FONDAZIONE BRUNO KESSLER

FMCAD 2011

Effective Word-Level Interpolation for Software Verification

Alberto Griggio
FBK-IRST

Motivations

- Craig interpolation applied succesfully for Formal Verification of both hardware and software
- Ongoing research (at least for 6-7 years) on efficient algorithms for computing interpolants in various useful (combinations of) theories
  - UF, LA (and fragments), data structures, arrays, quantifiers...
- Very little done for bit-vectors!
  - ...But BV are fundamental in both hardware and software verification
- This work: a “practical” procedure for BV interpolation
  - Using efficient SMT techniques
  - A first step, not a general-purpose solution
  - Optimized for problems arising in software verification

Outline

- Background
- Layered Interpolation for BV
- Discussion
- Experimental Evaluation

Interpolants

- (Craig) Interpolant for an ordered pair (A, B) of formulas s.t.

  $A \wedge B \models_{\mathcal{T}} \perp \;\; (\text{or: } A \models \neg B)$ is a formula I s.t.

  a) $A \models_{\mathcal{T}} I$

  b) $B \wedge I \models_{\mathcal{T}} \perp \;\; (I \models \neg B)$

  c) All the uninterpreted (in $\mathcal{T}$) symbols of I occur in both A and B

Lazy SMT and Interpolation

- DPLL(T) (i.e. "lazy") approach to SMT:
  SAT solver (DPLL) + decision procedure for *conjunctions* of *T*-constraints (*T*-solver)
- Interpolants from DPLL(T)-proofs [McMillan]:

Boolean part
(ground resolution)

+

$\mathcal{T}$ -specific part
(for conjunctions of constraints)

Standard Boolean
interpolation

$\mathcal{T}$-specific
interpolation
for conjunctions only

- State-of-the-art approach for solving and interpolation in several important theories (UF, LA, combinations, ...)

SMT for Bit-Vectors

FONDAZIONE BRUNO KESSLER

- State-of-the-art for SMT(BV) is NOT DPLL(T)!
  - All efficient SMT(BV) solvers are based on:
    - Aggressive preprocessing/simplification of the formula using word-level information
    - Eager encoding into SAT ("bit-blasting")
- Problem for interpolation: proofs are a "blob of bits"
  - No clear partitioning between Boolean part and BV-specific part
  - Word-level structure completely lost and difficult to recover
    - Some work done [Kroening&Weissenbacher 07], but limited to equality logic only

Outline

- Background
- Layered Interpolation for BV
- Discussion
- Experimental Evaluation

Interpolation via Bit-Blasting

Interpolation via bit-blasting is easy...

- From A_{BV} and B_{BV} generate $A_{\texttt{Bool}}$ and $B_{\texttt{Bool}}$
  - Each var x of width *n* encoded with *n* Boolean vars $b_1^x \dots b_n^x$
- Generate a Boolean interpolant $I_{\texttt{Bool}}$ for $(A_{\texttt{Bool}}, B_{\texttt{Bool}})$
- Replace every variable b_i^x in $I_{\texttt{Bool}}$ with the bit-selection $x[i]$ and every Boolean connective with the corresponding bit-wise connective: $\wedge \mapsto \&, \quad \vee \mapsto |, \quad \neg \mapsto \sim$

...but quite impractical

- Generates "ugly" interpolants
- Word-level structure of the original problem completely lost
  - How to apply word-level simplifications?

Interpolation via Bit-Blasting - Example

$$A \stackrel{\text{def}}{=} (a_{[8]} * b_{[8]} = 15_{[8]}) \wedge (a_{[8]} = 3_{[8]})$$

$$B \stackrel{\text{def}}{=} \neg(b_{[8]} \%_u c_{[8]} = 1_{[8]}) \wedge (c_{[8]} = 2_{[8]})$$

A word-level interpolant is:

$$I \stackrel{\text{def}}{=} (b_{[8]} * 3_{[8]} = 15_{[8]})$$

...but with bit-blasting we get:

$$I' \stackrel{\text{def}}{=} (b_{[8]}[0] = 1_{[1]}) \wedge ((b_{[8]}[0] \& \sim (((((((\sim b_{[8]}[7] \& \sim b_{[8]}[6]) \& \sim b_{[8]}[5]) \& \sim b_{[8]}[4]) \& \sim b_{[8]}[3]) \& b_{[8]}[2]) \& \sim b_{[8]}[1])) = 0_{[1]})$$

Lazy bit-blasting and DPLL(T) for BV

- Our goal: combine the benefits of bit-blasting for efficiently solving BV with those of DPLL(T) for interpolation

- Exploit *lazy bit-blasting*
  - Bit-blast only BV-atoms, not the whole formula
  - Boolean skeleton of the formula handled by the “main” DPLL, like in DPLL(T)
  - Conjunctions of BV-atoms handled (via bit-blasting) by a “sub”-DPLL (DPLL-BV) that acts as a BV-solver

Standard Boolean Interpolation + BV-specific Interpolation for *conjunctions of constraints*

  - Implemented using SAT solving under assumptions

Interpolation for BV constraints

A layered approach

- Apply in sequence a chain of procedures of increasing generality and cost

  - Interpolation in EUF
  - Interpolation via equality inlining
  - Interpolation via Linear Integer Arithmetic encoding
  - Interpolation via bit-blasting

Interpolation in EUF

- Treat all the BV-operators as uninterpreted functions
- Exploit cheap, efficient algorithms for solving and interpolating modulo EUF
  - Possible because we avoid bit-blasting upront!

Example:

$$A \stackrel{\text{def}}{=} (x_{1[32]} = 3_{[32]}) \wedge (x_{3[32]} = x_{1[32]} \cdot x_{2[32]})$$

$$B \stackrel{\text{def}}{=} (x_{4[32]} = x_{2[32]}) \wedge (x_{5[32]} = 3_{[32]} \cdot x_{4[32]}) \wedge \neg(x_{3[32]} = x_{5[32]})$$

$$I_{\mathsf{UF}} \stackrel{\text{def}}{=} x_3 = f^{\cdot}(f^3, x_2)$$

$$I_{\mathsf{BV}} \stackrel{\text{def}}{=} x_{3[32]} = 3_{[32]} \cdot x_{2[32]}$$

Interpolation via Equality Inlining

- Interpolation via quantifier elimination: given (A, B), an interpolant can be computed by eliminating quantifiers from $\exists_{x \not\in B} A$ or from $\exists_{x \not\in A} \neg B$

- In general, this can be very expensive for BV
  - Might require bit-blasting and can cause blow-up of the formula

- Cheap case: non-common variables occurring in "definitional" equalities
  - Example: $(x = e) \wedge \varphi$ and x does not occur in e, then

$$\exists_x ((x = e) \wedge \varphi) \Longrightarrow \varphi[x \mapsto e]$$

Interpolation via Equality Inlining

- Inline definitional equalities until either all all non-common variables are removed, or a fixpoint is reached
  - Try both from A and $\neg B$
  - If one of them succeeds, we have an interpolant

Example: $A \stackrel{\text{def}}{=} (0_{[24]} :: (x_{4[8]} \cdot x_{5[8]}) \leq_s (0_{[24]} :: x_{1[8]} - 1_{[32]})) \wedge$
$(x_{2[8]} = x_{1[8]}) \wedge (x_{4[8]} = 192_{[8]}) \wedge (x_{5[8]} = 128_{[8]})$

$B \stackrel{\text{def}}{=} ((x_{3[8]} \cdot x_{6[8]}) = (-(0_{[24]} :: x_{2[8]}))[7:0]) \wedge$
$(x_{3[8]} <_u 1_{[8]}) \wedge (0_{[8]} \leq_u x_{3[8]}) \wedge (x_{6[8]} = 1_{[8]})$

Interpolation via Equality Inlining

- Inline definitional equalities until either all all non-common variables are removed, or a fixpoint is reached
  - Try both from A and $\neg B$
  - If one of them succeeds, we have an interpolant

Example:

$$A \stackrel{\text{def}}{=} (0_{[24]} :: (x_{4\,[8]} \cdot x_{5\,[8]}) \leq_s (0_{[24]} :: x_{1\,[8]} - 1_{[32]}))\wedge \boxed{(x_{2\,[8]} = x_{1\,[8]})} \wedge \boxed{(x_{4\,[8]} = 192_{[8]})} \wedge \boxed{(x_{5\,[8]} = 128_{[8]})}$$

Definitional equalities

$$B \stackrel{\text{def}}{=} ((x_{3\,[8]} \cdot x_{6\,[8]}) = (-(0_{[24]} :: x_{2\,[8]}))[7:0])\wedge (x_{3\,[8]} <_u 1_{[8]}) \wedge (0_{[8]} \leq_u x_{3\,[8]}) \wedge \boxed{(x_{6\,[8]} = 1_{[8]})}$$

Interpolation via Equality Inlining

- Inline definitional equalities until either all all non-common variables are removed, or a fixpoint is reached
  - Try both from A and $\neg B$
  - If one of them succeeds, we have an interpolant

Example: $A \stackrel{\text{def}}{=} (0_{[24]} :: (x_{4\,[8]} \cdot x_{5\,[8]}) \leq_s (0_{[24]} :: x_{1\,[8]} - 1_{[32]})) \wedge$
$\boxed{(x_{2\,[8]} = x_{1\,[8]})} \wedge (x_{4\,[8]} = 192_{[8]}) \wedge (x_{5\,[8]} = 128_{[8]})$

$B \stackrel{\text{def}}{=} ((x_{3\,[8]} \cdot x_{6\,[8]}) = (-(0_{[24]} :: x_{2\,[8]}))[7:0]) \wedge$
$(x_{3\,[8]} <_u 1_{[8]}) \wedge (0_{[8]} \leq_u x_{3\,[8]}) \wedge (x_{6\,[8]} = 1_{[8]})$

Interpolation via Equality Inlining

- Inline definitional equalities until either all all non-common variables are removed, or a fixpoint is reached
  - Try both from A and $\neg B$
  - If one of them succeeds, we have an interpolant

Example:
$$A \stackrel{\text{def}}{=} (0_{[24]} :: (x_{4\,[8]} \cdot x_{5\,[8]}) \leq_s (0_{[24]} :: x_{2\,[8]} - 1_{[32]})) \wedge$$
$$\wedge\, (x_{4\,[8]} = 192_{[8]}) \wedge (x_{5\,[8]} = 128_{[8]})$$

$$B \stackrel{\text{def}}{=} ((x_{3\,[8]} \cdot x_{6\,[8]}) = (-(0_{[24]} :: x_{2\,[8]}))[7:0]) \wedge$$
$$(x_{3\,[8]} <_u 1_{[8]}) \wedge (0_{[8]} \leq_u x_{3\,[8]}) \wedge (x_{6\,[8]} = 1_{[8]})$$

Interpolation via Equality Inlining

- Inline definitional equalities until either all all non-common variables are removed, or a fixpoint is reached
  - Try both from A and $\neg B$
  - If one of them succeeds, we have an interpolant

Example:

$$A \stackrel{\text{def}}{=} (0_{[24]} :: ({x_4}_{[8]} \cdot {x_5}_{[8]}) \leq_s (0_{[24]} :: {x_2}_{[8]} - 1_{[32]})) \wedge$$
$$\wedge \boxed{({x_4}_{[8]} = 192_{[8]})} \wedge \boxed{({x_5}_{[8]} = 128_{[8]})}$$

$$B \stackrel{\text{def}}{=} (({x_3}_{[8]} \cdot {x_6}_{[8]}) = (-(0_{[24]} :: {x_2}_{[8]}))[7:0]) \wedge$$
$$({x_3}_{[8]} <_u 1_{[8]}) \wedge (0_{[8]} \leq_u {x_3}_{[8]}) \wedge ({x_6}_{[8]} = 1_{[8]})$$

Interpolation via Equality Inlining

- Inline definitional equalities until either all all non-common variables are removed, or a fixpoint is reached
  - Try both from A and $\neg B$
  - If one of them succeeds, we have an interpolant

Example: $A \stackrel{\text{def}}{=} (0_{[24]} :: (192_{[8]} \cdot 128_{[8]}) \leq_s (0_{[24]} :: x_{2[8]} - 1_{[32]}))$

$\wedge \qquad \wedge$

$B \stackrel{\text{def}}{=} ((x_{3[8]} \cdot x_{6[8]}) = (-(0_{[24]} :: x_{2[8]}))[7:0]) \wedge$
$(x_{3[8]} <_u 1_{[8]}) \wedge (0_{[8]} \leq_u x_{3[8]}) \wedge (x_{6[8]} = 1_{[8]})$

Interpolation via Equality Inlining

- Inline definitional equalities until either all all non-common variables are removed, or a fixpoint is reached
  - Try both from A and $\neg B$
  - If one of them succeeds, we have an interpolant

Example: $A \stackrel{\text{def}}{=} (0_{[24]} :: (192_{[8]} \cdot 128_{[8]}) \leq_s (0_{[24]} :: x_{2\,[8]} - 1_{[32]}))$

$$\wedge \qquad \wedge$$

$$I \stackrel{\text{def}}{=} (0_{32} \leq_s (0_{24} :: x_{2\,[8]} - 1_{[32]})$$

$$B \stackrel{\text{def}}{=} ((x_{3\,[8]} \cdot x_{6\,[8]}) = (-(0_{[24]} :: x_{2\,[8]}))[7:0]) \wedge$$
$$(x_{3\,[8]} <_u 1_{[8]}) \wedge (0_{[8]} \leq_u x_{3\,[8]}) \wedge (x_{6\,[8]} = 1_{[8]})$$

Interpolation via LIA Encoding

- Simple idea (in principle):
  - Encode a set of BV-constraints into an SMT(LIA)-formula
  - Generate a LIA-interpolant using existing algorithms
  - Map back to a BV-interpolant

- However, several problems to solve:
  - Efficiency *(see paper)*
  - More importantly, soundness

Encoding BV into LIA

- Use encoding of e.g. [PDPAR'06]
  - Encode each BV term $t_{[n]}$ as an integer variable x_t and the constraints $(0 \leq x_t) \wedge (x_t \leq 2^n - 1)$
  - Encode each BV operation as a LIA-formula.

Examples:

$$t_{[i-j+1]} \stackrel{\text{def}}{=} t_{1[n]}[i:j] \Rightarrow (x_t = m) \wedge (x_{t_1} = 2^{i+1}h + 2^j m + l) \wedge$$
$$l \in [0, 2^i) \wedge m \in [0, 2^{i-j+1}) \wedge h \in [0, 2^{n-i-1})$$

$$t_{[n]} \stackrel{\text{def}}{=} t_{1[n]} + t_{2[n]} \Rightarrow (x_t = x_{t_1} + x_{t_2} - 2^n \sigma) \wedge (0 \leq \sigma \leq 1)$$

$$t_{[n]} \stackrel{\text{def}}{=} t_{1[n]} \cdot k \Rightarrow (x_t = k \cdot x_{t_1} - 2^n \sigma) \wedge (0 \leq \sigma \leq k)$$

From LIA-interpolants to BV-interpolants

- "Invert" the LIA encoding to get a BV interpolant
- *Unsound* in general
  - Issues due to overflow and (un)signedness of operations
- Our (very simple) solution: *check the interpolants*
  - Given a candidate interpolant $\hat{I}$, use our SMT(BV) solver to check the unsatisfiability of $(A \wedge \neg\hat{I}) \vee (B \wedge \hat{I})$
    - If successful, then $\hat{I}$ is an interpolant

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} (y_{1[8]} = y_{5[4]} :: y_{5[4]}) \wedge (y_{1[8]} = y_{2[8]}) \wedge (y_{5[4]} = 1_{[4]})$$

$$B \stackrel{\text{def}}{=} \neg(y_{4[8]} + 1_{[8]} \leq_u y_{2[8]}) \wedge (y_{4[8]} = 1_{[8]})$$

- Encoding into LIA:

$$\begin{aligned} A_{\text{LIA}} \stackrel{\text{def}}{=} & (x_{y_2} = 16x_{y_5} + x_{y_5}) \wedge (x_{y_1} = x_{y_2}) \wedge (x_{y_5} = 1) \wedge \\ & (x_{y_1} \in [0, 2^8)) \wedge (x_{y_2} \in [0, 2^8)) \wedge (x_{y_5} \in [0, 2^4)) \end{aligned}$$

$$\begin{aligned} B_{\text{LIA}} \stackrel{\text{def}}{=} & \neg(x_{y_4+1} \leq x_{y_2}) \wedge (x_{y_4+1} = x_{y_4} + 1 - 2^8\sigma) \wedge \\ & (x_{y_4} = 1) \wedge \\ & (x_{y_4+1} \in [0, 2^8)) \wedge (x_{y_4} \in [0, 2^8)) \wedge (0 \leq \sigma \leq 1) \end{aligned}$$

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} (y_{1[8]} = y_{5[4]} :: y_{5[4]}) \wedge (y_{1[8]} = y_{2[8]}) \wedge (y_{5[4]} = 1_{[4]})$$

$$B \stackrel{\text{def}}{=} \neg(y_{4[8]} + 1_{[8]} \leq_u y_{2[8]}) \wedge (y_{4[8]} = 1_{[8]})$$

- LIA-Interpolant:

$$I_{\text{LIA}} \stackrel{\text{def}}{=} (17 \leq x_{y_2})$$

- BV-interpolant:

$$I \stackrel{\text{def}}{=} (17_{[8]} \leq_u y_{2[8]})$$

Good!

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} (y_{2[8]} = 81_{[8]}) \wedge (y_{3[8]} = 0_{[8]}) \wedge (y_{4[8]} = y_{2[8]})$$

$$B \stackrel{\text{def}}{=} (y_{13[16]} = 0_{[8]} :: y_{4[8]}) \wedge (255_{[16]} \leq_u y_{13[16]} + (0_{[8]} :: y_{3[8]}))$$

- Encoding into LIA:

$$A_{\text{LIA}} \stackrel{\text{def}}{=} (x_{y_2} = 81) \wedge (x_{y_3} = 0) \wedge (x_{y_4} = x_{y_2}) \wedge$$
$$(x_{y_2} \in [0, 2^8)) \wedge (x_{y_3} \in [0, 2^8)) \wedge (x_{y_4} \in [0, 2^8))$$

$$B_{\text{LIA}} \stackrel{\text{def}}{=} (x_{y_{13}} = 2^8 \cdot 0 + x_{y_4}) \wedge (255 \leq x_{y_{13}+(0::y_3)}) \wedge$$
$$(x_{y_{13}+(0::y_3)} = x_{y_{13}} + 2^8 \cdot 0 + x_{y_3} - 2^{16}\sigma) \wedge$$
$$(x_{y_{13}} \in [0, 2^{16})) \wedge (x_{y_{13}+(0::y_3)} \in [0, 2^{16})) \wedge$$
$$(0 \leq \sigma \leq 1)$$

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} (y_{2[8]} = 81_{[8]}) \wedge (y_{3[8]} = 0_{[8]}) \wedge (y_{4[8]} = y_{2[8]})$$

$$B \stackrel{\text{def}}{=} (y_{13[16]} = 0_{[8]} :: y_{4[8]}) \wedge (255_{[16]} \leq_u y_{13[16]} + (0_{[8]} :: y_{3[8]}))$$

- LIA-interpolant:

$$I_{\text{LIA}} \stackrel{\text{def}}{=} (x_{y_3} + x_{y_4} \leq 81)$$

- BV-interpolant:

$$\hat{I} \stackrel{\text{def}}{=} (y_{3[8]} + y_{4[8]} \leq_u 81_{[8]})$$

Wrong!

$$B \wedge \hat{I} \not\models \bot$$

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} (y_{2[8]} = 81_{[8]}) \wedge (y_{3[8]} = 0_{[8]}) \wedge (y_{4[8]} = y_{2[8]})$$

$$B \stackrel{\text{def}}{=} (y_{13[16]} = 0_{[8]} :: y_{4[8]}) \wedge (255_{[16]} \leq_u y_{13[16]} + (0_{[8]} :: y_{3[8]}))$$

- LIA-interpolant:

$$I_{\text{LIA}} \stackrel{\text{def}}{=} (x_{y_3} + x_{y_4} \leq 81)$$

Addition might overflow in BV!

- BV-interpolant:

$$\hat{I} \stackrel{\text{def}}{=} (y_{3[8]} + y_{4[8]} \leq_u 81_{[8]})$$

Wrong!
$B \wedge \hat{I} \not\models \bot$

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} (y_{2[8]} = 81_{[8]}) \wedge (y_{3[8]} = 0_{[8]}) \wedge (y_{4[8]} = y_{2[8]})$$

$$B \stackrel{\text{def}}{=} (y_{13[16]} = 0_{[8]} :: y_{4[8]}) \wedge (255_{[16]} \leq_u y_{13[16]} + (0_{[8]} :: y_{3[8]}))$$

- LIA-interpolant:

$$I_{\text{LIA}} \stackrel{\text{def}}{=} (x_{y_3} + x_{y_4} \leq 81)$$

Addition might overflow in BV!

- BV-interpolant:

A correct interpolant would be

$$I \stackrel{\text{def}}{=} (0_{[1]} :: y_{3[8]} + 0_{[1]} :: y_{4[8]} \leq_u 81_{[9]})$$

Wrong!

$B \wedge \hat{I} \not\models \perp$

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} \neg({y_4}_{[8]} + 1_{[8]} \leq_u {y_3}_{[8]}) \wedge ({y_2}_{[8]} = {y_4}_{[8]} + 1_{[8]})$$

$$B \stackrel{\text{def}}{=} ({y_2}_{[8]} + 1_{[8]} \leq_u {y_3}_{[8]}) \wedge ({y_7}_{[8]} = 3_{[8]}) \wedge ({y_7}_{[8]} = {y_2}_{[8]} + 1_{[8]})$$

- Encoding into LIA:

$$\begin{aligned} A_{\text{LIA}} \stackrel{\text{def}}{=} & \neg(x_{y_4+1} \leq x_{y_3}) \wedge (x_{y_2} = x_{y_4+1}) \wedge \\ & (x_{y_4+1} = x_{y_4} + 1 - 2^8\sigma_1) \wedge \\ & (x_{y_2} \in [0, 2^8)) \wedge (x_{y_3} \in [0, 2^8)) \wedge (x_{y_4} \in [0, 2^8)) \wedge \\ & (x_{y_4+1} \in [0, 2^8)) \wedge (0 \leq \sigma_1 \leq 1) \end{aligned}$$

$$\begin{aligned} B_{\text{LIA}} \stackrel{\text{def}}{=} & (x_{y_2+1} \leq x_{y_3}) \wedge (x_{y_7} = 3) \wedge (x_{y_7} = x_{y_2+1}) \wedge \\ & (x_{y_2+1} = x_{y_2} + 1 - 2^8\sigma_2) \wedge \\ & (x_{y_7} \in [0, 2^8)) \wedge (x_{y_2+1} \in [0, 2^8)) \wedge (0 \leq \sigma_2 \leq 1) \end{aligned}$$

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} \neg({y_4}_{[8]} + 1_{[8]} \leq_u {y_3}_{[8]}) \wedge ({y_2}_{[8]} = {y_4}_{[8]} + 1_{[8]})$$

$$B \stackrel{\text{def}}{=} ({y_2}_{[8]} + 1_{[8]} \leq_u {y_3}_{[8]}) \wedge ({y_7}_{[8]} = 3_{[8]}) \wedge ({y_7}_{[8]} = {y_2}_{[8]} + 1_{[8]})$$

- LIA-interpolant:

$$I_{\text{LIA}} \stackrel{\text{def}}{=} (-255 \leq x_{y_2} - x_{y_3} + 256\lfloor -1\frac{x_{y_2}}{256}\rfloor)$$

- BV-interpolant: (after fixing overflows)

$$\hat{I}' \stackrel{\text{def}}{=} (65281_{[16]} \leq_u (0_{[8]} :: {y_2}_{[8]}) - (0_{[8]} :: {y_3}_{[8]}) + 256_{[16]} \cdot (65535_{[16]} \cdot (0_{[8]} :: {y_2}_{[8]}) /_u 256_{[16]}))$$

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} \neg({y_4}_{[8]} + 1_{[8]} \leq_u {y_3}_{[8]}) \wedge ({y_2}_{[8]} = {y_4}_{[8]} + 1_{[8]})$$

$$B \stackrel{\text{def}}{=} ({y_2}_{[8]} + 1_{[8]} \leq_u {y_3}_{[8]}) \wedge ({y_7}_{[8]} = 3_{[8]}) \wedge ({y_7}_{[8]} = {y_2}_{[8]} + 1_{[8]})$$

- LIA-interpolant:

$$I_{\text{LIA}} \stackrel{\text{def}}{=} (-255 \leq x_{y_2} - x_{y_3} + 256\lfloor -1\frac{x_{y_2}}{256} \rfloor)$$

- BV-interpolant: (after fixing overflows)

$$\hat{I}' \stackrel{\text{def}}{=} (65281_{[16]} \leq_u (0_{[8]} :: {y_2}_{[8]}) - (0_{[8]} :: {y_3}_{[8]}) + 256_{[16]} \cdot (65535_{[16]} \cdot (0_{[8]} :: {y_2}_{[8]}) /_u 256_{[16]}))$$

In this case, the problem is also the sign

Still Wrong!

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} \neg(y_{4[8]} + 1_{[8]} \leq_u y_{3[8]}) \wedge (y_{2[8]} = y_{4[8]} + 1_{[8]})$$

$$B \stackrel{\text{def}}{=} (y_{2[8]} + 1_{[8]} \leq_u y_{3[8]}) \wedge (y_{7[8]} = 3_{[8]}) \wedge (y_{7[8]} = y_{2[8]} + 1_{[8]})$$

- LIA-interpolant:

$$I_{\text{LIA}} \stackrel{\text{def}}{=} (-255 \leq x_{y_2} - x_{y_3} + 256\lfloor -1\frac{x_{y_2}}{256}\rfloor)$$

- BV-interpolant:

$$I \stackrel{\text{def}}{=} (65281_{[16]} \leq_s (0_{[8]} :: y_{2[8]}) - (0_{[8]} :: y_{3[8]}) + 256_{[16]} \cdot (65535_{[16]} \cdot (0_{[8]} :: y_{2[8]}) /_u 256_{[16]}))$$

Correct interpolant

Outline

- Background
- Layered Interpolation for BV
- Discussion
- Experimental Evaluation

Discussion

- In the worst case, our algorithm is not much different than bit-blasting
- Actually, it can be even worse, performance-wise
  - Need to re-process the BV-lemmas after having checked unsatisfiability of $A \wedge B$
- However:
  - for interpolation problems arising in software verification, our specialized procedures succeed most of the times
  - In general, the overhead of running them is minor
    - The BV-lemmas occurring in the proof are only a small percentage of the total generated during search; and
    - They are typically small (close to minimal)

Interpolants in software verification

- Refinements of "spurious" paths in an abstract program unwinding
- Two observations:
  - Most arithmetic constraints are "simple"
    - Esp. In typical domains for sw verification (e.g. device drivers)
    - *LIA encoding works well*
  - Use of an SSA encoding:
    - Many "definitional" equalities, corresponding to assignment operations
    - *Exploited by our equality inlining layer*

SSA Example:

```
x := z
assume(x >= 0)
x := x + 2
z = y - 3
assume(z = 1)
```

$$x_0 = z_0 \wedge$$
$$x_0 \geq 0 \wedge$$
$$x_1 = x_0 + 2 \wedge$$
$$z_1 = y_0 - 3 \wedge$$
$$z_1 = 1$$

Outline

- Background
- Layered Interpolation for BV
- Discussion
- Experimental Evaluation

Experimental evaluation

- Implementation within the MathSAT 5 SMT solver
- Integration with the Kratos SW model checker
  - CEGAR-based lazy predicate abstraction with interpolation-based refinement
- Comparison with the other bit-precise engines available
  - Satabs
  - Wolverine

- Benchmarks that require a bit-precise semantics, collected from multiple sources

Results – programs requiring BV

Program	Kratos BV-1	BV-2	BV-3	BV-4	BV-5	SatAbs	Wolverine
byte_add_1.c	31.00	T.O.	M.O.	57.30	31.54	T.O.	T.O.
byte_add_2.c	47.98	T.O.	M.O.	72.17	44.42	T.O.	T.O.
num_conversion_1.c	1.85	3.20	3.67	2.67	1.13	23.78	2.16
num_conversion_2.c	48.04	776.53	72.12	763.16	47.73	T.O.	T.O.
gcd_1.c	1.75	20.45	20.56	1.05	1.27	FAIL	515.31
gcd_2.c	29.21	M.O.	M.O.	39.21	28.21	339.86	185.56
gcd_3.c	70.05	T.O.	M.O.	209.34	70.59	T.O.	290.03
gcd_4.c	3.58	M.O.	T.O.	T.O.	4.25	T.O.	1.26
interleave_bits.c	45.90	T.O.	T.O.	T.O.	49.01	836.78	T.O.
modulus.c	4.87	34.00	M.O.	3.30	4.15	T.O.	M.O.
parity.c	387.56	M.O.	M.O.	T.O.	391.84	T.O.	T.O.
soft_float_1.c.cil.c	48.02	T.O.	T.O.	T.O.	T.O.	T.O.	136.88
soft_float_2.c.cil.c	61.34	T.O.	T.O.	70.02	T.O.	1101.54	177.63
soft_float_3.c.cil.c	T.O.	T.O.	T.O.	T.O.	T.O.	T.O.	T.O.
soft_float_4.c.cil.c	51.67	T.O.	M.O.	247.31	49.88	T.O.	T.O.
soft_float_5.c.cil.c	61.70	T.O.	T.O.	78.54	T.O.	T.O.	193.76
s3_clnt_1.BV.c.cil.c	41.06	50.82	T.O.	48.77	42.32	FAIL	T.O.
s3_clnt_2.BV.c.cil.c	20.96	9.92	116.03	8.59	22.01	T.O.	T.O.
s3_clnt_3.BV.c.cil.c	7.66	T.O.	93.77	T.O.	6.68	T.O.	T.O.
s3_srvr_1.BV.c.cil.c	11.59	35.91	240.77	34.74	11.63	160.74	T.O.
s3_srvr_2.BV.c.cil.c	150.64	62.22	116.54	61.26	152.10	342.11	T.O.
s3_srvr_3.BV.c.cil.c	48.35	124.32	43.63	125.19	48.36	405.48	T.O.
jain_1.c	0.34	0.39	0.30	0.12	0.36	FAIL	T.O.
jain_2.c	0.43	0.48	0.35	0.21	0.44	FAIL	T.O.
jain_4.c	0.55	0.60	0.40	0.33	0.54	FAIL	T.O.
jain_5.c	T.O.	T.O.	T.O.	T.O.	T.O.	FAIL	T.O.
jain_6.c	0.18	0.12	0.09	0.15	0.16	FAIL	T.O.
jain_7.c	0.29	0.23	0.15	0.26	0.27	FAIL	T.O.
TOTAL (solved / time)	**26 / 1176.57**	**14 / 1119.19**	**13 / 708.38**	**21 / 1823.69**	**23 / 1008.89**	**7 / 3210.29**	**8 / 1500.43**

Conclusions

- Interpolation in BV is hard...
- ...this is a conceptually-simple approach:
  - Exploits efficient SMT solving and interpolation techniques
  - Aimed at “practical” problems arising in software verification
  - Promising experimental results
  - A first step, not a general-purpose solution
- Several directions for future work
  - Incorporate more layers
  - Investigate more deeply encoding into LIA
  - “Lifting” of bit-level proofs to word-level interpolants beyond equality logic

FONDAZIONE
BRUNO KESSLER

Thank You