Predicate Abstraction with SATABS

Outline
Introduction
Existential Abstraction
Predicate Abstraction for Software
Counterexample-Guided Abstraction Refinement
Computing Existential Abstractions of Programs
Checking the Abstract Model
Simulating the Counterexample
Refining the Abstraction

"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 in order to guarantee the reliability."

Bill Gates, April 18, 2002
Keynote address at WinHec 2002

"One of the least visible ways that Microsoft Research contributed to Vista, but something I like to talk about, is the work we did on what’s called the Static Driver Verifier. People who develop device drivers for Vista can verify the properties of their drivers before they ever even attempt to test that. What’s great about this technology is there is no testing involved. For the properties that it is proving, they are either true or false. You don’t have to ask yourself "Did I come up with a good test case or not?"

Rick Rashid, Microsoft Research chief
father of CMU’s Mach Operating System (Mac OS X)
news.cnet.com interview, 2008

Model Checking with Predicate Abstraction
▶ A heavy-weight formal analysis technique
▶ Recent successes in software verification, e.g., SLAM at Microsoft
▶ The abstraction reduces the size of the model by removing irrelevant detail

Goal: make the abstract model small enough for an analysis with a BDD-based Model Checker
▶ Idea: only track predicates on data, and remove data variables from model
▶ Mostly works with control-flow dominated properties
Predicate Abstraction as Abstract Domain

We are given a set of predicates over $S$, denoted by $\Pi_1, \ldots, \Pi_n$.

- An abstract state is a valuation of the predicates:
  $$\hat{S} = B^n$$
- The abstraction function:
  $$\alpha(s) = (\Pi_1(s), \ldots, \Pi_n(s))$$

Existential Abstraction

**Definition (Existential Abstraction)**

A model $\hat{M} = (\hat{S}, \hat{S}_0, \hat{T})$ is an **existential abstraction** of $M = (S, S_0, T)$ with respect to $\alpha : S \rightarrow \hat{S}$ iff
- $\exists s \in S_0, \alpha(s) = \hat{s} \quad \Rightarrow \quad \hat{s} \in \hat{S}_0$ and
- $\exists (s, s') \in T. \alpha(s) = \hat{s} \wedge \alpha(s') = \hat{s'} \quad \Rightarrow \quad (\hat{s}, \hat{s'}) \in \hat{T}$.

**Minimal Existential Abstractions**

There are obviously many choices for an existential abstraction for a given $\alpha$.

**Definition (Minimal Existential Abstraction)**

A model $\hat{M} = (\hat{S}, \hat{S}_0, \hat{T})$ is the **minimal existential abstraction** of $M = (S, S_0, T)$ with respect to $\alpha : S \rightarrow \hat{S}$ iff
- $\exists s \in S_0, \alpha(s) = \hat{s} \quad \iff \quad \hat{s} \in \hat{S}_0$ and
- $\exists (s, s') \in T. \alpha(s) = \hat{s} \wedge \alpha(s') = \hat{s'} \quad \iff \quad (\hat{s}, \hat{s'}) \in \hat{T}$.

This is the most precise existential abstraction.

**Existential Abstraction**

We write $\alpha(\pi)$ for the abstraction of a path $\pi = s_0, s_1, \ldots$:

$$\alpha(\pi) = \alpha(s_0), \alpha(s_1), \ldots$$

**Lemma**

Let $\hat{M}$ be an existential abstraction of $M$. The abstraction of every path (trace) $\pi$ in $M$ is a path (trace) in $\hat{M}$.

$$\pi \in M \quad \Rightarrow \quad \alpha(\pi) \in \hat{M}$$

Proof by induction.

We say that $\hat{M}$ **overapproximates** $M$. 

**Notation for Abstractions**

**Abstract Domain**

Approximate representation of sets of concrete values

$$S \xrightarrow{\alpha} \hat{S}$$

**Predicate Abstraction: the Basic Idea**

Concrete states over variables $x, y$:

- $x = 1$, $y = 0$
- $x = 1$, $y = 1$
- $x = 1$, $y = 2$
- $x = 2$, $y = \ldots$ ⇐ $x > y$
- $\hat{p}_2 \iff y = 0$

Predicates:

- $p_1 \iff x > y$
- $p_2 \iff y = 0$

Abstract Transitions?
Abstracting Properties

Reminder: we are using
- a set of atomic propositions (predicates) \( A \), and
- a state-labelling function \( L : S \rightarrow \mathcal{P}(A) \)
in order to define the meaning of propositions in our properties.

We define an abstract version of it as follows:
- First of all, the negations are pushed into the atomic propositions.
  E.g., we will have
  \[ x = 0 \in A \]
  and
  \[ x \neq 0 \in A \]

An abstract state \( \hat{s} \) is labelled with \( a \in A \) iff all of the corresponding concrete states are labelled with \( a \).

\[
a \in \hat{L}(\hat{s}) \iff \forall s | a(s) = \hat{s}, a \in L(s)
\]

This also means that an abstract state may have neither the label \( x = 0 \) nor the label \( x \neq 0 \) – this may happen if it concretizes to concrete states with different labels!

Conservative Abstraction

The keystone is that existential abstraction is conservative for certain properties:

**Theorem (Clarke/Grumberg/Long 1994)**

Let \( \phi \) be a \( \forall \)\(\text{CTL}^* \) formula where all negations are pushed into the atomic propositions, and let \( \hat{M} \) be an existential abstraction of \( M \). If \( \phi \) holds on \( \hat{M} \), then it also holds on \( M \).

\[
\hat{M} \models \phi \Rightarrow M \models \phi
\]

We say that an existential abstraction is conservative for \( \forall \)\(\text{CTL}^* \) properties. The same result can be obtained for LTL properties.

The proof uses the lemma and is by induction on the structure of \( \phi \). The converse usually does not hold.

Back to the Example

We hope: computing \( \hat{M} \) and checking \( \hat{M} \models \phi \) is easier than checking \( M \models \phi \).

\[
\begin{align*}
\text{p1, p2} & \quad \text{p1, p2} & \quad \text{p1, p2} \\
x = 2 & \quad x = 2 & \quad x = 2 \\
y = 0 & \quad y = 1 & \quad y = 0 \\
\end{align*}
\]
Let's try a Property

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Property: $x > y \lor y \neq 0 \iff p_1 \lor \neg p_2$

Another Property

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Property: $x > y \iff p_1$

But: the counterexample is spurious

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/

SLIC

- Finite state language for defining properties
- Monitors behavior of C code
- Temporal safety properties (security automata)
- Familiar C syntax
- Suitable for expressing control-dominated properties
- E.g., proper sequence of events
- Can track data values

SLIC Example

```c
enum {Locked, Unlocked}
s = Unlocked;
KeAcquireSpinLock . entry { if (s == Locked) abort;
else s = Locked;
}
KeReleaseSpinLock . entry { if (s == Unlocked) abort;
else s = Unlocked;
}
```

Refinement Example

```c
do {
KeAcquireSpinLock ();
nPacketsOld = nPackets;
if (request) {
request = request -> Next;
KeReleaseSpinLock ();
nPackets++;}
} while (nPackets != nPacketsOld);
KeReleaseSpinLock ()
```
Refinement Example
\[
\text{do }
\begin{align*}
\&\text{KeAcquireSpinLock();} \\
\&\text{if } (\ast) \\
\&\text{KeReleaseSpinLock();} \\
\end{align*}
\] while(\ast);
KeReleaseSpinLock();
Is this path concretizable?

Refinement Example
\[
\text{do }
\begin{align*}
\&\text{KeAcquireSpinLock();} \\
\&\text{nPacketsOld = nPackets; } \\
\&\text{if } (\text{request}) \\
\&\text{KeReleaseSpinLock();} \\
\&\text{nPackets++;} \\
\end{align*}
\] while(nPackets != nPacketsOld);
KeReleaseSpinLock();
This path is spurious!
Let's add the predicate nPacketsOld==nPackets

CEGAR Overview
C program
1.) Compute Abstraction
2.) Check Abstraction
[no error] OK
3.) Check Feasibility
[feasible] report counterexample
4.) Refine Predicates

Counterexample-guided Abstraction Refinement

Claims:
1. This never returns a false error.
2. This never returns a false proof.
3. This is complete for finite-state models.
4. But: no termination guarantee in case of infinite-state systems
Computing Existential Abstractions of Programs

1.) Compute Abstraction
2.) Check Abstraction
3.) Check Feasibility
4.) Refine Predicates

[no error] OK [feasible] report counterexample

C program

```c
void main() {  
  bool p1, p2;  
  p1=TRUE;  p2=TRUE;  
  while (p2) {  
    p1= p1 ? FALSE : *;  
    p2= ip2;  
  }  
}
```

C Program Predicates Boolean Program

Minimal?

Predicate Images

Reminder:

$$\text{Image}(X) = \{ s' \in S \mid \exists s \in X. T(s, s') \}$$

We need

$$\widehat{\text{Image}}(\widehat{X}) = \{ \widehat{s}' \in \widehat{S} \mid \exists \widehat{s} \in \widehat{X}. \widehat{T}(\widehat{s}, \widehat{s}') \}$$

$$\widehat{\text{Image}}(\widehat{X})$$ is equivalent to

$$\{ \widehat{s}, \widehat{s}' \in \widehat{S}^2 \mid \exists s, s' \in S^2. \alpha(s) = \widehat{s} \land \alpha(s') = \widehat{s}' \land T(s, s') \}$$

This is called the predicate image of $$T$$.

Enumeration

Let’s take existential abstraction seriously

Basic idea: with $$n$$ predicates, there are $$2^n \cdot 2^n$$ possible abstract transitions

Let’s just check them!

Predicate Abstraction with SATABS – http://www.cprover.org/

Predicate Images

Computing the minimal existential abstraction can be way too slow

Use an over-approximation instead

Fast(er) to compute

But has additional transitions

Examples:

- Cartesian approximation (SLAM)
- FastAbs (SLAM)
- Lazy abstraction (Blast)
- Predicate partitioning (VCEGAR)

Enumeration: Example

Predicates

- $$p_1 \iff i = 1$$
- $$p_2 \iff i = 2$$
- $$p_3 \iff \text{even}(i)$$

Query to Solver

- $$i \neq 1 \land i \neq 2 \land \text{even}(i) \land$$
- $$i' = i + 1 \land$$
- $$i' \neq 1 \land i' \neq 2 \land \text{even}(i')$$

... and so on ...
Checking the Abstract Model

1.) Compute Abstraction
2.) Check Abstraction
3.) Check Feasibility
4.) Refine Predicates

C program

OK

report counterexample

Finite-State Model Checkers: SMV

1. Variables

VAR b0_argc_ge_1 : boolean;  -- argc >= 1
VAR b1_argc_le_2147483646 : boolean;  -- argc <= 2147483646
VAR b2 : boolean;  -- argv[argc] == NULL
VAR b3_nmemb_ge_r : boolean;  -- nmemb >= r
VAR b4 : boolean;  -- p1 == &array[0]
VAR b5_i_ge_8 : boolean;  -- i >= 8
VAR b6_i_ge_s : boolean;  -- i >= s
VAR b7 : boolean;  -- 1 + i >= 8
VAR b8 : boolean;  -- 1 + i >= s
VAR b9_s_g1_0 : boolean;  -- s > 0
VAR b10_s_g1_1 : boolean;  -- s > 1

2. Control Flow

- program counter: 56 is the "terminating" PC
- ASSGN init(PC):=0;  -- initial PC
- ASSGN next(PC):=case
  PC:=0: 1;  -- other
  PC:=1: 2;  -- other
  PC:=19: case goto (with guard)
  guard19: 26;
  i: 20;
esac;

3. Data

TRANS (PC=0) ->
  next(b0_argc_ge_1)=b0_argc_ge_1
  & next(b1_argc_le_214746)=b1_argc_le_21646
  & next(b2)=b2
  & (i>b3 | b36)
  & (i>=b7 | b8)
  & (i>=b0 | b42)
  & (i>=(b7 | b8 | b42 | b54)
  & (i>=b0 | b60)

TRANS (PC=1) ->
  next(b0_argc_ge_1)=b0_argc_ge_1
  & next(b1_argc_le_214746)=b1_argc_le_214746
  & next(b2)=b2
  & next(b3_nmemb_ge_r)=b3_nmemb_ge_r
  & next(b4)=b4
  & next(b5_i_ge_8)=b5_i_ge_8
  & next(b6_i_ge_s)=b6_i_ge_s

4. Property

- the specification
- file main.c line 20 column 12
- function c::very_buggy_function
- SPEC AG ((PC=51) -> !b23)
**Finite-State Model Checkers: SMV**

- If the property holds, we can terminate
- If the property fails, SMV generates a counterexample with an assignment for all variables, including the PC

**Simulating the Counterexample**

1. Compute Abstraction
2. Check Abstraction
3. Check Feasibility
4. Refine Predicates

[no error]
OK
Feasible
report counterexample
C program

**Lazy Abstraction**

- The progress guarantee is only valid if the minimal existential abstraction is used.
- Thus, distinguish spurious transitions from spurious prefixes.
- Refine spurious transitions separately to obtain minimal existential abstraction
- SLAM: Constrain

**Example Simulation**

```c
int main() {
    int x, y;
    y = 1;
    x = 1;
    if (y > x)
        y--;
    else
        y++;
    assert(y > x);
}
```

Predicate: \( y > x \)

```c
int main() {
    bool b0; // y>x
    b0 = 0;
    b0 = 0;
    if (b0)
        b0 = 0;
    else
        b0 = 0;
    assert(b0);
}
```

We now do a path test, so convert to SSA.
1.) Compute Abstraction
2.) Check Abstraction
3.) Check Feasibility
4.) Refine Predicates

\[ \text{[no error]} \]

\[ \text{OK} \]

\[ \text{feasible} \]

report counterexample

C program

Refining the Abstraction

Example Simulation

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 53

Refining the Abstraction

```
1.) Compute Abstraction
2.) Check Abstraction
3.) Check Feasibility
4.) Refine Predicates

[no error]

OK

[feasible]

report counterexample

C program

Refining the Abstraction

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Manual Proof!

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 51

Manual Proof!

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 53

An Alternative Proof

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 52

An Alternative Proof

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 51

An Alternative Proof

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 54

An Alternative Proof

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 50

An Alternative Proof

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 55

An Alternative Proof

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 56

An Alternative Proof

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 57

An Alternative Proof

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 58

An Alternative Proof

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 59

An Alternative Proof

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 60

An Alternative Proof

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 61

An Alternative Proof

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 62

An Alternative Proof

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 63

An Alternative Proof

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```

Predicates Abstraction with SATABS – http://www.cprover.org/ 64

Refinement Algorithms

Using WP

1. Start with failed guard \( G \)
2. Compute \( wp(G) \) along the path

Using SP

1. Start at beginning
2. Compute \( sp(\ldots) \) along the path

- Both methods eliminate the trace
- Advantages/disadvantages?

Predicate Localization

Example:

```
int x, y;

x = 10;
{ x = 10 }

y = x + 10;
{ y = 20 }

y = y + 10;
{ y = 30 }

assert(y = 30);
```

We really only want to track specific predicates at each location!

Predicate Abstraction with SATABS – http://www.cprover.org/ 49

Refinement Algorithms

```
int main() {
  int x, y;
  y = 1;
  {y = 1}
  x = 1;
  {x = 1 ∧ y = 1}
  if (y > x)
    y = y - 1;
  else
    assert(y > x);
}
```
Predicate Localization

- Track a **separate set of predicates** for each location

- Makes predicate image easier
- Makes simulation of transitions easier
- Makes the check of the abstract model easier

Predicate Refinement for Paths

Recall the decision problem we build for simulating paths:

\[
\begin{align*}
A_1 & \Rightarrow A'_1 & A_2 & \Rightarrow A'_2 & A_3 & \Rightarrow A'_3 & \cdots & A_n & \Rightarrow A'_n
\end{align*}
\]

- Given \( A_1, \ldots, A_n \) with \( \bigwedge_i A_i = \text{false} \)
- \( A'_0 = \text{true} \) and \( A'_n = \text{false} \)
- \( (A'_{i-1} \land A_i) \Rightarrow A'_i \) for \( i \in \{1, \ldots, n\} \)
- Finally, \( \text{Vars}(A'_1) \subseteq (\text{Vars}(A_1 \ldots A_i) \cap \text{Vars}(A_{i+1} \ldots A_n)) \)

Craig Interpolation for Linear Inequalities

\[
\begin{align*}
0 & \leq x & 0 & \leq y \\
0 & \leq c_1 x + c_2 y & \text{with } 0 & \leq c_1, c_2
\end{align*}
\]

- “Cutting-planes”
- Naturally arise in Fourier-Motzkin or Simplex
Split Provers

Idea:

- Each prover $P_i$ only knows $A_i$, but they exchange facts
- We require that each prover only exchanges facts with common symbols
- Plus, we restrict the facts exchanged to some language $L$

Approximating Loop Invariants: WP

```plaintext
int x, y;

x = y = 0;
while (x != 10) {
  x ++;
  y ++;
}
assert (y == 10);
```

The WP refinement results in

- `wp(x=y=0, true)` = $x = 0 \land y = 0$
- `wp(x++; y++; ...)` = $x = 1 \land y = 1$
- `wp(x++; y++; ...)` = $x = 2 \land y = 2$
  ...

assert ($y == 10$);

- × It won’t work if we replace 10 by $n$.
- × Also requires 10 iterations.

Approximating Loop Invariants: SP

```plaintext
int x, y;

x = y = 0;
while (x != 10) {
  x ++;
  y ++;
}
assert (y == 10);
```

The SP refinement results in

- `sp(x=y=0, true)` = $x = 0 \land y = 0$
- `sp(x++; y++; ...)` = $x = 1 \land y = 1$
- `sp(x++; y++; ...)` = $x = 2 \land y = 2$
  ...

assert ($y == 10$);

- × 10 iterations required to prove the property.
- × It won’t work if we replace 10 by $n$.

Back to the Example

Restriction to language $L$ = “no new constants”:

```plaintext
1st Lt. 2nd Lt. 3rd Lt. Assertion
x1 = 0 x2 = 10 x3 = 10 x4 = 10
y1 = 0 x2 = 10 x3 = x4 = 10
y2 = y3 = y4 = y1 + 1
```

This proof will produce the same predicates as SP.
The language restriction forces the solver to generalize!

Algorithm:

- If the proof fails, increase $L$!
- If we fail to get a sufficiently strong invariant, increase $n$.

This does work if we replace 10 by $n$!

? Which $L_1, L_2, \ldots$ is complete for which programs?