Proposal for a Simple Static Analyzer for Verilog

서울대학교 전기·정보공학부 2018-12602 이준협

1 What language?

```
reg [15:0] a;
reg [15:0] b;
initial begin
  a = -1'b1 + {-1'b1}; // a becomes 0
  a = -1'b1 - 1'b1; // a becomes 2^16 - 2
  b = -1'b1 == {-1'b1}; // LHS: 1'b1, RHS: 1'b1 => b becomes 1
  b = (-1'b1 + 2'b0) == {-1'b1}; // LHS: 2'b11, RHS: 2'b01 => b becomes 0
end
```

Figure 1: Verilog code illustrating why it is easy to make mistakes.

The above figure is a snippet of fabulous Verilog code demonstrating the obfuscated conventions set by the IEEE committee for determining bit-width. Declaration of variables must always be carefully done to prevent overflow, and comparison operators must be used while considering whether all operands are signed or not. It is relatively easy, therefore, to make mistakes that create a state that can never be escaped from. For example, a variable with bit width 4 compared with 16 will always result in the result being evaluated to 1.

2 What property?

For each expression e, we want to identify the bit-width, signedness, and range of the expression. Furthermore, if for each state we overapproximate the memories that can enter the state, and after executing the command the changed memories may enter that state again, the state may not be escapable from.

3 Abstract Syntax

Width	w	$\in$	$\mathbb{N}_0$	
Integer value	n	$\in$	$\mathbb{Z}$	
Binary operation	$\circledcirc$	$\in$	$\{+, -, \&, \&\&, ==, <\}$	
	P	::=	`always @(posedge clk)` C	*program, at every rising edge do* C
	C	::=	$(S)^*\ (1 \Rightarrow \varepsilon)$	*command*
	S	::=	$E \Rightarrow A^*$	*state, when* $E \neq 0$ *perform assignments*
	A	::=	$x[w :\ w] := E$	*assignment of* x *from index* w_1 *to* $w_1 + w_2 - 1$
	E	::=	(w, s, n)	*signed integer with width* w *and value* n
		$\mid$	(w, u, n)	*unsigned integer with width* w *and value* n
		$\mid$	x	*variable*
		$\mid$	$x[E :\ w]$	*part-selection from index* E *to* $E + w - 1$
		$\mid$	$E \circledcirc E$	*binary operation*
		$\mid$	$\{E^+\}$	*concatenation*

Note that for each variable, there can be only <u>one</u> assignment to it in a single state. Multiple drivers result in undefined behavior! (What would happen if one assignment wants to drive the flip-flop high but some other assignment tries to drive the flip-flop low?)

4 Concrete Semantics

In Verilog, every expression needs to have a statically determined width and a signedness interpretation. In hardware terms, it corresponds to how many wires the expression will require and what the arithmetic comparison operations

will do. Will `0xffff` be interpreted as -1 or will it be interpreted as $2^{16}-1$? Thus the semantic domains are all annotated with their width and signedness.

Var = PgmVar $\to$ Width $\times$ Signed $\subset$ PgmVar $\times$ Width $\times$ Signed is the set of program variables in the program of interest(PgmVar) annotated with their bit-width(Width = $\mathbb{N}_0$) and their signed-ness(Signed = $\{s,u\}$). Since for every expression the width and signed-ness can be determined statically, the collecting semantics only has to collect the $\mathbb{Z}$ part of the expression result. The type of the (non-collecting) semantic domains are:

$$\text{Mem} = \text{Var} \to \mathbb{Z} \quad \underline{C} \in \text{Mem} \to \text{Mem} \quad \underline{S} \in \text{Mem} \to \text{Mem} \quad \underline{A} \in \text{Mem} \to \text{Mem} \quad \underline{E} \in \text{Mem} \to \text{Width} \times \text{Signed} \times \mathbb{Z}$$

The width and signedness of an expression is decided by the context of the expression, that is, the expression $x+y$ may or may not overflow depending on the context. If it is in a expression such as $x+y+(32,s,0)$, when 32 is wide enough to hold the calculation result of $x+y$, the most significant bit will not be lost. However, when it is in a expression such as $x+y+(16,s,0)$ when x or y has a larger bit width than 16, the result might overflow. Therefore, the integer value of the computation needs to be held in full precision *until the last moment.*

When the width is finally determined, the integer value needs to go through a restriction of width and sign. This restriction is defined as:

$$|\cdot| : \text{Width} \times \text{Signed} \times \mathbb{Z} \to \mathbb{Z} \qquad |(w,u,n)| \coloneqq (n \bmod 2^w) \qquad |(w,s,n)| \coloneqq 2 \times (n \bmod 2^{w-1}) - (n \bmod 2^w)$$

Also, the notation for what state the initial memory m corresponds to when it goes through the command C is written as:

$$m \Rightarrow_C S_i, \text{ when } \bigwedge_{j<i} [|\underline{E_j} m| = 0] \wedge [|\underline{E_i} m| \neq 0]$$

and we read this as "the memory m is *matched with* state S_i".

A Verilog program P with initial memory m corresponds to the infinite sequence $\underline{P}m \coloneqq \{m_i\}_{i\geq 0}$ with $m_0 \coloneqq m$, $m_{i+1} \coloneqq \underline{C} m_i (i \geq 0)$. We want to check if there exists a state S_i (except the trivial last state) such that if $m \Rightarrow_C S_i$, then $\forall n \geq 0, m_n \Rightarrow_C S_i$.

Such a state can only exist if an external signal stalls the state machine, for example when reading from an external server gets abruptly disconnected due to physical failure.

Now we dive into the actual definitions for the semantic functions.

4.1 Semantic function for expressions

In this section the recursive definition for $\underline{E}$ will be given.

Notation: $\underline{E_i} m = (w_i, \sigma_i, n_i)$, $\sigma_1 \sqcup \sigma_2 \coloneqq (\sigma_1 = \sigma_2 = s ? s : u)$

$$\begin{aligned}
m &\in \text{Mem} \\
\underline{(w,u,n)} m &\coloneqq (w,u,n) \\
\underline{(w,s,n)} m &\coloneqq (w,s,n) \\
\underline{x} m &\coloneqq (w_x, \sigma_x, m(w_x,\sigma_x,x)) && (w_x,\sigma_x,x) \in \text{Dom}(m) \\
\underline{x[E:w]} m &\coloneqq (w,u,|(w,u,\lfloor 2^{-|\underline{E}m|} m(w_x,\sigma_x,x)\rfloor)|) && (w_x,\sigma_x,x) \in \text{Dom}(m) \\
\underline{E_1 + E_2} m &\coloneqq (\max\{w_1,w_2\}, \sigma_1 \sqcup \sigma_2, n_1 + n_2) \\
\underline{E_1 - E_2} m &\coloneqq (\max\{w_1,w_2\}, \sigma_1 \sqcup \sigma_2, n_1 - n_2) \\
\underline{E_1 \& E_2} m &\coloneqq \left(\max\{w_1,w_2\}, \sigma_1 \sqcup \sigma_2, \sum_{i\geq 0} 2^i \left(\left\lfloor \frac{n_1}{2^i} \right\rfloor - 2\left\lfloor \frac{n_1}{2^{i+1}} \right\rfloor\right)\left(\left\lfloor \frac{n_2}{2^i} \right\rfloor - 2\left\lfloor \frac{n_2}{2^{i+1}} \right\rfloor\right)\right) && \text{when } n_1 \geq 0 \vee n_2 \geq 0 \\
&\coloneqq \left(\max\{w_1,w_2\}, \sigma_1 \sqcup \sigma_2, \sum_{i\geq 0} 2^i \left(\left(\left\lfloor \frac{n_1}{2^i} \right\rfloor - 2\left\lfloor \frac{n_1}{2^{i+1}} \right\rfloor\right)\left(\left\lfloor \frac{n_2}{2^i} \right\rfloor - 2\left\lfloor \frac{n_2}{2^{i+1}} \right\rfloor\right) - 1\right) - 1\right) && \text{otherwise} \\
\underline{E_1 \&\& E_2} m &\coloneqq (1,u,|(\max\{w_1,w_2\}, \sigma_1 \sqcup \sigma_2, n_1)| = 0 \vee |(\max\{w_1,w_2\}, \sigma_1 \sqcup \sigma_2, n_2)| = 0 ? 0 : 1) \\
\underline{E_1 == E_2} m &\coloneqq (1,u,|(\max\{w_1,w_2\}, \sigma_1 \sqcup \sigma_2, n_1)| = |(\max\{w_1,w_2\}, \sigma_1 \sqcup \sigma_2, n_2)| ? 1 : 0) \\
\underline{E_1 < E_2} m &\coloneqq (1,u,|(\max\{w_1,w_2\}, \sigma_1 \sqcup \sigma_2, n_1)| < |(\max\{w_1,w_2\}, \sigma_1 \sqcup \sigma_2, n_2)| ? 1 : 0) \\
\underline{\{E_1,\cdots,E_n\}} m &\coloneqq \left(\sum_{i=1}^n w_i, u, \sum_{i=1}^n 2^{\sum_{0\leq j<i} w_j} |(w_i,u,n_i)|\right)
\end{aligned}$$

As can be seen, *when* the restriction operation $|\cdot|$ is performed is a subtle issue that can possibly change the result of the computation. The rules above model the specification of Verilog well but is not exactly compliant with the IEEE

specs. For example, performing $(2, s, -1) + (2, u, 3) - (3, u, 7)$ does not work by reducing $(2, s, -1) + (2, u, 3)$ to $(2, u, 2)$ then reducing the expression to $(3, u, -5) = \texttt{011}$. The specs mandate that the bit-width of the expression $(= 3)$ be determined, then all of the operands that are "influenced" by that operand be extended to that bit-width, then all operations are performed *while* being restricted by that bit-width. So what actually happens is that $(2, s, -1)$ is sign-extended to `111`, $(2, u, 3)$ is zero-extended to `011`, and `111-011+111` is performed. `111-011=100` and `100+111=011`, so the result is `011`. This is same as what is predicted by applying $\underline{E}$.

Why this works is because the result of the computation is kept with "infinite width" before being restricted. If $(2, s, -1) + (2, u, 3)$ was prematurely restricted with width 2, then the MSB would've been lost. If one sees the definition of $\underline{E}$, the only places that restriction happens on the result of some computation $\underline{E_i}m$ is on (1) the index of part-selection (2) the operands of logical operators $(\&\&, ==, <)$ (3) the operands for concatenation. These are the "boundaries" set by the IEEE specification for determining bit-width, i.e how much expressions should one consider before deciding on the maximum bit-width. One more boundary is when an expression is assigned to a variable; that case will be dealt with in the next subsection.

4.2 Semantic function for assignments

Assignments of the form $x[w_1 : w_2] \coloneqq E_1$ update the value of x to

$$\left|\left(w_x, \sigma_x, n_x - \left\lfloor 2^{w_1} \left| \underline{x[(32, u, w_1) : w_2]}m \right| \right\rfloor + \lfloor 2^{w_1} |(w_2, u, n_1)| \rfloor\right)\right|$$

that is, the bits from w_1 to $w_1 + w_2 - 1$ are cleared and are filled with the lower w_2 bits of n_1.

Since all assignments occur parallelly and there can only be one assignment per variable in one state, this suffices to specify the behavior of a command.

4.3 Semantic function for states

The semantic function for a state $S = E \Rightarrow A_1 \cdots A_n$ is simply the composition of the semantic function of the assignments. That is, $\underline{S} \coloneqq \underline{A_1} \circ \cdots \circ \underline{A_n}$.

4.4 Semantic function for commands

The semantic function for a command consists of first matching the memory to a state, then performing the assignments that corresponds to that state. That is,

$$\underline{C}m \coloneqq \underline{S}m, \text{ when } m \Rightarrow_C S$$

Note that there is only one state S satisfying $m \Rightarrow_C S$, since by the definition of $\Rightarrow_C$, the conditions for each match $\bigwedge_{j<i}[|\underline{E_j}m| = 0] \wedge [|\underline{E_i}m| \neq 0]$ are mutually exclusive. Thus $\underline{C}$ is well-defined.

5 Abstract Semantics

Using the convention $f(A) \coloneqq \{f(x) | x \in A\}$, all the semantic functions defined in the last section can be extended to a function for $\wp(\text{Mem})$. Now we overapproximate $\wp(\text{Mem})$ by defining the abstract memory domain.

$$I = \{[a, b] | a \leq b, a, b \in [-\infty, \infty]\} \qquad \text{Mem}^\# = \text{Var} \to I$$

$\text{Mem}^\#$ is Galois connected with $\wp(\text{Mem})$ since $\wp(\text{Mem})$ is Galois connected with $\text{Var} \to \wp(\mathbb{Z})$, and I is Galois connected with $\wp(\mathbb{Z})$.

The types for the abstract semantic functions are obvious:

$$\underline{C}^\# \in \text{Mem}^\# \to \text{Mem}^\# \qquad \underline{S}^\# \in \text{Mem}^\# \to \text{Mem}^\# \qquad \underline{A}^\# \in \text{Mem}^\# \to \text{Mem}^\# \qquad \underline{E}^\# \in \text{Mem}^\# \to \text{Width} \times \text{Signed} \times I$$

The abstract restriction function must also be defined:

$$|(w, u, [a, b])|^\# \coloneqq \begin{cases} [0, 2^w - 1] & (b - a \geq 2^w - 1) \\ [0, 2^w - 1] & (b - a < 2^w - 1 \wedge |(w, u, a)| \geq |(w, u, b)|) \\ [|(w, u, a)|, |(w, u, b)|] & (\textit{otherwise}) \end{cases}$$

$$|(w, s, [a, b])|^\# \coloneqq \begin{cases} [-2^{w-1}, 2^{w-1} - 1] & (b - a \geq 2^w - 1) \\ [-2^{w-1}, 2^{w-1} - 1] & (b - a < 2^w - 1 \wedge |(w, s, a)| \geq |(w, s, b)|) \\ [|(w, s, a)|, |(w, s, b)|] & (\textit{otherwise}) \end{cases}$$

Now we define the abstract matching relation.

$$m^\# \Rightarrow_C^\# S_i, \text{ when } \bigwedge_{j<i} [0 \in |\underline{E_j}^\# m^\#|^\#] \wedge [|\underline{E_i}^\# m^\#|^\# \not\sqsubseteq [0,0]]$$

Note that an abstract memory can be matched with multiple states. For each state S_i we wish to compute a memory that can match with S_i, that is, a memory $m_i^\#$ such that $[m \Rightarrow_C S_i] \Rightarrow [m \in \gamma(m_i^\#)]$. The smallest $m_i^\#$ satisfying this condition will be the abstraction of the memories that are matched with S_i, that is: $\alpha(\{m | m \Rightarrow_C S_i\})$.

5.1 Semantic function for expressions

Note that there cannot be any infinite intervals, since all expressions must eventually be restricted by their bit-width.

Notation: $\underline{E_i}^\# m^\# = (w_i, \sigma_i, [a_i, b_i])$, $m^\#(w_x, \sigma_x, x) = [a_x, b_x]$

$$
\begin{aligned}
m^\# \in & \text{Mem}^\# \\
\underline{(w,u,n)}^\# m^\# &:= (w, u, [n,n]) \\
\underline{(w,s,n)}^\# m^\# &:= (w, s, [n,n]) \\
\underline{x}^\# m^\# &:= (w_x, \sigma_x, m^\#(w_x, \sigma_x, x)) && (w_x, \sigma_x, x) \in \text{Dom}(m^\#) \\
\underline{x[E : w]}^\# m^\# &:= (w, u, |(w, u, [\min\{\lfloor 2^{-a} a_x \rfloor, \lfloor 2^{-b} a_x \rfloor\}, \max\{\lfloor 2^{-a} b_x \rfloor, \lfloor 2^{-b} b_x \rfloor\}])|^\#) && |\underline{E}^\# m^\#|^\# = [a,b] \\
\underline{E_1 + E_2}^\# m^\# &:= (\max\{w_1, w_2\}, \sigma_1 \sqcup \sigma_2, [a_1 + a_2, b_1 + b_2]) \\
\underline{E_1 - E_2}^\# m^\# &:= (\max\{w_1, w_2\}, \sigma_1 \sqcup \sigma_2, [a_1 - b_2, b_1 - a_2]) \\
\underline{E_1 \& E_2}^\# m^\# &:= (\max\{w_1, w_2\}, \sigma_1 \sqcup \sigma_2, [0, \min\{b_1, b_2\}]) && a_1 \geq 0 \wedge a_2 \geq 0 \\
&:= (\max\{w_1, w_2\}, \sigma_1 \sqcup \sigma_2, [0, b_1]) && a_1 \geq 0 \wedge a_2 < 0 \leq b_2 \\
&:= (\max\{w_1, w_2\}, \sigma_1 \sqcup \sigma_2, [\max\{a_1 + a_2 + 1, 0\}, b_1]) && a_1 \geq 0 \wedge b_2 < 0 \\
&:= (\max\{w_1, w_2\}, \sigma_1 \sqcup \sigma_2, [0, b_2]) && a_1 < 0 \leq b_1 \wedge a_2 \geq 0 \\
&:= (\max\{w_1, w_2\}, \sigma_1 \sqcup \sigma_2, [a_1 + a_2 + 1, \max\{b_1, b_2\}]) && a_1 < 0 \leq b_1 \wedge a_2 < 0 \leq b_2 \\
&:= (\max\{w_1, w_2\}, \sigma_1 \sqcup \sigma_2, [a_1 + a_2 + 1, b_1]) && a_1 < 0 \leq b_1 \wedge b_2 < 0 \\
&:= (\max\{w_1, w_2\}, \sigma_1 \sqcup \sigma_2, [\max\{a_1 + a_2 + 1, 0\}, b_2]) && b_1 < 0 \wedge a_2 \geq 0 \\
&:= (\max\{w_1, w_2\}, \sigma_1 \sqcup \sigma_2, [a_1 + a_2 + 1, b_2]) && b_1 < 0 \wedge a_2 < 0 \leq b_2 \\
&:= (\max\{w_1, w_2\}, \sigma_1 \sqcup \sigma_2, [a_1 + a_2 + 1, \min\{b_1, b_2\}]) && b_1 < 0 \wedge b_2 < 0 \\
\underline{E_1 \&\& E_2}^\# m^\# &:= (1, u, [a_1', b_1'] \sqsubseteq [0,0] \vee [a_2', b_2'] \sqsubseteq [0,0] ? [0,0] : (&& |(\max\{w_1, w_2\}, \sigma_1 \sqcup \sigma_2, [a_i, b_i])|^\# = [a_i', b_i'] \\
&\quad 0 \notin [a_1', b_1'] \wedge 0 \notin [a_2', b_2'] ? [1,1] : [0,1])) \\
\underline{E_1 == E_2}^\# m^\# &:= (1, u, a_1' = b_1' = a_2' = b_2' ? [1,1] : (&& |(\max\{w_1, w_2\}, \sigma_1 \sqcup \sigma_2, [a_i, b_i])|^\# = [a_i', b_i'] \\
&\quad [a_1', b_1'] \sqcap [a_2', b_2'] = \bot ? [0,0] : [0,1])) \\
\underline{E_1 < E_2}^\# m^\# &:= (1, u, b_1' < a_2' ? [1,1] : (&& |(\max\{w_1, w_2\}, \sigma_1 \sqcup \sigma_2, [a_i, b_i])|^\# = [a_i', b_i'] \\
&\quad b_2' \leq a_1' ? [0,0] : [0,1])) \\
\underline{\{E_1, \cdots, E_n\}}^\# m^\# &:= \left(\sum_{i=1}^n w_i, u, \sum_{i=1}^n 2^{\sum_{0 \leq j < i} w_j} |(w_i, u, [a_i, b_i])|^\# \right) && +, \times \text{ are endpoint-wise}
\end{aligned}
$$

5.2 Semantic function for assignments

Consider the assignment $x[w_1 : w_2] := E_1$. The operation of clearing out consecutive bits from index w_1 to $w_1 + w_2 - 1$ of an integer a is equivalent to $\underbrace{a \bmod 2^{w_1}}_{\text{Lower } w_1 \text{ bits}} + \underbrace{2^{w_1+w_2} \lfloor 2^{-w_1-w_2} a \rfloor}_{\text{Upper bits}}$. Now, if we let $m^\#(w_x, \sigma_x, x) = [a_x, b_x]$, the "cleared out" version of x will be $[a_x', b_x'] = |(w_1, u, [a_x, b_x])|^\# + [2^{w_1+w_2} \lfloor 2^{-w_1-w_2} a_x \rfloor, 2^{w_1+w_2} \lfloor 2^{-w_1-w_2} b_x \rfloor]$, with the addition happening endpoint-wise, and the updated version will be

$$|(w_x, \sigma_x, [a_x' + a_1', b_x' + b_1'])|^\#$$

when $[a_1', b_1'] := |(w_2, u, [a_1, b_1])|^\#$.

5.3 Semantic function for states

The semantic function for a state $S = E \Rightarrow A_1 \cdots A_n$ is simply the composition of the semantic function of the assignments. That is, $\underline{S}^{\#} \coloneqq \underline{A_1}^{\#} \circ \cdots \circ \underline{A_n}^{\#}$.

5.4 Semantic function for commands

The semantic function for commands are simple. For an initial memory $m^{\#}$, collect the states that $m^{\#}$ matches with $\{S | m^{\#} \Rightarrow^{\#}_C S\}$, and for each state S, perform the assignments corresponding to that state, then join the memories after the assignments. That is,

$$\underline{C}^{\#} m^{\#} \coloneqq \bigsqcup_{m^{\#} \Rightarrow^{\#}_C S} \underline{S}^{\#} m^{\#}$$

6 Proofs

6.1 About the restriction function

It is trivial to show that if $n \in \gamma([a,b])$, then for any width w $|(w,u,n)| \in \gamma(|(w,u,[a,b])|^{\#})$ and $|(w,s,n)| \in \gamma(|w,s,[a,b]|^{\#})$. For intervals that go over the boundaries of width w, the abstract restriction function spits out the maximum interval possible, so it is trivial. For intervals that stays inside the boundary, the restriction function is an increasing function inside that interval, so the inequalities $a \le n$ and $n \le b$ are preserved after applying the restriction.

6.2 About expressions

We must prove that $m \in \gamma(m^{\#}) \Rightarrow \underline{E}m \in \gamma(\underline{E}^{\#} m^{\#})$, when γ preserves the width and signedness of the result and concretizes the interval part. That is, $\gamma(w,\sigma,[a,b]) = \{(w,\sigma,n) | n \in \gamma_I([a,b])\}$.

The only part that is not obvious is the bitwise and operation. For this, we need to prove some preliminary claims.

Claim 1. *For $n \in \mathbb{Z}$, if $n < 0$,*

$$n = \sum_{i \ge 0} 2^i \left(\left\lfloor \frac{n}{2^i} \right\rfloor - 2 \left\lfloor \frac{n}{2^{i+1}} \right\rfloor - 1 \right) - 1$$

and if $n \ge 0$,

$$n = \sum_{i \ge 0} 2^i \left(\left\lfloor \frac{n}{2^i} \right\rfloor - 2 \left\lfloor \frac{n}{2^{i+1}} \right\rfloor \right)$$

Proof. Note that the partial sum

$$\sum_{0 \le i < k} 2^i \left(\left\lfloor \frac{n}{2^i} \right\rfloor - 2 \left\lfloor \frac{n}{2^{i+1}} \right\rfloor \right) = 2^0 \left\lfloor \frac{n}{2^0} \right\rfloor - 2^k \left\lfloor \frac{n}{2^k} \right\rfloor = n - 2^k \left\lfloor \frac{n}{2^k} \right\rfloor$$

and

$$\sum_{0 \le i < k} 2^i \left(\left\lfloor \frac{n}{2^i} \right\rfloor - 2 \left\lfloor \frac{n}{2^{i+1}} \right\rfloor - 1 \right) = 2^0 \left\lfloor \frac{n}{2^0} \right\rfloor - 2^k \left\lfloor \frac{n}{2^k} \right\rfloor - 2^k + 1 = n - 2^k \left\lfloor \frac{n}{2^k} \right\rfloor - 2^k + 1$$

Note that if $n \ge 0$, there exists a N big enough that $k \ge N \Rightarrow \lfloor n/2^k \rfloor = 0$, and if $n < 0$, there exists a N big enough that $k \ge N \Rightarrow \lfloor n/2^k \rfloor = -1$. Therefore, for big enough k, the first partial sum for $n \ge 0$ is equal to n, and the second partial sum for $n < 0$ is equal to $n + 1$. □

This shows why the bitwise and operation is defined as such in section 4.1.

Now, we give upper and lower bounds for $x \& y$.

Claim 2. *For $a, b \in \mathbb{Z}$,*

$$\begin{cases} 0 \le a \& b \le \min\{a,b\} & (a,b \ge 0) \\ \max\{a+b+1,0\} \le a \& b \le a & (a \ge 0, b < 0) \\ \max\{a+b+1,0\} \le a \& b \le b & (a < 0, b \ge 0) \\ a+b+1 \le a \& b \le \min\{a,b\} & (a,b < 0) \end{cases}$$

Proof. First define the i-th bit for a and b.

$$a_i \coloneqq \left\lfloor \frac{a}{2^i} \right\rfloor - 2\left\lfloor \frac{a}{2^{i+1}} \right\rfloor \qquad b_i \coloneqq \left\lfloor \frac{b}{2^i} \right\rfloor - 2\left\lfloor \frac{b}{2^{i+1}} \right\rfloor$$

Since

$$\frac{a}{2^{i+1}} - 1 < \left\lfloor \frac{a}{2^{i+1}} \right\rfloor \leq \frac{a}{2^{i+1}} \Rightarrow \frac{a}{2^i} - 2 < 2\left\lfloor \frac{a}{2^{i+1}} \right\rfloor \leq \frac{a}{2^i} \Rightarrow 0 \leq \frac{a}{2^i} - 2\left\lfloor \frac{a}{2^{i+1}} \right\rfloor < 2$$

and since the equality can hold in

$$2\left\lfloor \frac{a}{2^{i+1}} \right\rfloor \leq \frac{a}{2^i}$$

only when $a/2^i$ is an integer, replacing $a/2^i$ with $\lfloor a/2^i \rfloor$ will still make the the inequality hold, thus $0 \leq a_i, b_i \leq 1$.

Now, we consider the inequality

$$0 \leq (1 - a_i)(1 - b_i) \Rightarrow a_i b_i \geq a_i + b_i - 1 \tag{1}$$

and

$$0 \leq a_i b_i \leq a_i \wedge 0 \leq a_i b_i \leq b_i \tag{2}$$

Since

$$a\&b = \sum_{i \geq 0} 2^i a_i b_i$$

when $a, b \geq 0$, the second inequality can be applied to get $0 \leq a\&b \leq \sum_{i\geq 0} 2^i a_i = a$ and $0 \leq a\&b \leq \sum_{i\geq 0} 2^i b_i = b$. Thus $0 \leq a\&b \leq \min\{a, b\}$.

When $a < 0$ and $b \geq 0$, the first inequality can be applied to get the lower bound $a\&b \geq \sum_{i\geq 0} 2^i(a_i + b_i - 1) = \sum_{i\geq 0} 2^i a_i + \sum_{i\geq 0} 2^i(b_i - 1) - 1 + 1 = a + b + 1$. Thus, we have the second claim and by symmetry, the third claim as well.

When $a, b < 0$, the expression for $a\&b$ is

$$a\&b = \sum_{i \geq 0} 2^i(a_i b_i - 1) - 1$$

Since $a_i b_i - 1 \geq a_i + b_i - 2 = (a_i - 1) + (b_i - 1)$ by the first inequality, we have the lower bound $a\&b \geq \sum_{i\geq 0} 2^i(a_i - 1 + b_i - 1) - 2 + 1 = a + b + 1$. Also, the upper bound can be obtained by noting $a_i b_i - 1 \leq a_i - 1$ and $a_i b_i - 1 \leq b_i - 1$, thus $a\&b \leq a$ and $a\&b \leq b$. Thus we have the fourth claim. □

Applying claim 2 between intervals $[a_1, b_1]$ and $[a_2, b_2]$ results in the bounds specified in section 5.1.

6.3 About commands

$m \in \gamma(m^\#) \Rightarrow \underline{A}m \in \gamma(\underline{A}^\# m^\#)$ is straightforward since (1) the "cleared out" version of n_x stays inside $[a'_x, b'_x]$, and by the soundness of the abstract expression function and the abstract restriction function (2) $|\underline{E}m|$ stays inside $[a'_1, b'_1]$. The only thing needed to show the soundness of the abstract command function is the soundness of the abstract matching relation $\Rightarrow^\#_C$. That is, we need to show that if $m \in \gamma(m^\#) \wedge m \Rightarrow_C S_i$, then $m^\# \Rightarrow^\#_C S_i$. This is straightforward from the soundness of the abstract expression function, since the predicate $\bigwedge_{j<i}[|\underline{E_j}m| = 0] \wedge [|\underline{E_i}m| \neq 0]$ will lead to $0 \in |\underline{E_j}^\# m^\#|^\#$ and $|\underline{E_i}^\# m^\#|^\# \not\sqsubseteq [0, 0]$.

7 Analysis

For each state S_i, we need to overapproximate $m_i^\# \coloneqq \alpha(\{m | m \Rightarrow_C S_i\})$. We note that if $m \Rightarrow_C S_i$, then $\alpha(\{m\}) \Rightarrow^\#_C S_i$. So we need to find a predicate that is a necessary condition for $m^\# \Rightarrow^\#_C S_i$, so that $m_i^\# \Rightarrow^\#_C S_i$ can be overapproximated. To do this, for each variable $x \in \text{PgmVar}$ make two variables a_x and b_x signifying $m^\#(w_x, \sigma_x, x) = [a_x, b_x]$. Then note that all the semantic functions defined in section 5 are linear inequalities and linear equations, except for $\underline{x[E : w]}^\# m^\#$. In this case, the abstract semantic function is further overapproximated by sending the result to the maximum range an unsigned number with width w can have, which is $[0, 2^w - 1]$. Then truly all semantic functions will have endpoints which are linear equations or linear inequalities(when max or min are used). Then the condition $m^\# \Rightarrow^\#_C S_i$ can be expressed as a system of linear inequalities in the variables a_x and b_x, which can be solved by a solver. The "least solution" to this system will be the least approximation to the state $m_i^\#$.

Then calculate $m_i^{\#\prime} \coloneqq \underline{S_i}^\# M^\#$. If $m_i^{\#\prime} \sqsubseteq m_i^\#$, then state S_i is possibly problematic. Also, if evaluating the predicate $\bigwedge_{j<i}[|\underline{E_j}^\# m_i'^\#|^\# = [0, 0]] \wedge [0 \notin |\underline{E_i}^\# m_i'^\#|^\#]$ evaluates to true, then the state S_i is *definitely* problematic.