Analysis of Binaries using SAT/SMT

Jörg Brauer
RWTH Aachen University
brauer@embedded.rwth-aachen.de

with Andy King and Thomas Reinbacher

01.03.2012 @ InBot, Aachen
The Ultimate Goal
Rather: What I want to get a PhD for

• **Given:** Binary program
• **Goal:** Compute over-approximations of all registers & memory locations
• **Approach:** SAT & SMT solving
• **Applications:** Jump-target recovery, and many more
Example #1

\[
\begin{align*}
\text{XOR} & \quad r0 & r1 \\
\text{XOR} & \quad r1 & r0 \\
\text{XOR} & \quad r0 & r1
\end{align*}
\]

Swaps contents of two registers without involving a third
Example #1

Introduce input-output variables and consider abstract domain of two-variable equalities:

\[(r0 = r1), (r0 = r0'), (r0 = r1'), \ldots, \top\]

\[
\begin{align*}
\text{XOR } r0 & \text{ r1} \\
\text{XOR } r1 & \text{ r0} \\
\text{XOR } r0 & \text{ r1}
\end{align*}
\]

Swaps contents of two registers without involving a third
Example #1

Traditional Abstraction

\[(r_0 = r_1)\]

XOR r0 r1

T

XOR r1 r0

T

XOR r0 r1

T
Example #1
Block-Wise Abstraction

\[(r_0 = r_1) \land (r_0 = r_1') \land (r_1 = r_0')\]
• Reasoning about blocks rather than instructions can increase precision
  – Problem: Blocks are program-dependent, whereas instructions are not
  – Solution: Automatic techniques to compute abstractions for each block in a program
Example #2

SBC R2 R2

Subtract with carry, result is either 00000000 or 11111111

XOR R0 R0

Reset register and program status word at once
Example #2

\[ \text{SBC R2 R2} \]
Subtract with carry, result is either 00000000 or 11111111

\[ \text{XOR R0 R0} \]
Reset register and program status word at once

Need to handle such cases (and many more) to get precise analysis results
Lesson #2

• Reasoning about binaries involves thinking about many nifty cases and obfuscated code

• Manual abstraction requires a lot of experience and engineering work

• Automatic abstraction deobfuscates the binary, represents the semantics as is
  – Nice for arithmetic obfuscations
  – There is no need to be smart here
Example #3

\[
\begin{align*}
\text{AND R0 } & \#15 \\
\text{AND R1 } & \#15 \\
\text{XOR R0 R1} & \\
\text{ADD R0 R1} & \\
\end{align*}
\]

\[
\begin{align*}
0 \leq r_0' & \leq 30 \\
0 \leq r_1' & \leq 15 \\
0 \leq r_0' + r_1' & \leq 45 \\
0 \leq r_0' - r_1' & \leq 15 \\
\end{align*}
\]

\[
\begin{align*}
\text{XOR R0 R1} & \\
\text{XOR R1 R0} & \\
\text{XOR R0 R1} & \\
\end{align*}
\]

\[
\begin{align*}
r_0' & = r_1 \\
r_1' & = r_0 \\
\end{align*}
\]
Lesson #3

• Need to combine different abstract domains that can model different properties
  – Ranges vs. equalities
  – Doing this by hand is a horrible task

• Automatic abstraction computes different abstractions for free
  – No need to be busy here!
Lessons Learnt

1. Block-wise abstraction improves precision
2. Automatic abstraction takes care of the evil cornercases
3. Automatic abstraction supports a variety of abstract domains for free

That’s enough blurb, let’s go technical!
Approach

• Specify semantics for each instruction once and for all
• Combine different semantics to form a basic block
• Apply SSA conversion within the block
• Use decision procedure to compute abstractions for variety of domains
  – Equalities, affine relations, congruences, intervals, value sets, etc.
  – SAT/SMT solvers are great to reason about bit-vectors
Conversion Into SSA

```
INC R0
MOV R1 R0
LSL R1
SBC R1 R1
EOR R0 R1
SUB R0 R1
```

```
R0_1 := INC R0
R1_1 := R0_1
R1_2 := LSL R1_1
R1' := SBC R1_2 R1_2
R0_2 := EOR R0_1 R1'
R0' := SUB R0_2 R1'
```
Conversion Into Logic

\[
\begin{align*}
R0_1 & := \text{INC } R0 \\
R1_1 & := R0_1 \\
R1_2 & := \text{LSL } R1_1 \\
R1' & := \text{SBC } R1_2 \ R1_2 \\
R0_2 & := \text{EOR } R0_1 \ R1' \\
R0' & := \text{SUB } R0_2 \ R1'
\end{align*}
\]

\[\varphi = \sigma_{\text{INC}}(r0_1, r0) \land \sigma_{\text{MOV}}(r1_1, r0_1) \land \sigma_{\text{LSL}}(r1_2, r1_1) \land \sigma_{\text{SBC}}(r1', r1_2, r1_2) \land \sigma_{\text{EOR}}(r0_2, r0_1, r1') \land \sigma_{\text{SUB}}(r0', r0_2, r1')\]
A Variety of Abstractions

- Intervals
- Value sets
- Affine equalities
- Octagons
Interval Abstraction

- Suppose $r_0 \in [-10, 20]$ on input
- Put $\varphi' = \varphi \land (r_0 \in [-10, 20])$
- What’s the upper bound of $r_0'$ on output?
Interval Abstraction

- Suppose \( r_0 \in [-10, 20] \) on input
- Put \( \varphi' = \varphi \land (r_0 \in [-10, 20]) \)
- What’s the upper bound of \( r_0' \) on output?

It’s somewhere in this range
Interval Abstraction

- Suppose $r_0 \in [-10, 20]$ on input
- Put $\varphi' = \varphi \land (r_0 \in [-10, 20])$
- What’s the upper bound of $r_0'$ on output?

\[ \varphi' \land \neg r_0'[7] ? \]

\[ \varphi' \land \neg r_0'[7] \land r_0'[6] ? \]
Interval Abstraction

• Suppose \( r0 \in [-10, 20] \) on input
• Put \( \varphi' = \varphi \land (r0 \in [-10, 20]) \)
• What’s the upper bound of \( r0' \) on output?

\[ \varphi' \land \neg r0'[7] \ ? \]
\[ \varphi' \land \neg r0'[7] \land r0'[6] \ ? \]
\[ \varphi' \land \neg r0'[7] \land r0'[6] \land r0'[5] \ ? \]
Interval Abstraction

- Suppose \( r_0 \in [-10, 20] \) on input
- Put \( \varphi' = \varphi \land (r_0 \in [-10, 20]) \)
- What’s the upper bound of \( r_0' \) on output?

\[
\varphi' \land \neg r_0'[7] \land r_0'[6] \land r_0'[5] \land r_0'[4]?
\]
Interval Abstraction

- Suppose $r_0 \in [-10, 20]$ on input
- Put $\varphi' = \varphi \land (r_0 \in [-10, 20])$
- What’s the upper bound of $r_0'$ on output?

```
-128  0  127

\varphi' \land \neg r_0'[7] \ ?

\varphi' \land \neg r_0'[7] \land r_0'[6] \ ?

\varphi' \land \neg r_0'[7] \land r_0'[6] \land r_0'[5] \ ?

\varphi' \land \neg r_0'[7] \land r_0'[6] \land r_0'[5] \land r_0'[4] \ ?
```

© 2012 Jörg Brauer, Embedded Software Laboratory, RWTH Aachen University
Interval Abstraction

• Suppose $r_0 \in [-10, 20]$ on input
• Put $\varphi' = \varphi \land (r_0 \in [-10, 20])$
• What’s the upper bound of $r_0'$ on output?

• Eventually: $r_0' \leq 21$
• Likewise, minimization gives $r_0' \geq 0$
Literature: Interval Abstraction

• Codish, Lagoon & Stuckey: Logic Programming with Satisfiability (TPLP’08)
• Barrett & King: Range and Set Abstraction using SAT (NSAD’10)
• Brauer, King & Kowalewski: Range Analysis of Microcontroller Code using Bit-Level Congruences (FMICS’10)
• Brauer & King: Transfer Function Synthesis without Quantifier Elimination (ESOP’11)
Value-Set Abstraction

• Suppose $r_0 \in \{-10, 4, 20\}$ on input
• Put $\varphi' = \varphi \land (r_0 \in \{-10, 4, 20\})$
• What’s the value set of $r_0'$ on output?
• Enumerate models of $\varphi'$ using incremental SAT
Value-Set Abstraction

• Suppose $r_0 \in \{-10, 4, 20\}$ on input
• Put $\varphi' = \varphi \land (r_0 \in \{-10, 4, 20\})$
• What’s the value set of $r_0'$ on output?
• Enumerate models of $\varphi'$ using incremental SAT
Value-Set Abstraction

- Suppose \( r_0 \in \{-10, 4, 20\} \) on input
- Put \( \varphi' = \varphi \land (r_0 \in \{-10, 4, 20\}) \)
- What’s the value set of \( r_0' \) on output?
- Enumerate models of \( \varphi' \) using incremental SAT

\[-128 \quad 0 \quad 5 \quad 127\]
Value-Set Abstraction

- Suppose $r_0 \in \{-10, 4, 20\}$ on input
- Put $\varphi' = \varphi \land (r_0 \in \{-10, 4, 20\})$
- What’s the value set of $r_0'$ on output?
- Enumerate models of $\varphi'$ using incremental SAT

- Consider $\varphi' \land (r_0' \neq 5) \land (r_0' \neq 21)$
Value-Set Abstraction

• Suppose $r_0 \in \{-10, 4, 20\}$ on input
• Put $\varphi' = \varphi \land (r_0 \in \{-10, 4, 20\})$
• What’s the value set of $r_0'$ on output?
• Enumerate models of $\varphi'$ using incremental SAT

$\varphi' \land (r_0' \neq 5) \land (r_0' \neq 21) \land (r_0' \neq 9)$ is unsat
Suppose $r_0' \in \{5, 9, 21\}$

Put $\varphi' = \varphi \land (r_0' \in \{5, 9, 21\})$

We apply the same algorithm backwards
  - Over-approximation of inputs we had before
  - Also works for intervals
Literature: Value-Set Abstraction

- Barrett & King: Range and Set Abstraction using SAT (NSAD’10)
- Reinbacher & Brauer: Precise Control Flow Reconstruction using Boolean Logic (EMSOFT’11)
- Brauer, King & Kriener: Existential Quantification as Incremental SAT (CAV’11)
Affine Equalities

• Consider again

\[ \varphi = \begin{cases} 
\sigma_{\text{INC}}(r_{01}, r_0) \\
\wedge \sigma_{\text{MOV}}(r_{11}, r_{01}) \\
\wedge \sigma_{\text{LSL}}(r_{12}, r_{11}) \\
\wedge \sigma_{\text{SBC}}(r_{1}', r_{12}, r_{12}) \\
\wedge \sigma_{\text{EOR}}(r_{02}, r_{01}, r_{1}') \\
\wedge \sigma_{\text{SUB}}(r_0', r_{02}, r_{1}') 
\end{cases} \]

restricted to normal operation of INC and SUB and overflow of LSL, denoted \( \psi \)
Affine Equalities

• Goal: compute affine equality that describes relation between $r^0$ on input and $r^0'$ on output
• Natural choice: represent affine equalities as matrices
Affine Hull
Iteration #1

- Pass $\psi$ to a solver
- Gives model $m_1 = (r_0 = -4 \land r_0' = 3)$
- Represent solution as matrix
  $$\begin{bmatrix}
  1 & 0 & -4 \\
  0 & 1 & 3 \\
\end{bmatrix}$$
  Focus on this row
- Put $\psi' = \psi \land (r_0' \neq 3)$
Affine Hull
Iteration #2

• Pass \( \psi' = \psi \land (r0' \neq 3) \) to a solver
• Gives model \( m_2 = (r0 = -5 \land r0' = 4) \)
• Represent solution as matrix
\[
\begin{bmatrix}
1 & 0 & -5 \\
0 & 1 & 4
\end{bmatrix}
\]

• Now compute
\[
\begin{bmatrix}
1 & 0 & -4 \\
0 & 1 & 3
\end{bmatrix} \sqcup \begin{bmatrix}
1 & 0 & -5 \\
0 & 1 & 4
\end{bmatrix} = \begin{bmatrix}
1 & 1 & -1
\end{bmatrix}
\]
Affine Hull
Iteration #3

\[
\begin{bmatrix}
1 & 0 & -4 \\
0 & 1 & 3
\end{bmatrix}
\sqcup
\begin{bmatrix}
1 & 0 & -5 \\
0 & 1 & 4
\end{bmatrix}
= \begin{bmatrix}
1 & 1 & -1
\end{bmatrix}
\]

• Put \( \psi'' = \psi \land (r0 + r1 \neq -1) \)
• Then, \( \psi'' \) is unsatisfiable

• Hence, \( r0' = -r0 - 1 \) is the optimal affine abstraction of \( \psi \)
Literature: Affine Relations

• Karr: Affine Relationships among Variables of a Program (Acta Informatica’76)
• Müller-Olm & Seidl: A Note on Karr’s Algorithm (ICALP’04)
• Müller-Olm & Seidl: Analysis of Modular Arithmetic (ACM TOPLAS’07)
• Brauer & King: Automatic Abstraction for Intervals using Boolean Formulae (SAS’10)
INC R0
MOV R1 R0
LSL R1
SBC R1 R1
EOR R0 R1
SUB R0 R1

\[(r_0 = 127) \implies (r_0' = -128)\]
\[(-128 \leq r_0 \leq -2) \implies (r_0' = -r_0 - 1)\]
\[(-1 \leq r_0 \leq 126) \implies (r_0' = r_0 + 1)\]

\[R_0' := \text{abs}(R_0 + 1)\]
Control Flow Reconstruction

\{r_7 : 1 - 3, 101 - 103\}  
\{flags : C := false\}

MOV 0x08 ← R7
MOV A ← 0x08
CLR A
SUBB A, #N_HANDL
JC C:0x00F
{r_8 : 1 - 3, 101 - 103}
{r_A : 251 - 253, 95 - 97}
{flags : C := true, false}

\{r_0:08 : 1 - 3\}  
\{flags : C := true\}

MOV R7 ← 0x08
MOV A ← R7
MOV B ← #0x03
MUL AB
ADD A, #0x26
MOV DPL ← A
CLR A
ADDC A, #0x00
MOV DPH ← A
AJMP C:0x038
\{r_dph : 0\}
\{r_dpl : 41, 44, 47\}

\{r_dpl : 41, 44, 47\}
CLR A
MOVC A ← @(A+DP)
MOV R3 ← A
MOV A ← #0x01
MOVC A ← @(A+DP)
MOV R2 ← A
MOV A ← #0x02
MOVC ← @(A+DP)
MOV R1 ← A
AJMP C:0x100
\{r_3 : 255\}
\{r_2 : 0\}
\{r_1 : 105, 110, 115\}

abstracted jump targets
hdr2()✓, hdr3()✓, hdr4()✓
# Experimental Results

<table>
<thead>
<tr>
<th>Binary Program</th>
<th>Compiler</th>
<th>$\text{loc}_C$</th>
<th>$\text{instr}_B$</th>
<th>JT</th>
<th>$\mathcal{F}$ interpreter</th>
<th>$\mathcal{F} + \overline{\mathcal{B}}$ interpreter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\text{RT}$</td>
<td>$\text{FT}$</td>
</tr>
<tr>
<td>Single row input</td>
<td>Keil</td>
<td>80</td>
<td>67</td>
<td>6</td>
<td>2401</td>
<td>2395</td>
</tr>
<tr>
<td></td>
<td></td>
<td>52</td>
<td></td>
<td></td>
<td>460</td>
<td>454</td>
</tr>
<tr>
<td>Keypad</td>
<td>Keil</td>
<td>113</td>
<td>113</td>
<td>9</td>
<td>3844</td>
<td>3835</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80</td>
<td></td>
<td></td>
<td>1508</td>
<td>1499</td>
</tr>
<tr>
<td>Communication Link</td>
<td>Keil</td>
<td>111</td>
<td>164</td>
<td>8</td>
<td>6889</td>
<td>6881</td>
</tr>
<tr>
<td></td>
<td></td>
<td>118</td>
<td></td>
<td></td>
<td>84</td>
<td>76</td>
</tr>
<tr>
<td>Task Scheduler</td>
<td>Keil</td>
<td>81</td>
<td>105</td>
<td>5</td>
<td>&gt;1000</td>
<td>&gt;995</td>
</tr>
<tr>
<td></td>
<td></td>
<td>97</td>
<td></td>
<td></td>
<td>&gt;1</td>
<td>&gt;995</td>
</tr>
<tr>
<td>Switch Case</td>
<td>Keil</td>
<td>82</td>
<td>166</td>
<td>19</td>
<td>&gt;5000</td>
<td>&gt;4981</td>
</tr>
<tr>
<td></td>
<td></td>
<td>180</td>
<td></td>
<td></td>
<td>3304</td>
<td>3285</td>
</tr>
<tr>
<td>Emergency Stop</td>
<td>Keil</td>
<td>138</td>
<td>150</td>
<td>9</td>
<td>768</td>
<td>759</td>
</tr>
<tr>
<td></td>
<td></td>
<td>141</td>
<td></td>
<td></td>
<td>256</td>
<td>247</td>
</tr>
</tbody>
</table>

| $\text{loc}_C$         | Lines of C code |
| $\text{instr}_B$       | Number of assembly instructions |
| JT                     | Number of jump targets |
| RT                     | Number of recovered targets |
| $\text{FT}$           | Number of recovered false targets |
| $\text{RS}$           | Number of refinement steps applied |
| $k$                   | Backtracking depth |
| Time                  | Analysis time in seconds |
So as to not cause offense

- Reps, Sagiv & Yorsh: Symbolic Implementation of the Best Transformer (VMCAI’04)
- Regehr & Reid: HOIST – A System For Automatically Deriving Static Analyzers for Embedded Systems (ASPLOS’04)
- Monniaux: Automatic Modular Abstractions for Linear Constraints (POPL’09)
- Monniaux: Automatic Modular Abstractions for Template Numerical Constraints (LMCS’10)
- Brauer & King: Automatic Abstraction for Intervals using Boolean Formulae (SAS’10)
- Brauer, King & Kowalewski: Range Analysis of Microcontroller Binary Code using Bit-Level Congruences (FMICS’10)
- Brauer & King: Transfer Function Synthesis without Quantifier Elimination (ESOP’11)
- Reinbacher & Brauer: Precise Control Flow Reconstruction using Boolean Logic (EMSOFT’11)
Conclusion

• We advocate automatic abstraction as opposed to manual design
• SAT/SMT solvers can easily solve thousands of structured problems per second
• All techniques rely on the same encoding of the semantics
  – Solving for different abstract domains is slightly different
  – Can be put into uniform framework, but it is more efficient the way we put it