Automatic Abstraction for Bit-Vector Relations

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

22.11.2011 @ CEA

Myself

- Studied at CAU Kiel
- Spent 1,5 years @ NICTA in Sydney
- Diploma (computer science) in 09/2008
- Since then
  - Embedded Software Laboratory at RWTH Aachen
  - [mc]square (project lead since 01/2010)
- Research interest
  - Circles around automatic abstraction
  - PhD thesis finished (hopefully) in spring 2012
  - Supervisors: S. Kowalewski (RWTH) & A. King (Kent)

What is the Talk about? (1/2)

```
1: INC R0;
2: MOV R1, R0;
3: LSL R1;
4: SBC R1, R1;
5: EOR R0, R1;
6: SUB R0, R1;
```

- Goal: Affine transfer functions that relate interval boundaries [Monniaux, POPL'09]
- Wraps are ubiquitous on 8-bit architecture
- Guard wrapping inputs using octagons [Mine, HOSC'06]

What is the Talk about? (2/2)

```
1: INC R0;
2: MOV R1, R0;
3: LSL R1;
4: SBC R1, R1;
5: EOR R0, R1;
6: SUB R0, R1;
```

$$\begin{array}{l}(127 \leq r0 \leq 127) \\ \Rightarrow (r0_l^\star = -128 \wedge r0_u^\star = -128)\end{array}$$

$$\begin{array}{l}(-128 \leq r0 \leq -2) \\ \Rightarrow (r0_l^\star = -r0_u - 1 \wedge r0_u^\star = -r0_l - 1)\end{array}$$

$$\begin{array}{l}(-1 \leq r0 \leq 126) \\ \Rightarrow (r0_l^\star = r0_l + 1 \wedge r0_u^\star = r0_u + 1)\end{array}$$

- Key idea: Boolean encodings of semantics
- Compute abstractions of affine relations and guards separately using SAT

Guards for Wrapping

- Consider instruction `ADD r0 r1`
- Boolean encoding (outputs are primed):

$$
\begin{aligned}
\varphi(\mathbf{c}) \quad = \quad & (\wedge_{i=0}^{7} \mathbf{r0}'[i] \oplus \mathbf{r0}[i] \oplus \mathbf{r1}[i] \oplus \mathbf{c}[i]) \wedge \neg \mathbf{c}[0] \wedge \\
& (\wedge_{i=0}^{6} \mathbf{c}[i+1] \leftrightarrow (\mathbf{r0}[i] \wedge \mathbf{r1}[i]) \vee (\mathbf{r0}[i] \wedge \mathbf{c1}[i]) \vee (\mathbf{r1}[i] \wedge \mathbf{c}[i])
\end{aligned}
$$

- For example, enforce overflow:

$$
\varphi(\mathbf{c})' \quad = \quad \varphi(\mathbf{c}) \wedge (\neg \mathbf{r0}[7] \wedge \neg \mathbf{r1}[7] \wedge \mathbf{r0}'[7])
$$

- Then $\varphi(\mathbf{c})'$ characterizes overflow-case only

Characterization of Optimal Bounds

- Guards are of the form $\pm v_1 \pm v_2 \leq d$
- d is characterized as [Monniaux, POPL'09]:
  - It is an upper bound for any $\pm v_1 \pm v_2$
  - For any other upper bound d', we have $d \leq d'$
- The „for any“ manifests itself in terms of universal quantification
  - Which is trivial for CNF formulae
  - Simply strike out all literals
- „Exists“ is more complicated

Guards in Boolean Logic

- Safety:

$$\nu \quad = \quad \forall r0 : \forall r1 : (\varphi \Rightarrow \pm r0 \pm r1 \leq d)$$

- Optimality:

$$\mu \quad = \quad \forall r0 : \forall r1 : \forall d' : ((\varphi \Rightarrow \pm r0 \pm r1 \leq d') \Rightarrow d \leq d')$$

- Then solve $\nu \wedge \mu$ using SAT after q-elimination
- Observe that μ interacts with ν to impose an optimal bound

Boolean Characterization for Intervals

- Very similar formulation for relation between input- and output-intervals (but more technically involved)
- Also uses two-staged formulation to
  - First characterize safe output intervals
  - And then impose optimality
- However, still need to compute affine relations

Boolean Characterization for Intervals

$$\left.\begin{array}{l}\forall r0 : \forall r1 : \forall r0' : \forall r1' : \\ ((r0_l \leq r0 \leq r0_u \wedge r1_l \leq r1 \leq r1_u) \wedge \varphi) \Rightarrow \\ (r0_l^\star \leq r0' \leq r0_u^\star \wedge r1_l^\star \leq r1' \leq r1_u^\star)\end{array}\right\} \text{safety}$$

$$\bigwedge$$

$$\left.\begin{array}{l}\forall r0_l' : \forall r0_u' : \forall r1_l' : \forall r1_u' : \forall r0 : \forall r1 : \forall r0' : \forall r1' : \\ (((r0_l \leq r0 \leq r0_u \wedge r1_l \leq r1 \leq r1_u) \wedge \varphi) \Rightarrow \\ \quad (r0_l' \leq r0' \leq r0_u' \wedge r1_l' \leq r1' \leq r1_u')) \Rightarrow \\ (r0_l' \leq r0_l^\star \wedge r0_u^\star \leq r0_u' \wedge r1_l' \leq r1_l^\star \wedge r1_u^\star \leq r1_u')\end{array}\right\} \text{optimality}$$

Key Idea: Affine Closure

- Obtain a solution of formula using SAT
- Represent solution as matrix
- Add disequality to obtain new solutions
- Join with previous matrix
- Add disequality to obtain new solutions
- ...
- Eventually stabilizes since domain is finite [Reps et al., VMCAI'04]

Example: Affine Closure

$$\varphi = \begin{cases} (\neg w[0]) \wedge \left(\wedge_{i=0}^{6} w[i+1] \leftrightarrow (v[i] \oplus \wedge_{j=0}^{i-1} v[j])\right) & \wedge \\ (\neg x[0]) & \wedge \\ \left(\wedge_{i=0}^{6} x[i+1] \leftrightarrow (w[i] \wedge x[i]) \vee (w[i] \wedge y[i]) \vee (x[i] \wedge y[i])\right) & \wedge \\ \left(\wedge_{i=0}^{7} z[i] \leftrightarrow w[i] \oplus x[i] \oplus y[i]\right) & \wedge \\ ((v[7] \leftrightarrow v[6]) \wedge (v[6] \leftrightarrow v[5])) \wedge ((y[7] \leftrightarrow y[6]) \wedge (y[6] \leftrightarrow y[5])) & \end{cases}$$

- Compute affine relations between variables z, v and y
- Could also be our Boolean characterization of intervals

Example: Affine Closure

- 1st solution: $(v = 0, y = 0, z = 2)$

$$\left[\begin{array}{ccc|c} 0 & 0 & 0 & 1 \end{array}\right] \sqcup \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{array}\right] = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{array}\right]$$

- 2nd solution: $(v = -1, y = 0, z = 0)$

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{array}\right] \sqcup \left[\begin{array}{ccc|c} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right] = \left[\begin{array}{ccc|c} 2 & 0 & -1 & -2 \\ 0 & 1 & 0 & 0 \end{array}\right]$$

- 3rd solution: $(v = 0, y = 1, z = 3)$

$$\left[\begin{array}{ccc|c} 2 & 0 & -1 & -2 \\ 0 & 1 & 0 & 0 \end{array}\right] \sqcup \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 3 \end{array}\right] = \left[\begin{array}{ccc|c} 2 & 1 & -1 & -2 \end{array}\right]$$

- Result: $2 \cdot v + y - z = -2$

Applying Transfer Functions

- Amounts to linear programming
- Given an octagonal guard g and input intervals i
- Treat affine transfer function f as cost function and maximize/minimize f subject to $g \wedge i$

- Solve using Simplex or branch-and-bound (runtime vs. precision)

Example: Applying Transfer Functions

- Input:

  $i = (-3 \leq r0 \leq 4)$

$$\begin{aligned}
&(127 \leq r0 \leq 127) \\
\Rightarrow\ &(r0_l^\star = -128 \wedge r0_u^\star = -128) \\
&(-128 \leq r0 \leq -2) \\
\Rightarrow\ &(r0_l^\star = -r0_u - 1 \wedge r0_u^\star = -r0_l - 1) \\
&(-1 \leq r0 \leq 126) \\
\Rightarrow\ &(r0_l^\star = r0_l + 1 \wedge r0_u^\star = r0_u + 1)
\end{aligned}$$

- Solving the two remaining linear programs then yields:

  $r0_l^\star = 0$

  $r0_u^\star = 5$

Drawbacks

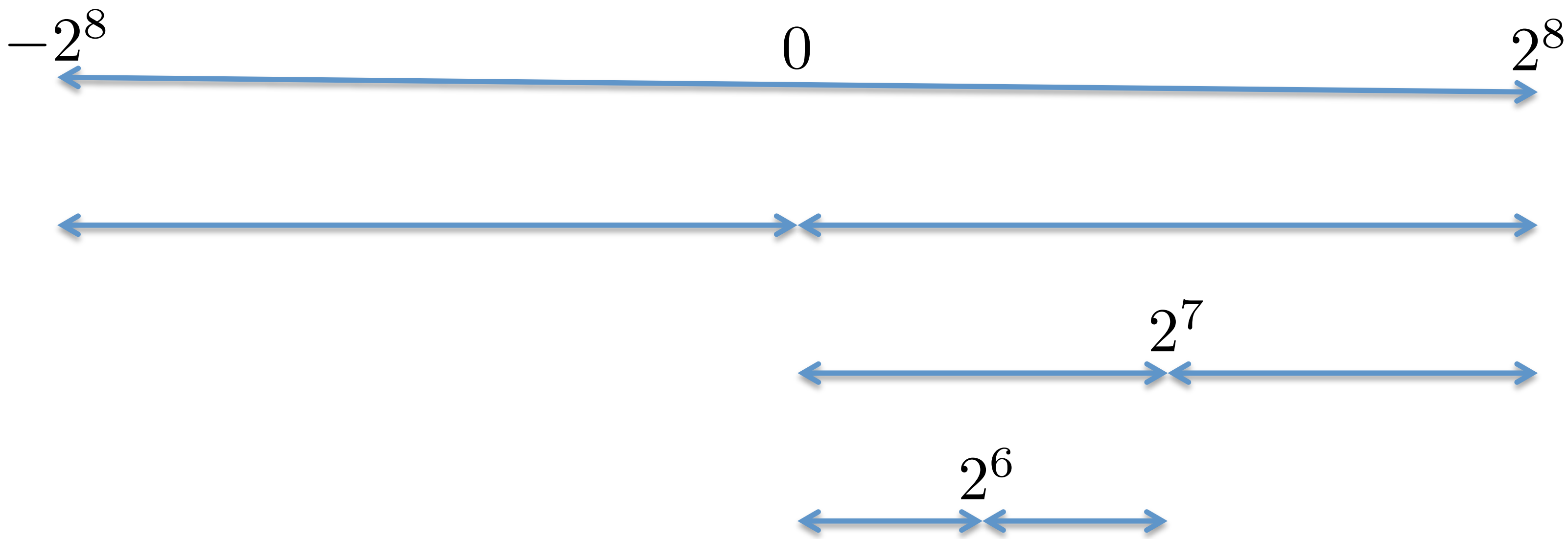
- Characterization requires quantifier alternation
- Especially existential quantifier elimination is difficult
  - Recall that we need output in CNF
  - Otherwise, universal quantification would be tricky
- Various techniques exist, e.g., resolution, BDDs [Lahiri et al., CAV'03 & CAV'06], SAT [Brauer and King, CAV'11]

Intuition

- Observe: abstraction is not dissimilar to universal quantification
  - Gives a relation that holds for all values
- Is it possible to come up with an approach that does not need alternating quantifiers?
  - Yes!
- Solution: Dichotomic/binary search
  - Implemented as incremental SAT [Codish et al., TPLP’08]

Algorithm by Picture

$$r0 + r1 = d$$



...

- $2^6 + 2^2 + 2^0$

Octagons using Dichotomic Search

- Consider computing guards for `ADD R0 R1` in overflow mode
- Then $\pm r0 \pm r1 \leq d$, thus

$$\begin{array}{rl} & -2^8 \leq d \leq 2^8 \\ \Leftrightarrow & (-2^8 \leq d \leq -1) \vee (0 \leq d \leq 2^8) \end{array}$$

- Use SAT solver to find out which disjunct holds

Octagons using Dichotomic Search

- φ encodes `ADD R0 R1`
- Then $\varphi' = \varphi \wedge (r0 + r1 = d)$
- Is $\varphi' \wedge \neg d[10]$ satisfiable? Yes!
  - Hence $(0 \leq d \leq 2^8)$

$$\Leftrightarrow \quad (0 \leq d \leq 2^7 - 1) \vee (2^7 \leq d \leq 2^8)$$

- Proceed with $\varphi'' = \varphi' \wedge \neg d[10] \wedge d[9]$ to give

$$2^7 \leq d \leq 2^8$$

Linear Templates using Dichotomic Search

- Have the form $\sum_{i=1}^{n} c_i \cdot v_i \leq d$ where the c_i are constants, hence d is bounded
- We can thus always apply binary search

How About Polynomials?

- Consider `MUL R0 R2; ADD R0 R1`
  - Assume neither operation overflows
- Relation is non-affine, analysis gives $\top$
- Idea:
  - Compute affine closure as before
  - While doing so, perform polynomial extension [Müller-Olm & Seidl, ICALP'04]

Polynomial Extension by Example

- 1st solution $\langle r0 = 2, r1 = 4, r2 = 3, r0' = 10 \rangle$
  - Matrix $\left[\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 & 4 \\ 0 & 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 1 & 10 \end{array}\right]$
  - Add monomial for $r0 \cdot r2$ to give

$$\left[\begin{array}{ccccc|c} 1 & 0 & 0 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 & 0 & 4 \\ 0 & 0 & 1 & 0 & 0 & 3 \\ 0 & 0 & 0 & 1 & 0 & 10 \\ 0 & 0 & 0 & 0 & 1 & 6 \end{array}\right]$$

Polynomial Extension by Example

- Now search for solution that violates $r0 \cdot r2 = 6$
- SAT gives $\langle r0 = 3, r1 = 4, r2 = 8, r0' = 28 \rangle$ which implies $r0 \cdot r2 = 24$
- Matrix representation

$$\left[\begin{array}{ccccc|c} 1 & 0 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & 0 & 4 \\ 0 & 0 & 1 & 0 & 0 & 8 \\ 0 & 0 & 0 & 1 & 0 & 28 \\ 0 & 0 & 0 & 0 & 1 & 24 \end{array}\right]$$

- Join with first solution

Polynomial Extension by Example

- After five iterations, we get the joined system

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

- Equivalent to $r0' = r1 + r0 \cdot r2$
  - Taken from code that indexes into two-dimensional array
- Observe: need to encode polynomials in SAT
  - It's well-known how to do that [Fuhs et al., SAT'07]

So as to not cause offense

- D. Monniaux: Automatic Modular Abstractions for Linear Constraints (POPL’09 & LMCS‘10)
- A. Mine: The Octagon Abstract Domain (HOSC’06)
- A. King, H. Søndergaard: Automatic Abstraction for Congruences (VMCAI‘10)
- T. Reps, M. Sagiv, G. Yorsh: Symbolic Implementation of the Best Transformer (VMCAI‘04)
- M. Müller-Olm, H. Seidl: A Note on Karr‘s Algorithm (ICALP‘04)
- J. Brauer, A. King: Automatic Abstraction for Intervals using Boolean Formulae (SAS‘10)
- J. Brauer, A. King: Transfer Function Synthesis without Quantifier Elimination (ESOP‘11 & submitted to LMCS)

Summary

- Deriving transfer functions for bit-vector programs using SAT solving
- Combination of octagons and affine equalities
- Two approaches:
  - Quantifier-based characterization
  - Use incremental SAT solving
- Easily extended to polynomial relations

Future Work

- Transfer functions for loops
  - Monniaux (POPL'09) did this for linear constraints
  - Complicated characterization explodes in Boolean logic though
- More general classes of linear constraints than

$$\sum_{i=1}^{n} c_i \cdot v_i \leq d$$

  - The c_i are constants
  - How about TVPI or polyhedra? Coefficients are variable then, probably requires approximation

Thank you very much!