Automatic Abstraction Using Decision Procedures

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

06.03.2012 @ Saarbrücken
Myself

- Diploma in CS from Kiel (2002-2008)
- 1.5 years at NICTA, Sydney (2006-2008)
  - Static analysis of C/C++
- Since 10/2008 at RWTH Aachen
  - Project leader for [mc]square since 01/2010, now ARCADE
  - Aalborg from 04/2011 till 06/2011
- Near future:
  - Submit dissertation in April
  - Work as “Verifikationsspezialist” with Verified Systems, Bremen
My Research Interests

• Automatic abstraction of binaries & decision procedures
  – with Andy King (Kent)
• Non-instrumenting runtime verification
  – with Thomas Reinbacher (TU Vienna)
• Model checking of PLCs
  – with Sebastian Biallas (RWTH)
Motivating Example

\begin{align*}
\text{INC} & \quad \text{R0} \\
\text{MOV} & \quad \text{R1} \quad \text{R0} \\
\text{LSL} & \quad \text{R1} \\
\text{SBC} & \quad \text{R1} \quad \text{R1} \\
\text{EOR} & \quad \text{R0} \quad \text{R1} \\
\text{SUB} & \quad \text{R0} \quad \text{R1}
\end{align*}

- Goal: Affine transfer functions that relate interval boundaries
- Wraps are ubiquitous on 8-bit architecture
- Guard wrapping inputs using octagons [Min06]
Motivating Example

\begin{itemize}
  \item Block can over/underflow in three different ways
\end{itemize}

\begin{align*}
  \text{INC} & \quad R0 \\
  \text{MOV} & \quad R1 \quad R0 \\
  \text{LSL} & \quad R1 \\
  \text{SBC} & \quad R1 \quad R1 \quad (r0 = 127) \\
  \text{EOR} & \quad R0 \quad R1 \quad (-128 \leq r0 \leq -2) \\
  \text{SUB} & \quad R0 \quad R1 \quad (-1 \leq r0 \leq 126)
\end{align*}

Guards
Motivating Example

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

- Block can over/underflow in three different ways

(r0 = 127) \[\Rightarrow (r0' = -128)\]

(−128 \leq r0 \leq −2) \[\Rightarrow (r0' = -r0 - 1)\]

(−1 \leq r0 \leq 126) \[\Rightarrow (r0' = r0 + 1)\]
Bit-Blasting

• Translate block into bit-vector logic, giving a formula $\varphi_b$ [CKSY04]

• Enforce combination of wrapping behavior, e.g. INC R0 and SUB R0 R1 behave normally, but LSL R2 overflows, denoted $\varphi_w$

• Then $\varphi = \varphi_b \land \varphi_w$ describes desired semantics

• Still need to abstract $\varphi$
Abstracting $\varphi$ With Linear Templates

- The block has input $R0$ and output $R0'$
- Consider symbolic interval $r0 \in [r0_l, r0_u]$  
- We know that $-128 \leq r0_u \leq 127$
- Key idea: dichotomic search
- $r0_u$ is uniquely determined, thus  
  
  $(-128 \leq r0_u \leq -1) \lor (0 \leq r0_u \leq 127)$
Abstracting $\varphi$ Using Binary Search
Abstracting $\varphi$ Using Binary Search

$$\varphi \land (\neg r0[7])$$
Abstracting $\varphi$ Using Binary Search

$\varphi \land (\neg r0[7])$ ? no!

$\varphi \land (\neg r0[7]) \land (r0[6])$ ?
Abstracting \( \varphi \) Using Binary Search

\[
\varphi \land (\neg r0[7]) \land (r0[6]) \land (r0[5]) ? \text{yes!}
\]

\[
\varphi \land (\neg r0[7]) \land (r0[6]) \land (r0[5]) ?
\]
Abstracting \( \varphi \) Using Binary Search

\[
\varphi \land (\neg r0[7]) \ ? \ \text{no!}
\]
\[
\varphi \land (\neg r0[7]) \land (r0[6]) \ ? \ \text{yes!}
\]
\[
\varphi \land (\neg r0[7]) \land (r0[6]) \land (r0[5]) \ ? \ \text{yes!}
\]

\[ r0_u = -2 \]
Summary: Range Abstraction

• Characterize feasible inputs of some mode combination
• Simple form of dichotomistic (or binary) search
• Efficient because of incremental SAT
• Can also be applied to octagons, etc.
• Alternative formulation using quantification is more complicated [Mon09,BK10]
  – Depends on alternating quantifiers, which is doable but difficult [BKK11]
Motivating Example Revisited

- Block can over/underflow in three different ways

\[
\begin{align*}
\text{INC } & \ R0 \\
\text{MOV } & \ R1 \ R0 \\
\text{LSL } & \ R1 \\
\text{SBC } & \ R1 \ R1 \quad (r0 = 127) \quad \Rightarrow \quad (r0' = -128) \\
\text{EOR } & \ R0 \ R1 \quad (-128 \leq r0 \leq -2) \quad \Rightarrow \quad (r0' = -r0 - 1) \\
\text{SUB } & \ R0 \ R1 \quad (-1 \leq r0 \leq 126) \quad \Rightarrow \quad (r0' = r0 + 1)
\end{align*}
\]

Done!  
Next!
Abstracting $\varphi$ With Affine Equalities

• Formula $\varphi$ describes relation between $r0$ on input and $r0'$ on output
• Question: How to extract affine equality that relates $r0$ and $r0'$?
• Answer: Incremental affine hull [MS04]
  – Similar in spirit to [RSY04]
Example: Affine Hull

- Pass $\varphi$ to a SAT/SMT solver
- Obtain model $m_1 = (r0 = -4 \land r0' = 3)$
- Represent as affine matrix

$$
\begin{bmatrix}
1 & 0 & -4 \\
0 & 1 & 3
\end{bmatrix}
$$
Example: Affine Hull

- Current abstraction: \[
\begin{bmatrix}
1 & 0 & -4 \\
0 & 1 & 3 \\
\end{bmatrix}
\]

- Pass \( \varphi \land (r0' \neq 3) \) to a SAT/SMT solver

- Obtain model \( m_2 = (r0 = -5 \land r0' = 4) \)

- Represent as affine matrix and join

\[
\begin{bmatrix}
1 & 0 & -4 \\
0 & 1 & 3 \\
\end{bmatrix} \cup \begin{bmatrix}
1 & 0 & -5 \\
0 & 1 & 4 \\
\end{bmatrix} = \begin{bmatrix}
1 & 1 & -1 \\
\end{bmatrix}
\]
Example: Affine Hull

- Current abstraction: \[ \begin{bmatrix} 1 & 1 & -1 \end{bmatrix} \]

- Pass \( \varphi \land (r0 + r0' \neq -1) \) to a SAT/SMT solver
- Formula is unsatisfiable
- Affine equality \( r0' = -r0 - 1 \) thus over-approximates \( \varphi \)
Back to the Example

- Block can over/underflow in three different ways

\[
\begin{align*}
\text{INC R0} & \quad (r_0 = 127) \quad \Rightarrow \quad (r_0' = -128) \\
\text{MOV R1 R0} & \quad (-128 \leq r_0 \leq -2) \quad \Rightarrow \quad (r_0' = -r_0 - 1) \\
\text{LSL R1} & \quad (-1 \leq r_0 \leq 126) \quad \Rightarrow \quad (r_0' = r_0 + 1) \\
\text{SBC R1 R1} & \quad (r_0 = 127) \quad \Rightarrow \quad (r_0' = -128) \\
\text{EOR R0 R1} & \quad (-128 \leq r_0 \leq -2) \quad \Rightarrow \quad (r_0' = -r_0 - 1) \\
\text{SUB R0 R1} & \quad (-1 \leq r_0 \leq 126) \quad \Rightarrow \quad (r_0' = r_0 + 1) 
\end{align*}
\]

\[R_0' := \text{abs}(R_0 + 1)\]
Some More Properties

- Fairly quick: \( \sim 0.02s \) using Z3
- Can apply this technique to relate intervals and octagons, too [BK10,BK11]

\[
\begin{align*}
(r0'_l &= -128) \land (r0'_u &= -128) \\
(r0'_l &= -r0_u - 1) \land (r0'_u &= -r0_l - 1) \\
(r0'_l &= r0_l + 1) \land (r0'_u &= r0_u + 1)
\end{align*}
\]

- Or even compute polynomial relations [MS04]
## Experimental Results

<table>
<thead>
<tr>
<th>Binary Program</th>
<th>(\mathcal{F}) interpreter</th>
<th>(\mathcal{F} + \overline{B}) interpreter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Compiler</strong></td>
<td><strong>loc(_C)</strong></td>
</tr>
<tr>
<td>Single row input</td>
<td>Keil</td>
<td>80</td>
</tr>
<tr>
<td>Keypad</td>
<td>Keil</td>
<td>113</td>
</tr>
<tr>
<td>Communication Link</td>
<td>Keil</td>
<td>111</td>
</tr>
<tr>
<td>Task Scheduler</td>
<td>Keil</td>
<td>81</td>
</tr>
<tr>
<td>Switch Case</td>
<td>Keil</td>
<td>82</td>
</tr>
<tr>
<td>Emergency Stop</td>
<td>Keil</td>
<td>138</td>
</tr>
</tbody>
</table>

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

- [Mon09] D. Monniaux: Automatic Modular Abstractions for Linear Constraints (POPL’09)
- [Min06] A. Mine: The Octagon Abstract Domain (HOSC’06)
- [CKSY04] E. Clarke, D. Kroening, N. Sharygina and K. Yorav: Predicate Abstraction of ANSI-C Programs using SAT (FMSD’04)
- [MS04] M. Müller-Olm and H. Seidl: A Note on Karr’s Algorithm (ICALP’04)
My Related Publications

• [BK11a] J. Brauer and A. King: Transfer Function Synthesis without Quantifier Elimination (ESOP’11 + hopefully LMCS’12)
Concluding Discussion

- We advocate automatic abstraction rather than manual design of transformers for binary analysis
  - Support for variety of different domains for free, e.g., affine equalities, intervals, value sets, octagons
  - Block-wise abstraction itself necessitates automation
- Key idea: decompose blocks based on wrapping modes
- SAT/SMT solvers can easily solve hundreds to thousands of instances per second now
- Future work: loop transformers + demand-driven abstraction
Thank you very much!