if $\forall 1 \leq i \leq n : G_i \models \psi$

Proceeds by manipulation of a set of **proof obligations**

$$\tilde{A} \vdash G \models \psi$$

Proof Rules

$$(DP) \quad \frac{\Pi \uplus \{\tilde{A} \vdash G \models \psi\}}{\Pi} \quad \text{There exists a substitution } \sigma \text{ s.t. } G \models \psi\sigma$$

$$(LU) \quad \frac{\Pi \uplus \{\tilde{A} \vdash G \models \psi\}}{\Pi \cup \bigcup_{i=1}^n \{\tilde{A} \cup \{G \models \psi\} \vdash G_i \models \psi\}} \quad \text{unfold}(G) = \{G_1, \dots, G_n\}$$

$$(AP) \quad \frac{\Pi \uplus \{\tilde{A} \vdash p(\tilde{X}), \phi \models \psi\}}{\Pi \cup \{\emptyset \vdash \psi'\theta, \tilde{X} = \tilde{Y}, \phi \models \psi\}} \quad \begin{array}{l} p(\tilde{Y}), \phi' \models \psi' \in \tilde{A}, \theta \text{ renames} \\ \text{apart } p(\tilde{Y}), \phi' \models \psi' \text{ from} \\ p(\tilde{X}), \phi \models \psi, \text{ and } p(\tilde{X}), \phi \\ \text{is } p\text{-subsumed by } p(\tilde{Y}), \phi'. \end{array}$$

$$(CUT) \quad \frac{\Pi \uplus \{\tilde{A} \vdash G \models \psi\}}{\Pi \cup \{\tilde{A} \vdash G' \models \psi\}} \quad G \models G'$$

Main Theorem

A safety assertion $G \models \psi$ holds if, starting with the proof obligation $\Pi = \{\emptyset \vdash G \models \psi\}$, there exists a sequence of applications of proof rules that results in $\Pi = \emptyset$.

Proof Method

Use original assertion to prove encountered assertion

$$p(X) \quad :- \quad X = 0.$$

$$p(X) \quad :- \quad p(X'), X = X' + 2.$$

$$p(X) \models \text{even}(X)$$

Unfold (LU)

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

Proof Method

Use original assertion to prove encountered assertion

$$p(X) \quad :- \quad X = 0.$$

$$p(X) \quad :- \quad p(X'), X = X' + 2.$$

$$p(X) \models \text{even}(X)$$

Unfold (LU)

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

Direct proof (DP)

Proof Method

Use original assertion to prove encountered assertion

$$p(X) \quad :- \quad X = 0.$$

$$p(X) \quad :- \quad p(X'), X = X' + 2.$$

$$p(X) \models \text{even}(X)$$

FULL UNFOLD

Unfold (LU)

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

Direct proof (DP)

$$X = X' + 2, p(X') \models \text{even}(X)$$

Proof Method

Use original assertion to prove encountered assertion

$$p(X) \quad :- \quad X = 0.$$

$$p(X) \quad :- \quad p(X'), X = X' + 2.$$

$$p(X) \models \text{even}(X)$$

FULL UNFOLD

Unfold (LU)

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

Direct proof (DP)

$$X = X' + 2, p(X') \models \text{even}(X)$$

Induction (AP)

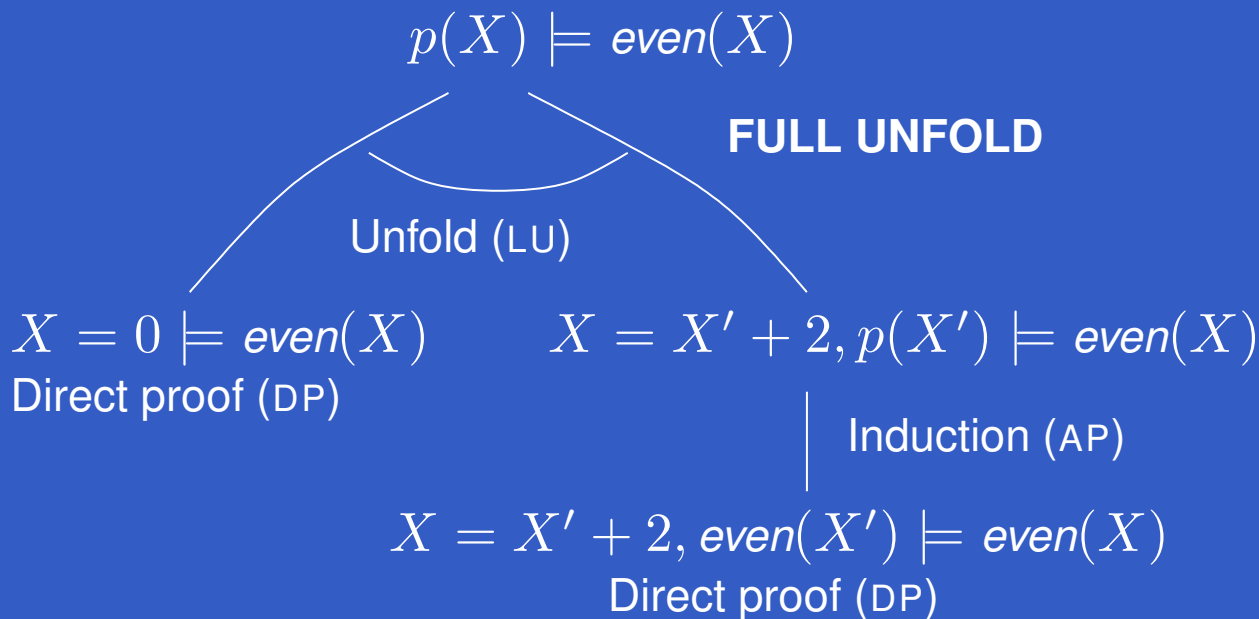
$$X = X' + 2, \text{even}(X') \models \text{even}(X)$$

Proof Method

Use original assertion to prove encountered assertion

$$p(X) \quad :- \quad X = 0.$$

$$p(X) \quad :- \quad p(X'), X = X' + 2.$$



Exact Traversal by Strongest Postcondition

Strongest postcondition computation using **UNFOLD**

Example:

$\langle 1 \rangle x := x + 1 \langle 2 \rangle$

modeled as clause R:

$p(1, X, Y) :- p(2, X + 1, Y).$

Assertion: $p(1, X, Y), Y \geq X \models \psi$

UNFOLD (LU) using R:

$p(2, X', Y), Y + 1 \geq X', X' = X + 1 \models \psi$

Same as:

$sp(x := x + 1, y \geq x) \equiv y + 1 \geq x$

Exact Traversal by Strongest Postcondition

Strongest postcondition computation using **UNFOLD**

Challenge No. 1:

Requires unbounded no. of variables

Example:

- $x := y + y$ followed by $y := 0$ results in condition $\langle \exists z : x = 2z \rangle \wedge y = 0$.

Solved by CLP projection

Exact Traversal by Strongest Postcondition

Strongest postcondition computation using **UNFOLD**

Challenge No. 2:

Proof Size

We want to join redundant subproofs as much as possible to reduce the size of proof tree

Our solution: *next*

Dynamic Summarization: Exact Traversal Key Method

```
⟨0⟩  if (a = 1) then  
⟨1⟩    skip  
      end if  
⟨2⟩  if (b = 1) then  
⟨3⟩    c := 0  
      end if  
⟨4⟩  if (c = 1) then  
⟨5⟩    x := x + 1  
      end if
```

```
p(0, X, A, B, C, Xf) :- A = 1, p(1, X, A, B, C, Xf).  
p(0, X, A, B, C, Xf) :- A ≠ 1, p(2, X, A, B, C, Xf).  
p(1, X, A, B, C, Xf) :- p(2, X, A, B, C, Xf).  
p(2, X, A, B, C, Xf) :- B = 1, p(3, X, A, B, C, Xf).  
p(2, X, A, B, C, Xf) :- B ≠ 1, p(4, X, A, B, C, Xf).  
p(3, X, A, B, C, Xf) :- p(4, X, A, B, 0, Xf).  
p(4, X, A, B, C, Xf) :- C = 1, p(5, X, A, B, C, Xf).  
p(4, X, A, B, C, Xf) :- C ≠ 1, p(Ω, X, A, B, C, Xf).  
p(5, X, A, B, C, Xf) :- p(Ω, X + 1, A, B, C, Xf).  
p(Ω, X, A, B, C, X).
```

Dynamic Summarization: Exact Traversal Key Method

$$1. p(0, X, A, B, C, X_f), \underline{X > 0} \models X_f > 0$$

LU

LU

$$2a. p(1, X, A, B, C, X_f), \underline{X > 0}, A = 1 \models X_f > 0$$

LU

$$2b. p(2, X, A, B, C, X_f), X > 0, A \neq 1 \models X_f > 0$$

(Becomes Redundant)

$$3. p(2, X, A, B, C, X_f), \underline{X > 0}, A = 1 \models X_f > 0$$

LU

LU

$$4a. p(3, X, A, B, C, X_f), \underline{X > 0}, A = 1, B = 1 \models X_f > 0$$

LU

$$4b. p(4, X, A, B, C, X_f), \underline{X > 0}, A = 1, B \neq 1 \models X_f > 0$$

(Proof Not Shown)

$$5. p(4, X, A, B, C', X_f), \underline{X > 0}, A = 1, B = 1, \underline{C' = 0} \models X_f > 0$$

LU

LU

$$6a. p(5, X, A, B, C', X_f), X > 0, A = 1, B = 1, \underline{C' = 0}, \underline{C' = 1} \models X_f > 0$$

DP

$$6b. p(\Omega, X, A, B, C', X_f), \underline{X > 0}, A = 1, B = 1, C' = 0 \models X_f > 0$$

7. $\neg \square$

$$8. \underline{X > 0}, A = 1, B = 1, C' = 0, \underline{X = X_f} \models X_f > 0$$

DP

9. $\neg \square$

Dynamic Summarization: Exact Traversal Key Method

$$1. p(0, X, A, B, C, X_f), \underline{X > 0} \models X_f > 0$$

LU

$$2a. p(1, X, A, B, C, X_f), \underline{X > 0}, A = 1 \models X_f > 0$$

LU

$$2b. p(2, X, A, B, C, X_f), X > 0, A \neq 1 \models X_f > 0$$

(Becomes Redundant)

$$3. p(2, X, A, B, C, X_f), \underline{X > 0}, A = 1 \models X_f > 0$$

LU

$$4a. p(3, X, A, B, C, X_f), \underline{X > 0}, A = 1, B = 1 \models X_f > 0$$

LU

$$4b. p(4, X, A, B, C, X_f), \underline{X > 0}, A = 1, B \neq 1 \models X_f > 0$$

(Proof Not Shown)

$$5. p(4, X, A, B, C', X_f), \underline{X > 0}, A = 1, B = 1, \underline{C' = 0} \models X_f > 0$$

LU

$$6a. p(5, X, A, B, C', X_f), X > 0, A = 1, B = 1, \underline{C' = 0}, \underline{C' = 1} \models X_f > 0$$

DP

$$6b. p(\Omega, X, A, B, C', X_f), \underline{X > 0}, A = 1, B = 1, C' = 0 \models X_f > 0$$

7. $\neg \square$

$$8. \underline{X > 0}, A = 1, B = 1, C' = 0, \underline{X = X_f} \models X_f > 0$$

DP

9. $\neg \square$

Automatic: Implementation by manipulating explicit list of constraints.

Intermittent Abstraction

Abstraction is **intermittent** (e.g., at particular program point)

Compared to *abstract interpretation* we gain:

Accuracy: abstract certain points only

Simpler Domain: precision maintained

Efficiency: not computing abstraction everywhere

Intermittent Abstraction

Abstraction is **intermittent** (e.g., at particular program point)

Compared to *abstract interpretation* we gain:

Accuracy: abstract certain points only

Simpler Domain: precision maintained

Efficiency: not computing abstraction everywhere

Can do program verification:

- $\{\phi\} t \{\psi\}$ holds iff $sp(t, \phi) \Rightarrow \psi$ s.t. we unfold $p(\tilde{X})$, ϕ completely and we prove ψ at frontier
- We introduce loop invariant using intermittent abstraction when needed

Intermittent Abstraction Example

Initially $x = s = 0, n \geq 0$.

$\langle 0 \rangle$ **while** $(x < n)$ **do**

$\langle 1 \rangle$ $s := s + x$

$\langle 2 \rangle$ $x := x + 1$

end do

In the proof we replace an assertion

$p(0, 0, 0, N, S_f), N \geq 0 \models S_f = (N^2 - N)/2$ with

$p(0, X, S, N, S_f), S = (X^2 - X)/2, X \leq N, N \geq 0$

$\models S_f = (N^2 - N)/2$

Proving at the side

$X = 0, S = 0, N \geq 0 \models S = (X^2 - X)/2, X \leq N, N \geq 0$

Algorithm for Analysis and Verification

- Tabling of proved assertions
- User-defined *abstraction points* for intermittent abstraction
 - ◆ Loop invariant placement in loop body
 - ◆ Procedure boundary
 - ◆ Fragment boundary
 - ◆ Arbitrary
- Optimized exact propagation between abstraction points via summarization

Our Approach

1. Modeling of programs in CLP: sequential, concurrent, multiprocedure, hardware constraints, arrays, pointer data structures, timed automata, and Statecharts
2. A proof method for general CLP:
 - a. Proving assertions $G \models \psi$, expressing *safety*.
 - b. Abstract *intermittently*.
 - c. Main algorithm for program verification and analysis + dynamic summarization.
3. Proving *recursive assertions* $G \models H$ where H contains CLP atoms.
 - a. Verification of *data structure properties*
 - b. Proving *non-behavioral properties* (e.g. *new classes of symmetries*) for *reduction*
4. Implementation & experiments

Recursive Assertions

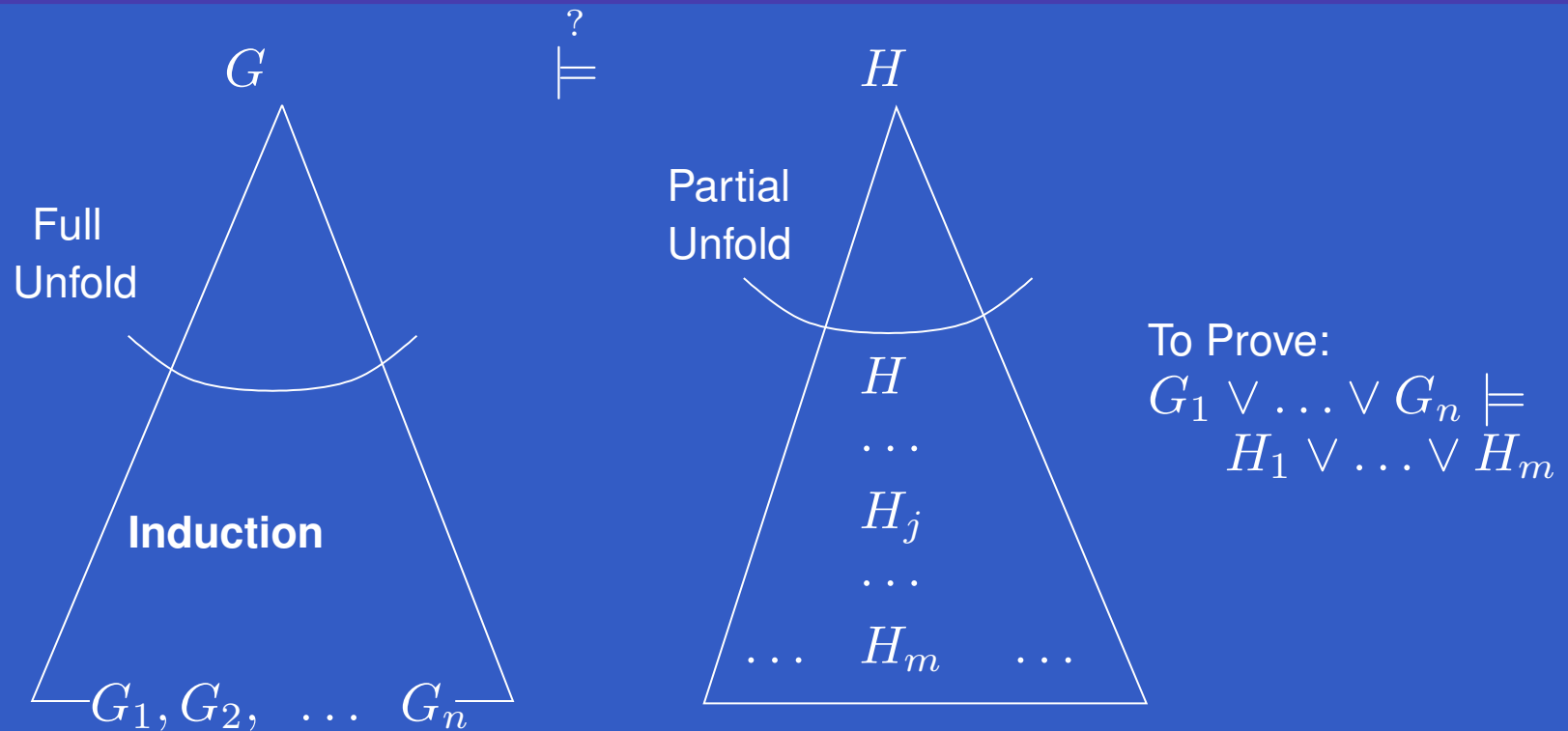
$$G \models H$$

where H contains atoms, not just constraints.

Specifying:

1. Data structure properties for data structure verification
 - General: User-defined data structure predicates
 - Arbitrary level of data structure expressiveness
2. Non-behavioral properties, e.g., symmetry for reduction
 - Handles more cases of symmetry, also commutativity, serializability

Recursive Assertions Proof Technique



Proof holds if $\forall 1 \leq i \leq n, \exists j : 1 \leq j \leq m : G_i \models H_j$

New rule: **RIGHT UNFOLD**

$$\text{(RU)} \quad \frac{\Pi \uplus \{\tilde{A} \vdash G \models H\}}{\Pi \cup \bigcup_{1 \leq i \leq k} \{\tilde{A} \vdash G \models H'\}} \quad H' \in \text{unfold}(H)$$

List Reset

Initially $p \neq 0$.

```
<0> while ( $p \neq 0$ ) do  
<1>    $p \rightarrow val := 0$   
<2>    $p := p \rightarrow next$   
end do
```

$$p(0, H, P, H_f, P_f) :-$$
$$P = 0, p(\Omega, H, P, H_f, P_f).$$
$$p(0, H, P, H_f, P_f) :-$$
$$P \neq 0, p(1, H, P, H_f, P_f).$$
$$p(1, H, P, H_f, P_f) :-$$
$$p(2, \langle H, P, 0 \rangle, P, H_f, P_f).$$
$$p(2, H, P, H_f, P_f) :-$$
$$p(0, H, H[P + 1], H_f, P_f).$$
$$p(\Omega, H, P, H, P).$$

Modeling **HEAP** as **ARRAY**

Data Structure Properties

Specification of nonempty all zero list:

$$\mathit{allz}(H, X, X, \langle H, X, 0 \rangle) \quad :- \quad X \neq 0.$$

$$\mathit{allz}(H, X, Y, \langle H_1, X, 0 \rangle) \quad :- \quad \mathit{allz}(H, T, Y, H_1), \\ H[X + 1] = T, X \neq 0.$$

We can prove:

$$p(0, H, P, H_f, P_f), P \neq 0, P = P_0 \\ \models \mathit{allz}(H, P_0, \mathit{Last}, H_f), P_f = H_f[\mathit{Last} + 1], P_f = 0$$

Array Index Principle (AIP)

Goal G contains $\langle H, I, J \rangle [K]$.

Rewrite G :

$G\{\langle H, I, J \rangle [K] \mapsto J\}$ when $G \Rightarrow I = K$, and

$G\{\langle H, I, J \rangle [K] \mapsto H[K]\}$ when $G \Rightarrow I \neq K$

From [McCarthy 1962].

Leave $\langle H, I, J \rangle [K]$ as is when we can't determine.

In addition,

$$(G \Rightarrow I = J) \Rightarrow H[I] = H[J].$$

Don't always know $G \Rightarrow I = J$.

Separation Principle

H : heap (array), X : root of data structure,

$$a(\langle H, I, J \rangle, X) \equiv a(H, X)$$

when $no_share(H, X, I)$ holds.

This simplifies $\langle H, I, J \rangle$ into H .

Relative Safety

$$p(\dots), \dots \models p(\dots), \dots$$

where p is the predicate of the CLP model

For representing and proving *non-behavioral* properties:

- Symmetry, including new cases
- Serializability
- Commutativity

Used in *reduction*

Non-Behavioral Property

Program m : 2-process mutex, $\langle 2 \rangle = \text{CRITICAL SECTION}$

loop forever

$\langle 0 \rangle$ **await** $(x_2 \neq 1)$ $x_1 := 1$

$\langle 1 \rangle$ **await** $(x_2 \neq 2)$ $x_1 := 2$

$\langle 2 \rangle$ $x_1 := 0$

end loop

loop forever

$\langle 0 \rangle$ $x_2 := 1$

$\langle 1 \rangle$ **await** $(x_1 = 0)$ $x_2 := 2$

$\langle 2 \rangle$ $x_2 := 0$

end loop

Non-Behavioral Property

Program m : 2-process mutex, $\langle 2 \rangle = \text{CRITICAL SECTION}$

loop forever

$\langle 0 \rangle$ **await** $(x_2 \neq 1)$ $x_1 := 1$

$\langle 1 \rangle$ **await** $(x_2 \neq 2)$ $x_1 := 2$

$\langle 2 \rangle$ $x_1 := 0$

end loop

loop forever

$\langle 0 \rangle$ $x_2 := 1$

$\langle 1 \rangle$ **await** $(x_1 = 0)$ $x_2 := 2$

$\langle 2 \rangle$ $x_2 := 0$

end loop

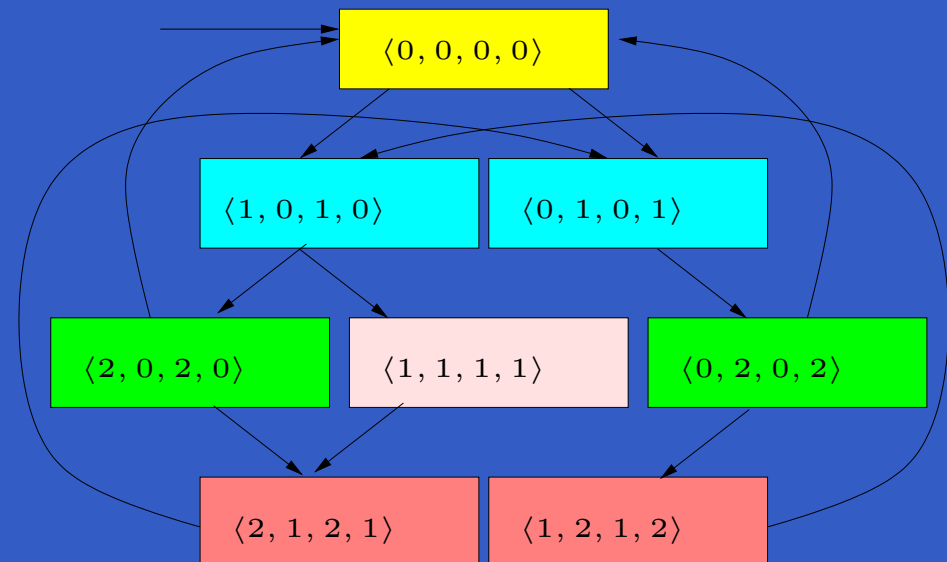
A state =

$\langle L_1, L_2, X_1, X_2 \rangle$

L_1, L_2 = program points of process 1, 2

X_1, X_2 = values of $x_1,$

x_2



Non-Behavioral Property

In the priority mutex program:

State $\langle L_1, L_2, X_1, X_2 \rangle$ is **REACHABLE ONLY IF** state $\langle L_2, L_1, X_2, X_1 \rangle$ is

$\langle 1, 0, 1, 0 \rangle$ is **REACHABLE ONLY IF** $\langle 0, 1, 0, 1 \rangle$ is

$\langle 2, 2, 0, 1 \rangle$ is **REACHABLE ONLY IF** $\langle 2, 2, 1, 0 \rangle$ is

(although $\langle 2, 2, 0, 1 \rangle$ is not reachable)

Non-Behavioral Property

In the priority mutex program:

State $\langle L_1, L_2, X_1, X_2 \rangle$ is **REACHABLE ONLY IF** state $\langle L_2, L_1, X_2, X_1 \rangle$ is

$\langle 1, 0, 1, 0 \rangle$ is **REACHABLE ONLY IF** $\langle 0, 1, 0, 1 \rangle$ is

$\langle 2, 2, 0, 1 \rangle$ is **REACHABLE ONLY IF** $\langle 2, 2, 1, 0 \rangle$ is

(although $\langle 2, 2, 0, 1 \rangle$ is not reachable)

SYMMETRY

$$p(L_1, L_2, X_1, X_2) \models p(L_2, L_1, X_2, X_1)$$

Non-Behavioral Property

In the priority mutex program:

State $\langle L_1, L_2, X_1, X_2 \rangle$ is **REACHABLE ONLY IF** state $\langle L_2, L_1, X_2, X_1 \rangle$ is

$\langle 1, 0, 1, 0 \rangle$ is **REACHABLE ONLY IF** $\langle 0, 1, 0, 1 \rangle$ is

$\langle 2, 2, 0, 1 \rangle$ is **REACHABLE ONLY IF** $\langle 2, 2, 1, 0 \rangle$ is

(although $\langle 2, 2, 0, 1 \rangle$ is not reachable)

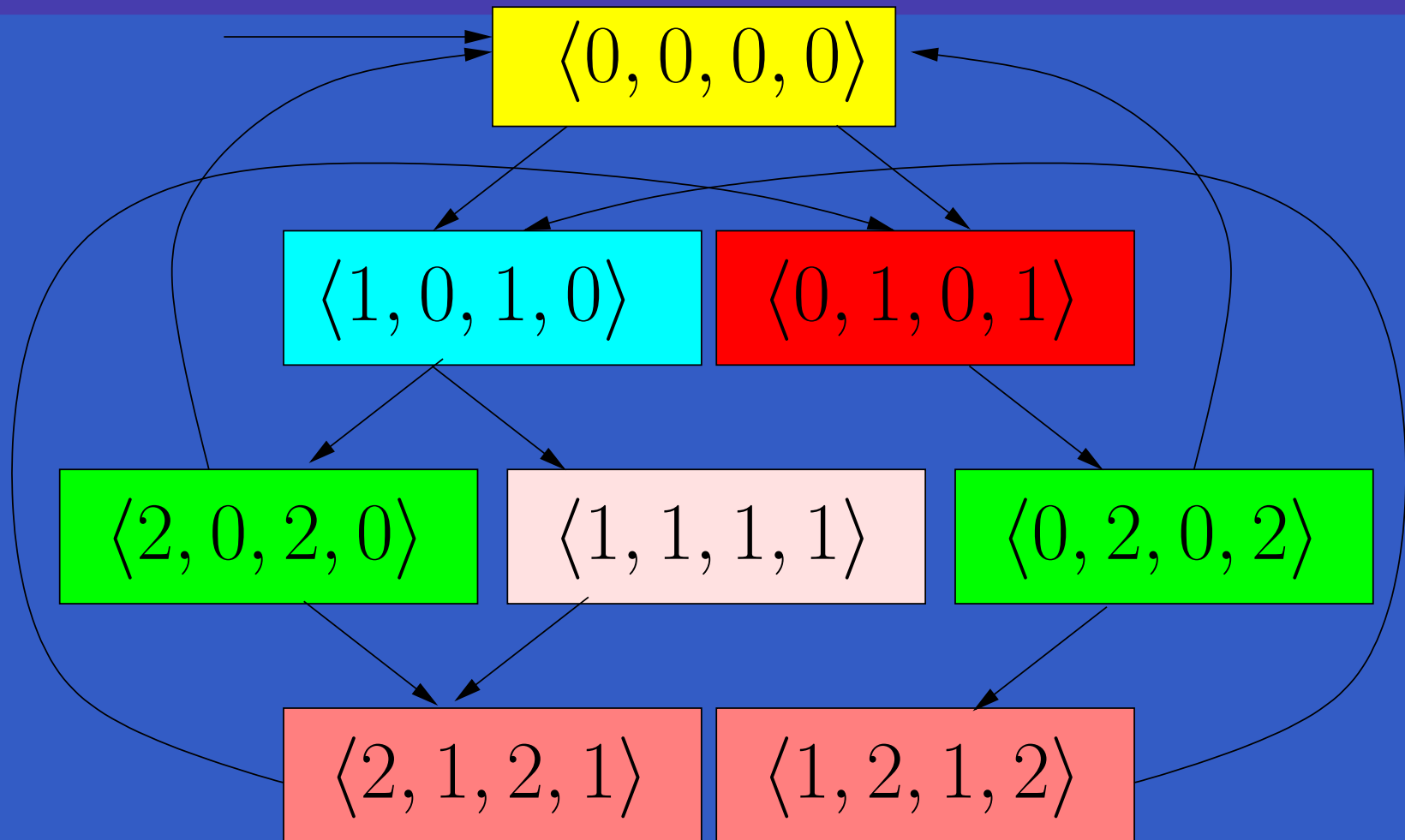
SYMMETRY

$$p(L_1, L_2, X_1, X_2) \models p(L_2, L_1, X_2, X_1)$$

Can also express

SERIALIZABILITY, COMMUTATIVITY

Automorphism of State Graph



[Emerson and Sistla 1993] [Ip and Dill 1993], [Sistla and Godefroid 2004], [Emerson et al. 2000] preserves TL

Automorphism on Collecting Semantics

$\langle 0, 0, 0, 0 \rangle$

$\langle 1, 0, 1, 0 \rangle$

$\langle 0, 1, 0, 1 \rangle$

$\langle 2, 0, 2, 0 \rangle$

$\langle 1, 1, 1, 1 \rangle$

$\langle 0, 2, 0, 2 \rangle$

$\langle 2, 1, 2, 1 \rangle$

$\langle 1, 2, 1, 2 \rangle$

$$p(L_1, L_2, X_1, X_2) \models p(L_2, L_1, X_2, X_1)$$

An *automorphism group* on the collecting semantics.
preserves safety, **can be proven.**

New Approach to Reduction

Previous approaches:

1. Restrict syntax of models
2. Model check with reduction

Examples: **Scalarsets**: Mur_φ : [Ip and Dill 1993], **Identical processes**: SMC [Sistla et al. 2000], RED [Wang 1997], **Priority specification**: PSMC [Sistla and Godefroid 2004]

Our approach:

1. Prove symmetry
2. Prove safety with reduction

Advantages:

1. Handles more cases: cyclic, priority, variable-value
2. Goes beyond symmetry: commutativity, serializability

Our Approach

1. Modeling of programs in CLP: sequential, concurrent, multiprocedure, hardware constraints, arrays, pointer data structures, timed automata, and Statecharts
2. A proof method for general CLP:
 - a. Proving assertions $G \models \psi$, expressing *safety*.
 - b. Abstract *intermittently*.
 - c. Main algorithm for program verification and analysis + dynamic summarization.
3. Proving *recursive assertions* $G \models H$ where H contains CLP atoms.
 - a. Verification of *data structure properties*
 - b. Proving *non-behavioral properties* (e.g. *new classes of symmetries*) for *reduction*
4. Implementation & experiments

- Prototypes implemented in $\text{CLP}(\mathcal{R})$
- Basic implementation is the CLP model itself

Dynamic Summarization Experiments

Problem	No Optim.		Optimized		% Space (Time) Reduction
	Spc.	Time	Spc.	Time	
encoder	494	0.91	252	0.41	48.99% (54.95%)
decoder	344	0.31	38	0.02	88.95% (93.55%)
sqrt	923	4.25	91	0.38	90.14% (91.06%)
qurt	1104	14.47	273	2.52	75.27% (82.58%)
janne_complex	1517	17.93	410	2.13	72.97% (88.12%)
bubblesort (5)	1034	94.49	58	0.58	94.39% (99.39%)
binary	381	10.88	170	4.00	55.38% (63.24%)
bubblesort (4)					

Time in seconds

Proving Non-Behavioral Properties

Problem	Assertions #	Left Size	Induction	Time (s)
<i>Bakery-4</i>	3	147	147	0.30
<i>Bakery-5</i>	4	424	424	2.11
<i>Bakery-6</i>	5	1145	1145	13.23
<i>Bakery-7</i>	6	3486	3486	81.38
<i>Priority</i>	1	30	17	0.05
<i>Szymanski-2</i>	8	1276	71	13.21
<i>Philosopher-3</i>	1	19	19	0.01
<i>Philosopher-4</i>	1	25	25	0.01
<i>Prod/Cons-4</i>	2	303	134	11.37
<i>Prod/Cons-5</i>	2	1487	619	70.84

Proving Safety

Problem	No Reduction		W/ Reduction	
	Nodes	Time	Nodes	Time
<i>Bakery-4</i>	4624	6.60	191	0.20
<i>Bakery-5</i>	∞	∞	677	2.88
<i>Bakery-6</i>	∞	∞	2569	49.08
<i>Bakery-7</i>	∞	∞	11865	1052.32
<i>Szymanski-2</i>	240	0.08	84	0.02
<i>Szymanski-3</i>	10883	35.43	3176	2.91
<i>Philosopher-3</i>	882	0.51	553	0.30
<i>Philosopher-4</i>	4293	27.77	2783	9.67
<i>Prod/Cons-10</i>	664	0.10	171	0.02
<i>Prod/Cons-20</i>	2314	1.90	331	0.04

Publications

1. J. Jaffar, A. E. Santosa, R. Voicu. A CLP Method for Compositional and Intermittent Predicate Abstraction. VMCAI '06
2. J. Jaffar, A. E. Santosa, R. Voicu. Relative Safety. VMCAI '06.
3. J. Jaffar, A. E. Santosa, R. Voicu. Modeling Systems in CLP. ICLP '05.
4. J. Jaffar, A. E. Santosa, R. Voicu. A CLP Proof Method for Timed Automata. RTSS '04.

Ongoing (drafts available):

1. Dynamic Summarizations.
2. General-Purpose Assertions for Data Structures.
3. An Integrated Approach to Proving Liveness.

Conclusion

- Modeling of programs in CLP
- Assertions $G \models \psi$, or $G \models H$
- A proof method for general CLP
 - ◆ Intermittent abstraction:
 - “Abstract interpretation” but intermittent
 - “Program verification” but with automatic propagation
 - ◆ Main algorithm for program verification and analysis + dynamic summarization.
- Verification of data structures
- Reduction via unusual symmetries

- Liveness: proof tree also represents traces, hence can be used to reason about liveness
- Loop invariant discovery:
 - ◆ Use *context* of the loop to discover invariant.
 - ◆ Keep constraints in context that are invariant in 1 iteration.
- Resource analysis (optimization)
- Modeling formal specifications for reasoning and simulation