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)

Demand for guarantees

Methods have other applications:
  • Study other (natural & engineered) systems.
  • Characterize decidability & complexity.
  • Provide tools (constraint solvers, static analyses)
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?

```c
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

```java
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

```plaintext
@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]

```plaintext
@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
    {[i ≤ u ∧ |a| = |a₀|
        ∧ ∀i, j. m ≤ i ≤ j ≤ u → a[i] ≤ a[j]
        ∧ ∀i. 0 ≤ i < ℓ → a[i] = a₀[i]
        ∧ ∀i. u < i < |a| → a[i] = a₀[i]
        (int m := u; m > ℓ; m := m - 1)
    for
    {[ℓ < i ≤ u ∧ ℓ ≤ j ≤ i ∧ |a| = |a₀|
        ∧ ∀i, j. m ≤ i ≤ j ≤ u → a[i] ≤ a[j]
        ∧ ∀i. ℓ ≤ i ≤ m < j ≤ u → a[i] ≤ a[j]
        ∧ ∀i. ℓ ≤ i < n → a[i] ≤ a[n]
        ∧ ∀i. 0 ≤ i < ℓ → a[i] = a₀[i]
        ∧ ∀i. u < i < |a| → a[i] = a₀[i]
        (int n := ℓ; n < m; n := n + 1)
    }
}
```
What Comprises Verification?

- Implementation
- Specification
- Strengthen invariant generation [BMS05c, BM06, BM07]
- Check argument decision procedures [BMS06]

```plaintext
int[] BubbleSort(int[] a0, int ℓ, int u) {
    int[] a := a0;
    for @L1 : @pre 0 ≤ ℓ, u < |a0|
        (int m := u; m > ℓ; m := m − 1)
        for [i ≤ u ∧ |a| = |a0|]
            ∧ ∀i, j. ℓ ≤ i ≤ j ≤ u → a[i] ≤ a[j]
            ∧ ∀i. 0 ≤ i < ℓ → a[i] = a0[i]
            ∧ ∀i. u < i < |a| → a[i] = a0[i]
        (int m := u; m > ℓ; m := m − 1)
    for [ℓ < i ≤ u ∧ ℓ ≤ j ≤ i ∧ |a| = |a0|]
        ∧ ∀i, j. m ≤ i ≤ j ≤ u → a[i] ≤ a[j]
        ∧ ∀i. ℓ ≤ i ≤ n → a[i] ≤ a[n]
        ∧ ∀i. 0 ≤ i < ℓ → a[i] = a0[i]
        ∧ ∀i. u < i < |a| → a[i] = a0[i]
    (int n := ℓ; n < m; n := n + 1)
}
```
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]
Invariant:

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

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

Inductive Invariant:

- Initiation: Includes initial states
- Consecution: Closed under transitions
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
- \( \rho[\overline{x}, \overline{x}'] \): transition relation

\[
x' = x + 1 \quad \lor \quad x' = 0
\]

\[
x \geq 0
\]
Formally...

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

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

Inductive invariant $\varphi$:

- $\Theta \Rightarrow \varphi$
- $\varphi \land \rho \Rightarrow \varphi'$

1. $x \geq 0 \Rightarrow x \geq 0$
2. $x \geq 0 \land (x' = x + 1 \lor x' = 0) \Rightarrow x' \geq 0$
Given $\langle \overline{x}, \Theta, \rho \rangle$ and property $\Pi$.

Goal: Prove that $\Pi$ is invariant.
Given \( \langle x, \Theta, \rho \rangle \) and property \( \Pi \).

Goal: Prove that \( \Pi \) is invariant.

Inductive method:
Find strengthening assertion \( \chi \) such that

\[
\begin{align*}
\Theta & \Rightarrow \Pi \land \chi \\
\Pi \land \chi \land \rho & \Rightarrow \Pi' \land \chi'
\end{align*}
\]
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

- $\Theta \Rightarrow \Pi \land \chi$
- $\Pi \land \chi \land \rho \Rightarrow \Pi' \land \chi'$
Challenges

1. Prove initiation and consecution automatically (especially for infinite-state systems)
   ⇒ decision procedures

2. Discover strengthening invariants automatically
   ⇒ invariant generation procedures
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 $T_A$

$$\Sigma_A : \{ a[i], \ a\langle i \triangleleft e \rangle, = \}$$

- $\forall a, v, i, j. \ i = j \rightarrow a\langle i \triangleleft v \rangle[j] = v$  \hspace{1cm} (ROW 1)
- $\forall a, v, i, j. \ i \neq j \rightarrow a\langle i \triangleleft v \rangle[j] = a[j]$  \hspace{1cm} (ROW 2)
Theory $T_A$

$\Sigma_A : \{ a[i], \ a(i < e), \ = \}$

- $\forall a, v, i, j. \ i = j \rightarrow a(i < v)[j] = v$  \hspace{1cm} (ROW 1)
- $\forall a, v, i, j. \ i \neq j \rightarrow a(i < v)[j] = a[j]$  \hspace{1cm} (ROW 2)

Combination: index & element theories.

$$\text{sorted}(a(i < 0)(j < 1), i, j)$$
$$\land \ \text{sorted}(a(i < 2)(j < 3), i, j)$$
$$\land \ i + 1 < j$$

$$\text{sorted}(a, \ell, u) \overset{\text{def}}{=} \forall i, j. \ \ell \leq i \leq j \leq u \rightarrow a[i] \leq a[j]$$
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:
  \[ \forall i. \ \ell \leq i \leq u \rightarrow F[\bar{a}, i] \]
Useful Properties

For sorting [Mat81, Jaf81]:

- \text{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.

NP-complete for bounded stack of $\mathcal{A}$.
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 $I$ s.t.:

- each $\ell \in I$ occurs in $F$;
- instantiating universal quantifiers over $I$ 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?

\[ 3(x + 2)^2 - xy = 5 \]

\[ \begin{align*}
  u & : 3(x + 2)^2 - xy - 5 \\
  v & : 15 + 6x - y \\
  w & : -x \\
  6 & \quad -1 \\
\end{align*} \]

- \( x = y = 0: \)
  \( u = 7, v = 15, w = 0 \)
- \( x := x + 1: \)
  \( (u, v, w) := (u + v, v + 6, w - 1) \)
- \( y := y + 1: \)
  \( (u, v, w) := (u + w, v - 1, w) \)
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 \textit{sum} array \(s\).
- \(\Theta(z)\): simulate at origin
- \(\rho(z_1, z_2)\): relate positions
  1. possibly move one step in one direction
  2. \(s(z_2) = s(z_1) - u\)
- \(\Pi(z)\): \(s(z) > 0\)

\[
\exists z, u, v, w, s. \forall i. \exists j. \Theta(z) \land \rho(i, j) \land \Pi(i)
\]

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 \text{sorted}(rv, 0, |rv| - 1)

```c
int[] BubbleSort(int[] a0) {
    int[] a := a0;
    for
        \[ -1 \leq i < |a| \]
        \[ \wedge \text{partitioned}(a, 0, i, i + 1, |a| - 1) \]
        \[ \wedge \text{sorted}(a, i, |a| - 1) \]
    (int i := |a| - 1; i > 0; i := i - 1) {
        int k := \text{maxi}(a, 0, i, \geq);
        a[k] := a[i];
    }

    return a;
}
```

\( (\forall j. 0 \leq j \leq i \rightarrow a[k] \geq a[j]) \)
\( \wedge a' = a\langle k < a[i]\rangle\langle i < a[k]\rangle \)
Vectorized Loops

Goal: Reduce annotation burden.

Encode array/collection operations in \( \exists \forall \) fragments:

- \( \text{map}(a, f) \)
- \( \text{filter}(a, p^{(1)}) \)
- \( \text{choose}(a) \)
- \( \text{max}(a, q^{(2)}) \)
- \( \forall a \in a, p^{(1)} \)
- \( \exists a \in a, p^{(1)} \)

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

- Decide VCs quickly.
Outline

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

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

Safety property: $\Pi$

- $\bar{x}$ 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)
- \( \Pi \land \chi \land \rho \Rightarrow \Pi' \land \chi' \)  
  (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:

\[ \Pi \land \chi_i \land \rho \not\Rightarrow \Pi' \]

Counterexample:

\[ s : x_1 \land \neg x_2 \land \neg x_3 \land \cdots \land x_n \]

Negation is clause:

\[ \neg s : \neg x_1 \lor x_2 \lor x_3 \lor \cdots \lor \neg x_n \]

Goal: Find minimal inductive subclause \( d \sqsubseteq \neg s \)

\[ d : x_2 \lor x_{13} \lor \neg x_{41} \]
Finding a MIC

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

Given $c$: 

\[ \begin{array}{c}
\circ & \circ \\
\downarrow & \downarrow \\
\circ & \circ \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\circ & \circ \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\circ & \circ \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\circ & \circ \\
\end{array} \]
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.

$$d \Rightarrow \text{wp}(d, \rho)$$
Goal: Compute largest $d \sqsubseteq c$ s.t.

$$d \Rightarrow \text{wp}(d, \rho)$$

Slow method:

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

Compute $d_{i+1} \Rightarrow \text{wp}(d_i, \rho)$ literal-by-literal.

Cost: $O(n^2)$ SAT problems. $O(n)$ per iteration.
Goal: Compute largest $d \sqsubseteq c$ s.t.

$$d \Rightarrow \wp(d, \rho)$$

Fast method:

- Find counterexample to $c \Rightarrow \wp(c, \rho)$
  $c_1$: drop satisfied literals of $c$

- Find counterexample to $c_1 \Rightarrow \wp(c_1, \rho)$

- ...

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: \( \Omega \left( |S_0| \log \frac{|S|-|S_0|}{|S_0|} \right) \) \( p \)-queries.

Yet the typical solution is linear in \( |S| \)!
Finding Consequences
Finding Consequences
Finding Consequences
Finding Consequences
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: $O \left( |S₀| + |S₀| \log \frac{|S|}{|S₀|} \right)$ p-queries.
Use counterexamples to induction:

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

Complete: Proves $\Pi$ or discovers counterexample.
Use counterexamples to induction:

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

**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

1. Introduction
2. Decision Procedure for Arrays
3. Invariant Generation of Clauses
4. Course: The Calculus of Computation
5. Directions for Research
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.
    - 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
```c
#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;
      }
  }
}
```
\begin{verbatim}
bsort
[invalid]: 16 15 13 10
\texttt{@ (0 <= i) && ((i < length(a)) && ((0 <= j) && ((j < i)));}
i = i - 1;
\texttt{@ (-1 <= i) && ((i < length(a)) && ((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 <= j
&& 0 <= j
&& j <= i
\end{verbatim}
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
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
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


[BRCZ05] R. Bagnara, E. Rodríguez-Carbonell, and E. Zaffanella. Generation of basic semi-algebraic in-


[CSS03] Michael Colón, Sriram Sankaranarayanan, and Henny B. Sipma. Linear invariant generation us-


