SLAM

- A Microsoft tool for checking safety of device drivers
- Inspired BLAST
BLAST

Berkeley Lazy Abstraction
Software * Tool

www.eecs.berkeley.edu/~blast/
Counter Example
Guided Refinement
CEGAR

Mooly Sagiv
Recap

• Many abstract domains
  – Signs
  – Odd/Even
  – Constant propagation
  – Intervals
  – [Polyhedra]
  – Canonic abstraction
  – Domain constructors
  – …

• Static Algorithms
  – Iterative Chaotic Iterations
  – Widening/Narrowing
  – Interprocedural Analysis
  – Concurrency
  – Modularity
  – Non-Iterative methods
A Lattice of Abstractions

• Every element is an abstract domain
• $A \subseteq A'$ if there exists a Galois Connection from $A$ to $A'$
But how to find the appropriate abstract domain

- Precision vs. Scalability
- Sometimes precision improves scalability
- Specialize the abstraction for the desired property
Counter Example Guided Refinement (CEGAR)

- Run the analysis with a simple abstract domain
- When the analysis verifies the property declare done
- If the analysis reports an error employs a theorem prover to identify if the error is feasible
  - If the error is feasible generate a concrete trace
  - If the error is spurious refine the abstract domain and repeat
A Simple Example

\[ z = 5 \]

if \( y > 0 \)
\[ x = z; \]
else
\[ x = -y; \]
assert \( x > 0 \)

sign(x)

\[ \text{assert } x > 0 \]
A Simple Example

z = 5
if (y > 0)
    x = z;
else
    x = -y;
assert x > 0

sign(x), sign(y)
A Simple Example

\[ z = 5 \]

if \((y > 0)\)
  \[ x = z; \]
else
  \[ x = -y; \]
assert \(x > 0\)

\[ \text{sign}(x), \text{sign}(y), \text{sign}(z) \]

\[ \begin{align*}
  [x \mapsto & T, \ y \mapsto T, \ z \mapsto T] \\
  [x \mapsto & T, \ y \mapsto T, \ z \mapsto P] \\
  [x \mapsto & T, \ y \mapsto P, \ z \mapsto T] \\
  [x \mapsto & T, \ y \mapsto P, \ z \mapsto P] \\
  [x \mapsto & T, \ y \mapsto N, \ z \mapsto P] \\
  [x \mapsto & P, \ y \mapsto P, \ z \mapsto T] \\
  [x \mapsto & P, \ y \mapsto P, \ z \mapsto P] \\
  [x \mapsto & P, \ y \mapsto N, \ z \mapsto P] \\
  [x \mapsto & P, \ y \mapsto T, \ z \mapsto P] \\
  [x \mapsto & P, \ y \mapsto P, \ z \mapsto P] \\
  [x \mapsto & P, \ y \mapsto P, \ z \mapsto P] \\
  \end{align*} \]
Simple Example (local abstractions)

\[ z = 5 \]

\[
\text{if } (y > 0) \quad x = z; \\
\text{else} \quad x = -y; \\
\text{assert } x > 0
\]
Plan

• CEGAR in BLAST (inspired by SLAM) POPL’04
• Limitations
Abstractions from Proofs
Scalable Program Verification

• *Little theorems about big programs*
  – Partial Specifications
    • Device drivers use kernel API correctly
    • Applications use root privileges correctly
  – Behavioral, path-sensitive properties
Predicate Abstraction: A crash course

- Abstraction: **Predicates** on program state
  - Signs: \( x > 0 \)
  - Aliasing: \( \&x \neq \&y \)

- States satisfying the same predicates are equivalent
  - Merged into single abstract state
(Predicate) Abstraction: A crash course

Q1: Which predicates are required to verify a property?
The Predicate Abstraction Domain

- Fixed set of predicates $\text{Pred}$
- The relational domain is $\langle P(P(\text{Pred})), \emptyset, P(\text{Pred}), \cup, \cap \rangle$
  - Join is set union
  - State space explosion
- Special case of canonic abstraction
Scalability vs. Verification

- Few predicates tracked
  - *e.g.* type of variables

- Imprecision hinders Verification
  - Spurious counterexamples

- Many predicates tracked
  - *e.g.* values of variables

- State explosion
  - Analysis drowned in detail
Example

while(*){
  1: if (p₁) lock();
      if (p₁) unlock();
  ...
  2: if (p₂) lock();
      if (p₂) unlock();
  ...
  n: if (pₙ) lock();
      if (pₙ) unlock();
}

Only track lock

Bogus Counterexample
- Must correlate branches

Predicate $p₁$ makes trace abstractly infeasible

$pᵢ$ required for verification
Example

```c
while(*){
  1: if (p_1) lock();
      if (p_1) unlock();
  ...
  2: if (p_2) lock();
      if (p_2) unlock();
  ...
  n: if (p_n) lock();
      if (p_n) unlock();
}
```

Only track `lock`

Track `lock, p_i`s

Bogus Counterexample
- Must *correlate* branches

State Explosion
- > $2^n$ distinct states
- intractable

How can we get scalable verification?
By Localizing Precision

while (*) {
    1: if (p₁) lock();
    if (p₁) unlock();
    ...
    2: if (p₂) lock();
    if (p₂) unlock();
    ...
    n: if (pₙ) lock();
    if (pₙ) unlock();
}

Preds. Used locally
Ex: 2 £ n states

Preds. used globally
Ex: 2ⁿ states

P₁
P₂
Pₙ

Q2: *Where* are the predicates required?
Counterexample Guided Refinement

1. **What predicates** remove trace?  
   - Make it abstractly infeasible

2. **Where** are predicates needed?

Seed Abstraction > Abstract > Check

*Is model safe?*

[Why infeasible?](#)

[Explanation](#)

Refine

SAFE

BUG

[Clarke et al. '00]

[Ball, Rajamani '01]
Counterexample Guided Refinement

Seed Abstraction Program → Abstract → Check

- explanation
- NO! (Trace)
- Why infeasible?
- Refine
- feasible
- YES
- SAFE
- BUG

Is model safe?

YES

SAFE

BUG
Counterexample Guided Refinement

Seed Abstraction Program → Abstract

Why infeasible?

NO! (Trace)

Check

Is model safe?

Refine

Why infeasible?

explanation

feasible

SAFE

BUG

safe
This Talk: Counterexample Analysis

1. What predicates remove trace?
   - Make it abstractly infeasible
2. Where are predicates needed?

Seed Abstraction Program → Abstract → Check

- explanation
- Why infeasible?
- NO! (Trace)
- feasible
- YES
- SAFE
- BUG
Plan

1. Motivation

2. Refinement using Traces
   • Simple
   • Procedure calls

3. Results
Trace Formulas

• A single abstract trace represents infinite number of traces
  – Different loop iterations
  – Different concrete values

• Solution
  – Only considers concrete traces with the same number of executions
  – Use formulas to represent sets of states
Representing **States as** *Formulas*

<table>
<thead>
<tr>
<th>[F]</th>
<th>(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>states satisfying (F) ({s \mid s \models F})</td>
<td>(F) FO formula over prog. vars</td>
</tr>
<tr>
<td>([F_1] \cap [F_2])</td>
<td>(F_1 \land F_2)</td>
</tr>
<tr>
<td>([F_1] \cup [F_2])</td>
<td>(F_1 \lor F_2)</td>
</tr>
<tr>
<td>([F])</td>
<td>(\neg F)</td>
</tr>
<tr>
<td>([F_1] \subseteq [F_2])</td>
<td>(F_1) implies (F_2)</td>
</tr>
</tbody>
</table>

\(i.e. \ F_1 \land \neg F_2\) unsatisfiable
Counterexample Analysis

Q0: Is trace feasible?
Q1: What predicates remove trace?
Q2: Where are preds required?

SSA
Trace Feasibility Formula

Thm Pvr
Proof of Unsat.

Extract
Predicate Map:
Prog Ctr ! Predicates

Refine
Feasible
Explanation of Infeasibility
Trace
Counterexample Analysis

Q0: Is trace feasible?
Q1: What predicates remove trace?
Q2: Where are preds required?

Trace → Refine → Feasible
  Explanation of Infeasibility

Trace → SSA → Thm Pvr → Feasible
  Trace Feasibility Formula
  Proof of Unsat.
  Extract
  Predicate Map: Prog Ctr ! Predicates
\[ pc_1: x = \text{ctr}; \]
\[ pc_2: \text{ctr} = \text{ctr} + 1; \]
\[ pc_3: y = \text{ctr}; \]
\[ pc_4: \text{if } (x = i-1)\{ \]
\[ pc_5: \text{if } (y \neq i)\{ \text{ERROR: } \}
\]

\[ \text{pc}_1: x = \text{ctr} \]
\[ \text{pc}_2: \text{ctr} = \text{ctr} + 1 \]
\[ \text{pc}_3: y = \text{ctr} \]
\[ \text{pc}_4: \text{assume}(x = i-1) \]
\[ \text{pc}_5: \text{assume}(y \neq i) \]
### Trace Feasibility Formulas

<table>
<thead>
<tr>
<th>Trace</th>
<th>SSA Trace</th>
<th>Trace Feasibility Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pc_1$: $x = \text{ctr}$</td>
<td>$pc_1$: $x_1 = \text{ctr}_0$</td>
<td>$x_1 = \text{ctr}_0$</td>
</tr>
<tr>
<td>$pc_2$: $\text{ctr} = \text{ctr}+1$</td>
<td>$pc_2$: $\text{ctr}_1 = \text{ctr}_0+1$</td>
<td>$\land \ \text{ctr}_1 = \text{ctr}_0 + 1$</td>
</tr>
<tr>
<td>$pc_3$: $y = \text{ctr}$</td>
<td>$pc_3$: $y_1 = \text{ctr}_1$</td>
<td>$\land \ y_1 = \text{ctr}_1$</td>
</tr>
<tr>
<td>$pc_4$: $\text{assume}(x=i-1)$</td>
<td>$pc_4$: $\text{assume}(x_1=i_0-1)$</td>
<td>$\land \ x_1 = i_0 - 1$</td>
</tr>
<tr>
<td>$pc_5$: $\text{assume}(y\neq i)$</td>
<td>$pc_5$: $\text{assume}(y_1\neq i_0)$</td>
<td>$\land \ y_1 \neq i_0$</td>
</tr>
</tbody>
</table>

**Theorem:** Trace is **Feasible**, TFF is **Satisfiable**

Compact Verification Conditions [Flanagan,Saxe ’00]
Counterexample Analysis

Q0: Is trace feasible?
Q1: What predicates remove trace?
Q2: Where are preds required?

Trace → Refine → Feasible
Refine → Explanation of Infeasibility

Trace → SSA → Trace Feasibility Formula
SSA → Thm Pvr → Proof of Unsat.
Thm Pvr → Y
Y → Feasible
No → Extract
Extract → Predicate Map:

Prog Ctrl ! Predicates
Counterexample Analysis

Q0: Is trace feasible?
Q1: What predicates remove trace?
Q2: Where are preds required?

Refine

Feasible
Explanation of Infeasibility

SSA
Trace Feasibility Formula

Thm Pvr
Proof of Unsat.

Extract
Predicate Map:
Prog Ctr ! Predicates
Proof of Unsatisfiability

\[ x_1 = \text{ctr}_0 \]
\[ \land \text{ctr}_1 = \text{ctr}_0 + 1 \]
\[ \land y_1 = \text{ctr}_1 \]
\[ \land x_1 = i_0 - 1 \]
\[ \land y_1 \neq i_0 \]

Trace Formula

Proof of Unsatisfiability

PROBLEM

Proof uses entire \textit{history} of execution

\begin{itemize}
  \item Information flows up and down
\end{itemize}

No \textit{localized} or \textit{state} information!
The Present State...

Trace

\[ pc_1: x = \text{ctr} \]
\[ pc_2: \text{ctr} = \text{ctr} + 1 \]
\[ pc_3: y = \text{ctr} \]
\[ pc_4: \text{assume}(x = i-1) \]
\[ pc_5: \text{assume}(y \neq i) \]

... is all the information the executing program has **here**

State...

1. ... after executing trace **prefix**
2. ... knows **present values** of variables
3. ... makes trace **suffix** infeasible

At \( pc_4 \), which predicate on **present state** shows infeasibility of **suffix**?
What Predicate is needed?

<table>
<thead>
<tr>
<th>Trace</th>
<th>Trace Formula (TF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pc_1$: $x = ctr$</td>
<td>$x_1 = ctr_0$</td>
</tr>
<tr>
<td>$pc_2$: $ctr = ctr + 1$</td>
<td>$\land ctr_1 = ctr_0 + 1$</td>
</tr>
<tr>
<td>$pc_3$: $y = ctr$</td>
<td>$\land y_1 = ctr_1$</td>
</tr>
<tr>
<td>$pc_4$: $\text{assume}(x = i-1)$</td>
<td>$\land x_1 = i_0 - 1$</td>
</tr>
<tr>
<td>$pc_5$: $\text{assume}(y \neq i)$</td>
<td>$\land y_1 \neq i_0$</td>
</tr>
</tbody>
</table>

State...

1. ... after executing trace **prefix**

2. ... has *present values* of variables

3. ... makes trace **suffix** infeasible

Predicate ...

... implied by TF **prefix**
What Predicate is needed?

Trace

\( pc_1: \ x = \text{ctr} \)
\( pc_2: \ \text{ctr} = \text{ctr} + 1 \)
\( pc_3: \ y = \text{ctr} \)
\( pc_4: \ \text{assume}(x = i - 1) \)
\( pc_5: \ \text{assume}(y \neq i) \)

Trace Formula (TF)

\[
\begin{align*}
x_1 &= \text{ctr}_0 \\
\land \quad \text{ctr}_1 &= \text{ctr}_0 + 1 \\
\land \quad y_1 &= \text{ctr}_1 \\
\land \quad x_1 &= i_0 - 1 \\
\land \quad y_1 &\neq i_0
\end{align*}
\]

State...

1. ... after executing trace \textit{prefix} ...
2. ... has \textit{present values} of variables ...
3. ... makes trace \textit{suffix} infeasible ...

Predicate ...

... implied by TF \textit{prefix} ...
... on \textit{common} variables
What Predicate is needed?

<table>
<thead>
<tr>
<th>Trace</th>
<th>Trace Formula (TF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pc_1$: $x = ctr$</td>
<td>$x_1 = ctr_0$</td>
</tr>
<tr>
<td>$pc_2$: $ctr = ctr + 1$</td>
<td>$\land ctr_1 = ctr_0 + 1$</td>
</tr>
<tr>
<td>$pc_3$: $y = ctr$</td>
<td>$\land y_1 = ctr_1$</td>
</tr>
<tr>
<td>$pc_4$: assume($x = i-1$)</td>
<td>$\land x_1 = i_0 - 1$</td>
</tr>
<tr>
<td>$pc_5$: assume($y \neq i$)</td>
<td>$\land y_1 \neq i_0$</td>
</tr>
</tbody>
</table>

State...

1. ... after executing trace **prefix**
2. ... has **present values** of variables
3. ... makes trace **suffix** infeasible

Predicate...

... implied by TF **prefix**
... on **common** variables
... & TF **suffix** is **unsatisfiable**
What Predicate is needed?

<table>
<thead>
<tr>
<th>Trace</th>
<th>Trace Formula (TF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pc_1$: $x = \text{ctr}$</td>
<td>$\quad x_1 = \text{ctr}_0$</td>
</tr>
<tr>
<td>$pc_2$: $\text{ctr} = \text{ctr} + 1$</td>
<td>$\quad \land \quad \text{ctr}_1 = \text{ctr}_0 + 1$</td>
</tr>
<tr>
<td>$pc_3$: $y = \text{ctr}$</td>
<td>$\quad \land \quad y_1 = \text{ctr}_1$</td>
</tr>
<tr>
<td>$pc_4$: $\text{assume}(x = i - 1)$</td>
<td>$\quad \land \quad x_1 = i_0 - 1$</td>
</tr>
<tr>
<td>$pc_5$: $\text{assume}(y \neq i)$</td>
<td>$\quad \land \quad y_1 \neq i_0$</td>
</tr>
</tbody>
</table>

State...

1. ... after executing trace \textit{prefix} ... implied by TF \textit{prefix}
2. ... knows \textit{present values} of variables ... on \textit{common} variables
3. ... makes trace \textit{suffix} infeasible ... & TF \textit{suffix} is \textit{unsatisfiable}
Craig’s Interpolation Theorem [Craig ’57]

Given formulas $\psi^-, \psi^+$ s.t. $\psi^- \land \psi^+$ is *unsatisfiable*

There exists an *Interpolant* $\Phi$ for $\psi^-, \psi^+$, s.t.

1. $\psi^-$ *implies* $\Phi$
2. $\Phi$ has symbols *common* to $\psi^-$, $\psi^+$
3. $\Phi \land \psi^+$ is *unsatisfiable*
Examples of Craig’s Interpolation

- $\psi^- = b \land (\neg b \lor c)$
  $\psi^+ = \neg c$

- $\psi^- = x_1 = \text{ctr}_0 \land \text{ctr}_1 = \text{ctr}_0 + 1 \land y_1 = \text{ctr}_1$
  $\psi^+ = x_1 = i_0 - 1 \land y_1 \neq i_0$
  - $y_1 = x_1 + 1$
Craig’s Interpolation Theorem [Craig ’57]

Given formulas $\psi^-$, $\psi^+$ s.t. $\psi^- \land \neg \psi^+$ is **unsatisfiable**

There exists an **Interpolant** $\Phi$ for $\psi^-$, $\psi^+$, s.t.

1. $\psi^-$ **implies** $\Phi$
2. $\Phi$ has only symbols **common** to $\psi^-$, $\psi^+$
3. $\Phi \land \psi^+$ is **unsatisfiable**

$\Phi$ computable from **Proof of Unsat.** of $\psi^- \land \psi^+$

[Krajicek ’97] [Pudlak ’97]
(booleans) SAT–based Model Checking [McMillan ’03]
Interpolant = Predicate!

<table>
<thead>
<tr>
<th>Trace</th>
<th>Trace Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pc_1$: $x = \text{ctr}$</td>
<td>$x_1 = \text{ctr}_0$</td>
</tr>
<tr>
<td>$pc_2$: $\text{ctr} = \text{ctr} + 1$</td>
<td>$\wedge \hspace{1em} \text{ctr}_1 = \text{ctr}_0 + 1$</td>
</tr>
<tr>
<td>$pc_3$: $y = \text{ctr}$</td>
<td>$\wedge \hspace{1em} y_1 = \text{ctr}_1$</td>
</tr>
<tr>
<td>$pc_4$: assume($x = i-1$)</td>
<td>$\wedge \hspace{1em} x_1 = i_0 - 1$</td>
</tr>
<tr>
<td>$pc_5$: assume($y \neq i$)</td>
<td>$\wedge \hspace{1em} y_1 \neq i_0$</td>
</tr>
</tbody>
</table>

**Require:**

1. Predicate *implied* by trace *prefix*
2. Predicate on *common* variables
   common = *current* value
3. Predicate & *suffix* yields a *contradiction*

**Interpolant:**

1. $\psi^-$ *implies* $\Phi$
2. $\Phi$ has symbols *common* to $\psi^-$, $\psi^+$
3. $\Phi \wedge \psi^+$ is *unsatisfiable*
Interpolant = Predicate!

<table>
<thead>
<tr>
<th>Trace</th>
<th>Trace Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pc_1$: $x = ctr$</td>
<td>$x_1 = ctr_0$</td>
</tr>
<tr>
<td>$pc_2$: $ctr = ctr + 1$</td>
<td>$\land ctr_1 = ctr_0 + 1$</td>
</tr>
<tr>
<td>$pc_3$: $y = ctr$</td>
<td>$\land y_1 = ctr_1$</td>
</tr>
<tr>
<td>$pc_4$: assume($x = i-1$)</td>
<td>$\land x_1 = i_0 - 1$</td>
</tr>
<tr>
<td>$pc_5$: assume($y \neq i$)</td>
<td>$\land y_1 \neq i_0$</td>
</tr>
</tbody>
</table>

Interpolant:

1. $\psi^-$ implies $\Phi$
2. $\Phi$ has symbols common to $\psi^-$, $\psi^+$
3. $\Phi \land \psi^+$ is unsatisfiable

Require:

1. Predicate implied by trace prefix
2. Predicate on common variables
3. Predicate & suffix yields a contradiction
Interpolant = Predicate!

Trace

<table>
<thead>
<tr>
<th>Trace</th>
<th>Trace Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>pc₁: x = ctr</td>
<td>$x₁ = ctr₀$</td>
</tr>
<tr>
<td>pc₂: ctr = ctr + 1</td>
<td>$ctr₁ = ctr₀ + 1$</td>
</tr>
<tr>
<td>pc₃: y = ctr</td>
<td>$y₁ = ctr₁$</td>
</tr>
<tr>
<td>pc₄: assume(x = i-1)</td>
<td>$x₁ = i₀ - 1$</td>
</tr>
<tr>
<td>pc₅: assume(y ≠ i)</td>
<td>$y₁ ≠ i₀$</td>
</tr>
</tbody>
</table>

Require:
1. Predicate **implied** by trace **prefix**
2. Predicate on **common** variables
3. Predicate & **suffix** yields a **contradiction**

Interpolant:
1. $ψ⁻$ implies $Φ$
2. $Φ$ has symbols **common** to $ψ⁻$, $ψ⁺$
3. $Φ ∀ Φψ⁺$ is **unsatisfiable**
Building Predicate Maps

<table>
<thead>
<tr>
<th>Trace</th>
<th>Trace Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pc_1$: $x = \text{ctr}$</td>
<td>$x_1 = \text{ctr}_0$</td>
</tr>
<tr>
<td>$pc_2$: $\text{ctr} = \text{ctr} + 1$</td>
<td>$\land \text{ctr}_1 = \text{ctr}_0 + 1$</td>
</tr>
<tr>
<td>$pc_3$: $y = \text{ctr}$</td>
<td>$\land y_1 = \text{ctr}_1$</td>
</tr>
<tr>
<td>$pc_4$: $\text{assume}(x = i - 1)$</td>
<td>$\land x_1 = i_0 - 1$</td>
</tr>
<tr>
<td>$pc_5$: $\text{assume}(y \neq i)$</td>
<td>$\land y_1 \neq i_0$</td>
</tr>
</tbody>
</table>

- Cut + Interpolate at each point
- Pred. Map: $pc_i \mapsto$ Interpolant from cut $i$
# Building Predicate Maps

<table>
<thead>
<tr>
<th>Trace</th>
<th>Trace Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pc_1$: $x = ctr$</td>
<td>$x_1 = ctr_0$</td>
</tr>
<tr>
<td>$pc_2$: $ctr = ctr + 1$</td>
<td>$\land ; ctr_1 = ctr_0 + 1$</td>
</tr>
<tr>
<td>$pc_3$: $y = ctr$</td>
<td>$\land ; y_1 = ctr_1$</td>
</tr>
<tr>
<td>$pc_4$: assume($x = i-1$)</td>
<td>$\land ; x_1 = i_0 - 1$</td>
</tr>
<tr>
<td>$pc_5$: assume($y \neq i$)</td>
<td>$\land ; y_1 \neq i_0$</td>
</tr>
</tbody>
</table>

- Cut + Interpolate at each point
- Pred. Map: $pc_i \mapsto$ Interpolant from cut $i$

Predicate Map
- $pc_2$: $x = ctr$
- $pc_3$: $x = ctr - 1$
Building Predicate Maps

Trace

\[ pc_1: \ x = \text{ctr} \]
\[ pc_2: \ \text{ctr} = \text{ctr} + 1 \]
\[ pc_3: \ y = \text{ctr} \]
\[ pc_4: \ \text{assume}(x = i-1) \]
\[ pc_5: \ \text{assume}(y \neq i) \]

Trace Formula

\[ x_1 = \text{ctr}_0 \]
\[ \land \ \text{ctr}_1 = \text{ctr}_0 + 1 \]
\[ \land \ y_1 = \text{ctr}_1 \]
\[ \land \ x_1 = i_0 - 1 \]
\[ \land \ y_1 \neq i_0 \]

Predicate Map

\[ pc_2: \ x = \text{ctr} \]
\[ pc_3: \ x = \text{ctr} - 1 \]
\[ pc_4: \ y = x + 1 \]
\[ pc_5: \ y = i \]

• Cut + Interpolate at each point
• Pred. Map: \[ pc_i \mapsto \text{Interpolant from cut } i \]
Building Predicate Maps

<table>
<thead>
<tr>
<th>Trace</th>
<th>Trace Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pc_1$: $x = ctr$</td>
<td>$x_1 = ctr_0$</td>
</tr>
<tr>
<td>$pc_2$: $ctr = ctr + 1$</td>
<td>$\land ctr_1 = ctr_0 + 1$</td>
</tr>
<tr>
<td>$pc_3$: $y = ctr$</td>
<td>$\land y_1 = ctr_1$</td>
</tr>
<tr>
<td>$pc_4$: $\text{assume}(x = i-1)$</td>
<td>$\land x_1 = i_0 - 1$</td>
</tr>
<tr>
<td>$pc_5$: $\text{assume}(y \neq i)$</td>
<td>$\land y_1 \neq i_0$</td>
</tr>
</tbody>
</table>

Theorem: *Predicate map* makes trace *abstractly infeasible*
Plan

1. Motivation

2. Refinement using Traces
   - Simple
   - Procedure calls

3. Results
Traces with Procedure Calls

Trace Formula

\[ \text{Find predicate needed at point } i \]
Interprocedural Analysis

Trace

Trace Formula

Require at each point \( i \):
- **Well-scoped** predicates
- **YES**: Variables **visible** at \( i \)
- **NO**: Caller’s local variables

Find predicate needed at point \( i \)

Procedure Summaries [Reps, Horwitz, Sagiv ’95]
Polymorphic Predicate Abstraction [Ball, Millstein, Rajamani ’02]
Problems with Cutting

Trace

Trace Formula

Caller variables common to $\psi^-$ and $\psi^+$

• Unsuitable interpolant: not well-scoped
Interprocedural Cuts

Trace

Trace Formula

Call begins

-i
Interprocedural Cuts

Trace

Trace Formula

Predicate at $pc_i = \text{Interpolant from cut } i$
Predicate at $pc_i$ = Interpolant from i-cut
Plan

1. Motivation

2. Refinement using Traces
   - Simple
   - Procedure calls

3. Results
Implementation

• Algorithms implemented in BLAST
  – Verifier for C programs, Lazy Abstraction [POPL ’02]

• FOCI : Interpolating decision procedure

• Examples:
  – Windows Device Drivers (DDK)
  – IRP Specification: 22 state FSM
  – Current: Security properties of Linux programs
<table>
<thead>
<tr>
<th>Program</th>
<th>LOC*</th>
<th>Previous Time</th>
<th>New Time</th>
<th>Predicates Total</th>
<th>Predicates Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>kbfiltr</td>
<td>12k</td>
<td>1m12s</td>
<td>3m48s</td>
<td>72</td>
<td>6.5</td>
</tr>
<tr>
<td>floppy</td>
<td>17k</td>
<td>7m10s</td>
<td>25m20s</td>
<td>240</td>
<td>7.7</td>
</tr>
<tr>
<td>diskperf</td>
<td>14k</td>
<td>5m36s</td>
<td>13m32s</td>
<td>140</td>
<td>10</td>
</tr>
<tr>
<td>cdaudio</td>
<td>18k</td>
<td>20m18s</td>
<td>23m51s</td>
<td>256</td>
<td>7.8</td>
</tr>
<tr>
<td>parport</td>
<td>61k</td>
<td>DNF</td>
<td>74m58s</td>
<td>753</td>
<td>8.1</td>
</tr>
<tr>
<td>parclass</td>
<td>138k</td>
<td>DNF</td>
<td>77m40s</td>
<td>382</td>
<td>7.2</td>
</tr>
</tbody>
</table>

* Pre-processed
Windows DDK

IRP

Localizing works…

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC*</th>
<th>Previous Time</th>
<th>New Time</th>
<th>Predicates Total</th>
<th>Predicates Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>kbfiltr</td>
<td>12k</td>
<td>1m12s</td>
<td>3m48s</td>
<td>72</td>
<td>6.5</td>
</tr>
<tr>
<td>floppy</td>
<td>17k</td>
<td>7m10s</td>
<td>25m20s</td>
<td>240</td>
<td>7.7</td>
</tr>
<tr>
<td>diskperf</td>
<td>14k</td>
<td>5m36s</td>
<td>13m32s</td>
<td>140</td>
<td>10</td>
</tr>
<tr>
<td>cdaudio</td>
<td>18k</td>
<td>20m18s</td>
<td>23m51s</td>
<td>256</td>
<td>7.8</td>
</tr>
<tr>
<td>parport</td>
<td>61k</td>
<td>DNF</td>
<td>74m58s</td>
<td>753</td>
<td>8.1</td>
</tr>
<tr>
<td>parclass</td>
<td>138k</td>
<td>DNF</td>
<td>77m40s</td>
<td>382</td>
<td>7.2</td>
</tr>
</tbody>
</table>

* Pre–processed
Conclusion

- Scalability *and* Precision by *localizing*
- Craig Interpolation
  - Interprocedural cuts give well-scoped predicates

- Some Current and Future Work:
  - Multithreaded Programs
    - Project local info of thread to predicates over globals
  - Hierarchical trace analysis
Limitations of CEGAR

- Limited to powerset/relational abstract domains
- Interpolant computations
- Interactions with widening
- Starting on the right foot
- Unnecessary refinement steps
- Long and infinite number of refinement steps
- Long traces
Unnecessary Refinements

\[ x = 0 \]

while \((x < 10^6)\) do
  \[ x = x + 1 \]
assert \(x < 100\)
x = malloc();
y = x ;
while (...) 
    t = malloc();
    t->next = x

x = t;
...
while (x != y) do
    assert x != null;
    x = x->next
Long Traces

Example () {
1: c = 0;
2: for (i = 1; i < 1000; i++)
3: c = c + f(i);
4: if (a > 0) {
5:   if (x == 0) {
ERR: ;
    }
  }
}

• Assume f always terminates

• ERR is reachable
  – a and x are unconstrained

• Any feasible path to error must unroll the loop 1000 times AND find feasible paths through f

• Any other path must be dismissed as a false positive
Long Traces

Example ( ) {
1: c = 0;
2: for(i=1; i<1000; i++)
3: c = c + f(i);
4: if (a>0) {
  5: if (x==0) {
     ERR: ;
  }
}
}

• Intuitively, the for loop is irrelevant

• **ERR** reachable as long as there exists some path from 2 to 4 that does not modify a or x

• Can we use static analysis to precisely report a statement is reachable *without* finding a feasible path?
Example ( ) {
  1: c = 0;
  2: for (i = 1; i < 1000; i++)
  3:   c = c + f(i);
  4: if (a > 0) {
  5:     if (x == 0) {
ERR: ;
      }
  }
}
Path Slice (PLDI’05)

The **path slice** of a program path $\pi$ is a subsequence of the edges of $\pi$ such that if the sequence of operations along the subsequence is:

1. **infeasible**, then $\pi$ is **infeasible**, and
2. **feasible**, then the last location of $\pi$ is **reachable** (but not necessarily along $\pi$)