# Safety Analysis of Systems

Aaron R. Bradley

Stanford University

# Why Analyze Systems?

- Two trends:
  - increasing prominence in controlling and decision-making roles
  - rising complexity (multi-core processors)

# Why Analyze Systems?

- Two trends:
  - increasing prominence in controlling and decision-making roles
  - rising complexity (multi-core processors)

Demand for guarantees

# Why Analyze Systems?

- Two trends:
  - increasing prominence in controlling and decision-making roles
  - rising complexity (multi-core processors)

  Demand for guarantees
- Methods have other applications:
  - Study other (natural & engineered) systems.
  - Characterize decidability & complexity.
  - Provide tools (constraint solvers, static analyses).

# What Comprises Verification?

```
int[] BubbleSort(int[] a₀, int ℓ, int u) {
   int[] a := a₀;
   for
      (int i := u; i > ℓ; i := i − 1)
      for
         (int j := ℓ; j < i; j := j + 1)
         if (a[j] > a[j + 1]) {
            int t := a[j];
            a[j] := a[j + 1];
            a[j + 1] := t;
         }
   return a;
}
```

# What Comprises Verification?

- Implementation

```
int[] BubbleSort(int[] a0, int ℓ, int u) {
  {int[] a := a0;
  for
    (int m := u; m > ℓ; m := m − 1)
    for
      (int n := ℓ; n < m; n := n + 1)
      if (a[n] > a[n + 1]) {
        int t := a[n];
        a[n] := a[n + 1];
        a[n + 1] := t;
      }
  return a;
}
```

# What Comprises Verification?

- Implementation
- Specification

```
@pre 0 ≤ ℓ, u < |a₀|
@post ∀i, j. ℓ ≤ i ≤ j ≤ u → rv[i] ≤ rv[j]
        ∧ |rv| = |a₀|
        ∧ ∀i. 0 ≤ i < ℓ → rv[i] = a₀[i]
        ∧ ∀i. u < i < |rv| → rv[i] = a₀[i]
int[] BubbleSort(int[] a₀, int ℓ, int u) {
  {int[] a := a₀;
  for
    (int m := u; m > ℓ; m := m − 1)
    for
      (int n := ℓ; n < m; n := n + 1)
      if (a[n] > a[n + 1]) {
        int t := a[n];
        a[n] := a[n + 1];
        a[n + 1] := t;
      }
  return a;
}
```

# What Comprises Verification?

- Implementation
- Specification

- Strengthen

  invariant generation

  [BMS05c, BM06, BM07]

$$
\begin{array}{l}
\text{@pre } 0 \leq \ell, u < |a_0| \\
\text{@post } \forall i, j.\ \ell \leq i \leq j \leq u \ \rightarrow\ rv[i] \leq rv[j] \\
\qquad\quad \wedge\ |rv| = |a_0| \\
\qquad\quad \wedge\ \forall i.\ 0 \leq i < \ell \ \rightarrow\ rv[i] = a_0[i] \\
\qquad\quad \wedge\ \forall i.\ u < i < |rv| \ \rightarrow\ rv[i] = a_0[i] \\
\texttt{int}[]\ \text{BubbleSort}(\texttt{int}[]\ a_0,\ \texttt{int}\ \ell,\ \texttt{int}\ u)\ \{ \\
\quad \{\texttt{int}[]\ a := a_0; \\
\quad \texttt{for} \\
\qquad @L_1 : \left[\begin{array}{l}
i \leq u \ \wedge\ |a| = |a_0| \\
\wedge\ \forall i, j.\ m \leq i \leq j \leq u \ \rightarrow\ a[i] \leq a[j] \\
\wedge\ \forall i, j.\ \ell \leq i \leq m < j \leq u \ \rightarrow\ a[i] \leq a[j] \\
\wedge\ \forall i.\ 0 \leq i < \ell \ \rightarrow\ a[i] = a_0[i] \\
\wedge\ \forall i.\ u < i < |a| \ \rightarrow\ a[i] = a_0[i]
\end{array}\right] \\
\quad\ (\texttt{int}\ m := u;\ m > \ell;\ m := m - 1) \\
\quad\ \texttt{for} \\
\qquad\quad @L_2 : \left[\begin{array}{l}
\ell < i \leq u \ \wedge\ \ell \leq j \leq i \ \wedge\ |a| = |a_0| \\
\wedge\ \forall i, j.\ m \leq i \leq j \leq u \ \rightarrow\ a[i] \leq a[j] \\
\wedge\ \forall i, j.\ \ell \leq i \leq m < j \leq u \ \rightarrow\ a[i] \leq a[j] \\
\wedge\ \forall i.\ \ell \leq i < n \ \rightarrow\ a[i] \leq a[n] \\
\wedge\ \forall i.\ 0 \leq i < \ell \ \rightarrow\ a[i] = a_0[i] \\
\wedge\ \forall i.\ u < i < |a| \ \rightarrow\ a[i] = a_0[i]
\end{array}\right] \\
\qquad (\texttt{int}\ n := \ell;\ n < m;\ n := n + 1)
\end{array}
$$

# What Comprises Verification?

- Implementation
- Specification
- Strengthen

  invariant generation

  [BMS05c, BM06, BM07]
- Check argument

  decision procedures

  [BMS06]

$$
\begin{array}{l}
\text{@pre } 0 \le \ell, u < |a_0| \\
\text{@post } \forall i, j.\ \ell \le i \le j \le u \ \rightarrow\ rv[i] \le rv[j] \\
\qquad\quad \wedge\ |rv| = |a_0| \\
\qquad\quad \wedge\ \forall i.\ 0 \le i < \ell \ \rightarrow\ rv[i] = a_0[i] \\
\qquad\quad \wedge\ \forall i.\ u < i < |rv| \ \rightarrow\ rv[i] = a_0[i] \\
\texttt{int}[]\ \text{BubbleSort}(\texttt{int}[]\ a_0,\ \texttt{int}\ \ell,\ \texttt{int}\ u)\ \{ \\
\quad \{\texttt{int}[]\ a := a_0; \\
\quad \texttt{for} \\
\qquad @L_1 : \left[\begin{array}{l}
i \le u \ \wedge\ |a| = |a_0| \\
\wedge\ \forall i, j.\ m \le i \le j \le u \ \rightarrow\ a[i] \le a[j] \\
\wedge\ \forall i, j.\ \ell \le i \le m < j \le u \ \rightarrow\ a[i] \le a[j] \\
\wedge\ \forall i.\ 0 \le i < \ell \ \rightarrow\ a[i] = a_0[i] \\
\wedge\ \forall i.\ u < i < |a| \ \rightarrow\ a[i] = a_0[i]
\end{array}\right] \\
\qquad (\texttt{int}\ m := u;\ m > \ell;\ m := m - 1) \\
\qquad \texttt{for} \\
\qquad\quad @L_2 : \left[\begin{array}{l}
\ell < i \le u \ \wedge\ \ell \le j \le i \ \wedge\ |a| = |a_0| \\
\wedge\ \forall i, j.\ m \le i \le j \le u \ \rightarrow\ a[i] \le a[j] \\
\wedge\ \forall i, j.\ \ell \le i \le m < j \le u \ \rightarrow\ a[i] \le a[j] \\
\wedge\ \forall i.\ \ell \le i < n \ \rightarrow\ a[i] \le a[n] \\
\wedge\ \forall i.\ 0 \le i < \ell \ \rightarrow\ a[i] = a_0[i] \\
\wedge\ \forall i.\ u < i < |a| \ \rightarrow\ a[i] = a_0[i]
\end{array}\right] \\
\qquad\quad (\texttt{int}\ n := \ell;\ n < m;\ n := n + 1)
\end{array}
$$

## What Comprises Verification?

- Imp
- Sp
- Str
  inva
  [BM
- Ch
  dec
  [BM

Contributions:

- Decision procedures:
  - theory of arrays [BMS06]
- Property-guided invariant generation:
  - clauses (hardware) [BM07]
  - linear/polynomial inequalities (software) [BM06]
  - linear inequalities of integers (mixed) [BMS05c]
- Termination analysis [BMS05d, BMS05b, BMS05a, BMS05c]

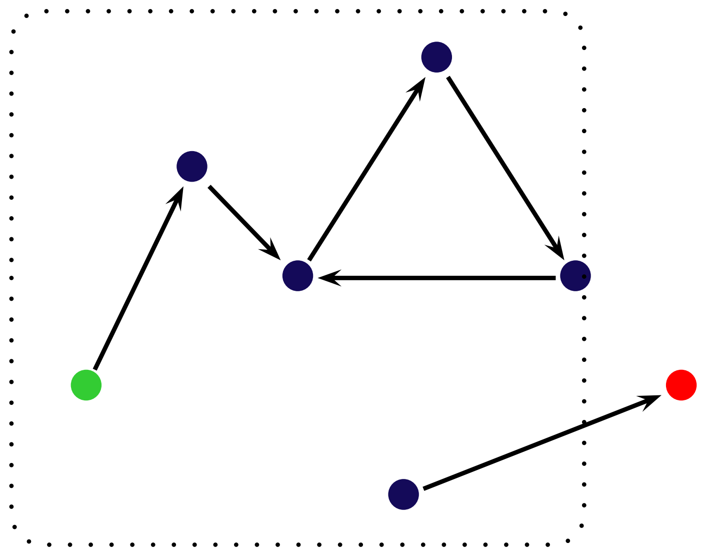
$j]$

$a[j]$

# Invariant

Invariant:

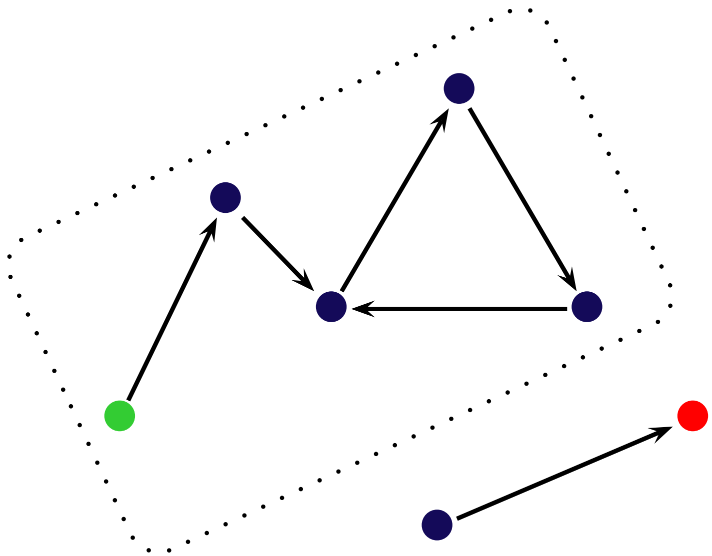
- Over-approximates reachable states
- Represented as formula in practice



# Invariant

Invariant:

- Over-approximates reachable states
- Represented as formula in practice



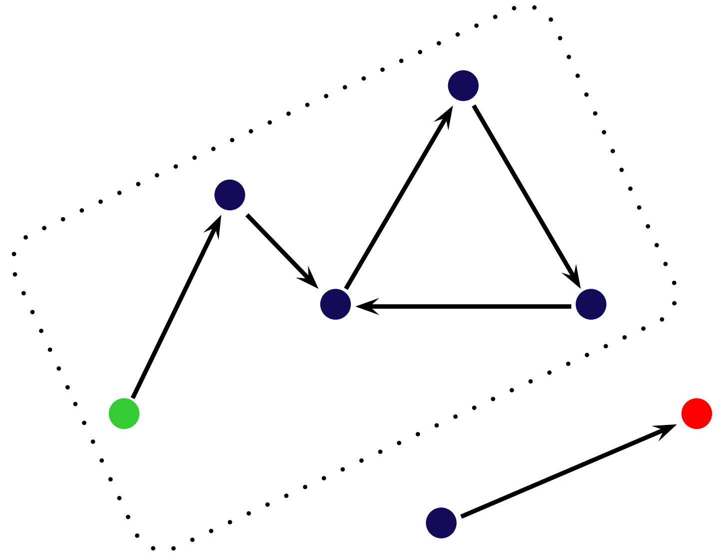
Inductive Invariant:

- Initiation: Includes initial states
- Consecution: Closed under transitions

# Invariant

Invariant:

- Over-approximates reachable states
- Represented as formula in practice



Inductive Invariant:

- Initiation: Includes initial states
- Consecution: Closed under transitions

Based on mathematical induction:

- Base case: Initiation
- Inductive case: Consecution

# Formally...

Transition system $\langle \overline{x}, \Theta, \rho \rangle$:

- $\Theta[\overline{x}]$: initial states $\qquad x \geq 0$
- $\rho[\overline{x}, \overline{x}']$: transition relation $\qquad x' = x + 1 \;\vee\; x' = 0$

# Formally...

## Transition system $\langle \overline{x}, \Theta, \rho \rangle$:

- $\Theta[\overline{x}]$: initial states $\quad x \geq 0$
- $\rho[\overline{x}, \overline{x}']$: transition relation $\quad x' = x + 1 \ \vee \ x' = 0$

## Inductive invariant $\varphi$:

- $\Theta \ \Rightarrow \ \varphi$ (initiation)
- $\varphi \wedge \rho \ \Rightarrow \ \varphi'$ (consecution)

1. $x \geq 0 \ \Rightarrow \ x \geq 0$
2. $x \geq 0 \ \wedge \ (x' = x + 1 \ \vee \ x' = 0) \ \Rightarrow \ x' \geq 0$

# Formally. . .

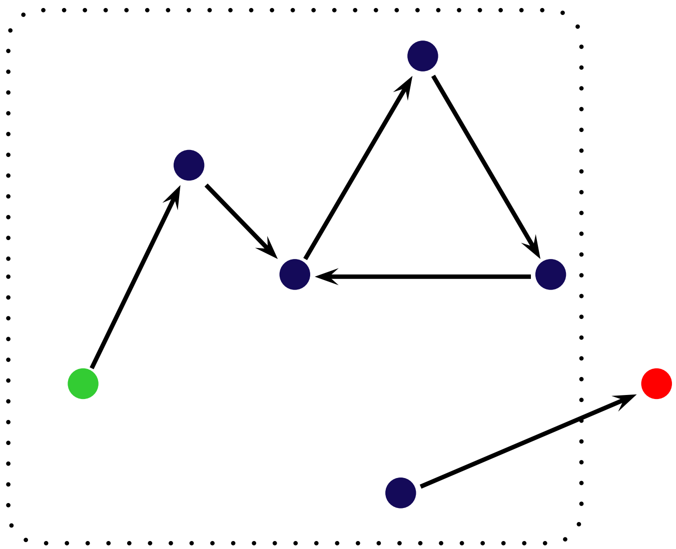
Given $\langle \overline{x}, \Theta, \rho \rangle$ and property $\Pi$.
Goal: Prove that $\Pi$ is invariant.

# Formally. . .

Given $\langle \overline{x}, \Theta, \rho \rangle$ and property $\Pi$.
Goal: Prove that $\Pi$ is invariant.

Inductive method:
Find strengthening assertion $\chi$ such that

- $\Theta \;\Rightarrow\; \Pi \wedge \chi$
- $\Pi \wedge \chi \wedge \rho \;\Rightarrow\; \Pi' \wedge \chi'$

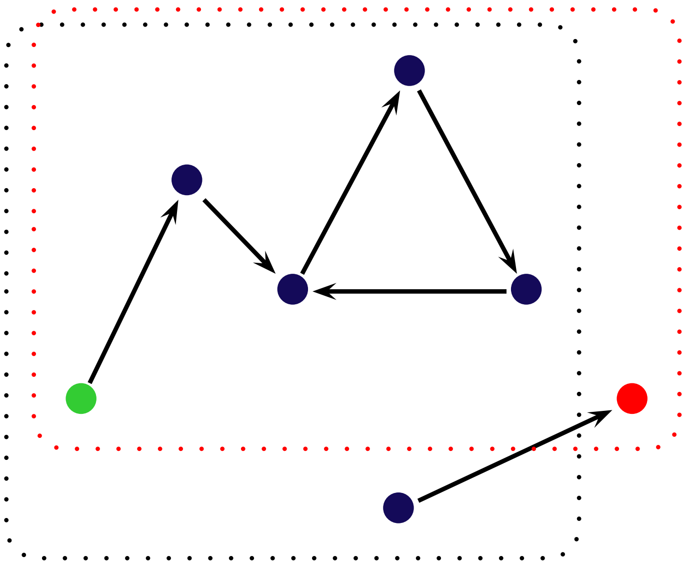


# Formally...

Given $\langle \overline{x}, \Theta, \rho \rangle$ and property $\Pi$.
Goal: Prove that $\Pi$ is invariant.

Inductive method:
Find strengthening assertion $\chi$ such that

- $\Theta \;\Rightarrow\; \Pi \wedge \chi$
- $\Pi \wedge \chi \wedge \rho \;\Rightarrow\; \Pi' \wedge \chi'$



# Challenges

1. Prove initiation and consecution automatically (especially for infinite-state systems)
   $\Rightarrow$ decision procedures
2. Discover strengthening invariants automatically
   $\Rightarrow$ invariant generation procedures

# Outline

1. Introduction
2. Decision Procedure for Arrays
3. Invariant Generation of Clauses
4. Course: The Calculus of Computation
5. Directions for Research

# Theory of Arrays: Context

Important theory with long history:

- axioms [McC62]; DP for QFF [Kin69]
- Early 1980s
  - sorting [Mat81, Jaf81, SJ80]
  - (restricted) permutation [SJ80]
- 2001: QFF of extensional theory [SBDL01]

# Theory of Arrays: Context

Important theory with long history:

- axioms [McC62]; DP for QFF [Kin69]
- Early 1980s
  - sorting [Mat81, Jaf81, SJ80]
  - (restricted) permutation [SJ80]
- 2001: QFF of extensional theory [SBDL01]

Questions:

1. Unifying decidable fragment?
2. Upper bounds on decidability?

Goal: Combination theories (for indices & elements)

# Decidability Landscape

“unnatural extensions”

sorting extensionality

quantifier-free

undecidable

?

decidable

[BMS06]

# Arrays

## Theory $T_{\mathsf{A}}$

$$\Sigma_{\mathsf{A}} : \ \{a[i],\ a\langle i \triangleleft e \rangle,\ =\}$$

- $\forall a, v, i, j.\ i = j \ \rightarrow\ a\langle i \triangleleft v\rangle[j] = v$ (ROW 1)
- $\forall a, v, i, j.\ i \neq j \ \rightarrow\ a\langle i \triangleleft v\rangle[j] = a[j]$ (ROW 2)

# Arrays

## Theory $T_\mathsf{A}$

$$\Sigma_\mathsf{A}: \ \{a[i],\ a\langle i \triangleleft e\rangle,\ =\}$$

- $\forall a, v, i, j.\ i = j \ \rightarrow \ a\langle i \triangleleft v\rangle[j] = v$ (ROW 1)
- $\forall a, v, i, j.\ i \neq j \ \rightarrow \ a\langle i \triangleleft v\rangle[j] = a[j]$ (ROW 2)

## Combination: index & element theories.

$$\begin{aligned} &\mathsf{sorted}(a\langle i \triangleleft 0\rangle\langle j \triangleleft 1\rangle, i, j) \\ &\ \wedge\ \mathsf{sorted}(a\langle i \triangleleft 2\rangle\langle j \triangleleft 3\rangle, i, j) \\ &\ \wedge\ i + 1 < j \end{aligned}$$

$$\mathsf{sorted}(a, \ell, u) \overset{\mathsf{def}}{=} \forall i, j.\ \ell \leq i \leq j \leq u \ \rightarrow \ a[i] \leq a[j]$$

# Useful Properties

- Equality between arrays [SBDL01]:

$$\forall i.\ a[i] = b[i]$$

- Bounded equality:

$$\forall i.\ \ell \leq i \leq u \ \rightarrow\ a[i] = b[i]$$

- (Bounded) universal properties:

$$\forall i.\ \ell \leq i \leq u \ \rightarrow\ F[\overline{a}, i]$$

# Useful Properties

For sorting [Mat81, Jaf81]:

- sorted$(a, \ell, u)$:

$$\forall i, j.\ \ell \leq i \leq j \leq u \ \rightarrow\ a[i] \leq a[j]$$

- $a[\ell_1..u_1] < a[\ell_2..u_2]$

$$\forall i, j.\ \ell_1 \leq i \leq u_1 < \ell_2 \leq j \leq u_2 \ \rightarrow\ a[i] < b[j]$$

# What is Decidable?

## Array Property Fragment

- QFF + universal quantification of indices, with restrictions.
- Includes mentioned properties.

# What is Decidable?

## Array Property Fragment

- QFF + universal quantification of indices, with restrictions.
- Includes mentioned properties.

Why? Given $F$, there is a finite set of symbolic indices $\mathcal{I}$ s.t.:

- each $\ell \in \mathcal{I}$ occurs in $F$;
- instantiating universal quantifiers over $\mathcal{I}$ produces an equisatisfiable formula.

NP-complete for bounded stack of $\forall$.

# What is Undecidable?

Possible extensions:

- One more quantifier alternation.

$$\forall i.\ \exists j.\ a[j] > a[i]$$

- Arithmetic on universally quantified indices.

$$\forall i.\ a[i+1] > a[i]$$

- Nested reads.

$$\forall i.\ a[b[i]] > a[i]$$

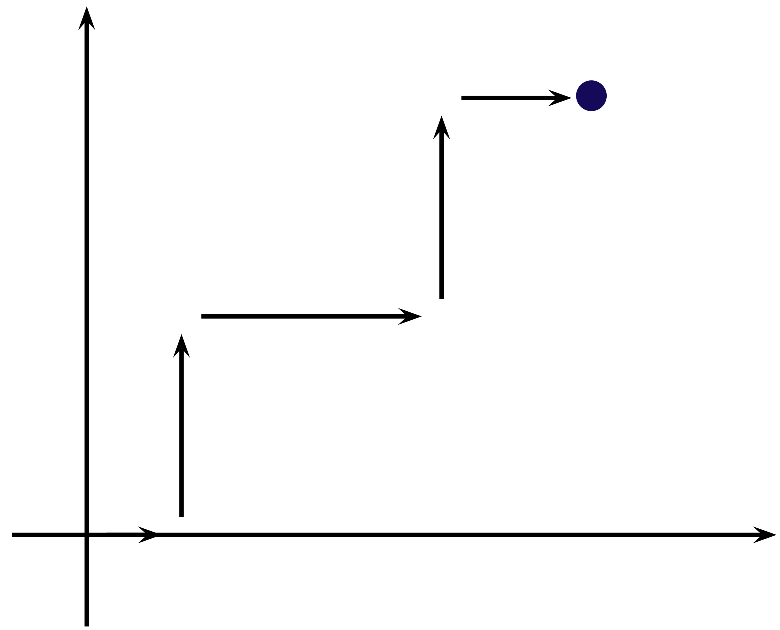
- Other technical relaxations.

# What is Undecidable?

- From undecidability of Diophantine equations.

# What is Undecidable?

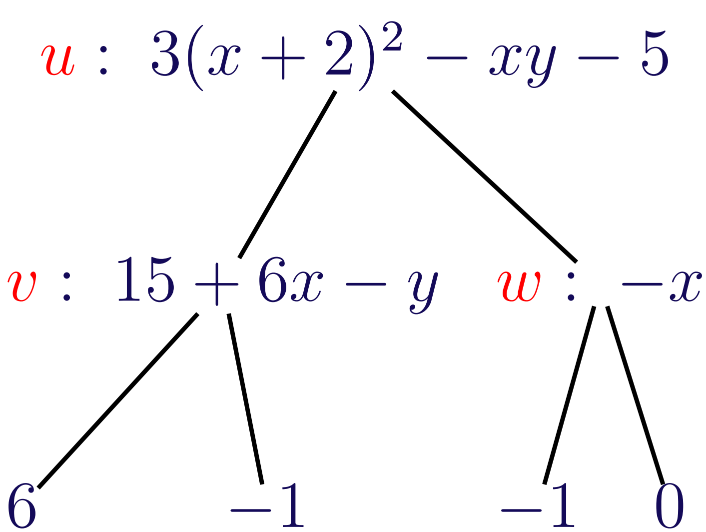
- From undecidability of Diophantine equations.
- Encode traces of walks from origin.
  Is there a path that leads to a solution?



# What is Undecidable?

- From undecidability of Diophantine equations.
- Encode traces of walks from origin.
  Is there a path that leads to a solution?

$$3(x+2)^2 - xy \stackrel{?}{=} 5$$




- $x = y = 0$:
  $u = 7,\ v = 15,\ w = 0$
- $x := x + 1$:
  $(u, v, w) := (u + v, v + 6, w - 1)$
- $y := y + 1$:
  $(u, v, w) := (u + w, v - 1, w)$

# So What is Undecidable?

Anything that can encode a trace:

- One more quantifier alternation.
- Arithmetic on universally quantified indices.
- Nested reads.
- Add permutation predicate.
- Other technical relaxations.
  Interesting: express injectivity

$$\forall i, j.\ i < j \ \rightarrow \ a[i] \neq a[j]$$

Open: uninterpreted indices.

# Additional Quantifier Alternation

- One array per introduced variable ($u$, $v$, $w$).
- One sum array $s$.
- $\Theta(z)$: simulate at origin
- $\rho(z_1, z_2)$: relate positions
  1. possibly move one step in one direction
  2. $s(z_2) = s(z_1) - u$
- $\Pi(z)$: $s(z) > 0$

$$\exists z, u, v, w, s.\ \forall i.\ \exists j.\ \Theta(z)\ \wedge\ \rho(i, j)\ \wedge\ \Pi(i)$$

Satisfiable iff (finite) path to solution iff solution exists.

# Future Directions

- Other data structures:
  - hashtables [BMS06]
  - collections
  - recursive data structures
- More heuristics for speed
  avoid full instantiation if possible
- Application-driven heuristics
  Ex: scientific programming
- Vectorized loops

# Vectorized Loops

```
@pre ⊤
@post sorted(rv, 0, |rv| − 1)
int[] BubbleSort(int[] a₀) {
  int[] a := a₀;
  for
    @ [ −1 ≤ i < |a|
        ∧ partitioned(a, 0, i, i + 1, |a| − 1)
        ∧ sorted(a, i, |a| − 1) ]
    (int i := |a| − 1; i > 0; i := i − 1) {
    int k := maxi(a, 0, i, ≥);
    a[k] :=: a[i];
  }
  return a;
}
```

# Vectorized Loops

@pre $\top$
@post sorted$(rv, 0, |rv| - 1)$
`int[]` BubbleSort(`int[]` $a_0$) $\{$
  `int[]` $a$ := $a_0$;
  `for`
    $@\left[\begin{array}{l} -1 \le i < |a| \\ \wedge\ \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \wedge\ \text{sorted}(a, i, |a|-1) \end{array}\right]$
    (`int` $i$ := $|a| - 1$; $i > 0$; $i$ := $i - 1$) $\{$
    `int` $k$ := maxi$(a, 0, i, \ge)$;
    $a[k]$ :=: $a[i]$;
  $\}$
  `return` $a$;
$\}$

$$(\forall j.\ 0 \le j \le i \ \rightarrow\ a[k] \ge a[j])$$
$$\wedge\ \ a' = a\langle k \triangleleft a[i]\rangle\langle i \triangleleft a[k]\rangle$$

# Vectorized Loops

Goal: Reduce annotation burden.

Encode array/collection operations in $\exists\forall$ fragments:

$$\begin{array}{ll} \mathsf{map}(a, f) & \mathsf{filter}(a, p^{(1)}) \\ \mathsf{choose}(a) & \mathsf{max}(a, q^{(2)}) \\ \mathsf{forall}(a, p^{(1)}) & \mathsf{exists}(a, p^{(1)}) \end{array}$$

- Recognize vectorizable loops.
  - Characterization of decidability
  - Parameterized systems [ES93, EN95, PRZ01]
  - Analyses from vectorizing compilers
- Decide VCs quickly.

# Outline

# Finite-State Systems

Transition system: $\langle \overline{x}, \Theta, \rho \rangle$
Safety property: $\Pi$

- $\overline{x}$ are Boolean
- $\Theta$, $\rho$, $\Pi$ are propositional formulae

# Finite-State Systems

Transition system: $\langle \overline{x}, \Theta, \rho \rangle$
Safety property: $\Pi$

- $\overline{x}$ are Boolean
- $\Theta$, $\rho$, $\Pi$ are propositional formulae

Goal: Given $\Pi$, discover $\chi$ such that

- $\Theta \;\Rightarrow\; \chi$ (initiation)
- $\Pi \wedge \chi \wedge \rho \;\Rightarrow\; \Pi' \wedge \chi'$ (consecution)

# Context

Standard: model checking
[CES86, QS82, BCM$^{+}$92, BCCZ99]

- monolithic: one detailed inductive invariant
- abstraction: use coarser transition relation

Our approach:

- incremental: many weak inductive invariants
- abstraction: violating states seeds for invariants

# Directed Generation

Idea: Suppose induction fails:

$$\Pi \land \chi_i \land \rho \ \not\Rightarrow \ \Pi'$$

Counterexample:

$$s : \ x_1 \land \neg x_2 \land \neg x_3 \land \cdots \land x_n$$

Negation is clause:

$$\neg s : \ \neg x_1 \lor x_2 \lor x_3 \lor \cdots \lor \neg x_n$$

# Directed Generation

Idea: Suppose induction fails:

$$\Pi \wedge \chi_i \wedge \rho \;\not\Rightarrow\; \Pi'$$

Counterexample:

$$s : \;\; x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \cdots \wedge x_n$$

Negation is clause:

$$\neg s : \;\; \neg x_1 \vee x_2 \vee x_3 \vee \cdots \vee \neg x_n$$

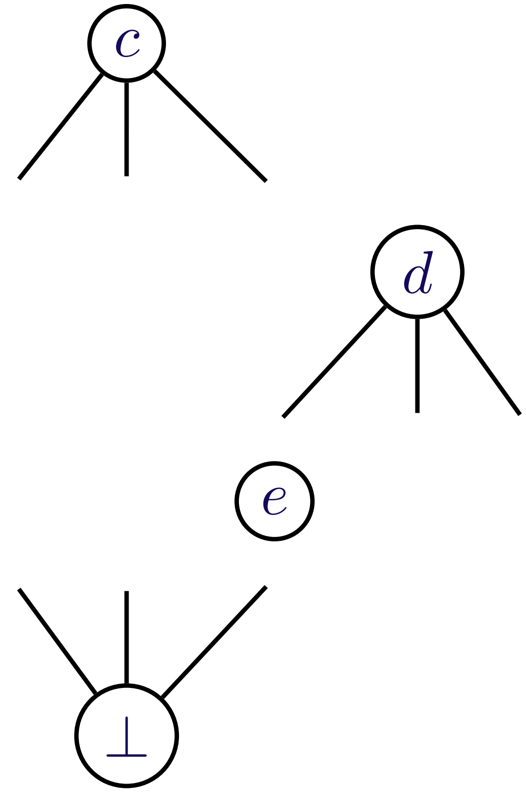
Goal: Find minimal inductive subclause $d \sqsubseteq \neg s$

$$d : \;\; x_2 \vee x_{13} \vee \neg x_{41}$$

# Finding a MIC

$L_c$: Lattice of subclauses of $c$.

Given $c$:

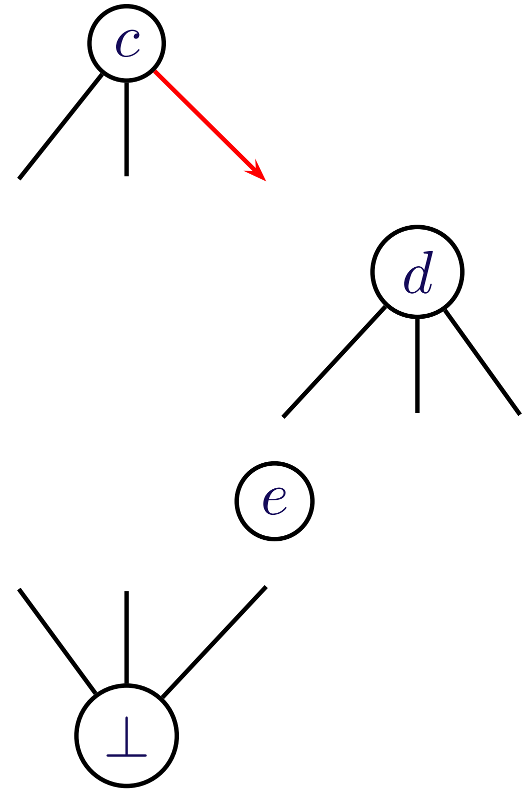


# Finding a MIC

$L_c$: Lattice of subclauses of $c$.

Given $c$:

1. Descend $L_c$ in search of largest inductive $d \sqsubseteq c$.

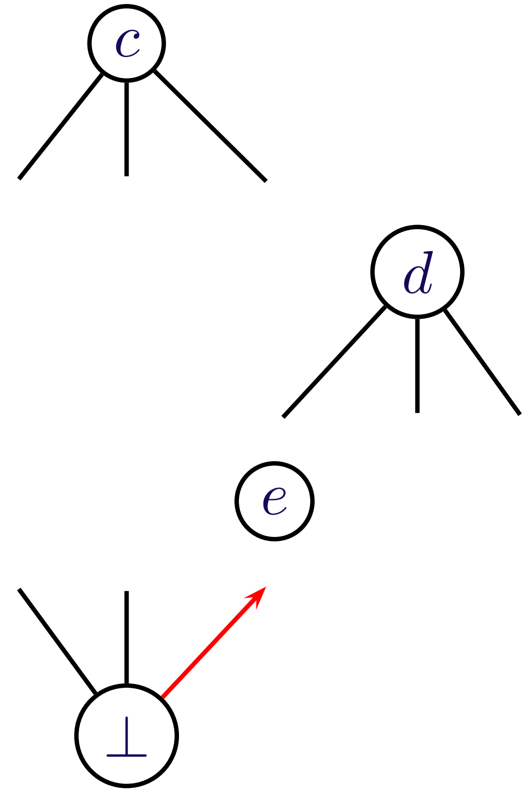


# Finding a MIC

$L_c$: Lattice of subclauses of $c$.

Given $c$:

1. Descend $L_c$ in search of largest inductive $d \sqsubseteq c$.
2. Ascend $L_d$ in search of "small" inductive $e \sqsubseteq d$.

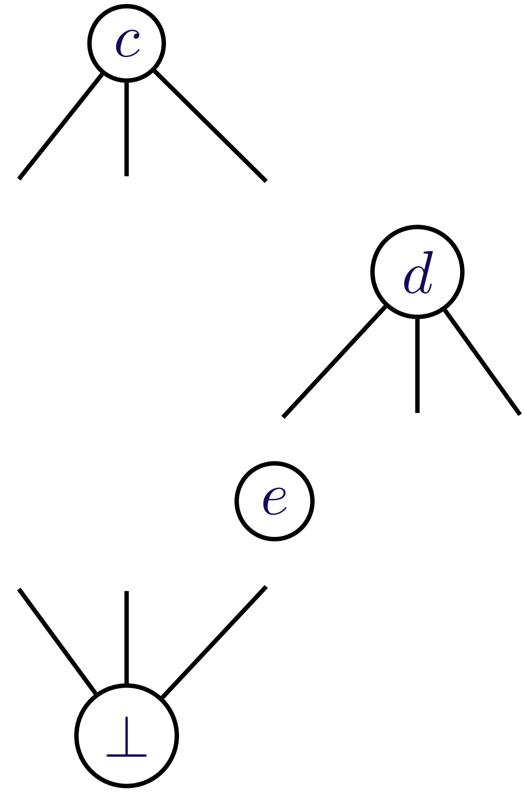


# Finding a MIC

$L_c$: Lattice of subclauses of $c$.

Given $c$:

1. Descend $L_c$ in search of largest inductive $d \sqsubseteq c$.
2. Ascend $L_d$ in search of "small" inductive $e \sqsubseteq d$.
3. Drop literal of $e$ and recurse.



# Descending

Goal: Compute largest $d \sqsubseteq c$ s.t.

$$d \;\Rightarrow\; \mathsf{wp}(d,\; \rho)$$

# Descending

Goal: Compute largest $d \sqsubseteq c$ s.t.

$$d \;\Rightarrow\; \mathsf{wp}(d,\ \rho)$$

Slow method:

- Compute $d_1 \sqsubseteq c$: $\;d_1 \;\Rightarrow\; \mathsf{wp}(c,\ \rho)$
- Compute $d_2 \sqsubseteq d_1$: $d_2 \;\Rightarrow\; \mathsf{wp}(d_1,\ \rho)$
- ...

Compute $d_{i+1} \;\Rightarrow\; \mathsf{wp}(d_i,\ \rho)$ literal-by-literal.
Cost: $O(n^2)$ SAT problems. $O(n)$ per iteration.

# Descending

Goal: Compute largest $d \sqsubseteq c$ s.t.

$$d \;\Rightarrow\; \mathsf{wp}(d,\ \rho)$$

Fast method:

- Find counterexample to $c \;\Rightarrow\; \mathsf{wp}(c,\ \rho)$
  $c_1$: drop satisfied literals of $c$
- Find counterexample to $c_1 \;\Rightarrow\; \mathsf{wp}(c_1,\ \rho)$
- $\ldots$

Converges to same solution.
Cost: $O(n)$ SAT problems. $O(1)$ per iteration.

# Ascending: Finding Consequences

Goal: Find minimal $d \sqsubseteq c$ s.t. $F \Rightarrow d$.

More general: Given monotonic predicate $p$ over sets

$$S' \subseteq S \;\Rightarrow\; p(S') \rightarrow p(S)$$

find minimal subset $S_0 \subseteq S$ s.t. $p(S_0)$.

# Ascending: Finding Consequences

Goal: Find minimal $d \sqsubseteq c$ s.t. $F \Rightarrow d$.

More general: Given monotonic predicate $p$ over sets

$$S' \subseteq S \;\Rightarrow\; p(S') \rightarrow p(S)$$

find minimal subset $S_0 \subseteq S$ s.t. $p(S_0)$.

Lower bound: $\Omega\left(|S_0| \text{ lg } \frac{|S|-|S_0|}{|S_0|}\right)$ $p$-queries.

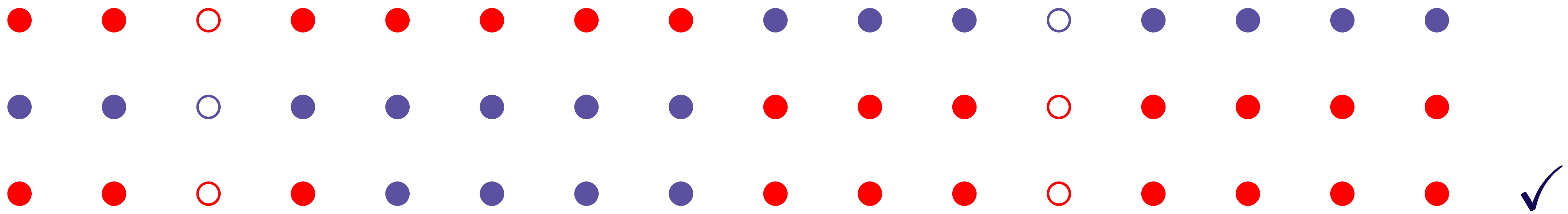
Yet the typical solution is linear in $|S|$!

# Finding Consequences

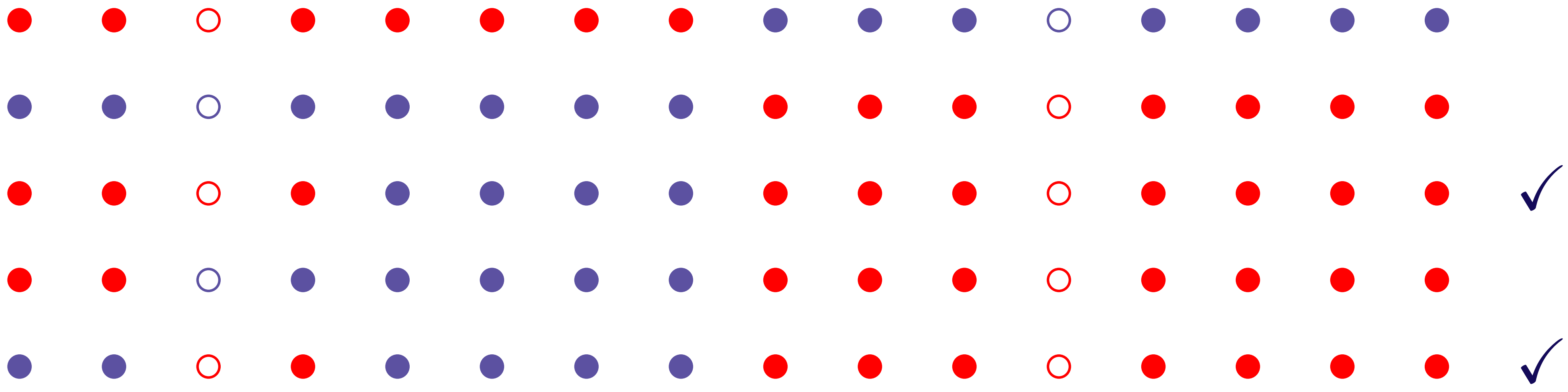
# Finding Consequences

# Finding Consequences

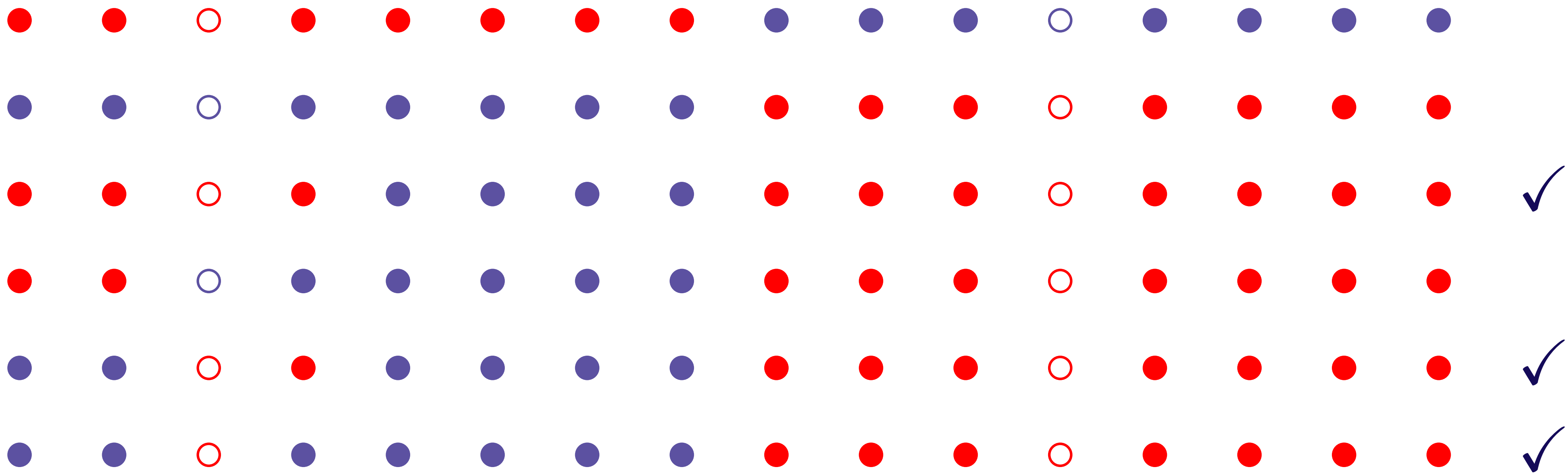
# Finding Consequences



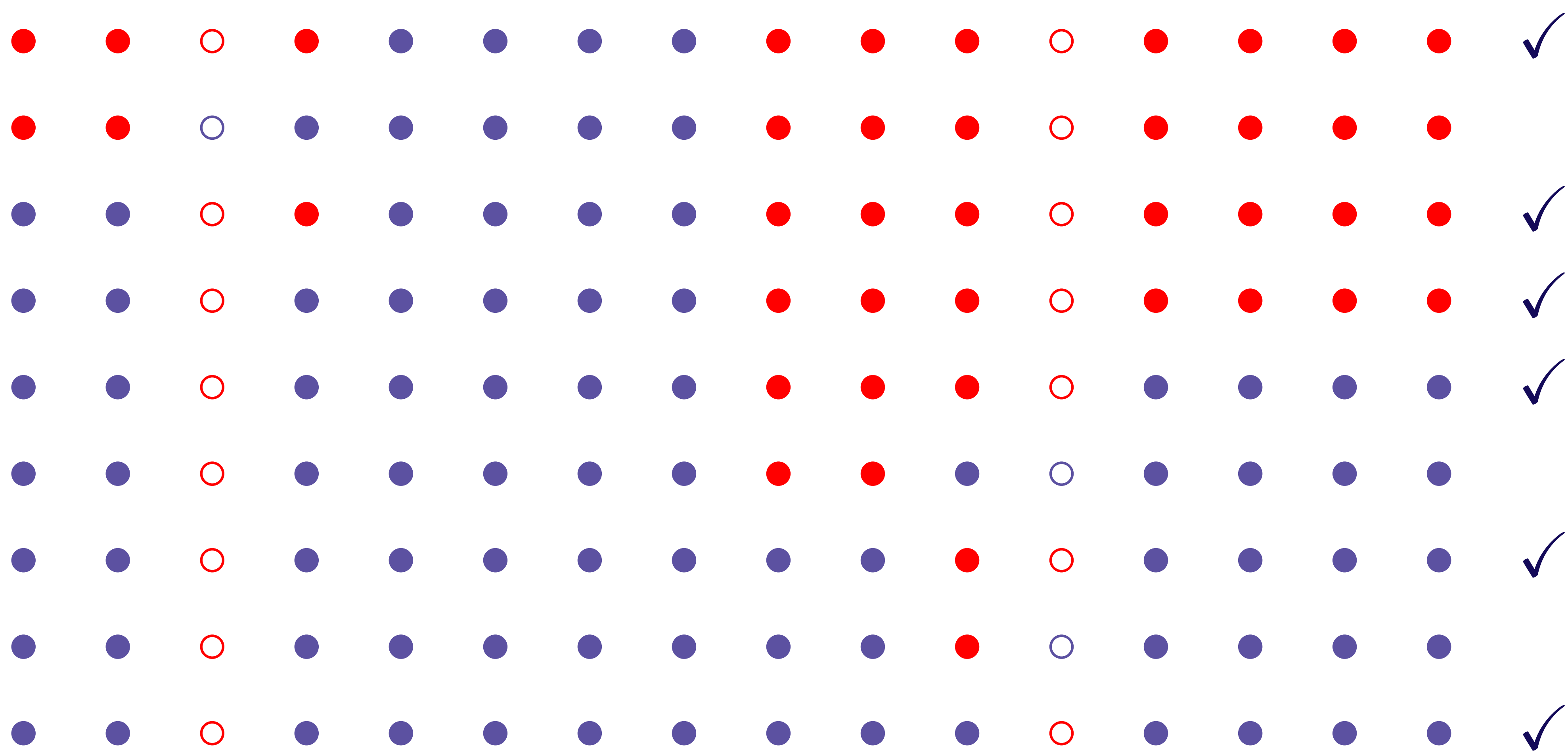
# Finding Consequences



# Finding Consequences



# Finding Consequences



# Finding Consequences

```
let rec MIN (p, support, S) =
   if |S| = 1
   then S
   else let S1, S2 = SPLIT(S) in
         if p(support ∪ S1)
         then MIN(p, support, S1)
         else if p(support ∪ S2)
               then MIN(p, support, S2)
               else let T1 = MIN(p, S2 ∪ support, S1) in
                     let T2 = MIN(p, T1 ∪ support, S2) in
                     (T1 ∪ T2)

let MINIMUM (p, S) = MIN(p, ∅, S)
```

Cost: $O\left(|S_0| + |S_0| \lg \frac{|S|}{|S_0|}\right)$ $p$-queries.

# Safety Analysis

Use counterexamples to induction:

- Find violating state $s$.
- If $\neg s$ has MIC $c$, $\chi \stackrel{\text{def}}{=} \chi \wedge c$.
- Otherwise, $\Pi \stackrel{\text{def}}{=} \Pi \wedge \neg s$.

Complete: Proves $\Pi$ or discovers counterexample.

# Safety Analysis

Use counterexamples to induction:

- Find violating state $s$.
- If $\neg s$ has MIC $c$, $\chi \stackrel{\text{def}}{=} \chi \wedge c$.
- Otherwise, $\Pi \stackrel{\text{def}}{=} \Pi \wedge \neg s$.

Complete: Proves $\Pi$ or discovers counterexample.

Results:

- Parallel implementation.
- Benchmarks: derived from PicoJava II microprocessor [McM03, McM05].
- Solved 20/20 (first to our knowledge).

# Related & Future Directions

- Similar approach to directed generation of inequalities [BM06]
- Extend to termination [BMS05a]
  - counterexample for existence of LRF: $n+1$ transitions
- Continuous, hybrid, stochastic systems
  - "barrier certificates" [PJ04]
  - applications to biology

# Related & Future Directions

- Similar approach to directed generation of inequalities [BM06]
- Extend to termination [BMS05a]
  - counterexample for existence of LRF: $n+1$ transitions
- Continuous, hybrid, stochastic systems
- Blocking clauses [McM02, JS05]
- $k$-induction [SSS00, dMRS03, AFF$^{+}$04, VH06, AS06]

# Related & Future Directions

- Similar approach to directed generation of inequalities [BM06]
- Extend to termination [BMS05a]
  - counterexample for existence of LRF: $n+1$ transitions
- Continuous, hybrid, stochastic systems
- Blocking clauses [McM02, JS05]
- $k$-induction [SSS00, dMRS03, AFF$^+$04, VH06, AS06]
- Fixed-width integer arithmetic
  - applications to embedded systems
  - initial ideas in [BMS05c]

# Outline

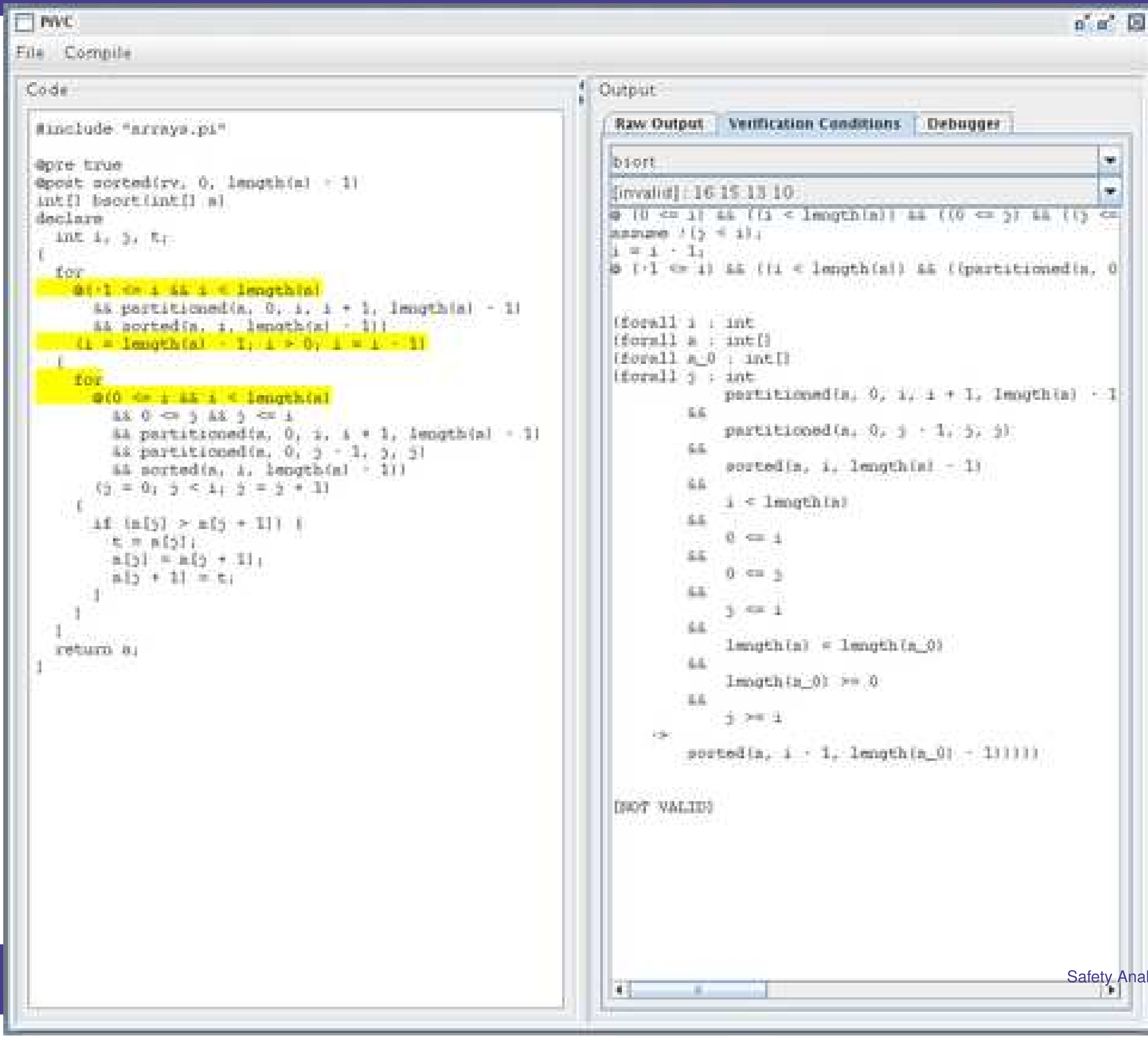
# Course: The Calculus of Computation

- Co-taught with Zohar Manna
- Co-authored textbook with Zohar
  (to be published by *Springer*)
- Goals:
  - Foundations of robust system design/analysis.
  - Computationally-oriented presentation of logic.
    $\Rightarrow$ Focus on decision procedures.
  - Complement compiler course: static analyses.

# Topics

- First-order logic, first-order theories
- Decision procedures:
  - quantifier-elimination for arithmetic
  - simplex algorithm for QF of rationals
  - QF equality (congruence closure)
  - QF recursive data structures
  - arrays (QFF & array property fragment)
  - Nelson-Oppen combination
- Formal specification and verification
- Static analysis

# πVC



PiVC
File Compile
Code
#include "arrays.pi"
@pre true
@post sorted(rv, 0, length(a) - 1)
int[] bsort(int[] a)
declare
int i, j, t;
{
for
@(-1 <= i && i < length(a)
&& partitioned(a, 0, i, i + 1, length(a) - 1)
&& sorted(a, i, length(a) - 1))
(i = length(a) - 1; i > 0; i = i - 1)
{
for
@(0 <= i && i < length(a)
&& 0 <= j && j <= i
&& partitioned(a, 0, i, i + 1, length(a) - 1)
&& partitioned(a, 0, j - 1, j, j)
&& sorted(a, i, length(a) - 1))
(j = 0; j < i; j = j + 1)
{
if (a[j] > a[j + 1]) {
t = a[j];
a[j] = a[j + 1];
a[j + 1] = t;
}
}
}
return a;
}
Output
Raw Output
Verification Conditions
Debugger
bsort
[invalid] : 16 15 13 10
(forall i : int
(forall a : int[]
(forall a_0 : int[]
(forall j : int
partitioned(a, 0, i, i + 1, length(a) - 1
partitioned(a, 0, j - 1, j, j)
sorted(a, i, length(a) - 1)
i < length(a)
0 <= i
0 <= j
j <= i
length(a) = length(a_0)
length(a_0) >= 0
j >= i
sorted(a, i - 1, length(a_0) - 1)))))
(NOT VALID)

Code

```
#include "arrays.pi"

@pre true
@post sorted(rv, 0, length(a) - 1)
int[] bsort(int[] a)
declare
  int i, j, t;
{
  for
    @(-1 <= i && i < length(a)
      && partitioned(a, 0, i, i + 1, length(a) - 1)
      && sorted(a, i, length(a) - 1))
    (i = length(a) - 1; i > 0; i = i - 1)
  {
    for
      @(0 <= i && i < length(a)
        && 0 <= j && j <= i
        && partitioned(a, 0, i, i + 1, length(a) - 1)
        && partitioned(a, 0, j - 1, j, j)
        && sorted(a, i, length(a) - 1))
      (j = 0; j < i; j = j + 1)
    {
      if (a[j] > a[j + 1]) {
        t = a[j];
        a[j] = a[j + 1];
        a[j + 1] = t;
      }
```

πVC

Output

Raw Output | Verification Conditions | Debugger

bsort

[invalid] : 16 15 13 10

```
@ (0 <= i) && ((i < length(a)) && ((0 <= j) && ((
assume !(j < i);
i = i - 1;
@ (-1 <= i) && ((i < length(a)) && ((partitioned(


(forall i : int
(forall a : int[]
(forall a_0 : int[]
(forall j : int
            partitioned(a, 0, i, i + 1, length(a)
        &&
            partitioned(a, 0, j - 1, j, j)
        &&
            sorted(a, i, length(a) - 1)
        &&
            i < length(a)
        &&
            0 <= i
        &&
            0 <= j
        &&
            j <= i
        &&
```

# 2006 Offering

- Organization:
  - 9 problem sets
  - 3+ verification projects using $\pi$VC
    (*e.g.*, InsertionSort, MergeSort, QuickSort)
  - Take-home final
- Student comments on material (from evaluations):
  "fun", "interesting", "wide-reaching foundations"

# Outline

1. Introduction
2. Decision Procedure for Arrays
3. Invariant Generation of Clauses
4. The Calculus of Computation
5. Directions for Research

# Directions for Research

Invariant generation

- Directed invariant generation beyond safety
- Hybrid, continuous, stochastic systems
- Fixed-width integers
- Procedures for other temporal properties
  Ex: existential invariants
  What must the system do to maintain $\Pi$?
- Connection with mathematical induction
  Ex: finite Hintikka set $\Rightarrow$ strengthen hypothesis

# Directions for Research

Decision procedures

- Heuristics for speed
- Collection and recursive data structures
- Application: vectorizing loops
- Integrate with mechanical theorem proving
  straightforward for instantiation-based procedures

# Directions for Research

Parallel systems (observations & questions):

- Obs: Poor data access patterns across threads
  Q: Static optimizations; language models?
- Obs: Clear data structure invariants
  Q: Determine synchronization from invariants?

# Directions for Research

Biology (challenges):

- Quantitative models (with noise)
  metabolic and signaling pathways
- Analysis of hierarchical models
  stochastic + continuous + discrete
- Ex: Given (noisy) data for protein interactions, what are possible behaviors?
  (soft) invariants, stable ratios, feedback loops

# For More Information

- Homepage:
  `http://theory.stanford.edu/~arbrad`
- Course website:
  `http://cs156.stanford.edu`
- Draft of textbook available upon request

# References


[AFF$^+$04] R. Armoni, L. Fix, R. Fraer, S. Huddleston, N. Piterman, and M. Vardi. Sat-based induction for temporal safety properties. In *BMC*, 2004.

[Aik99] Alex Aiken. Introduction to set constraint-based program analysis. *Science of Computer Programming*, 35, 1999.

[AS06] Mohammad Awedh and Fabio Somenzi. Automatic invariant strengthening to prove properties in bounded model checking. In *DAC*. ACM Press, 2006.

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *TACAS*. Springer-Verlag, 1999.

[BCM$^+$92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10ˆ20 states and beyond. *Inf. Comput.*, 98(2), 1992.

[BM06] Aaron R. Bradley and Zohar Manna. Verification constraint problems with strengthening. In *ICTAC*, volume 3722 of *LNCS*. Springer-Verlag, 2006.

[BM07] Aaron R. Bradley and Zohar Manna. *Property-directed Invariant Generation of Clauses*, 2007.

[BMS05a] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. Linear ranking with reachability. In *CAV*, volume 3576 of *LNCS*. Springer-Verlag, 2005.

[BMS05b] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. The polyranking principle. In *ICALP*, volume 3580 of *LNCS*, pages 1349–1361. Springer-Verlag, 2005.

[BMS05c] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. Termination analysis of integer linear loops. In *CONCUR*, volume 3653 of *LNCS*. Springer-Verlag, 2005.

[BMS05d] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. Termination of polynomial programs. In *VMCAI*, volume 3385 of *LNCS*. Springer-Verlag, 2005.

[BMS06] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. What's decidable about arrays? In *VMCAI*, volume 3855 of *LNCS*. Springer-Verlag, 2006.

[BRCZ05] R. Bagnara, E. Rodríguez-Carbonell, and E. Zaffanella. Generation of basic semi-algebraic invariants using convex polyhedra. In *SAS*, LNCS. Springer-Verlag, 2005.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*. ACM Press, 1977.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2), 1986.

[CH78] Patrick Cousot and Nicholas Halbwachs. Automatic discovery of linear restraints among the variables of a program. In *POPL*. ACM Press, 1978.

[Cou05] P. Cousot. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In *VMCAI*, volume 3385 of *LNCS*. Springer-Verlag, 2005.

[CS01] Michael Colón and Henny B. Sipma. Synthesis of linear ranking functions. In *TACAS*, volume 2031 of *LNCS*. Springer-Verlag, 2001.

[CSS03] Michael Colón, Sriram Sankaranarayanan, and Henny B. Sipma. Linear invariant generation using non-linear constraint solving. In *CAV*, volume 2725 of *LNCS*. Springer-Verlag, 2003.

[dMRS03] L. de Moura, H. Ruess, and M. Sorea. Bounded model checking and induction: From refutation to verification. In *CAV*, LNCS. Springer-Verlag, 2003.

[EN95] E. Allen Emerson and Kedar S. Namjoshi. Reasoning about rings. In *POPL*. ACM Press, 1995.

[ES93] E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. In *CAV*, LNCS. Springer-Verlag, 1993.

[Jaf81] Joxan Jaffar. Presburger arithmetic with array segments. *Inf. Processing Letters*, 12(2), 1981.

[JS05] HoonSang Jin and Fabio Somenzi. Prime clauses for fast enumeration of satisfying assignments to boolean circuits. In *DAC*. ACM Press, 2005.

[Kin69] James King. *A Program Verifier*. PhD thesis, Carnegie Mellon University, September 1969.

[Mat81] Prabhaker Mateti. A decision procedure for the correctness of a class of programs. *J. ACM*, 28(2), 1981.

[McC62] John McCarthy. Towards a mathematical science of computation. In *IFIP Congress 62*, 1962.

[McM02] Kenneth L. McMillan. Applying sat methods in unbounded symbolic model checking. In *CAV*, volume 2404 of *LNCS*. Springer-Verlag, 2002.

[McM03] Kenneth L. McMillan. Interpolation and sat-based model checking. In *CAV*, volume 2725 of *LNCS*. Springer-Verlag, 2003.

[McM05] Kenneth L. McMillan. Applications of Craig interpolants in model checking. In *TACAS*, volume 3440 of *LNCS*. Springer-Verlag, 2005.

[PJ04] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *HSCC*, volume 2993 of *LNCS*. Springer-Verlag, 2004.

[PP02] A. Papachristodoulou and Stephen Prajna. On the construction of lyapunov functions using the sum of squares decomposition. In *CDC*, 2002.

[PRZ01] Amir Pnueli, Sitvanit Ruah, and Lenore Zuck. Automatic deductive verification with invisible invariants. In *TACAS*, LNCS. Springer-Verlag, 2001.

[QS82] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *International Symposium on Programming*, volume 137 of *LNCS*. Springer-Verlag, 1982.

[SBDL01] Aaron Stump, Clark W. Barrett, David L. Dill, and Jeremy R. Levitt. A decision procedure for an extensional theory of arrays. In *LICS*, 2001.

[SJ80] Norihisa Suzuki and David Jefferson. Verification decidability of Presburger array programs. *J. ACM*, 27(1), 1980.

[SSM04] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Constraint-based linear relations analysis. In *SAS*, volume 3148 of *LNCS*. Springer-Verlag, 2004.

[SSS00] Mary Sheeran, Satnam Singh, and Gunnar Stalmarck. Checking safety properties using induction and a SAT-solver. In *FMCAD*, volume 1954 of *LNCS*. Springer-Verlag, 2000.

[VH06] Vishnu C. Vimjam and Michael S. Hsiao. Fast illegal state identification for improving SAT-based induction. In *DAC*. ACM Press, 2006.