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: 0x0000000A 0xffffffff802880010A,
0x0000000000000002, 0x0000000000000000, 0xffffffff8000185e251)
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

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

```
assert x > 0
```

```
sign(x)
```

```
[x \mapsto T]
```

```
z = 5
[x \mapsto T]
```

```
y \geq 0
[x \mapsto P]
```

```
[x \mapsto T]
```

```
x = z
```

```
T
```

```
F
```

```
[x \mapsto P]
```

```
x = -y
```

```
[x \mapsto T]
```

```
assert x > 0
```

```
[x \mapsto T]
```

```
[x \mapsto T]
```

A Simple Example

\[
z = 5
\]

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

\[
\text{else } \quad x = -y;
\]

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

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

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

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

T

F

x = z

x = −y

assert x > 0
Simple Example (local abstractions)

\texttt{z = 5}
\texttt{if \ (y > 0)\ x = z;}
\texttt{else\ x = -y;}
\texttt{assert \ x > 0}

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

\texttt{\begin{align*}
\text{if } (y > 0) \ &\ x = z; \\
\text{else} \ &\ x = -y; \\
\text{assert } x > 0
\end{align*}}
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 $\text{Pred}$
- The meaning of each predicate $p_i \in \text{Pred}$ is a closed first order formula $f_i$
- The relational domain is
  $<\text{P(\text{P(\text{Pred}})), \emptyset, \text{P(\text{Pred}}), \cup, \cap>$
  - 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;
```
```c
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();

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

Bogus Counterexample
- Must correlate branches

Predicate $p_1$ makes trace abstractly infeasible

$p_i$ required for verification
Only track \textit{lock}

Track \textit{lock, }\textit{p}_i\textit{ s}

\textbf{Bogus Counterexample}
- Must \textit{correlate} branches

\textbf{State Explosion}
- \( > 2^n \) distinct states
- intractable

\textbf{How can we get scalable verification ?}
By Localizing Precision

Q2: Where are the predicates required?

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
Counterexample Guided Refinement

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

Seed Abstraction Program → Abstract → Check → Refine

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

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

Seed Abstraction Program

Abstract

Check

Refine

Why infeasible?

explanation

NO! (Trace)

Is model safe?

SAFE

BUG

feasible

feastible
Counterexample Guided Refinement

Seed Abstraction
Program

Abstract

Check

Refine

Is model safe?

Why infeasible?

NO! (Trace)

feasible

SAFE

BUG

YES

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)
- Refine
- Is model safe?
- YES
- SAFE
- BUG
- feasible
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>States satisfying $F { s \mid s \vDash F }$</th>
<th>$F$ FO formula over prog. vars</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[F]$</td>
<td>$F$</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

Trace → Refine

Feasible

Explanation of Infeasibility

Q0: Is trace feasible?

Q1: What predicates remove trace?

Q2: Where are preds required?

SSA → Thm Pvr

Trace Feasibility Formula

Proof of Unsat.

Feasible

Predicates Map: Prog Ctr!Predicates
Counterexample Analysis

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

Trace → Refine
Refine → Feasible
Refine → Explanation of Infeasibility

Trace → SSA
SSA → Trace Feasibility Formula

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

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

\[pc_1: \ x_1 = \text{ctr}_0\]
\[pc_2: \ \text{ctr}_1 = \text{ctr}_0 + 1\]
\[pc_3: \ y_1 = \text{ctr}_1\]
\[pc_4: \ \text{assume}(x_1 = i_0 - 1)\]
\[pc_5: \ \text{assume}(y_1 \neq i_0)\]

Trace Feasibility Formula

Theorem: Trace is \textbf{Feasible}, TFF is \textbf{Satisfiable}

Compact Verification Conditions [Flanagan, Saxe ’00]
Q0: Is trace feasible?

Q1: What predicates remove trace?

Q2: Where are preds required?

Counterexample Analysis

Refine

Trace

Feasible

Explanation of Infeasibility

SSA

Trace Feasibility Formula

Proof of Unsat.

Thm Pvr

Feasible

Extract

Predicate Map: Prog Ctr ! Predicates
Trace

Refine

Feasible
Explanation of Infeasibility

Q0: Is trace feasible?

Q1: What predicates remove trace?

Q2: Where are preds required?

SSA

Trace Feasibility Formula

Proof of Unsat.

Thm Pvr

Y

N

Extract

Feasible

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 *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 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?

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)

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

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

$pc_1$: $x = ctr$

$pc_2$: $ctr = ctr + 1$

$pc_3$: $y = ctr$

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

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

Trace Formula (TF)

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

State...

1. … after executing trace **prefix**

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

3. … makes trace **suffix** infeasible

Predicate ...

... implied by TF **prefix**

... on **common** variables
What Predicate is needed?

Trace

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

Trace Formula (TF)

\[
\begin{align*}
  x_1 &= ctr_0 \\
  \land \quad ctr_1 &= ctr_0 + 1 \\
  \land \quad y_1 &= ctr_1 \\
  \land \quad x_1 &= i_0 - 1 \\
  \land \quad y_1 &\neq i_0
\end{align*}
\]

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

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

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 \textit{prefix}

2. … knows \textit{present values} of variables

3. … makes trace \textit{suffix} infeasible

Predicate ...

… implied by TF \textit{prefix}

… on \textit{common} variables

… & 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
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

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

- $\psi^- = x_1 = \text{ctr}_0 \land \text{ctr}_1 = \text{ctr}_0 + 1 \land y_1 = \text{ctr}_1$
- $\psi^+ = x_1 = i_0 - 1 \land y_1 \neq i_0$
- $y_1 = x_1 + 1$
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 $= \text{Predicate}$ !

### Trace

<table>
<thead>
<tr>
<th>(pc_1):</th>
<th>(x = \text{ctr})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(pc_2):</td>
<td>(\text{ctr} = \text{ctr} + 1)</td>
</tr>
<tr>
<td>(pc_3):</td>
<td>(y = \text{ctr})</td>
</tr>
<tr>
<td>(pc_4):</td>
<td>(\text{assume}(x = i - 1))</td>
</tr>
<tr>
<td>(pc_5):</td>
<td>(\text{assume}(y \neq i))</td>
</tr>
</tbody>
</table>

### Trace Formula

- \(x_1 = \text{ctr}_0\)
- \(\text{ctr}_1 = \text{ctr}_0 + 1\)
- \(y_1 = \text{ctr}_1\)
- \(x_1 = i_0 - 1\)
- \(y_1 \neq i_0\)

### Interpolant:

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

### Require:

1. Predicate \textit{implied} by trace \textit{prefix}
2. Predicate on \textit{common} variables
   \(\text{common} = \text{current value}\)
3. Predicate \& \textit{suffix} yields a \textit{contradiction}
Interpolant = Predicate!

Trace

\( \text{pc}_1: \ x = \text{ctr} \)
\( \text{pc}_2: \ \text{ctr} = \text{ctr} + 1 \)
\( \text{pc}_3: \ y = \text{ctr} \)
\( \text{pc}_4: \ \text{assume}(x = i-1) \)
\( \text{pc}_5: \ \text{assume}(y \neq i) \)

Trace Formula

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

\( \phi \) is interpolant

Require:

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

Interpolant:

1. \( \psi^- \) \textit{implies} \( \phi \)
2. \( \phi \) has symbols \textit{common} to \( \psi^- \), \( \psi^+ \)
3. \( \phi \land \psi^+ \) is \textit{unsatisfiable}
Interpolant = Predicate!

Trace

$\text{pc}_1$: $x = \text{ctr}$

$\text{pc}_2$: $\text{ctr} = \text{ctr} + 1$

$\text{pc}_3$: $y = \text{ctr}$

$\text{pc}_4$: assume($x = i - 1$)

$\text{pc}_5$: 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 at $\text{pc}_4$:

$y = x + 1$

Interpolant:

1. $\psi^-$ implies $\Phi$

2. $\Phi$ has symbols common to $\psi^-$, $\psi^+$

3. $\Phi \land \lnot \psi^+$ is unsatisfiable
Building Predicate Maps

Trace Formula

<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 $i$
### 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

<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
### 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>

**Predicate Map**

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

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

1. Motivation

2. Refinement using Traces
   • Simple
   • Procedure calls

3. Results
Traces with Procedure Calls

Find predicate needed at point $i$
Interprocedural Analysis

Trace

Trace Formula

Find predicate needed at point i

Require at each point i:

- Well-scoped predicates
- YES: Variables visible at i
- NO: Caller’s local variables

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

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

IRP

22 state

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 \textit{and} Precision by \textit{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

\[
x = 0 \\
\text{while (}x < 10^6\text{) do} \\
\quad x = x + 1 \\
\text{assert } x < 100
\]
Unsuccessful Refinement Set

```c
x = malloc();
y = x ;
while (...)
    t = malloc();
    t->next = x
    x = t;
...
while (x != y) do
    assert x != null;
    x = x->next
```
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: ;
  6:     }
  7: }
  8: }

• 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 ( ) {
7: 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 $\pi$)
Limitations of CEGAR

- Interpolants
- Non-disjunctive analysis
• CEGAR can be effective heuristics for finding abstraction
• More methods later