Counter Example
Guided Refinement
CEGAR

Mooly Sagiv
A problem has been detected and Windows has been shut down to prevent damage to your computer.

IRQL_NOT_LESS_OR_EQUAL

If this is the first time you have seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to be sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need. If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing.

If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x00000000A 0xFFFFFA80280010A, 0x0000000000000002, 0x0000000000000000, 0xFFFF8000185E251)
SLAM

- Microsoft blames most Windows crashes on third party device drivers
- The Windows device driver API is quite complicated
- Drivers are low level C code
- SLAM: Tool to automatically check device drivers for certain errors
- SLAM is shipped with Device Driver Development Kit
- Full detail available at http://research.microsoft.com/slam/
Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we’re building tools that can do actual proof about the software and how it works.
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)

z = 5

[y ≥ 0]

assert x > 0
A Simple Example

\[ z = 5 \]

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

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

\[
\begin{bmatrix}
\text{x} & \rightarrow & \text{T}, \text{y} & \rightarrow & \text{T} \\
\text{z} & = & 5
\end{bmatrix}
\]

\[
\begin{bmatrix}
\text{x} & \rightarrow & \text{T}, \text{y} & \rightarrow & \text{T} \\
\text{y} & > & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
\text{x} & \rightarrow & \text{T}, \text{y} & \rightarrow & \text{P} \\
\text{x} & = & z
\end{bmatrix}
\]

\[
\begin{bmatrix}
\text{x} & \rightarrow & \text{T}, \text{y} & \rightarrow & \text{P} \\
\text{x} & = & -y
\end{bmatrix}
\]

\[
\begin{bmatrix}
\text{T} & \quad & \text{F} \\
\text{x} & \rightarrow & \text{T}, \text{y} & \rightarrow & \text{P} \\
\text{x} & \rightarrow & \text{P}, \text{y} & \rightarrow & \text{N}
\end{bmatrix}
\]

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

\[ z = 5 \]

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

sign(\( x \)), sign(\( y \)), sign(\( z \))

\[
\begin{align*}
[x &\rightarrow T, y \rightarrow T, z \rightarrow T] \\
z = 5 &
\end{align*}
\]

\[
\begin{align*}
[x &\rightarrow T, y \rightarrow T, z \rightarrow P] \\
y > 0 &
\end{align*}
\]

\[
\begin{align*}
[x &\rightarrow T, y \rightarrow P, z \rightarrow P] \\
T &
\end{align*}
\]

\[
\begin{align*}
[x &\rightarrow P, y \rightarrow P, z \rightarrow P] \\
F &
\end{align*}
\]

\[
\begin{align*}
[x &\rightarrow P, y \rightarrow P, z \rightarrow P] \\
T &
\end{align*}
\]

\[
\begin{align*}
[x &\rightarrow P, y \rightarrow N, z \rightarrow P] \\
T &
\end{align*}
\]

\[
\begin{align*}
[x &\rightarrow P, y \rightarrow T, z \rightarrow P] \\
T &
\end{align*}
\]

assert \( x > 0 \)
Simple Example (local abstractions)

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

```
sign(x), sign(y), sign(z)  []
```

```
[y⇒P, z⇒P]
```

```
x = z
```

```
[x ⇒P]
```

```
assert x > 0
```

```
[y⇒N]
```

```
x = -y
```

```
[x ⇒P]
```

```
assert x > 0
```

Plan

• Predicate Abstraction
• CEGAR in BLAST (inspired by SLAM) POPL’04
• Limitations
BLAST

Berkeley Lazy Abstraction Software Tool

www.eecs.berkeley.edu/~blast/
Abstractions from Proofs: POPL’04

Ranjit Jhala
UCSD

Rupak Majumdar
MPI

Ken McMillan
MSR

Thomas Henzinger
IST
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
Q1: *Which predicates* are required to verify a property?
The Predicate Abstraction Domain

• Fixed set of predicates Pred
• The meaning of each predicate $p_i \in \text{Pred}$ is a closed first order formula $f_i$
• The relational domain is $<P(P(\text{Pred})), \emptyset, P(\text{Pred}), \cup, \cap>$
  – Join is set union
  – State space explosion
• Special case of canonic abstraction
A Simple Example

Predicates: \( p_1 = x > 0 \) \( p_2 = y \geq 0 \)

```c
int x, y;
x = 1;
y = 2;
while (*) do {
    x = x + y;
}
assert x > 0;
```

```c
bool p1, p2;
p1 = true;
p2 = true;
while (*) do {
    p1 = (p1&&p2 ? 1 : *);
}
assert p1;
```
do {
    KeAcquireSpinLock();
    nPacketsOld = nPackets;
    if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
    }
} while (nPackets != nPacketsOld);
KeReleaseSpinLock();
do {
    KeAcquireSpinLock();
    if(*){
        KeReleaseSpinLock();
    }
} while (*);
KeReleaseSpinLock();
do {
    KeAcquireSpinLock();
    nPacketsOld = nPackets;
    if (request) {
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
    }
} while (nPackets != nPacketsOld);
KeReleaseSpinLock();

SLAM Example

Is error path feasible in C program? (newton)
do {
    KeAcquireSpinLock();
    nPacketsOld = nPackets; b = true;
    if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++; b = !b;
    }
} while (nPackets != nPacketsOld); !b
KeReleaseSpinLock();
b : (nPacketsOld == nPackets)

do {
    KeAcquireSpinLock();
    b = true;
    if(*){
        KeReleaseSpinLock();
        b = b ? false : *;
    }
}
while (!b);

KeReleaseSpinLock();

SLAM Example

Model checking refined boolean program (bebop)
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 (p_n) lock();
      if (p_n) unlock();
}

Only track \textit{lock}

Bogus Counterexample
- Must \textit{correlate} branches

Predicate \( p_1 \) makes trace \textit{abstractly infeasible}

\( p_i \) required for verification
Example

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

Bogus Counterexample
- Must *correlate* branches

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

How can we get scalable verification?
By Localizing Precision

```
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();
}
```

Preds. Used locally

Ex: 2 * n states

Preds. used globally

Ex: 2^n states

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 Program → Abstract → Check

- explanation
- Why infeasible?

Refine

Is model safe?

- YES: SAFE
- NO! (Trace): BUG

[Clarke et al. ’00]
[Kurshan et al. ’93]
[Ball, Rajamani ’01]
Counterexample Guided Refinement

Seed Abstraction Program → Abstract → Check

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

Is model safe?
Counterexample Guided Refinement

Seed Abstraction Program → Abstract → Check

- Abstract
- Check
  - Is model safe?
    - YES
      - SAFE
    - NO!
      - (Trace)
        - Why infeasible?
          - explanation
            - feasible
              - BUG
            - infeasible
              - SAFE

- Refine
This Talk: Counterexample Analysis

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

Seed Abstraction Program → Abstract → Check → Is model safe?

Abstract:
- explanation
- Why infeasible?

Check:
- NO! (Trace)

Refine:
- feasible
- 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>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>$\overline{[F]}$</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?

Trace → Refine
Feasible
Explanation of Infeasibility

SSA → Thm Pvr
Trace Feasibility Formula
Proof of Unsat.

Extract
Feasible
Predicate Map: Prog Ctr ! Predicates
Counterexample Analysis

Q0: Is trace feasible?

Q1: What predicates remove trace?

Q2: Where are preds required?

Trace → Refine → Feasible → Explanation of Infeasibility

SSA → Trace Feasibility Formula → Thm Pvr → Proof of Unsat. → Extract → Feasible

Predicate Map: Prog Ctr ! Predicates
Traces

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

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

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

Trace Feasibility Formula

\[ pc_1: x_1 = ctr_0 \]
\[ pc_2: ctr_1 = ctr_0 + 1 \]
\[ pc_3: y_1 = ctr_1 \]
\[ pc_4: \text{assume}(x_1 = i_0 - 1) \]
\[ pc_5: \text{assume}(y_1 \neq i_0) \]

\[ x_1 = ctr_0 \]
\[ \land \quad ctr_1 = ctr_0 + 1 \]
\[ \land \quad y_1 = ctr_1 \]
\[ \land \quad x_1 = i_0 - 1 \]
\[ \land \quad y_1 \neq i_0 \]

Theorem: Trace is \textbf{Feasible}, TFF is \textbf{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 → Explanation of Infeasibility

Trace → SSA → Trace Feasibility Formula → Thm Pvr → Proof of Unsat.

Feasible → Extract → Predicate Map: Prog Ctr ! 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

Proof of Unsat.

Thm Pvr

Extract

Predicate Map: Prog Ctr ! Predicates
Proof of Unsatisfiability

\[ x_1 = ctr_0 \]
\[ \wedge \text{ctr}_1 = \text{ctr}_0 + 1 \]
\[ \wedge y_1 = \text{ctr}_1 \]
\[ \wedge x_1 = i_0 - 1 \]
\[ \wedge y_1 \neq i_0 \]

Trace Formula

**PROBLEM**

Proof uses entire *history* of execution

- Information flows up and down

No *localized* or *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 \textit{prefix}
2. ... knows \textit{present values} of variables
3. ... makes trace \textit{suffix} infeasible

At \textit{pc}_4, which predicate on \textit{present state} shows infeasibility of \textit{suffix}?
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)

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

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?

Trace

\( pc_1: x = ctr \)

\( pc_2: ctr = ctr + 1 \)

\( pc_3: y = ctr \)

\( pc_4: \text{assume}(x = i-1) \)

\( pc_5: \text{assume}(y \neq i) \)

Trace Formula (TF)

\[ x_1 = ctr_0 \]
\[ \land \quad ctr_1 = ctr_0 + 1 \]
\[ \land \quad y_1 = ctr_1 \]
\[ \land \quad x_1 = i - 1 \]
\[ \land \quad y_1 \neq i_0 \]

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?

Trace

$pc_1$: $x = \text{ctr}$

$pc_2$: $\text{ctr} = \text{ctr} + 1$

$pc_3$: $y = \text{ctr}$

$pc_4$: assume($x = i - 1$)

$pc_5$: assume($y \neq i$)

Trace Formula (TF)

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

State...

1. ... after executing trace $\text{prefix}$
2. ... knows $\text{present values}$ of variables
3. ... makes trace $\text{suffix}$ infeasible

Predicate...

... implied by TF $\text{prefix}$

... on $\text{common}$ variables

... & TF $\text{suffix}$ is 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 \textit{common} to $\psi^-$, $\psi^+$
3. $\Phi \land \psi^+$ is unsatisfiable
Craig’s Interpolation Theorem (take 2)

Given formulas $\psi^-, \neg\psi^+$ s.t. $\psi^-$ implies $\neg\psi^+$

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

1. $\psi^-$ implies $\Phi$ implies $\neg\psi^+$
2. $\Phi$ has symbols common to $\psi^-$, $\neg\psi^+$
Examples of Craig’s Interpolation

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

- $\psi^- = x_1 = \text{ctr}_0 \land \text{ctr}_1 = \text{ctr}_0 + 1 \land y_1 = \text{ctr}_1$
  $\psi^+ = x_1 = i_0 - 1 \land y_1 \neq i_0$
  $\quad \land y_1 = x_1 + 1$
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 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]
(boolean) SAT–based Model Checking [McMillan ’03]
Interpolant = Predicate!

**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**

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

**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. \( \Phi \land \psi^+ \) is *unsatisfiable*
Interpolant = Predicate !

Trace

$pc_1$: $x = \text{ctr}$

$pc_2$: $\text{ctr} = \text{ctr} + 1$

$pc_3$: $y = \text{ctr}$

$pc_4$: assume ($x = i-1$)

$pc_5$: assume ($y \neq i$)

Trace Formula

$\Psi^-$

$x_1 = \text{ctr}_0$

$^\wedge ctr_1 = ctr_0 + 1$

$^\wedge y_1 = ctr_1$

$^\wedge x_1 = i_0 - 1$

$^\wedge y_1 \neq i_0$

$\Psi^+$

$\Phi$

$y_1 = x_1 + 1$

Require:

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

Interpolant:

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

Trace

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

Trace Formula

\( x_1 = ctr_0 \)
\( \wedge ctr_1 = ctr_0 + 1 \)
\( \wedge y_1 = ctr_1 \)
\( \wedge x_1 = i_0 - 1 \)
\( \wedge y_1 \neq i_0 \)

Predicate at \( pc_4 \):
\( y = x + 1 \)

Interpolant:

1. \( \psi^- \) implies \( \Phi \)
2. \( \Phi \) has symbols \textit{common} to \( \psi^- \), \( \psi^+ \)
3. \( \Phi \land \neg \psi^+ \) is \textit{unsatisfiable}
# Building Predicate Maps

## Trace

<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: \ ctr = \text{ctr} + 1 )</td>
<td>( ctr_1 = \text{ctr}_0 + 1 )</td>
</tr>
<tr>
<td>( pc_3: \ y = \text{ctr} )</td>
<td>( y_1 = \text{ctr}_1 )</td>
</tr>
<tr>
<td>( pc_4: \ \text{assume}(x = i-1) )</td>
<td>( x_1 = i_0 - 1 )</td>
</tr>
<tr>
<td>( pc_5: \ \text{assume}(y \neq i) )</td>
<td>( y_1 \neq i_0 )</td>
</tr>
</tbody>
</table>

- Cut + Interpolate at *each* point
- Pred. Map: \( pc_i \mapsto \text{Interpolant from cut i} \)

**Predicate Map**

\( pc_2: x = \text{ctr} \)
# Building Predicate Maps

## Trace

<table>
<thead>
<tr>
<th>Trace</th>
<th>Trace Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>( pc_1 ):  ( x = 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 \ \text{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>

\[ \Psi^- \quad \text{Interpolate} \quad \Psi^+ \]

- 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 = \text{ctr} )</td>
<td>( x_1 = ctr_0 )</td>
</tr>
<tr>
<td>( pc_2: \ \text{ctr} = \text{ctr} + 1 )</td>
<td>( \land \ ctr_1 = ctr_0 + 1 )</td>
</tr>
<tr>
<td>( pc_3: \ y = \text{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>

- 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

\[ pc_1: \ x_1 = 3 \]
\[ pc_2: \ \text{assume} \ (x_1 > 0) \]
\[ pc_3: \ x_3 = f_{\pi}^1(x_1)(x_1) \]
\[ pc_4: \ y_1 = y_1 \]
\[ pc_5: \ y_3 = f_{\pi}^2(y_2) = f_2(y_2) \]
\[ pc_6: \ z_2 = z_2 + 1 \]
\[ pc_7: \ z_3 = 2z_2 \]
\[ pc_8: \ \text{return} \ z_3 \]
\[ pc_9: \ \text{return} \ y_3 \]
\[ pc_{10}: \ x_4 = x_3 + 1 \]
\[ pc_{11}: \ x_5 = f_{\pi}^3(x_4)(x_4) \]
\[ pc_{12}: \ \text{assume} \ (w_1 < 5) \]
\[ pc_{13}: \ \text{return} \ w_1 \]
\[ pc_{14}: \ \text{assume} \ x_4 > 5 \]
\[ pc_{15}: \ \text{assume} \ (x_1 - x_5 > 2) \]

Trace Formula

Find predicate needed at point i
Interprocedural Analysis

Procedure Summaries [Reps, Horwitz, Sagiv '95]
Polymorphic Predicate Abstraction [Ball, Millstein, Rajamani '02]

Trace

Find predicate needed at point i

Trace Formula

Require at each point i:

Well-scoped predicates
YES: Variables \textit{visible} at i
NO: Caller’s local variables
Problems with Cutting

**Caller variables** common to $\psi^-$ and $\psi^+$
- Unsuitable interpolant: not well-scoped
Interprocedural Cuts

Trace

Trace Formula

Call begins

\[ i \]
Interprocedural Cuts

Trace

[Diagram of Trace]

Trace Formula

[Diagram of Trace Formula with symbols $\psi^+$ and $\psi^-$]

Predicate at $pc_i = \text{Interpolant from cut i}$
Predicate at $\text{pc}_i = \text{Interpolant from i-cut}$
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
Windows DDK

IRP
22 state

Results

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

*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
Unsuccessful Refinement Set

\[
x = \text{malloc}(); \\
y = x; \\
\text{while (\ldots)} \\
\hspace{1cm} t = \text{malloc}(); \\
\hspace{1cm} t->\text{next} = x \\
x = t; \\
\]
\[\ldots\]
\[\text{while (x \neq y) do}\]
\[\hspace{1cm} \text{assert x \neq null;}\]
\[\hspace{1cm} x = x->\text{next}\]
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
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:  ;
    }
  }
}
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: ;
   }
}
}
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 $\pi$)