Block-wise abstract interpretation by combining abstract domains with SMT

Jiahong Jiang, Liqian Chen, Xueguang Wu, Ji Wang
National University of Defense Technology, China

01/16/2017 VMCAI 2017
Overview

• Motivation
• Block-wise Abstract Interpretation (BWAI) Framework
• Practical Concerns for BWAI
• Implementation and Experiments
• Conclusion
Statement-wise Abstract Interpretation (SWAI)

- SWAI
  - each statement as an individual transfer function
- Advantage
  - scalable
Statement-wise Abstract Interpretation (SWAI)

- SWAI
  - each statement as an individual transfer function
- Advantage
  - scalable
- Disadvantage
  - may cause precision loss

// x ∈ [-2, 2], y ∈ [-3, 3]
x = y + 1; // x ∈ [-2, 4], y ∈ [-3, 3]
y = x - y; // x ∈ [-2, 4], y ∈ [-5, 7]
y = 1 / (y - 2); // y ∈ [-5, 7]

Ex. 1

if (brandom())
  y = 1;
else
  y = -1;
x = 1 / y; // y ∈ [-1, 1]

Ex. 2
Main Idea

• **Block-wise** abstract interpretation (BWAI)
  
  • partition the program into several blocks
  
  • analyze the program **block by block** under AI

SWAI

BWAI

multiple statements as a block
Main Idea

- **Block-wise** abstract interpretation (BWAI)
  - partition the program into several blocks
  - analyze the program **block by block** under AI

SWAI

BWAI

multiple statements as a block

BWAI could see **more information** than SWAI at one step
Overview

• Motivation

• Block-wise Abstract Interpretation (BWAI) Framework

• Practical Concerns for BWAI

• Implementation and Experiments

• Conclusion
Questions

• How to partition the program into blocks

• How to encode semantics of a block

• How to transmit information between blocks
Choices for Expressing Transfer Semantics of a Block

• Abstract domains
  • pros: efficient
  • cons: most domains have limitations in expressing disjunctions

• SMT
  • pros: expressive for disjunctions
    • E.g., (cond == true ∧ x1 == 2) ∨ (cond == false ∧ x1 == -2)
  • cons: loops are challenging to cope with when using SMT
Workflow of BWAI

• BWAI by combining *abstract domains* (AD) with *SMT*
  • partition the program into several blocks
  • encode transfer semantics of a block via SMT
  • use *abstract domains* between blocks
  • use *widening* of *abstract domains* at loop heads

```plaintext
...  
...  
while(brandom()){
  ...
}  
...  
...  
```
Block Partitioning

• Partitioning based on cutpoints
  • a set of cutpoints: a subset of program points
    • entry/exit points, loop heads, ...

• two extreme partitioning strategies
  • minimize the size of a block
    • each statement as a block (SWAI)
  • maximize the size of a block
    • only at necessary points (loop heads, etc.)

[Beyer et al., FMCAD’09]
Block Encoding

- Encoding of the transfer semantics of a block
  - via SMT formula in $\mathcal{T}$-theory (e.g., Linear Real Arithmetic)

```plaintext
while(brandom()){
    if(phase == 1){
        x = x - 1;
        y = y + 2;
    }else{
        x = x + 2;
        y = y - 1;
    }
    phase = 1 - phase;
}
```

$\varphi_2^\text{trans} \triangleq \text{ite}(\text{phase0} == 1, \ (x_1 = x_0 - 1) \land (y_1 = y_0 + 2), \ (x_1 = x_0 + 2) \land (y_1 = y_0 - 1)) \land (\text{phase1} = 1 - \text{phase0})$
• Conversion between abstract domain representation and SMT

\[ \phi_1^\text{pre} \triangleq \nu(a_1) \]

\[ \phi_1^\text{pre} \land \phi_1^\text{trans} \]

\[ a_2 \triangleq \zeta(\phi_1^\text{pre} \land \phi_1^\text{trans}) \]
Symbolic Abstraction: SMT to Abstract Domain Representation

• Symbolic abstraction [Thakur et al., SAS’12]
  • the consequence “a” of an SMT formula \( \varphi \) in the abstract domain

• sound symbolic abstraction “a”
  • \( \text{Sol}(\varphi) \subseteq \text{Sol}(a) \)
Symbolic Abstraction: SMT to Abstract Domain Representation

- Using optimization techniques based on SMT (SMT-opt)
  - SMT-opt problem: \( \text{max } e \text{ s.t. } \varphi \) 

- fit for abstract domains based on templates
  - e.g., boxes, octagons, TCMs

“\( \text{max}(x + y) \text{ s.t. } (2x+y > 10 \lor 3x-2y < -5) \)” for Octagon domain

[Li et al., POPL’14]
Block-wise Iteration Strategy

• “iteration + widening” on abstract domains
  • iterating on CFG with blocks
  • use widening at loop heads
Overview

• Motivation
• Block-wise Abstract Interpretation (BWAI) Framework
• Practical Concerns for BWAI
  • precision
  • efficiency
• Implementation and Experiments
• Conclusion
Precision Loss Problem in BWAI

- SMT is often more expressive than abstract domain

```c
phase = [0, 1];
x = y = 0;
while(brandom()){  
  if(phase == 1){  
    x = x - 1;
    y = y + 2;
  }else{  
    x = x + 2;
    y = y - 1;
  }
  phase = 1 - phase;
}  
if(x - y > 3) { /* error() */ };  
...  
```

\[
\varphi_2^{\text{pre}} \land \varphi_2^{\text{trans}} \triangleq (0 \leq \text{phase0} \leq 1) \land (x0 ==1) \land (y0 ==1) \\
\land (\text{ite}(
\text{phase0} ==1),
(x1 = x0 - 1) \land (y1 = y0 +2),
(x1 = x0 +2) \land (y1 = y0 - 1))
\land (\text{phase1} = 1 - \text{phase0})
\]

SMT-opt for Octagon

\[
((-3 \leq x - y \leq 3) \land (0 \leq \text{phase} \leq 1) \\
\land (-1 \leq x \leq 2) \land (-1 \leq y \leq 2) \land ...)
\]

\[
((-\infty \leq x - y \leq +\infty) \land ...)
\]

\[
((-\infty \leq x - y \leq +\infty) \land ...)
\]
Precision Loss Problem in BWAI

- SMT is often more expressive than abstract domain

```plaintext
phase = [0, 1];
x = y = 0;
while (brandom()){
    if (phase == 1){
        x = x - 1;
        y = y + 2;
    } else{
        x = x + 2;
        y = y - 1;
    }
    phase = 1 - phase;
}
if (x - y > 3) { /* error() */
...}
```

\[
\varphi_2^{\text{pre}} \land \varphi_2^{\text{trans}} \triangleq (0 \leq \text{phase}_0 \leq 1) \land (x_0 == 1) \land (y_0 == 1) \\
\land (\text{ite}(\text{phase}_0 == 1),
    (x_1 = x_0 - 1) \land (y_1 = y_0 + 2),
    (x_1 = x_0 + 2) \land (y_1 = y_0 - 1)) \\
\land (\text{phase}_1 = 1 - \text{phase}_0)
\]

SMT-opt for Octagon

\[
((-3 \leq x - y \leq 3) \land (0 \leq \text{phase} \leq 1) \\
\land (-1 \leq x \leq 2) \land (-1 \leq y \leq 2) \land ...)
\]

loss of disjunctive information

\[
((-\infty \leq x - y \leq +\infty) \land ...)
\]
Our Solution

• Abstract domain lifting functor for BWAI
  • **goal**: pass necessary **disjunctive** information between blocks
  • **idea**:
    • choose a set of predicates for each block
      • **branch conditions** in direct **syntactic successor** blocks
    • partition the post-state according to predicate values

\[
(p \land a_3') \lor (\neg p \land a_3'')
\]
Our Solution

- SMT is often more expressive than abstract domain

```plaintext
phase = [0, 1];
```
```plaintext
x = y = 0;
```
```plaintext
while (brandom()){
    if (phase == 1){
        x = x - 1;
        y = y + 2;
    } else{
        x = x + 2;
        y = y - 1;
    }
    phase = 1 - phase;
}
```
```plaintext
if (x - y > 3) { /* error() */};
```
```plaintext
...
```

Our Solution

\[
\phi_2^{\text{pre}} \land \phi_2^{\text{trans}} \triangleq (0 \leq \text{phase}_0 \leq 1) \land (x_0 == 1) \land (y_0 == 1) \land \text{ite}(\text{phase}_0 == 1), (x_1 = x_0 - 1) \land (y_1 = y_0 + 2), (x_1 = x_0 + 2) \land (y_1 = y_0 - 1)) \land (\text{phase}_1 = 1 - \text{phase}_0)
\]

SMT-opt for Octagon

((phase == 1) \land ...

\lor ((phase != 1) \land ...

check “x - y > 3”
Overview

• Motivation
• Block-wise Abstract Interpretation (BWAI) framework
• Practical Concerns for BWAI
  • precision
  • efficiency
• Implementation and Experiments
• Conclusion
Scalability Problem due to Large Blocks

- Big-size formula for a large block
- Large predicate set
  - when many branches in a large block

```plaintext
while(brandom()){
  if(p1 != 0)
    lk1 = 1;
  if(p2 != 0)
    lk2 = 1;
  if(p1 != 0 && lk1 != 0)
    // ...
  if(p2 != 0 && lk2 != 0)
    // ...
}
```

at least 4 predicates for this large block
Our Solution

• Dividing a large block into small blocks

• exploiting variable clustering based on data dependency

```c
while(brandom()){
    if(p1 != 0)
        lk1 = 1;
    if(p2 != 0)
        lk2 = 1;
    if(p1 != 0 && lk1 != 0)
        // ...
    if(p2 != 0 && lk2 != 0)
        // ...
}
```

variable clusters:
{p1, lk1} for b1 and b3
{p2, lk2} for b2 and b4
Our Solution

• Considering direct semantic successive blocks
  • the closest successive blocks that share the same variable cluster with the current block

• Benefits of using direct semantic successive blocks
  • more effective information transfer
  • more useful predicates
Our Solution

• BWAI by considering direct semantic successive blocks

while(brandom()){
    if(p1 != 0)
        lk1 = 1;
    if(p2 != 0)
        lk2 = 1;
    if(p1 != 0 && lk1 != 0)
        // ...
    if(p2 != 0 && lk2 != 0)
        // ...
}
Our Solution

- BWAI by considering direct semantic successive blocks

```c
while(brandom()){
  if(p1 != 0)
    lk1 = 1;
  if(p2 != 0)
    lk2 = 1;
  if(p1 != 0 && lk1 != 0)
    // ...
  if(p2 != 0 && lk2 != 0)
    // ...
}
```
Overview

• Motivation
• Block-wise Abstract Interpretation (BWAI) Framework
• Practical Concerns for BWAI
• Implementation and Experiments
• Conclusion
Implementation

- BWCAI: a prototype under BWAI framework

```
Programs
↓
Block partitioning
↓
CFG with blocks

Abstract domains

Semantic equations

Fixpoint solver

inside blocks

between blocks

SMT

invariants
```
Implementation

- BWCAI: a prototype under BWAI framework
### Experiments

#### BWAI vs. SWAI

<table>
<thead>
<tr>
<th>SV-COMP Directories (Numbers of files)</th>
<th>SWAI</th>
<th>BWAI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Box</td>
<td>Oct</td>
</tr>
<tr>
<td></td>
<td>#Y</td>
<td>t(s)</td>
</tr>
<tr>
<td>locks(11)</td>
<td>0</td>
<td>0.28</td>
</tr>
<tr>
<td>loop-lit(14)</td>
<td>1</td>
<td>0.09</td>
</tr>
<tr>
<td>systemc(20)</td>
<td>0</td>
<td>24.77</td>
</tr>
<tr>
<td>termination-crafted(16)</td>
<td>13</td>
<td>0.08</td>
</tr>
<tr>
<td>termination-crafted-lit(12)</td>
<td>10</td>
<td>0.08</td>
</tr>
<tr>
<td>termination-restricted-15(12)</td>
<td>6</td>
<td>0.09</td>
</tr>
</tbody>
</table>
 BWAI could check around 66% properties (65 out of 98 ones), around one times more than SWAI (33 out of 98 ones)

<table>
<thead>
<tr>
<th>SV-COMP Directories (Numbers of files)</th>
<th>SWAI</th>
<th>BWAI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Box</td>
<td>Oct</td>
</tr>
<tr>
<td></td>
<td>#Y</td>
<td>t(s)</td>
</tr>
<tr>
<td>locks(11)</td>
<td>0</td>
<td>0.28</td>
</tr>
<tr>
<td>loop-lit(14)</td>
<td>1</td>
<td>0.09</td>
</tr>
<tr>
<td>systemc(20)</td>
<td>0</td>
<td>24.77</td>
</tr>
<tr>
<td>termination-crafted(16)</td>
<td>13</td>
<td>0.08</td>
</tr>
<tr>
<td>termination-crafted-lit(12)</td>
<td>10</td>
<td>0.08</td>
</tr>
<tr>
<td>termination-restricted-15(12)</td>
<td>6</td>
<td>0.09</td>
</tr>
</tbody>
</table>
Overview

• Motivation
• Block-wise Abstract Interpretation (BWAI) Framework
• Practical Concerns under BWAI
• Implementation and Experiments
• Conclusion
Conclusion

• Block-wise AI instead of statement-wise AI
  • by combining abstract domains with SMT
Conclusion

• A block-wise AI instead of statement-wise AI
  • by combining abstract domains with SMT

abstract domain lifting functor
(\textit{precision})
• A block-wise AI instead of statement-wise AI
• by combining abstract domains with SMT

divide a large block into small blocks (efficiency)

Block partitioning

Abstract domains

semantic successive blocks (efficiency)

SMT
Future Work

• More flexible block partitioning strategies
  • trade off between precision and efficiency

• Support more SMT theories
  • e.g., floating point, array, ...