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;

⇒ \((127 \leq r_0 \leq 127)\)
⇒ \((r_0^* = -128 \land r_0^* = -128)\)
⇒ \((-128 \leq r_0 \leq -2)\)
⇒ \((r_0^* = -r_0^u - 1 \land r_0^* = -r_0^l - 1)\)
⇒ \((-1 \leq r_0 \leq 126)\)
⇒ \((r_0^* = r_0^l + 1 \land r_0^* = r_0^u + 1)\)

• 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):

  \[
  \varphi(c) = \left( \bigwedge_{i=0}^{7} r0'[i] \oplus r0[i] \oplus r1[i] \oplus c[i] \right) \land \neg c[0] \land \\
  \left( \bigwedge_{i=0}^{6} c[i+1] \leftrightarrow (r0[i] \land r1[i]) \lor (r0[i] \land c1[i]) \lor (r1[i] \land c[i]) \right)
  \]

- For example, enforce overflow:

  \[
  \varphi(c)' = \varphi(c) \land (\neg r0[7] \land \neg r1[7] \land r0'[7])
  \]

- Then \( \varphi(c)' \) 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:
  \[ \nu = \forall r_0 : \forall r_1 : (\varphi \Rightarrow \pm r_0 \pm r_1 \leq d) \]

• Optimality:
  \[ \mu = \forall r_0 : \forall r_1 : \forall d' : ((\varphi \Rightarrow \pm r_0 \pm r_1 \leq d') \Rightarrow d \leq d') \]

• Then solve \( \nu \land \mu \) using SAT after q-elimination

• Observe that \( \mu \) interacts with \( \nu \) 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

\[ \forall r_0 : \forall r_1 : \forall r_0' : \forall r_1' : 
\]
\[ ((r_{0l} \leq r_0 \leq r_{0u} \land r_{1l} \leq r_1 \leq r_{1u}) \land \varphi) \Rightarrow 
(r_{0_1}^* \leq r_0' \leq r_{0u}^* \land r_{1_1}^* \leq r_1' \leq r_{1u}^*) 
\]

\[ \land 
\]
\[ \forall r_{0_1} : \forall r_{0_1}^* : \forall r_{1_1} : \forall r_{1_1}^* : \forall r_0 : \forall r_1 : \forall r_0' : \forall r_1' : 
\]
\[ (((r_{0l} \leq r_0 \leq r_{0u} \land r_{1l} \leq r_1 \leq r_{1u}) \land \varphi) \Rightarrow 
(r_0' \leq r_{0_1} \leq r_{0u}' \land r_{1_1} \leq r_1' \leq r_{1u}')) \Rightarrow 
(r_0' \leq r_{0_1}^* \land r_{0u}' \leq r_{0_1}^* \leq r_{1_1} \land r_{1u}^* \leq r_{1u}') 
\]

\text{safety} \quad \text{optimality}

© 2011 Jörg Brauer, Embedded Software Laboratory, RWTH Aachen University
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

\[ \varphi = \left\{ \begin{array}{l}
(\neg w[0]) \land (\land_{i=0}^{6} w[i + 1] \leftrightarrow (v[i] \oplus \land_{j=0}^{i-1} v[j])) \\
(\neg x[0]) \\
(\land_{i=0}^{6} x[i + 1] \leftrightarrow (w[i] \land x[i]) \lor (w[i] \land y[i]) \lor (x[i] \land y[i])) \\
(\land_{i=0}^{7} z[i] \leftrightarrow w[i] \oplus x[i] \oplus y[i]) \\
\end{array} \right. \]

- Compute affine relations between variables \( z, v \) and \( y \)
- Could also be our Boolean characterization of intervals
Example: Affine Closure

• 1st solution: \((v = 0, y = 0, z = 2)\)

\[
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix} \cup \begin{bmatrix} 1 \\ 0 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix}
\]

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

\[
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix} \cup \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 & -1 & -2 \\ 0 & 1 & 0 & 0 \end{bmatrix}
\]

• 3rd solution: \((v = 0, y = 1, z = 3)\)

\[
\begin{bmatrix} 2 & 0 & -1 & -2 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cup \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 1 & -1 & -2 \end{bmatrix}
\]

• 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:
  \[ i = (-3 \leq r_0 \leq 4) \]
  \[ \Rightarrow (127 \leq r_0 \leq 127) \]
  \[ \Rightarrow (r_{0_i}^* = -128 \land r_{0_u}^* = -128) \]
  \[ (-128 \leq r_0 \leq -2) \]
  \[ \Rightarrow (r_{0_i}^* = -r_{0_u} - 1 \land r_{0_u}^* = -r_{0_i} - 1) \]
  \[ (-1 \leq r_0 \leq 126) \]
  \[ \Rightarrow (r_{0_i}^* = r_{0_i} + 1 \land r_{0_u}^* = r_{0_u} + 1) \]

• Solving the two remaining linear programs then yields:
  \[ r_{0_i}^* = 0 \]
  \[ r_{0_u}^* = 5 \]
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

\[ r_0 + r_1 = d \]

\[ -2^8 \quad 0 \quad 2^8 \]

\[ \ldots \]

\[ 2^7 \]

\[ 2^6 \]

\[ 2^6 + 2^2 + 2^0 \]

© 2011 Jörg Brauer, Embedded Software Laboratory, RWTH Aachen University
Octagons using Dichotomic Search

• Consider computing guards for `ADD R0 R1` in overflow mode
• Then \( \pm r0 \pm r1 \leq d \), thus
  \[ -2^8 \leq d \leq 2^8 \]
  \[ \Leftrightarrow ( -2^8 \leq d \leq -1 ) \lor ( 0 \leq d \leq 2^8 ) \]
• Use SAT solver to find out which disjunct holds
Octagons using Dichotomotic Search

• \( \varphi \text{ encodes } \text{ADD R0 R1} \)
• Then \( \varphi' = \varphi \land (r0 + r1 = d) \)
• Is \( \varphi' \land \neg d[10] \) satisfiable? Yes!
  – Hence \( (0 \leq d \leq 2^8) \)
    \[ \Leftrightarrow (0 \leq d \leq 2^7 - 1) \lor (2^7 \leq d \leq 2^8) \]
• Proceed with \( \varphi'' = \varphi' \land \neg d[10] \land d[9] \) 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** \( \langle r_0 = 2, r_1 = 4, r_2 = 3, r_0' = 10 \rangle \)
  - Matrix
    \[
    \begin{bmatrix}
    1 & 0 & 0 & 0 & 2 \\
    0 & 1 & 0 & 0 & 4 \\
    0 & 0 & 1 & 0 & 3 \\
    0 & 0 & 0 & 1 & 10
    \end{bmatrix}
    \]
  - Add monomial for \( r_0 \cdot r_2 \) to give
    \[
    \begin{bmatrix}
    1 & 0 & 0 & 0 & 0 & 2 \\
    0 & 1 & 0 & 0 & 0 & 4 \\
    0 & 0 & 1 & 0 & 0 & 3 \\
    0 & 0 & 0 & 1 & 0 & 10 \\
    0 & 0 & 0 & 0 & 1 & 6
    \end{bmatrix}
    \]
Polynomial Extension by Example

• Now search for solution that violates $r_0 \cdot r_2 = 6$

• SAT gives $\langle r_0 = 3, r_1 = 4, r_2 = 8, r_0' = 28 \rangle$ which implies $r_0 \cdot r_2 = 24$

• Matrix representation

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & | & 3 \\
0 & 1 & 0 & 0 & 0 & | & 4 \\
0 & 0 & 1 & 0 & 0 & | & 8 \\
0 & 0 & 0 & 1 & 0 & | & 28 \\
0 & 0 & 0 & 0 & 1 & | & 24
\end{bmatrix}
$$

• Join with first solution
Polynomial Extension by Example

• After five iterations, we get the joined system

\[
\begin{bmatrix} 1 & 0 & -1 & 0 & -1 & | & 0 \end{bmatrix}
\]

• 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)
- 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!