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

```
XOR r0 r1
XOR r1 r0
XOR r0 r1
```

Swaps contents of two registers without involving a third

Example #1

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

$$(\mathbf{r0} = \mathbf{r1}), (\mathbf{r0} = \mathbf{r0}'), (\mathbf{r0} = \mathbf{r1}'), \ldots, \top$$

```
XOR r0 r1
XOR r1 r0
XOR r0 r1
```

} Swaps contents of two registers without involving a third

Example #1
Traditional Abstraction

$(\mathbf{r0} = \mathbf{r1})$

XOR r0 r1

$\top$

XOR r1 r0

$\top$

XOR r0 r1

$\top$

Example #1
Block-Wise Abstraction

$(\mathbf{r0} = \mathbf{r1})$

XOR r0 r1

$\top$

XOR r1 r0

$\top$

XOR r0 r1

$\top$

$(\mathbf{r0} = \mathbf{r1})$

```
XOR r0 r1
XOR r1 r0
XOR r0 r1
```

$$(\mathbf{r0} = \mathbf{r1}) \wedge (\mathbf{r0} = \mathbf{r1}') \wedge (\mathbf{r1} = \mathbf{r0}')$$

Lesson #1

- 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

`SBC R2 R2` — Subtract with carry, result is either 00000000 or 11111111

`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

```
AND R0 #15
AND R1 #15
XOR R0 R1
ADD R0 R1
```

$$\begin{array}{ccccc} 0 & \leq & \mathbf{r0}' & \leq & 30 \\ 0 & \leq & \mathbf{r1}' & \leq & 15 \\ 0 & \leq & \mathbf{r0}' + \mathbf{r1}' & \leq & 45 \\ 0 & \leq & \mathbf{r0}' - \mathbf{r1}' & \leq & 15 \end{array}$$

```
XOR R0 R1
XOR R1 R0
XOR R0 R1
```

$$\begin{array}{ccc} \mathbf{r0}' & = & \mathbf{r1} \\ \mathbf{r1}' & = & \mathbf{r0} \end{array}$$

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
```

$R0_1 := \mathtt{INC}\ R0$

$R1_1 := R0_1$

$R1_2 := \mathtt{LSL}\ R1_1$

$R1' := \mathtt{SBC}\ R1_2\ R1_2$

$R0_2 := \mathtt{EOR}\ R0_1\ R1'$

$R0' := \mathtt{SUB}\ R0_2\ R1'$

Conversion Into Logic

$\texttt{R0}_1 \; := \; \texttt{INC R0}$

$\texttt{R1}_1 \; := \; \texttt{R0}_1$

$\texttt{R1}_2 \; := \; \texttt{LSL R1}_1$

$\texttt{R1'} \; := \; \texttt{SBC R1}_2 \; \texttt{R1}_2$

$\texttt{R0}_2 \; := \; \texttt{EOR R0}_1 \; \texttt{R1'}$

$\texttt{R0'} \; := \; \texttt{SUB R0}_2 \; \texttt{R1'}$

$$\underbrace{\begin{array}{ll} & \sigma_{\texttt{INC}}(\mathbf{r0}_1, \mathbf{r0}) \\ \wedge & \sigma_{\texttt{MOV}}(\mathbf{r1}_1, \mathbf{r0}_1) \\ \wedge & \sigma_{\texttt{LSL}}(\mathbf{r1}_2, \mathbf{r1}_1) \\ \wedge & \sigma_{\texttt{SBC}}(\mathbf{r1}', \mathbf{r1}_2, \mathbf{r1}_2) \\ \wedge & \sigma_{\texttt{EOR}}(\mathbf{r0}_2, \mathbf{r0}_1, \mathbf{r1}') \\ \wedge & \sigma_{\texttt{SUB}}(\mathbf{r0}', \mathbf{r0}_2, \mathbf{r1}') \end{array}}_{\varphi}$$

A Variety of Abstractions

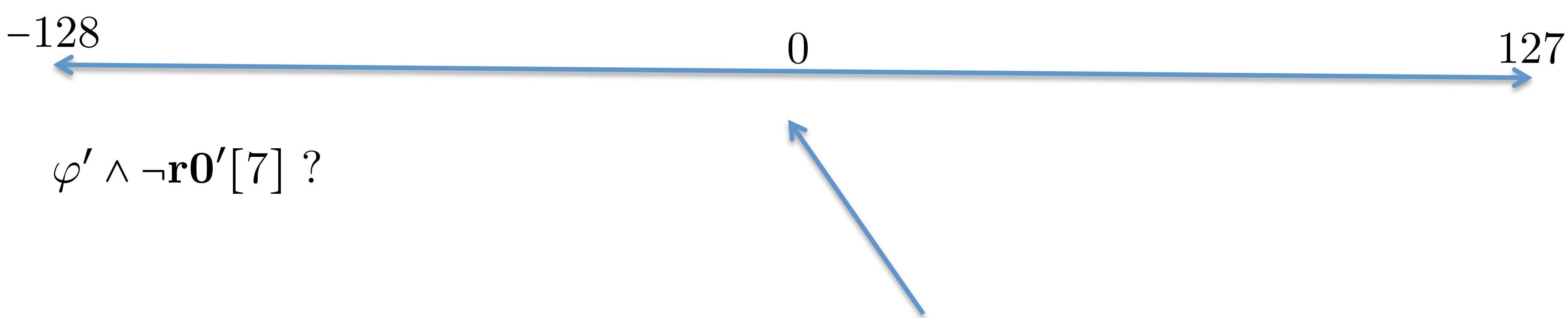
- Intervals
- Value sets
- Affine equalities
- Octagons

Interval Abstraction

- Suppose $\mathbf{r0} \in [-10, 20]$ on input
- Put $\varphi' = \varphi \wedge (\mathbf{r0} \in [-10, 20])$
- What's the upper bound of $\mathbf{r0}'$ on output?

Interval Abstraction

- Suppose $\mathbf{r0} \in [-10, 20]$ on input
- Put $\varphi' = \varphi \wedge (\mathbf{r0} \in [-10, 20])$
- What's the upper bound of $\mathbf{r0}'$ on output?

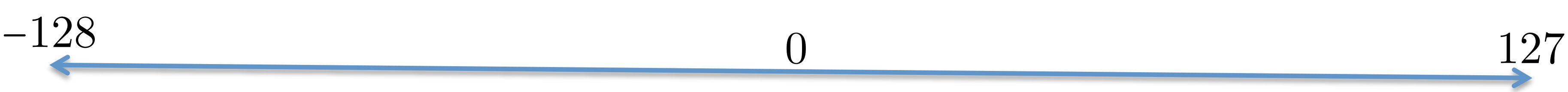


$\varphi' \wedge \neg \mathbf{r0}'[7]$?

It's somewhere in this range

Interval Abstraction

- Suppose $\mathbf{r0} \in [-10, 20]$ on input
- Put $\varphi' = \varphi \wedge (\mathbf{r0} \in [-10, 20])$
- What's the upper bound of $\mathbf{r0}'$ on output?

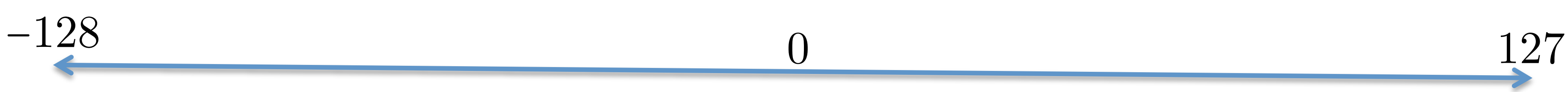


$\varphi' \wedge \neg \mathbf{r0}'[7]$?

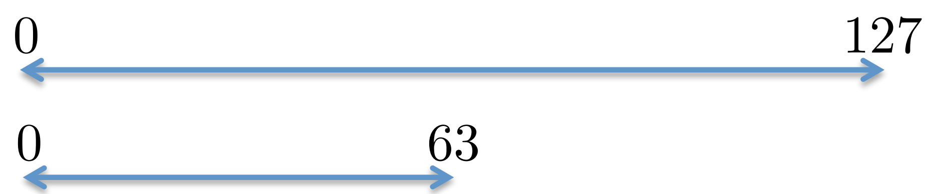
$\varphi' \wedge \neg \mathbf{r0}'[7] \wedge \mathbf{r0}'[6]$?

Interval Abstraction

- Suppose $\mathbf{r0} \in [-10, 20]$ on input
- Put $\varphi' = \varphi \wedge (\mathbf{r0} \in [-10, 20])$
- What's the upper bound of $\mathbf{r0}'$ on output?



$\varphi' \wedge \neg \mathbf{r0}'[7]$?

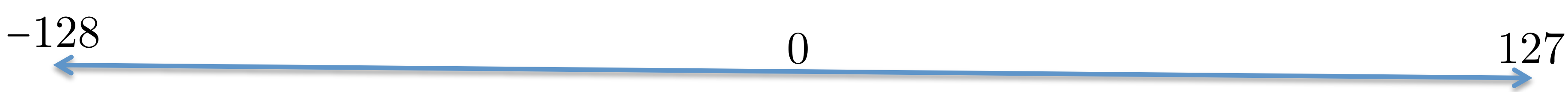


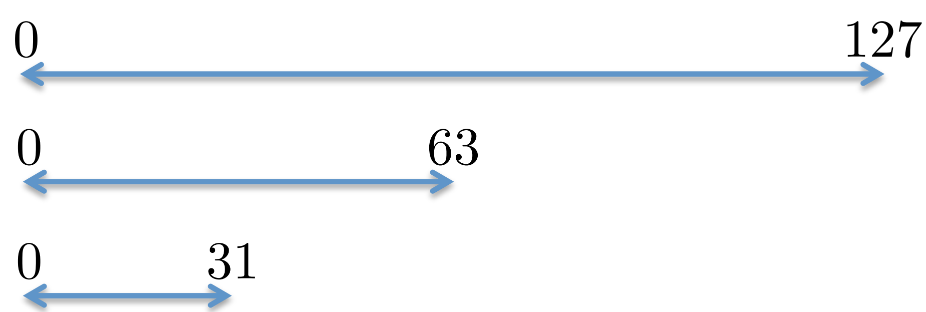
$\varphi' \wedge \neg \mathbf{r0}'[7] \wedge \mathbf{r0}'[6]$?

$\varphi' \wedge \neg \mathbf{r0}'[7] \wedge \mathbf{r0}'[6] \wedge \mathbf{r0}'[5]$?

Interval Abstraction

- Suppose $\mathbf{r0} \in [-10, 20]$ on input
- Put $\varphi' = \varphi \wedge (\mathbf{r0} \in [-10, 20])$
- What's the upper bound of $\mathbf{r0}'$ on output?





$\varphi' \wedge \neg\mathbf{r0}'[7]$?

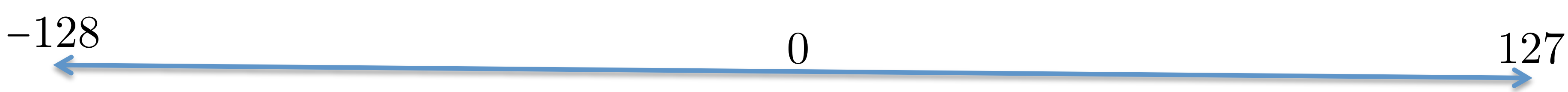
$\varphi' \wedge \neg\mathbf{r0}'[7] \wedge \mathbf{r0}'[6]$?

$\varphi' \wedge \neg\mathbf{r0}'[7] \wedge \mathbf{r0}'[6] \wedge \mathbf{r0}'[5]$?

$\varphi' \wedge \neg\mathbf{r0}'[7] \wedge \mathbf{r0}'[6] \wedge \mathbf{r0}'[5] \wedge \mathbf{r0}'[4]$?

Interval Abstraction

- Suppose $\mathbf{r0} \in [-10, 20]$ on input
- Put $\varphi' = \varphi \wedge (\mathbf{r0} \in [-10, 20])$
- What's the upper bound of $\mathbf{r0}'$ on output?

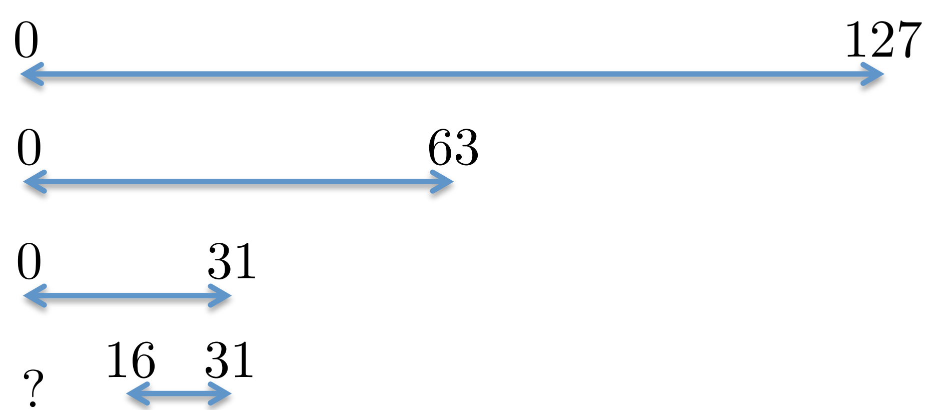


$\varphi' \wedge \neg \mathbf{r0}'[7]$?

$\varphi' \wedge \neg \mathbf{r0}'[7] \wedge \mathbf{r0}'[6]$?

$\varphi' \wedge \neg \mathbf{r0}'[7] \wedge \mathbf{r0}'[6] \wedge \mathbf{r0}'[5]$?

$\varphi' \wedge \neg \mathbf{r0}'[7] \wedge \mathbf{r0}'[6] \wedge \mathbf{r0}'[5] \wedge \mathbf{r0}'[4]$?



Interval Abstraction

- Suppose $\mathbf{r0} \in [-10, 20]$ on input
- Put $\varphi' = \varphi \wedge (\mathbf{r0} \in [-10, 20])$
- What's the upper bound of $\mathbf{r0}'$ on output?

- Eventually: $\mathbf{r0}' \leq 21$
- Likewise, minimization gives $\mathbf{r0}' \geq 0$

Literature: Interval Abstraction

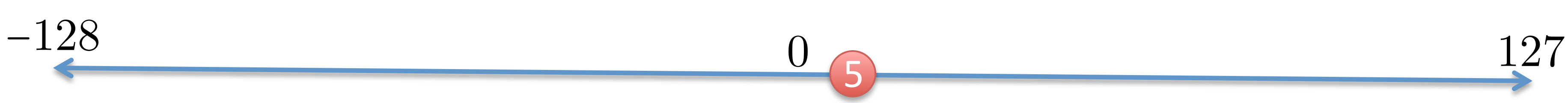
- Codish, Lagoon & Stuckey: Logic Programming with Satisfiability (TPLP'08)
- Barrett & King: Range and Set Abstraction using SAT (NSAD'10)
- Brauer, King & Kowalewski: Range Analysis of Microcontroller Code using Bit-Level Congruences (FMICS'10)
- Brauer & King: Transfer Function Synthesis without Quantifier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

Value-Set Abstraction

- Suppose $\mathbf{r0} \in \{-10, 4, 20\}$ on input
- Put $\varphi' = \varphi \wedge (\mathbf{r0} \in \{-10, 4, 20\}$
- What's the value set of $\mathbf{r0}'$ on output?
- Enumerate models of φ' using incremental SAT

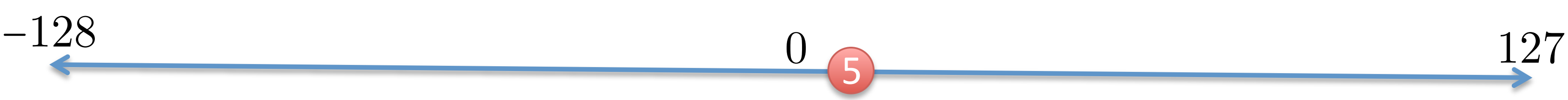
Value-Set Abstraction

- Suppose $\mathbf{r0} \in \{-10, 4, 20\}$ on input
- Put $\varphi' = \varphi \wedge (\mathbf{r0} \in \{-10, 4, 20\})$
- What's the value set of $\mathbf{r0}'$ on output?
- Enumerate models of φ' using incremental SAT



Value-Set Abstraction

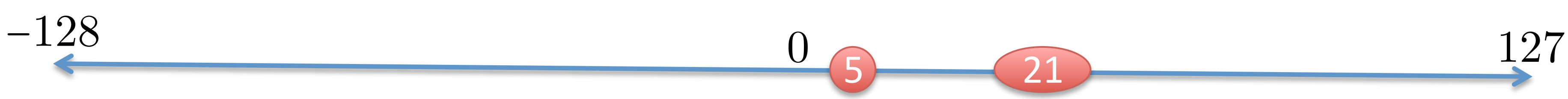
- Suppose $\mathbf{r0} \in \{-10, 4, 20\}$ on input
- Put $\varphi' = \varphi \wedge (\mathbf{r0} \in \{-10, 4, 20\})$
- What's the value set of $\mathbf{r0}'$ on output?
- Enumerate models of φ' using incremental SAT



- Consider $\varphi' \wedge (\mathbf{r0}' \neq 5)$

Value-Set Abstraction

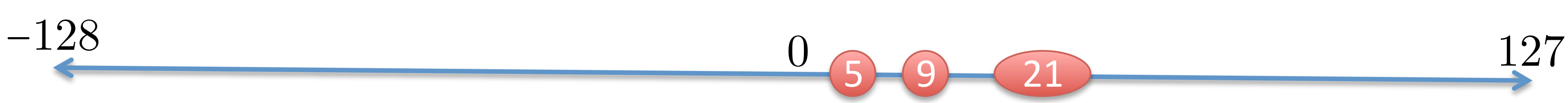
- Suppose $\mathbf{r0} \in \{-10, 4, 20\}$ on input
- Put $\varphi' = \varphi \wedge (\mathbf{r0} \in \{-10, 4, 20\}$
- What's the value set of $\mathbf{r0}'$ on output?
- Enumerate models of φ' using incremental SAT



- Consider $\varphi' \wedge (\mathbf{r0}' \neq 5) \wedge (\mathbf{r0}' \neq 21)$

Value-Set Abstraction

- Suppose $\mathbf{r0} \in \{-10, 4, 20\}$ on input
- Put $\varphi' = \varphi \wedge (\mathbf{r0} \in \{-10, 4, 20\})$
- What's the value set of $\mathbf{r0}'$ on output?
- Enumerate models of φ' using incremental SAT

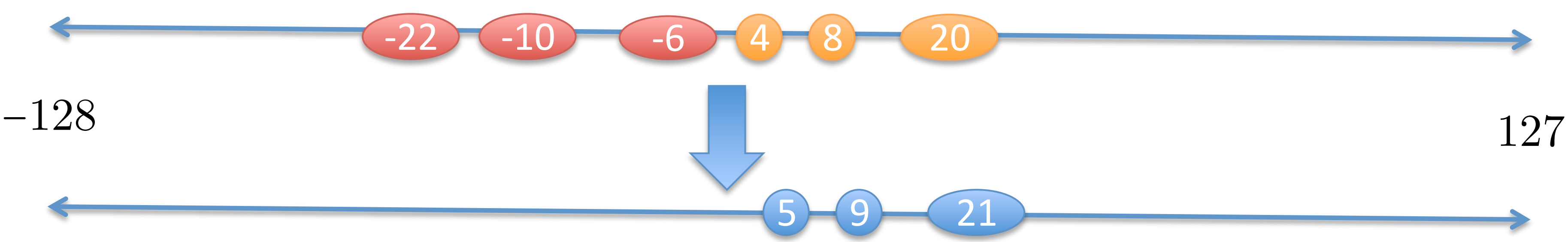


- $\varphi' \wedge (\mathbf{r0}' \neq 5) \wedge (\mathbf{r0}' \neq 21) \wedge (\mathbf{r0}' \neq 9)$ is unsat

Value-Set Abstraction

The Other Direction

- Suppose $\mathbf{r0}' \in \{5, 9, 21\}$
- Put $\varphi' = \varphi \wedge (\mathbf{r0}' \in \{5, 9, 21\})$



- 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

$$\varphi = \begin{cases} & \sigma_{\mathtt{INC}}(\mathbf{r0}_1, \mathbf{r0}) \\ \wedge & \sigma_{\mathtt{MOV}}(\mathbf{r1}_1, \mathbf{r0}_1) \\ \wedge & \sigma_{\mathtt{LSL}}(\mathbf{r1}_2, \mathbf{r1}_1) \\ \wedge & \sigma_{\mathtt{SBC}}(\mathbf{r1}', \mathbf{r1}_2, \mathbf{r1}_2) \\ \wedge & \sigma_{\mathtt{EOR}}(\mathbf{r0}_2, \mathbf{r0}_1, \mathbf{r1}') \\ \wedge & \sigma_{\mathtt{SUB}}(\mathbf{r0}', \mathbf{r0}_2, \mathbf{r1}') \end{cases}$$

restricted to normal operation of `INC` and `SUB` and overflow of `LSL`, denoted ψ

Affine Equalities

- Goal: compute affine equality that describes relation between $\mathbf{r0}$ on input and $\mathbf{r0}'$ on output
- Natural choice: represent affine equalities as matrices

Affine Hull

Iteration #1

- Pass ψ to a solver
- Gives model $m_1 = (\mathbf{r0} = -4 \wedge \mathbf{r0}' = 3)$
- Represent solution as matrix

$$\left[\begin{array}{cc|c} 1 & 0 & -4 \\ 0 & 1 & 3 \end{array}\right]$$

← Focus on this row

- Put $\psi' = \psi \wedge (\mathbf{r0}' \neq 3)$

Affine Hull

Iteration #2

- Pass $\psi' = \psi \wedge (\mathbf{r0}' \neq 3)$ to a solver
- Gives model $m_2 = (\mathbf{r0} = -5 \wedge \mathbf{r0}' = 4)$
- Represent solution as matrix $\left[\begin{array}{cc|c} 1 & 0 & -5 \\ 0 & 1 & 4 \end{array}\right]$
- Now compute

$$\left[\begin{array}{cc|c} 1 & 0 & -4 \\ 0 & 1 & 3 \end{array}\right] \sqcup \left[\begin{array}{cc|c} 1 & 0 & -5 \\ 0 & 1 & 4 \end{array}\right] = \left[\begin{array}{cc|c} 1 & 1 & -1 \end{array}\right] \longleftarrow$$

Affine Hull

Iteration #3

$$\left[\begin{array}{cc|c} 1 & 0 & -4 \\ 0 & 1 & 3 \end{array}\right] \sqcup \left[\begin{array}{cc|c} 1 & 0 & -5 \\ 0 & 1 & 4 \end{array}\right] = \left[\begin{array}{cc|c} 1 & 1 & -1 \end{array}\right] \leftarrow$$

- Put $\psi'' = \psi \wedge (\mathbf{r0} + \mathbf{r1} \neq -1)$
- Then, ψ'' is unsatisfiable

- Hence, $\mathbf{r0}' = -\mathbf{r0} - 1$ is the optimal affine abstraction of ψ

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)

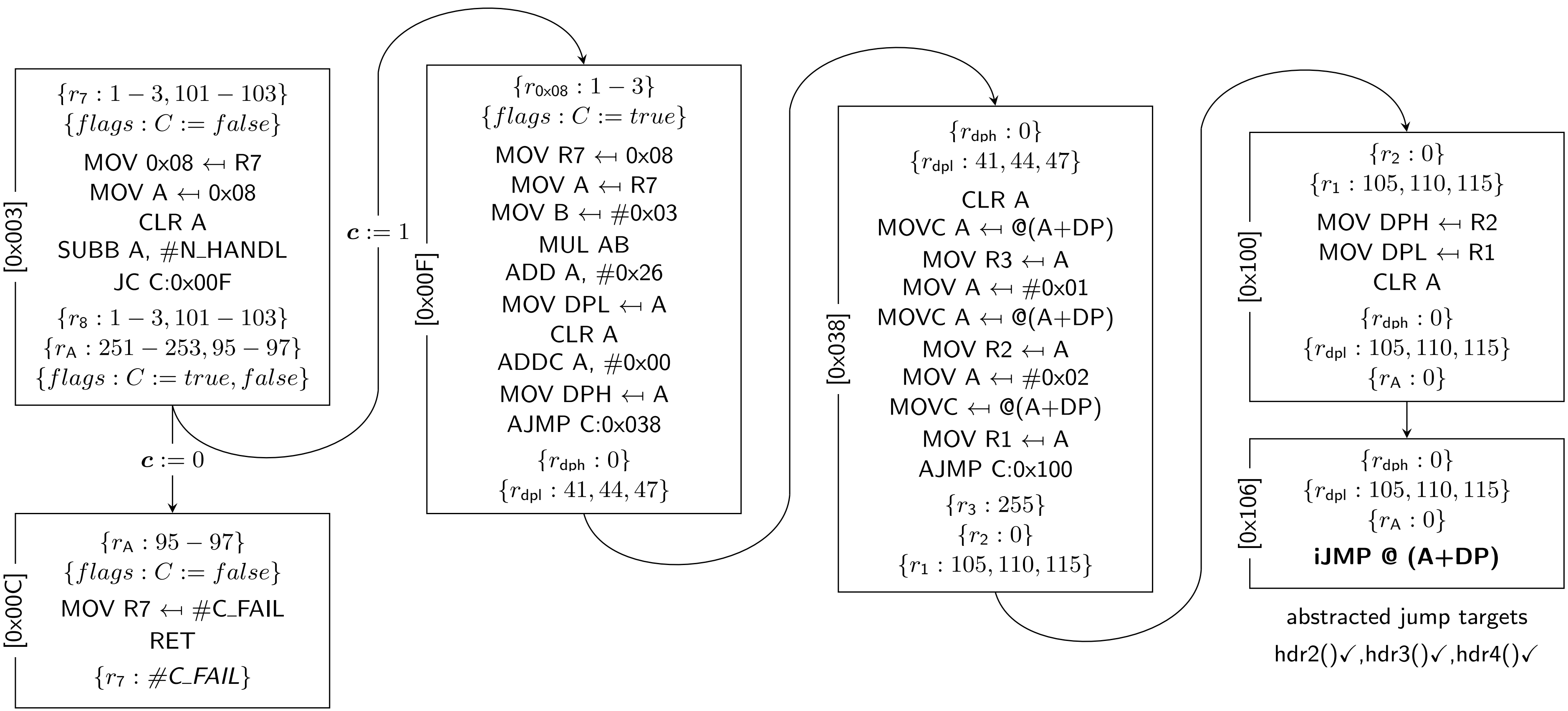
Combined Analyses

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

$$
\begin{aligned}
(\mathbf{r0} = 127) &\Rightarrow (\mathbf{r0}' = -128)\\
(-128 \le \mathbf{r0} \le -2) &\Rightarrow (\mathbf{r0}' = -\mathbf{r0} - 1)\\
(-1 \le \mathbf{r0} \le 126) &\Rightarrow (\mathbf{r0}' = \mathbf{r0} + 1)
\end{aligned}
$$

```
R0' := abs(R0 + 1)
```

Control Flow Reconstruction



Experimental Results

Binary Program					$\overrightarrow{\mathcal{F}}$ interpreter			$\overrightarrow{\mathcal{F}} + \overleftarrow{\mathcal{B}}$ interpreter				
Name	Compiler	loc_C	$instr_B$	JT	RT	FT	Time	RS	k	RT	FT	Time
Single row input	Keil	80	67	6	2401	2395	2.6	2	2	6	–	3.32
	Sdcc		52		460	454	2.4	2	2	6	–	2.0
Keypad	Keil	113	113	9	3844	3835	3.49	4	2	9	–	4.33
	Sdcc		80		1508	1499	3.08	4	2	9	–	2.57
Communication Link	Keil	111	164	8	6889	6881	4.56	2	2	8	–	4.37
	Sdcc		118		84	76	3.38	2	2	8	–	4.29
Task Scheduler	Keil	81	105	5	>1000	>995	>5m	17	2	5	–	14.03
	Sdcc		97					23	2	5	–	10.23
Switch Case	Keil	82	166	19	>5000	>4981	>5m	94	2	19	–	17.49
	Sdcc		180		3304	3285	2.31	6	2	38	19	2.6
Emergency Stop	Keil	138	150	9	768	759	2.8	2	2	9	–	2.6
	Sdcc		141		256	247	2.9	2	2	9	–	3.1

loc_C ... Lines of C code
$instr_B$ ... Number of assembly instructions
JT ... Number of jump targets
RT ... Number of recovered targets
FT ... Number of recovered false targets
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


Excellent