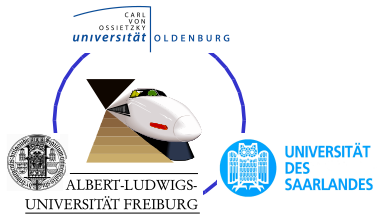


AVACS – Automatic Verification and Analysis of Complex Systems

REPORTS

of SFB/TR 14 AVACS

Editors: Board of SFB/TR 14 AVACS

Extending iSAT3 with ICP-Contractors for Bitwise Integer Operations

by

Karsten Scheibler Felix Neubauer Ahmed Mahdi
Martin Fränzle Tino Teige Tom Bienmüller Detlef Fehrer
Bernd Becker

AVACS Technical Report No. 116
August 2016
ISSN: 1860-9821

Publisher: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
Editors: Bernd Becker, Werner Damm, Bernd Finkbeiner, Martin Fränzle,
Ernst-Rüdiger Olderog, Andreas Podelski
ATRs (AVACS Technical Reports) are freely downloadable from `www.avacs.org`

Copyright © August 2016 by the author(s)
Author(s) contact: Karsten Scheibler (`scheibler@informatik.uni-freiburg.de`).

Extending iSAT3 with ICP-Contractors for Bitwise Integer Operations

Karsten Scheibler[(FR)], Felix Neubauer[(FR)], Ahmed Mahdi[(OL)], Martin Fränzle[(OL)], Tino Teige[(BTC)], Tom Bienmüller[(BTC)], Detlef Fehrer[(SICK)], Bernd Becker[(FR)]

[(FR)] Chair of Computer Architecture, University of Freiburg, Germany
[(OL)] Research Group Hybrid Systems, Carl von Ossietzky University of Oldenburg, Germany
[(BTC)] BTC Embedded Systems AG, Oldenburg, Germany
[(SICK)] Sick AG, Waldkirch, Germany

Abstract

Up to now SMT-solvers addressing floating-point arithmetic were based on bit-blasting. Quite recently, methods based on interval constraint propagation (ICP) for accurate reasoning over floating-point arithmetic were proposed. One prominent use-case for such methods is automatic dead-code detection in floating-point dominated embedded C programs. However, C programs usually contain a mix of floating-point arithmetic, integer arithmetic and bitwise integer operations. Thus, adding ICP-based support for floating-point arithmetic is not enough – bitwise integer operations have to be supported as well. Therefore, this report gives a detailed overview how these operations can be handled with ICP.

1. Introduction

The iSAT algorithm [3, 4] was originally developed with hybrid systems verification in mind. Therefore, it aims at solving complex-structured arithmetic constraint systems involving linear and non-linear arithmetic as well as transcendental functions. The three implementations of the iSAT algorithm (HySAT[1], iSAT[2] and iSAT3[3]) may differ in their supported features, but they all share the key idea of tightly integrating interval constraint propagation (ICP) into the confict-driven clause learning (CDCL) framework – yielding a unified lattice-based view in the meantime also known as abstract CDCL [1]. Similar to CDCL-based SAT-solvers which operate on a conjuctive normal form (CNF), the iSAT algorithm operates on a CNF as well – but enriched with so-called primitive constraints (PCs). The PCs are the result of a Tseitin-like transformation of the original theory atoms. This transformation ensures that each PC contains only one arithmetic operation – which helps to keep the ICP-contractors for each PC-type simple. Thus, ICP-contractors are responsible for performing the arithmetic reasoning. Adding support for new operations essentially requires adding further ICP-contractors. In this report we solely concentrate on such newly

[1] http://hysat.informatik.uni-oldenburg.de/
[2] http://projects.informatik.uni-freiburg.de/projects/isat/
[3] http://projects.informatik.uni-freiburg.de/projects/isat3/

added operations and therefore neglect the remaining details of the overall solving algorithm – the interested reader might refer to [3, 4] for a detailed overview.

Over the years we were surprised to see how powerful the iSAT algorithm also performs in other domains, e.g. systems biology [7], analysis of analogue circuits [11], on-the-fly safety analysis in robotics [5], or automatic test pattern generation for detecting interconnect open defects [2, 8]. Recently, we adapted iSAT3 [9, 8] in order to support accurate reasoning for floating-point arithmetic [10] – allowing iSAT3 to be used as a backend tool for automatic dead-code detection in floating-point dominated embedded C programs. However, C programs usually contain a mix of floating-point arithmetic, integer arithmetic and bitwise integer operations. Thus, adding support for floating-point arithmetic is not enough – we have to include support for bitwise integer operations as well. Therefore, this report gives a detailed overview of the ICP-contractors for bitwise integer operations and thus complements [10] where these contractors were proposed.

This paper is structured as follows: after introducing the desired features of an ICP-contractor in Section 2, we present the newly added contractors in Section 3, before concluding the report in Section 4.

2. Desired Features of ICP-Contractors inside the iSAT Algorithm

As already mentioned, we focus on ICP-contractors for primitive constraints (PCs). A PC assigns the result of an operation to a helper variable (denoted h) which was introduced during the Tseitin-like transformation. Examples for PCs are: $(h = -x)$, $(h = x^2)$, $(h = x + y)$, $(h = max(x,y))$ or $(h = x \cdot y)$. Of course an auxiliary variables can occur itself as an argument in a different PC. But we neglect this fact here, because in this report we treat each PC individually and independent of other PCs. This allows us to keep the notation simple: h will always represent the result of the operation contained in the PC, while the variables x and y denote the arguments. As all operations are performed on intervals, let $I(v)$ denote the interval valuation of a variable and let v_{lb} and v_{ub} represent the lower and upper bound of $I(v)$, e.g. for $I(h) = [0, 10]$ the lower bound h_{lb} is 0 and the upper bound h_{ub} is 10. Furthmore, $I()$ is also used to denote the resulting interval of an operation, e.g. $I(x+y)$. Please keep in mind that for a PC like $(h = x + y)$ the interval valuations $I(h)$ and $I(x+y)$ are not necessarily equal – in fact they could be even disjoint. This is due to possible interval splits (which are part of the superordinate search process) and the dependencies between the PCs (which are neglected here).

In the iSAT algorithm an ICP-contractor has to accomplish three tasks. It has to:

1. check the consistency of a PC (i.e. check whether the PC is still satisfiable under the current interval valuations of its variables),

2. prune the intervals of the variables contained in the PC by cutting off definitive non-solutions (i.e. deduce stronger lower and upper bounds for a variable), and

3. determine which reasons (i.e. which bounds of the other variables) are responsible for the deduction of a stronger lower or upper bound

The first point is the most important one – otherwise the solver would be unable to operate. Although being optional, the last two points play an important role for the efficiency of the solver.

Without interval pruning the performance of the solver would be very poor. The following example illustrates all three tasks and motivates why the consistency check can be moved outside of the ICP-contractor.

Example 1: The PC $(h = x + y)$ with $I(h) = [0, 10]$, $I(x) = [1, 10]$ and $I(y) = [2, 10]$ is consistent, because there exists at least one assignment satisfying it, e.g. $h = 5$, $x = 1$ and $y = 4$. The following calculations are performed in order to check whether stronger bounds could be deduced. Hereby, the variables h', x' and y' are used to hold the intermediate results. After all calculations are performed, these results are used to update the intervals of h, x and y.

$$\begin{array}{lllllllllll} h'_{lb} & = & x_{lb} + y_{lb} & = & 1+2 & = & 3 & h'_{ub} & = & x_{ub} + y_{ub} & = & 10+10 & = & 20 \\ x'_{lb} & = & h_{lb} - y_{ub} & = & 0-10 & = & -10 & x'_{ub} & = & h_{ub} - y_{lb} & = & 10-2 & = & 8 \\ y'_{lb} & = & h_{lb} - x_{ub} & = & 0-10 & = & -10 & y'_{ub} & = & h_{ub} - x_{lb} & = & 10-1 & = & 9 \end{array}$$

While the interval bounds for h' can be directly calculated with the operation contained in the PC (also called forward deduction), a redirect is needed for x' and y' (also called backward or inverse deduction).

Three stronger bounds were deduced in this example: h_{lb} is improved from 0 to 3 while x_{ub} and y_{ub} are now 8 and 9 instead of 10. As mentioned above, when a stronger bound is deduced, the reasons for this deduction are needed as well. The reasons for h_{lb} are x_{lb} and y_{lb}, because these two values are used to calculate the improved lower bound for h. Similarly, the reasons for x_{ub} are h_{ub} and y_{lb} (and for y_{ub} they are h_{ub} and x_{lb}). The deductions could be therefore also written as:

$$\begin{array}{lll} ((x_{lb} \geq 1) \wedge (y_{lb} \geq 2)) & \Rightarrow & (h_{lb} \geq 3) \\ ((h_{ub} \leq 10) \wedge (y_{lb} \geq 2)) & \Rightarrow & (x_{ub} \leq 8) \\ ((h_{ub} \leq 10) \wedge (x_{lb} \geq 1)) & \Rightarrow & (y_{ub} \leq 9) \end{array}$$

Each lower (upper) bound calculation is performed with the following question in mind: what is the smallest (largest) possible value regarding the performed operation when all value combinations from both argument intervals are considered. In fact the notation above indicates that this question could be rephrased even more generally. Instead of only considering values within the current intervals, it is also possible to consider all values above (below) the current lower (upper) bound – making the deduction independent of the current upper (lower) bound of the interval.

Obviously, for addition it suffices to operate on the interval bounds of the arguments in order to calculate the enclosing interval of the result, because enlarging a summand also enlarges the sum $(\forall y \forall \hat{x} > x : \hat{x} + y > x + y)$. Thus, h'_{lb} (h'_{ub}) is equal to the sum of x_{lb} and y_{lb} (x_{ub} and y_{ub}). Similar arguments apply for subtraction.

But this does not mean that all operations can be handled this way. E.g. for the bitwise AND between x and y – for the moment denoted as $(h = x \,\&\, y)$ – with $I(x) = [1, 7]$ and $I(y) = [1, 8]$, the lower bound of $I(x \,\&\, y)$ would be 1 when solely relying on x_{lb} and y_{lb}. But in fact it is 0, e.g. with $x = 1$ and $y = 2$. Similarly, if only x_{ub} and y_{ub} are considered, the upper bound of $I(x \,\&\, y)$ would be 0 – but in fact it is 7 (with $x = 7$ and $y = 7$).

Although consistency checking, interval pruning and reason determination are listed as distinct tasks, they are closely related. During each bound calculation the involved argument bounds are known. Therefore, determining the reasons is intertwined with the bound calculation. Similarly, the consistency check (which determines whether $I(h) \cap I(x+y) \neq \emptyset$ holds or not) relies on the calculation of h'_{lb} and h'_{ub} in order to compare them against h_{lb} and h_{ub}. Thus, the forward deduction is already part of the this check. In fact the consistency check can be moved outside of the ICP-contractor. Hence, the ICP-contractor is only responsible for deducing stronger bounds and determining the reasons – the comparison of the current bounds with the deduced bounds of a variable is performed separately.

Please note that it is not required to always deduce the tightest possible bounds, e.g. $((x_{lb} \geq 1) \wedge (y_{lb} \geq 2)) \Rightarrow (h_{lb} \geq -100)$ instead of $((x_{lb} \geq 1) \wedge (y_{lb} \geq 2)) \Rightarrow (h_{lb} \geq 3)$ would be perfectly valid as well. Further note, that backward deductions could be omitted – as the consistency check only depends on the forward deduction. But of course the solver will profit from backward deductions and tighter bounds in general.

The next example illustrates the ICP-contractor for the unary minus operation. This is in preparation for the bitwise NOT operation which has similarities to the unary minus and will be presented in the next section.

Example 2: The PC $(h = -x)$ with $I(h) = [1,10]$ and $I(x) = [-9,2]$ is consistent. The following calculations are performed in order to check whether stronger bounds could be deduced:

$$\begin{array}{lllll} h'_{lb} & = & -x_{ub} & = & -2 \\ x'_{lb} & = & -h_{ub} & = & -10 \end{array} \qquad \begin{array}{lllll} h'_{ub} & = & -x_{lb} & = & 9 \\ x'_{ub} & = & -h_{lb} & = & -1 \end{array}$$

In contrast to the PC containing addition, the PC in this example uses the unary minus operation also in the redirected case in order to calculate bounds for x'. Two stronger bounds can be deduced:

$$\begin{array}{lcl} (x_{lb} \geq -9) & \Rightarrow & (h_{ub} \leq 9) \\ (h_{lb} \geq 1) & \Rightarrow & (x_{ub} \leq -1) \end{array}$$

3. ICP-Contractors for Bitwise Integer Operations

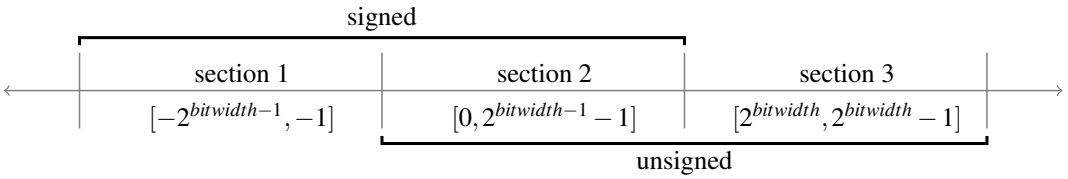
The iSAT algorithm operates on interval valuations and uses ICP to reason about arithmetic constraints. In C programs an integer with n bits can be interpreted either as signed or unsigned. Therefore, the same bit-pattern can represent two different numbers depending on its signedness interpretation. The example in the following table illustrates this with two bit-patterns for the 8 bit data type `char`.

	00010001	10000001
`signed char`	17	-127
`unsigned char`	17	129

Thus, it is important to know the bitwidth and whether the result should be interpreted as signed or unsigned integer. Hence, each ICP-contractor for a bitwise operation expects an additional constant parameter specifying the bitwidth. Furthermore, each operation exists in a signed and an unsigned variant:

- s_bw_not($arg, bitwidth$), u_bw_not($arg, bitwidth$)
  signed, unsigned bitwise NOT
- s_bw_and($arg1, arg2, bitwidth$), u_bw_and($arg1, arg2, bitwidth$)
  signed, unsigned bitwise AND
- s_bw_or($arg1, arg2, bitwidth$), u_bw_or($arg1, arg2, bitwidth$)
  signed, unsigned bitwise OR
- s_bw_xor($arg1, arg2, bitwidth$), u_bw_xor($arg1, arg2, bitwidth$)
  signed, unsigned bitwise XOR
- s_cast($arg, bitwidth$), u_cast($arg, bitwidth$)
  signed, unsigned cast between different bitwidths

While the cast operation allows arguments with arbitrary bitwidths (which are then casted to the specified bitwidth), the arguments of the bitwise NOT, AND, OR and XOR operations are expected to be either in the range of a signed or an unsigned integer regarding the constant bitwidth parameter (for arguments with larger intervals a cast has to be applied first). Therefore, for these operations additional case distictions are performed regarding the interval bounds of their arguments.



In order to allow the deduction of tighter bounds, these case distinctions are based on sections and thus are more fine-grained than dividing the arguments solely into signed and unsigned. Up to five cases are distinguished for each argument[4]:

1) the argument interval lies in section 1

1+2) the argument interval lies in sections 1 and 2

2) the argument interval lies in section 2

2+3) the argument interval lies in sections 2 and 3

3) the argument interval lies in section 3

Furthermore, we will make use of the section bounds and the interval width induced by the bitwidth:

	$width$	$=$	$2^{bitwidth}$	$half\_width$	$=$	$width/2$
section 1:	$s1_{lb}$	$=$	$-half\_width$	$s1_{ub}$	$=$	-1
section 2:	$s2_{lb}$	$=$	0	$s2_{ub}$	$=$	$half\_width-1$
section 3:	$s3_{lb}$	$=$	$half\_width$	$s3_{ub}$	$=$	$width-1$

[4] As a bit-pattern of a given bitwidth can be interpreted either signed or unsigned, it is not possible that an argument interval can cover all three sections.

3.1. Bitwise NOT

If the signedness of the argument and the result are the same, the bitwise NOT has similarities to the unary minus. While the bounds are "mirrored at 0" when applying the unary minus, the bounds are "mirrored at -1" for the signed bitwise NOT. Therefore, the following bound calculations are performed in order to determine whether stronger bounds can be deduced:

$$\begin{array}{rcl} h'_{lb} & = & s1_{ub} - x_{ub} \\ x'_{lb} & = & s1_{ub} - h_{ub} \end{array} \qquad \begin{array}{rcl} h'_{ub} & = & s1_{ub} - x_{lb} \\ x'_{ub} & = & s1_{ub} - h_{lb} \end{array}$$

For the unsigned bitwise NOT with unsigned argument these calculations are relevant:

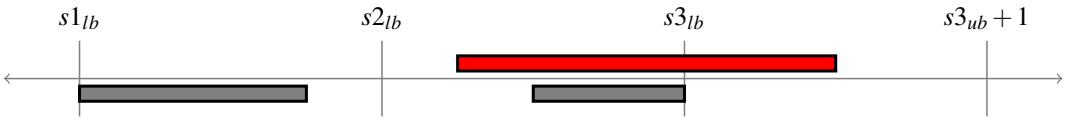
$$\begin{array}{rcl} h'_{lb} & = & s3_{ub} - x_{ub} \\ x'_{lb} & = & s3_{ub} - h_{ub} \end{array} \qquad \begin{array}{rcl} h'_{ub} & = & s3_{ub} - x_{lb} \\ x'_{ub} & = & s3_{ub} - h_{lb} \end{array}$$

As for the unary minus, forward and backward deductions are similar. This means: swapping h and x in the forward deduction yields the calculations for the backward deduction. If the argument and the result differ in their signedness, a case distinction is made. If the argument interval lies in only one section then similar calculations as above can be applied. The following pictures illustrate this. The red area denotes an example range of $I(x)$ while the blue area denotes the resulting range of $I(h')$.

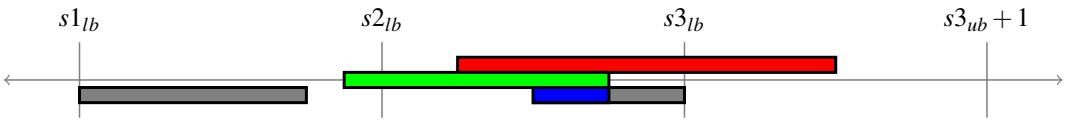
If the result is signed and $I(x)$ lies in section 3 then $I(h')$ will lie in section 2:

If the result is unsigned and $I(x)$ lies in section 1 then $I(h')$ will lie in section 2:

In the remaining two cases when the argument interval lies in two sections, the result would consist of two disjoint intervals. For example if the result is signed, $bitwidth = 8$ and $I(x) = [32, 191]$. The argument sub-interval $[32, 127]$ would be mapped to $[-128, -33]$ while the sub-interval $[128, 191]$ would be mapped to $[64, 127]$ (both in grey color):



For these two resulting sub-intervals the enclosing interval bounds have to be calculated – which are equal to the trivial bounds for all signed results with $bitwidth = 8$, namely $[-128, 127]$. In order to deduce stronger bounds, the current result bounds are additionally considered in these cases. For example if in the above example $I(h) = [-16, 96]$ (in green color) then it could be deduced that $I(h') = [64, 96]$:



Thus, h_{lb} and h_{ub} are also relevant as reasons in these cases. The following table gives an overview of all case distinctions in the forward deduction. The backward deduction is similar, only the result and the argument have to be swapped then the same case distinctions apply. Please note that no reasons are needed (denoted with: none) in cases where the trivial bounds are deduced (for signed results $[s1_{lb}, s2_{ub}]$ and $[s2_{lb}, s3_{ub}]$ for unsigned results) – as these bounds hold in all cases regardless of the argument bounds. Further note that for arguments solely lying in section 2 sometimes an additional reason is needed. For example regarding s_bw_not when comparing the calculation for h'_{ub} in sections 1+2 and 2. In the case of 1+2 x_{lb} is guaranteed to lie in section 1. Therefore, x_{ub} can never be in section 3. But the picture changes in the case of 2. Here, x_{lb} lies in section 2. Thus, x_{ub} could in principle lie in section 3 – but in this case the deduction would be invalid (as the bound calculation for 2+3 is different). Hence, x_{ub} has to be added as reason in this case.

bitwise NOT forward deduction			
operation	x sections	bound calculation	reasons
s_bw_not	1	see 1+2	
	1+2	$h'_{lb} = s1_{ub} - x_{ub}$ $h'_{ub} = s1_{ub} - x_{lb}$	x_{ub} x_{lb}
	2	$h'_{lb} = s1_{ub} - x_{ub}$ $h'_{ub} = s1_{ub} - x_{lb}$	x_{ub} x_{lb}, x_{ub}
	2+3	$h''_{lb} = s3_{ub} - x_{ub}$ $h''_{ub} = s1_{ub} - x_{lb}$ $h'_{lb} = h''_{lb}$ if $(h''_{ub} < h_{lb})$ $h'_{lb} = s1_{lb}$ if $(h''_{ub} \geq h_{lb})$ $h'_{ub} = h''_{ub}$ if $(h''_{lb} > h_{ub})$ $h'_{ub} = s2_{ub}$ if $(h''_{lb} \leq h_{ub})$	 x_{lb}, x_{ub}, h_{lb} none x_{lb}, x_{ub}, h_{ub} none
	3	$h'_{lb} = s3_{ub} - x_{ub}$ $h'_{ub} = s3_{ub} - x_{lb}$	x_{lb}, x_{ub} x_{lb}
u_bw_not	1	$h'_{lb} = s1_{ub} - x_{ub}$ $h'_{ub} = s1_{ub} - x_{lb}$	x_{ub} x_{lb}, x_{ub}
	1+2	$h''_{lb} = s3_{ub} - x_{ub}$ $h''_{ub} = s1_{ub} - x_{lb}$ $h'_{lb} = h''_{lb}$ if $(h''_{ub} < h_{lb})$ $h'_{lb} = s2_{lb}$ if $(h''_{ub} \geq h_{lb})$ $h'_{ub} = h''_{ub}$ if $(h''_{lb} > h_{ub})$ $h'_{ub} = s3_{ub}$ if $(h''_{lb} \leq h_{ub})$	 x_{lb}, x_{ub}, h_{lb} none x_{lb}, x_{ub}, h_{ub} none
	2	$h'_{lb} = s3_{ub} - x_{ub}$ $h'_{ub} = s3_{ub} - x_{lb}$	x_{lb}, x_{ub} x_{lb}
	2+3	$h'_{lb} = s3_{ub} - x_{ub}$ $h'_{ub} = s3_{ub} - x_{lb}$	x_{ub} x_{lb}
	3	see 2+3	

3.2. Bitwise AND, OR and XOR

For bitwise AND, OR and XOR we only implemented the forward deduction. Furthermore, in order to keep the bound calculations fast, a safe overapproximation of $I(h')$ is calculated by using basic operations like addition, subtraction, minimum and maximum. Hereby, we exploit that every interval that lies in sections 1+2 contains the numbers $s1_{ub} = -1$ and $s2_{lb} = 0$. While the bit-pattern of $s1_{ub}$ solely consists of one-bits, the bit-pattern of $s2_{lb}$ contains only zero-bits. Similarly, every interval that lies in sections 2+3 contains the numbers $s2_{ub}$ and $s3_{lb}$ whose bit-patterns are similar to $s1_{ub}$ and $s2_{lb}$ (only the first bit is flipped compared to $s1_{ub}$ and $s2_{lb}$).

The following tables list all distinguished cases together with their bound calculations. For each case there are trivial bounds which always hold (and need no reasons), e.g. for a signed h all values will reside in $[s1_{lb}, s2_{ub}]$. These trivial bounds are only listed whenever no stronger bound can be calculated. Furthermore, some bounds are listed with more than one calculation. It is assumed that always the strongest bound is taken. Please note that in certain situations a trivial bound could be stronger than a calculated bound (e.g. when adding two bounds, the result may be outside the trivial bound). In such situations the trivial bound is taken – as it is the stronger one.

Additionally, certain cases make use of a bound calculation which analyzes the leading bits of the interval bounds of both arguments. For each argument it is checked whether the bounds have a common bit-prefix. This allows to apply the bitwise operation to the prefixes of both arguments. Hereby, the remaining trailing bits are choosen in such a way that the lower or upper bound is obtained. The next example illustrates this for a signed bitwise AND: assume $bitwidth = 8$, $I(x) = [18, 30]$, $I(y) = [89, 92]$ and let "&" denote the bitwise AND operation.

$$
\begin{array}{lcll}
x_{lb} & = & 18 = 0001\,0010 & \\
x_{ub} & = & 30 = 0001\,1110 & \\
 & & 0001 & \text{common bit-prefix for values in } I(x) \\
 & & & \\
y_{lb} & = & 89 = 0101\,1001 & \\
y_{ub} & = & 92 = 0101\,1100 & \\
 & & 0101\,1 & \text{common bit-prefix for values in } I(y) \\
 & & & \\
h'_{lb} & = & 0001\,0000 \;\&\; 0101\,1000 = 0001\,0000 = 16 & \text{trailing bits are 0} \\
h'_{ub} & = & 0001\,1111 \;\&\; 0101\,1111 = 0001\,1111 = 31 & \text{trailing bits are 1}
\end{array}
$$

While for bitwise AND and bitwise OR it suffices to set all trailing bits to zero for the calculation of a lower bound, this will not work in general for bitwise XOR. As the bit-prefixes of x and y might have different lengths, the trailing bits of the shorter prefix have to be chosen in such a way that the last bits of the longer prefix are canceled out in order to obtain the lower bound (similar arguments apply for the calculation of the upper bound). This can be realized with additional AND- and OR-bit-masks which are applied during the calculation.

Besides the possibility to return stronger bounds compared to the other calculations, the bit-prefix approach also ensures that $I(h')$ will be a point interval when both argument intervals $I(x)$ and $I(y)$ are points themselves. In the tables the bound calculations based on bit-prefixes are denoted as:

- bitwise AND: s_bw_and_lb_prefix(), s_bw_and_ub_prefix(), u_bw_and_lb_prefix(), u_bw_and_ub_prefix()

- bitwise OR: s_bw_or_lb_prefix(), s_bw_or_ub_prefix(), u_bw_or_lb_prefix(), u_bw_or_ub_prefix()
- bitwise XOR: s_bw_xor_lb_prefix(), s_bw_xor_ub_prefix(), u_bw_xor_lb_prefix(), u_bw_xor_ub_prefix()

While the prefix calculation is straightforward, the bound calculations relying on addition, subtraction, minimum and maximum may look counterintuitive at first glance. Therefore, we explain some examples more detailed for the signed bitwise AND in order to clearify the underlying idea:

1) When $I(x)$ and $I(y)$ both lie in section 1 then $h'_{ub} = \min(x_{ub}, y_{ub})$:
Obviously, in order to get a bit at position i in h' to zero, it suffices that a bit at the same position in x or y is zero. Therefore, even with $x_{ub} = s1_{ub}$ it follows that $h'_{ub} = y_{ub}$. Thus, there can be no h'_{ub} which is larger than x_{ub} or y_{ub}.

2) When $I(x)$ lies in section 1 and $I(y)$ lies in sections 1+2 then $h'_{ub} = \min(x_{ub} + half\_width, y_{ub})$:
Obviously, when comparing two numbers, every number in section 2 is larger than every number in section 1. Therefore, if one argument contains values in section 2, the result of the signed bitwise AND will be in section 2 as well. Thus, with $I(y)$ lying in sections 1+2, h'_{ub} has to be in section 2. The same reasoning as in 1) applies here too – but we already know that the most significant bit is zero. Therefore, x_{ub} is shifted from section 1 to section 2 by adding $half\_width$.

3) When $I(x)$ lies in section 1 and $I(y)$ lies in sections 2+3 then $h'_{ub} = x_{ub} + half\_width$:
All numbers in section 3 will be interpreted as negative numbers by the signed bitwise AND. Therefore, with a similar reasoning as in 2), h'_{ub} has to be in section 2. Because $I(y)$ contains $s2_{ub}$, a minimum operation is not necessary and $h'_{ub} = x_{ub} + half\_width$.

4) When $I(x)$ and $I(y)$ both lie in sections 1+2 then $h'_{ub} = \max(x_{ub}, y_{ub})$:
$I(x)$ and $I(y)$ contain $s1_{ub}$. Therefore, the larger of the upper bounds x_{ub} and y_{ub} can be combined with $s1_{ub}$ as the other argument. Thus, $h'_{ub} = \max(x_{ub}, y_{ub})$.

bitwise AND forward deduction			
operation	*x*/*y* sections	bound calculation	reasons
s_bw_and	1/1	$h'_{lb} = \text{s\_bw\_and\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \min(x_{ub}, y_{ub})$	x_{ub}, y_{ub}
		$h'_{ub} = \text{s\_bw\_and\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1/1+2	$h'_{lb} = \text{s\_bw\_and\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, s1_{ub})$	x_{lb}, x_{ub}, y_{lb}
		$h'_{ub} = \min(x_{ub} + half\_width, y_{ub})$	x_{ub}, y_{ub}
		$h'_{ub} = \text{s\_bw\_and\_ub\_prefix}(x_{lb}, x_{ub}, s2_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{ub}
	1/2	$h'_{lb} = \text{s\_bw\_and\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \min(x_{ub} + half\_width, y_{ub})$	x_{ub}, y_{ub}
		$h'_{ub} = \text{s\_bw\_and\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1/2+3	$h'_{lb} = s1_{lb}$	none
		$h'_{ub} = x_{ub} + half\_width$	x_{ub}, y_{ub}
	1/3	$h'_{lb} = \text{s\_bw\_and\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \min(x_{ub}, y_{ub} - width)$	x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{s\_bw\_and\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}

bitwise AND forward deduction			
operation	*x*/*y* sections	bound calculation	reasons
s_bw_and	1+2/1	swap x and y and use 1/1+2	
	1+2/1+2	$h'_{lb} = \text{s\_bw\_and\_lb\_prefix}(x_{lb}, s1_{ub}, y_{lb}, s1_{ub})$ $h'_{ub} = \max(x_{ub}, y_{ub})$	x_{lb}, y_{lb} x_{ub}, y_{ub}
	1+2/2	$h'_{lb} = s2_{lb}$ $h'_{ub} = y_{ub}$	y_{lb}, y_{ub} y_{lb}, y_{ub}
	1+2/2+3	$h'_{lb} = s1_{lb}$ $h'_{ub} = s2_{ub}$	none none
	1+2/3	$h'_{lb} = \text{s\_bw\_and\_lb\_prefix}(x_{lb}, s1_{ub}, y_{lb}, y_{ub})$ $h'_{ub} = \min(x_{ub}, y_{ub} - half\_width)$ $h'_{ub} = \text{s\_bw\_and\_ub\_prefix}(s2_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, y_{lb}, y_{ub} x_{ub}, y_{lb}, y_{ub} x_{ub}, y_{lb}, y_{ub}
	2/1	swap x and y and use 1/2	
	2/1+2	swap x and y and use 1+2/2	
	2/2	$h'_{lb} = \text{s\_bw\_and\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$ $h'_{ub} = \min(x_{ub}, y_{ub})$ $h'_{ub} = \text{s\_bw\_and\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub}
	2/2+3	$h'_{lb} = s2_{lb}$ $h'_{ub} = x_{ub}$	x_{lb}, x_{ub} x_{lb}, x_{ub}
	2/3	$h'_{lb} = \text{s\_bw\_and\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$ $h'_{ub} = \min(x_{ub}, y_{ub} - half\_width)$ $h'_{ub} = \text{s\_bw\_and\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{ub}, y_{lb}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub}
	2+3/1	swap x and y and use 1/2+3	
	2+3/1+2	swap x and y and use 1+2/2+3	
	2+3/2	swap x and y and use 2/2+3	
	2+3/2+3	$h'_{lb} = s1_{lb}$ $h'_{ub} = s2_{ub}$	none none
	2+3/3	$h'_{lb} = s1_{lb}$ $h'_{ub} = y_{ub} - half\_width$	none y_{lb}, y_{ub}
	3/1	swap x and y and use 1/3	
	3/1+2	swap x and y and use 1+2/3	
	3/2	swap x and y and use 2/3	
	3/2+3	swap x and y and use 2+3/3	
	3/3	$h'_{lb} = \text{s\_bw\_and\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$ $h'_{ub} = \min(x_{ub} - width, y_{ub} - width)$ $h'_{ub} = \text{s\_bw\_and\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub}
u_bw_and	1/1	$h'_{lb} = \text{u\_bw\_and\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$ $h'_{ub} = \min(x_{ub} + width, y_{ub} + width)$ $h'_{ub} = \text{u\_bw\_and\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{ub}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1/1+2	$h'_{lb} = s2_{lb}$ $h'_{ub} = x_{ub} + width$	none x_{ub}
	1/2	$h'_{lb} = s2_{lb}$ $h'_{ub} = \min(x_{ub} + half\_width, y_{ub})$	none x_{ub}, y_{lb}, y_{ub}

bitwise AND forward deduction			
operation	*x*/*y* sections	bound calculation	reasons
u_bw_and	1/2+3	$h'_{lb} = \text{u\_bw\_and\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, s2_{ub})$	x_{lb}, x_{ub}, y_{lb}
		$h'_{ub} = \min(x_{ub} + width, y_{ub})$	x_{ub}, y_{ub}
		$h'_{ub} = \text{u\_bw\_and\_ub\_prefix}(x_{lb}, x_{ub}, s3_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{ub}
	1/3	$h'_{lb} = \text{u\_bw\_and\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \min(x_{ub} + width, y_{ub})$	x_{ub}, y_{ub}
		$h'_{ub} = \text{u\_bw\_and\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1+2/1	swap x and y and use 1/1+2	
	1+2/1+2	$h'_{lb} = s2_{lb}$	none
		$h'_{ub} = s3_{ub}$	none
	1+2/2	$h'_{lb} = s2_{lb}$	none
		$h'_{ub} = y_{ub}$	y_{lb}, y_{ub}
	1+2/2+3	$h'_{lb} = s2_{lb}$	none
		$h'_{ub} = y_{ub}$	y_{ub}
	1+2/3	$h'_{lb} = s2_{lb}$	none
		$h'_{ub} = y_{ub}$	y_{ub}
	2/1	swap x and y and use 1/2	
	2/1+2	swap x and y and use 1+2/2	
	2/2	$h'_{lb} = \text{u\_bw\_and\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \min(x_{ub}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{u\_bw\_and\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	2/2+3	$h'_{lb} = s2_{lb}$	none
		$h'_{ub} = x_{ub}$	x_{lb}, x_{ub}
	2/3	$h'_{lb} = \text{u\_bw\_and\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \min(x_{ub}, y_{ub} - half\_width)$	x_{lb}, x_{ub}, y_{ub}
		$h'_{ub} = \text{u\_bw\_and\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	2+3/1	swap x and y and use 1/2+3	
	2+3/1+2	swap x and y and use 1+2/2+3	
	2+3/2	swap x and y and use 2/2+3	
	2+3/2+3	$h'_{lb} = s2_{lb}$	none
		$h'_{ub} = \min(x_{ub}, y_{ub})$	x_{ub}, y_{ub}
		$h'_{ub} = \text{u\_bw\_and\_ub\_prefix}(s3_{lb}, x_{ub}, s3_{lb}, y_{ub})$	x_{ub}, y_{ub}
	2+3/3	$h'_{lb} = \text{u\_bw\_and\_lb\_prefix}(x_{lb}, s2_{ub}, y_{lb}, y_{ub})$	x_{lb}, y_{lb}, y_{ub}
		$h'_{ub} = \min(x_{ub}, y_{ub})$	x_{ub}, y_{ub}
		$h'_{ub} = \text{u\_bw\_and\_ub\_prefix}(s3_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{ub}, y_{lb}, y_{ub}
	3/1	swap x and y and use 1/3	
	3/1+2	swap x and y and use 1+2/3	
	3/2	swap x and y and use 2/3	
	3/2+3	swap x and y and use 2+3/3	
	3/3	$h'_{lb} = \text{u\_bw\_and\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \min(x_{ub}, y_{ub})$	x_{ub}, y_{ub}
		$h'_{ub} = \text{u\_bw\_and\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}

bitwise OR forward deduction			
operation	*x*/*y* sections	bound calculation	reasons
s_bw_or	1/1	$h'_{lb} = \max(x_{lb}, y_{lb})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{lb} = \text{s\_bw\_or\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + y_{ub} + half\_width$	x_{ub}, y_{ub}
		$h'_{ub} = \text{s\_bw\_or\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1/1+2	$h'_{lb} = x_{lb}$	x_{lb}, x_{ub}
		$h'_{ub} = s1_{ub}$	x_{ub}
	1/2	$h'_{lb} = \max(x_{lb}, y_{lb} - half\_width)$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{lb} = \text{s\_bw\_or\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + y_{ub}$	x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{s\_bw\_or\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1/2+3	$h'_{lb} = x_{lb}$	x_{lb}, x_{ub}
		$h'_{ub} = s1_{ub}$	x_{ub}
	1/3	$h'_{lb} = \max(x_{lb}, y_{lb} - width)$	x_{lb}, x_{ub}, y_{lb}
		$h'_{lb} = \text{s\_bw\_or\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + y_{ub} - half\_width$	x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{s\_bw\_or\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1+2/1	swap x and y and use 1/1+2	
	1+2/1+2	$h'_{lb} = \min(x_{lb}, y_{lb})$	x_{lb}, y_{lb}
		$h'_{ub} = x_{ub} + y_{ub}$	x_{ub}, y_{ub}
		$h'_{ub} = \text{s\_bw\_or\_ub\_prefix}(s2_{lb}, x_{ub}, s2_{lb}, y_{ub})$	x_{ub}, y_{ub}
	1+2/2	$h'_{lb} = \max(x_{lb}, y_{lb} - half\_width)$	x_{lb}, y_{lb}, y_{ub}
		$h'_{lb} = \text{s\_bw\_or\_lb\_prefix}(x_{lb}, s1_{ub}, y_{lb}, y_{ub})$	x_{lb}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + y_{ub}$	x_{ub}, y_{ub}
		$h'_{ub} = \text{s\_bw\_or\_ub\_prefix}(s2_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{ub}, y_{lb}, y_{ub}
	1+2/2+3	$h'_{lb} = s1_{lb}$	none
		$h'_{ub} = s2_{ub}$	none
	1+2/3	$h'_{lb} = y_{lb} - width$	y_{lb}
		$h'_{ub} = s1_{ub}$	y_{lb}
	2/1	swap x and y and use 1/2	
	2/1+2	swap x and y and use 1+2/2	
	2/2	$h'_{lb} = \max(x_{lb}, y_{lb})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{lb} = \text{s\_bw\_or\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + y_{ub}$	x_{ub}, y_{ub}
		$h'_{ub} = \text{s\_bw\_or\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	2/2+3	$h'_{lb} = x_{lb} - half\_width$	x_{lb}, x_{ub}
		$h'_{ub} = s2_{ub}$	none
	2/3	$h'_{lb} = \max(x_{lb} - half\_width, y_{lb} - width)$	x_{lb}, x_{ub}, y_{lb}
		$h'_{lb} = \text{s\_bw\_or\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + y_{ub} - width$	x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{s\_bw\_or\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	2+3/1	swap x and y and use 1/2+3	

bitwise OR forward deduction			
operation	*x*/*y* sections	bound calculation	reasons
s_bw_or	2+3/1+2	swap x and y and use 1+2/2+3	
	2+3/2	swap x and y and use 2/2+3	
	2+3/2+3	$h'_{lb} = s1_{lb}$	none
		$h'_{ub} = s2_{ub}$	none
	2+3/3	$h'_{lb} = y_{lb} - width$	y_{lb}
		$h'_{ub} = s1_{ub}$	y_{lb}
	3/1	swap x and y and use 1/3	
	3/1+2	swap x and y and use 1+2/3	
	3/2	swap x and y and use 2/3	
	3/2+3	swap x and y and use 2+3/3	
	3/3	$h'_{lb} = \max(x_{lb} - width, y_{lb} - width)$	x_{lb}, y_{lb}
		$h'_{lb} = \text{s\_bw\_or\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} - half\_width + y_{ub} - width$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{s\_bw\_or\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
u_bw_or	1/1	$h'_{lb} = \max(x_{lb} + width, y_{lb} + width)$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{lb} = \text{u\_bw\_or\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + half\_width + y_{ub} + width$	x_{ub}, y_{ub}
		$h'_{ub} = \text{u\_bw\_or\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1/1+2	$h'_{lb} = x_{lb} + width$	x_{lb}, x_{ub}
		$h'_{ub} = s3_{ub}$	none
	1/2	$h'_{lb} = \max(x_{lb} + width, y_{lb} + half\_width)$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{lb} = \text{u\_bw\_or\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + width + y_{ub}$	x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{u\_bw\_or\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1/2+3	$h'_{lb} = x_{lb} + width$	x_{lb}, x_{ub}
		$h'_{ub} = s3_{ub}$	none
	1/3	$h'_{lb} = \max(x_{lb} + width, y_{lb})$	x_{lb}, x_{ub}, y_{lb}
		$h'_{lb} = \text{u\_bw\_or\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + half\_width + y_{ub}$	x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{u\_bw\_or\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1+2/1	swap x and y and use 1/1+2	
	1+2/1+2	$h'_{lb} = s2_{lb}$	none
		$h'_{ub} = s3_{ub}$	none
	1+2/2	$h'_{lb} = y_{lb}$	y_{lb}
		$h'_{ub} = s3_{ub}$	none
	1+2/2+3	$h'_{lb} = y_{lb}$	y_{lb}
		$h'_{ub} = s3_{ub}$	none
	1+2/3	$h'_{lb} = y_{lb}$	y_{lb}
		$h'_{ub} = s3_{ub}$	none
	2/1	swap x and y and use 1/2	
	2/1+2	swap x and y and use 1+2/2	

bitwise OR forward deduction			
operation	*x*/*y* sections	bound calculation	reasons
u_bw_or	2/2	$h'_{lb} = \max(x_{lb}, y_{lb})$	x_{lb}, y_{lb}
		$h'_{lb} = \text{u\_bw\_or\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + y_{ub}$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{u\_bw\_or\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	2/2+3	$h'_{lb} = \max(x_{lb}, y_{lb})$	x_{lb}, y_{lb}
		$h'_{lb} = \text{u\_bw\_or\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, s2_{ub})$	x_{lb}, x_{ub}, y_{lb}
		$h'_{ub} = x_{ub} + y_{ub}$	x_{lb}, x_{ub}, y_{ub}
		$h'_{ub} = \text{u\_bw\_or\_ub\_prefix}(x_{lb}, x_{ub}, s3_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{ub}
	2/3	$h'_{lb} = \max(x_{lb} + half\_width, y_{lb})$	x_{lb}, x_{ub}, y_{lb}
		$h'_{lb} = \text{u\_bw\_or\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + y_{ub}$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{u\_bw\_or\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	2+3/1	swap x and y and use 1/2+3	
	2+3/1+2	swap x and y and use 1+2/2+3	
	2+3/2	swap x and y and use 2/2+3	
	2+3/2+3	$h'_{lb} = \max(x_{lb}, y_{lb})$	x_{lb}, y_{lb}
		$h'_{lb} = \text{u\_bw\_or\_lb\_prefix}(x_{lb}, s2_{ub}, y_{lb}, s2_{ub})$	x_{lb}, y_{lb}
		$h'_{ub} = s3_{ub}$	none
	2+3/3	$h'_{lb} = y_{lb}$	y_{lb}
		$h'_{ub} = s3_{ub}$	none
	3/1	swap x and y and use 1/3	
	3/1+2	swap x and y and use 1+2/3	
	3/2	swap x and y and use 2/3	
	3/2+3	swap x and y and use 2+3/3	
	3/3	$h'_{lb} = \max(x_{lb}, y_{lb})$	x_{lb}, y_{lb}
		$h'_{lb} = \text{u\_bw\_or\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} - half\_width + y_{ub}$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{u\_bw\_or\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}

bitwise XOR forward deduction			
operation	*x*/*y* sections	bound calculation	reasons
s_bw_xor	1/1	$h'_{lb} = \text{s\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + half\_width + y_{ub} + half\_width$	x_{ub}, y_{ub}
		$h'_{ub} = \text{s\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1/1+2	$h'_{lb} = \text{s\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, s2_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{ub}
		$h'_{ub} = \text{s\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, s1_{ub})$	x_{lb}, x_{ub}, y_{lb}
	1/2	$h'_{lb} = \text{s\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + y_{ub}$	x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{s\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1/2+3	$h'_{lb} = \text{s\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, s2_{ub})$	x_{lb}, x_{ub}, y_{lb}
		$h'_{ub} = \text{s\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, s3_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{ub}

bitwise XOR forward deduction			
operation	*x*/*y* sections	bound calculation	reasons
s_bw_xor	1/3	$h'_{lb} = \text{s\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$ $h'_{ub} = x_{ub} + half\_width + y_{ub} - half\_width$ $h'_{ub} = \text{s\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{ub}, y_{lb}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1+2/1	swap *x* and *y* and use 1/1+2	
	1+2/1+2	$h'_{lb} = s1_{lb}$ $h'_{ub} = s2_{ub}$	none none
	1+2/2	$h'_{lb} = \text{s\_bw\_xor\_lb\_prefix}(x_{lb}, s1_{ub}, y_{lb}, y_{ub})$ $h'_{ub} = x_{ub} + y_{ub}$ $h'_{ub} = \text{s\_bw\_xor\_ub\_prefix}(s2_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, y_{lb}, y_{ub} x_{ub}, y_{lb}, y_{ub} x_{ub}, y_{lb}, y_{ub}
	1+2/2+3	$h'_{lb} = s1_{lb}$ $h'_{ub} = s2_{ub}$	none none
	1+2/3	$h'_{lb} = \text{s\_bw\_xor\_lb\_prefix}(s2_{lb}, x_{ub}, y_{lb}, y_{ub})$ $h'_{ub} = x_{ub} + half\_width + y_{ub} - half\_width$ $h'_{ub} = \text{s\_bw\_xor\_ub\_prefix}(x_{lb}, s1_{ub}, y_{lb}, y_{ub})$	x_{ub}, y_{lb}, y_{ub} x_{ub}, y_{lb}, y_{ub} x_{lb}, y_{lb}, y_{ub}
	2/1	swap *x* and *y* and use 1/2	
	2/1+2	swap *x* and *y* and use 1+2/2	
	2/2	$h'_{lb} = \text{s\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$ $h'_{ub} = x_{ub} + y_{ub}$ $h'_{ub} = \text{s\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub}
	2/2+3	$h'_{lb} = \text{s\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, s3_{lb}, y_{ub})$ $h'_{ub} = \text{s\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, s2_{ub})$	x_{lb}, x_{ub}, y_{ub} x_{lb}, x_{ub}, y_{lb}
	2/3	$h'_{lb} = \text{s\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$ $h'_{ub} = x_{ub} + y_{ub} - width$ $h'_{ub} = \text{s\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub}
	2+3/1	swap *x* and *y* and use 1/2+3	
	2+3/1+2	swap *x* and *y* and use 1+2/2+3	
	2+3/2	swap *x* and *y* and use 2/2+3	
	2+3/2+3	$h'_{lb} = s1_{lb}$ $h'_{ub} = s2_{ub}$	none none
	2+3/3	$h'_{lb} = \text{s\_bw\_xor\_lb\_prefix}(x_{lb}, s2_{ub}, y_{lb}, y_{ub})$ $h'_{ub} = x_{ub} - half\_width + y_{ub} - half\_width$ $h'_{ub} = \text{s\_bw\_xor\_ub\_prefix}(s3_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, y_{lb}, y_{ub} x_{ub}, y_{lb}, y_{ub} x_{ub}, y_{lb}, y_{ub}
	3/1	swap *x* and *y* and use 1/3	
	3/1+2	swap *x* and *y* and use 1+2/3	
	3/2	swap *x* and *y* and use 2/3	
	3/2+3	swap *x* and *y* and use 2+3/3	
	3/3	$h'_{lb} = \text{s\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$ $h'_{ub} = x_{ub} - half\_width + y_{ub} - half\_width$ $h'_{ub} = \text{s\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub}

bitwise XOR forward deduction			
operation	*x*/*y* sections	bound calculation	reasons
u_bw_xor	1/1	$h'_{lb} = \text{u\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + half\_width + y_{ub} + half\_width$	x_{ub}, y_{ub}
		$h'_{ub} = \text{u\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1/1+2	$h'_{lb} = \text{u\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, s1_{ub})$	x_{lb}, x_{ub}, y_{lb}
		$h'_{ub} = \text{u\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, s2_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{ub}
	1/2	$h'_{lb} = \text{u\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + width + y_{ub}$	x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{u\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1/2+3	$h'_{lb} = \text{u\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, s3_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{ub}
		$h'_{ub} = \text{u\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, s2_{ub})$	x_{lb}, x_{ub}, y_{lb}
	1/3	$h'_{lb} = \text{u\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + half\_width + y_{ub} - half\_width$	x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{u\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	1+2/1	swap x and y and use 1/1+2	
	1+2/1+2	$h'_{lb} = s2_{lb}$	none
		$h'_{ub} = s3_{ub}$	none
	1+2/2	$h'_{lb} = \text{u\_bw\_xor\_lb\_prefix}(s2_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{u\_bw\_xor\_ub\_prefix}(x_{lb}, s1_{ub}, y_{lb}, y_{ub})$	x_{lb}, y_{lb}, y_{ub}
	1+2/2+3	$h'_{lb} = s2_{lb}$	none
		$h'_{ub} = s3_{ub}$	none
	1+2/3	$h'_{lb} = \text{u\_bw\_xor\_lb\_prefix}(x_{lb}, s1_{ub}, y_{lb}, y_{ub})$	x_{lb}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + y_{ub}$	x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{u\_bw\_xor\_ub\_prefix}(s2_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{ub}, y_{lb}, y_{ub}
	2/1	swap x and y and use 1/2	
	2/1+2	swap x and y and use 1+2/2	
	2/2	$h'_{lb} = \text{u\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + y_{ub}$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{u\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	2/2+3	$h'_{lb} = \text{u\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, s2_{ub})$	x_{lb}, x_{ub}, y_{lb}
		$h'_{ub} = x_{ub} + y_{ub}$	x_{lb}, x_{ub}, y_{ub}
		$h'_{ub} = \text{u\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, s3_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{ub}
	2/3	$h'_{lb} = \text{u\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = x_{ub} + y_{ub}$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{u\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub}
	2+3/1	swap x and y and use 1/2+3	
	2+3/1+2	swap x and y and use 1+2/2+3	
	2+3/2	swap x and y and use 2/2+3	
	2+3/2+3	$h'_{lb} = s2_{lb}$	
		$h'_{ub} = s3_{ub}$	
	2+3/3	$h'_{lb} = \text{u\_bw\_xor\_lb\_prefix}(s3_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{ub}, y_{lb}, y_{ub}
		$h'_{ub} = \text{u\_bw\_xor\_ub\_prefix}(x_{lb}, s2_{ub}, y_{lb}, y_{ub})$	x_{lb}, y_{lb}, y_{ub}

bitwise XOR forward deduction			
operation	x/y sections	bound calculation	reasons
u_bw_xor	3/1	swap x and y and use 1/3	
	3/1+2	swap x and y and use 1+2/3	
	3/2	swap x and y and use 2/3	
	3/2+3	swap x and y and use 2+3/3	
	3/3	$h'_{lb} = \text{u\_bw\_xor\_lb\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$ $h'_{ub} = x_{ub} - half\_width + y_{ub} - half\_width$ $h'_{ub} = \text{u\_bw\_xor\_ub\_prefix}(x_{lb}, x_{ub}, y_{lb}, y_{ub})$	x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub} x_{lb}, x_{ub}, y_{lb}, y_{ub}

3.3. Cast between different Bitwidths

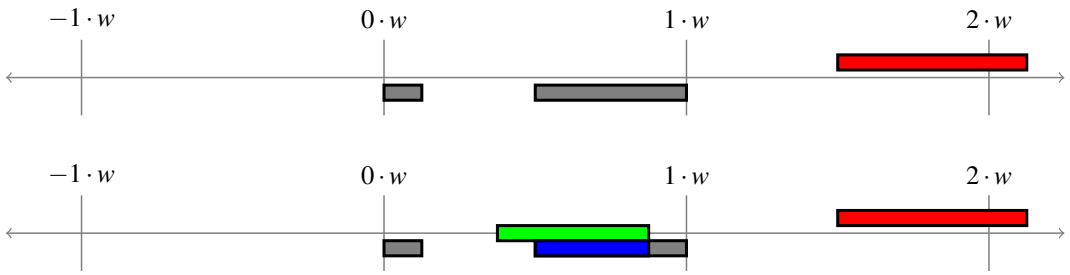
The C programming language allows casts between integers with different bitwidths and signedness, e.g. casting an `int` (signed 32 bit integer) to an `unsigned short` (unsigned 16 bit integer). In order to mimic the behaviour of these cases, the s_cast and u_cast operation were added to iSAT3 (based on the preliminary work of [6]). Intuitively, both operations are similar to modulo operations. They expect an argument interval (which has to be casted) and a constant parameter specifying the bitwidth of the result. While s_cast interprets the result of the cast as signed, u_cast interprets it as unsigned. In the following we will describe the u_cast operation in more detail – the details of s_cast are very similar.

In contrast to the bitwise NOT, AND, OR and XOR operations which were discussed earlier, the argument interval is not partitioned into sections for the u_cast operation. Instead, the argument interval is divided into segments. The width of each segment is $w = 2^{bitwidth}$ and thus is determined by the constant bitwidth parameter. Every segment is mapped to the interval $[0, w-1]$. Thus, if the argument interval $I(x)$ is larger than w then for each value v in the result interval there is at least one value in the argument interval which maps to v. Therefore, $I(h') = [0, w-1]$. This is the first case in the case distinction below. In order to keep the pictures simple, only three segments are shown.

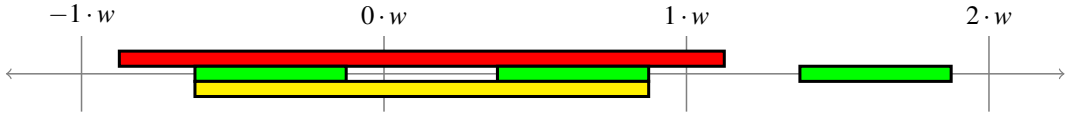
If the width of $I(x)$ (in red color) is larger or equal to w then $I(h') = [0, w-1]$ (in blue color):

If the width of $I(x)$ is less than w and all values of $I(x)$ lie in one segment then do a direct mapping to $[0, w-1]$:

If the width of $I(x)$ is less than w, but $I(x)$ spans two segments then two disjoint sub-intervals (in grey color) would be the result which have the trivial enclosing interval $[0, w-1]$. Therefore, $I(h)$ (in green color) is additionally considered:



For the backward deduction the current interval of $I(h)$ is projected into all segments. If the current bounds of $I(x)$ are in a "gap" between the projections then its bounds could be strengthened (in yellow color).



u_cast forward and backward deduction			
direction	case	bound calculation	reasons
forward	$x_{ub} - x_{lb} \geq w$	$h'_{lb} = 0$ $h'_{ub} = w - 1$	none none
	$x_{ub} - x_{lb} < w$ one segment	$s =$ segment of x_{lb} $h'_{lb} = x_{lb} - s \cdot w$ $h'_{ub} = x_{ub} - s \cdot w$	 x_{lb}, x_{ub} x_{lb}, x_{ub}
	$x_{ub} - x_{lb} < w$ two segments	$s =$ segment of x_{lb} $h''_{lb} = x_{lb} - s \cdot w$ $h''_{ub} = x_{ub} - (s+1) \cdot w$ $h'_{lb} = h''_{lb}$ if $(h''_{ub} < h_{lb})$ $h'_{lb} = 0$ if $(h''_{ub} \geq h_{lb})$ $h'_{ub} = h''_{ub}$ if $(h''_{lb} > h_{ub})$ $h'_{ub} = w - 1$ if $(h''_{lb} \leq h_{ub})$	 x_{lb}, x_{ub}, h_{lb} none x_{lb}, x_{ub}, h_{ub} none
backward	$x_{lb} < h_{lb} + s \cdot w$	$s =$ segment of x_{lb} $x'_{lb} = h_{lb} + s \cdot w$	 x_{lb}, h_{lb}
	$x_{lb} > h_{ub} + s \cdot w$	$s =$ segment of x_{lb} $x'_{lb} = h_{lb} + (s+1) \cdot w$	 x_{lb}, h_{lb}, h_{ub}
	$x_{ub} > h_{ub} + s \cdot w$	$s =$ segment of x_{ub} $x'_{ub} = h_{ub} + s \cdot w$	 x_{ub}, h_{ub}
	$x_{ub} < h_{lb} + s \cdot w$	$s =$ segment of x_{ub} $x'_{ub} = h_{ub} + (s-1) \cdot w$	 x_{ub}, h_{lb}, h_{ub}

The s_cast operation can be build upon u_cast by adding $w/2$ to x_{lb}, x_{ub}, h_{lb} and h_{ub} before applying u_cast. Afterwards $w/2$ has to be subtracted from x'_{lb}, x'_{ub}, h'_{lb} and h'_{ub}.

4. Conclusion

When solely considering bitwise operations then bit-blasting would be the method of choice. On the other hand, arithmetic constraints and its operations (like addition, subtraction, multiplication in its integer and floating-point variants) are better handled on the arithmetic level (e.g. with ICP) – a translation into a propositional formula is sub-optimal in this case. However, when aiming for the full range of basic data types and operations in C programs, both operation types (arithmetic and bitwise) have to be handled. While SMT solvers based on bit-blasting already provided support for both operation types, iSAT3 is the first SMT solver not relying on bit-blasting which now supports these operations.

Nonetheless, the ICP-contractors for bitwise operations presented in this report are not optimal – e.g. some contractors suffer from the missing backward deduction. Furthermore, in order to get tighter bounds for bitwise AND, OR and XOR, both argument intervals can be partitioned into sets of sub-intervals S_1 and S_2. Then the bit-prefix approach could be applied by combining each sub-interval from S_1 with every sub-interval from S_2. The lower and upper bound could be obtained by determing the minimum and the maximum outcome over all bit-prefix operations. So this work can be seen as a starting point for further research in this direction – perhaps resulting in a hybrid approach which combines the strengths of ICP and bit-blasting. The method proposed in [12] already goes in this direction by combining bit-blasting and a reasoning mechanism similar to ICP in a prototypical implementation in Z3.

References

[1] Martin Brain, Vijay D'Silva, Leopold Haller, Alberto Griggio, and Daniel Kroening. An Abstract Interpretation of DPLL(T). In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, volume 7737 of *Lecture Notes in Computer Science*, pages 455–475. Springer, 2013.

[2] Dominik Erb, Karsten Scheibler, Matthias Sauer, and Bernd Becker. Efficient SMT-based ATPG for Interconnect Open Defects. In Gerhard Fettweis and Wolfgang Nebel, editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*, pages 1–6. European Design and Automation Association, 2014.

[3] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient Solving of Large Non-Linear Arithmetic Constraint Systems with Complex Boolean Structure. *JSAT*, 1(3-4):209–236, 2007.

[4] Christian Herde. *Efficient Solving of Large Arithmetic Constraint Systems with Complex Boolean Structure: Proof Engines for the Analysis of Hybrid Discrete-Continuous Systems*. PhD thesis, Carl von Ossietzky University of Oldenburg, 2011.

[5] Giorgio Metta, Lorenzo Natale, Shashank Pathak, Luca Pulina, and Armando Tacchella. Safe and Effective Learning: A Case Study. In *IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010*, pages 4809–4814. IEEE, 2010.

[6] Felix Neubauer. *Deadcode-Detection mit Hilfe von SMT-Techniken*. Masterarbeit, Albert-Ludwigs-Universität Freiburg im Breisgau, Germany, 2014. Becker / Scheibler.

[7] Andreas Schäfer and Mathias John. Conceptional Modeling and Analysis of Spatio-Temporal Processes in Biomolecular Systems. In Markus Kirchberg and Sebastian Link, editors, *Conceptual Modelling 2009, Sixth Asia-Pacific Conference on Conceptual Modelling (APCCM 2009), Wellington, New Zealand, January 20-23 2009.*, volume 96 of *CRPIT*, pages 39–48. Australian Computer Society, 2009.

[8] Karsten Scheibler and Bernd Becker. Using Interval Constraint Propagation for Pseudo-Boolean Constraint Solving. In *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*, pages 203–206. IEEE, 2014.

[9] Karsten Scheibler, Stefan Kupferschmid, and Bernd Becker. Recent Improvements in the SMT Solver iSAT. In Christian Haubelt and Dirk Timmermann, editors, *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV), Warnemünde, Germany, March 12-14, 2013.*, pages 231–241. Institut für Angewandte Mikroelektronik und Datentechnik, Fakultät für Informatik und Elektrotechnik, Universität Rostock, 2013.

[10] Karsten Scheibler, Felix Neubauer, Ahmed Mahdi, Martin Fränzle, Tino Teige, Tom Bienmüller, Detlef Fehrer, and Bernd Becker. Accurate ICP-based Floating-Point Reasoning. In *Formal Methods in Computer-Aided Design, FMCAD 2016*. IEEE, 2016.

[11] Mohamed H. Zaki, Ian M. Mitchell, and Mark M. Greenstreet. DC Operating Point Analyis: a Formal Approach. Workshop on Formal Verification of Analog Circuits (FAC'09), 2009.

[12] Aleksandar Zeljic, Christoph M. Wintersteiger, and Philipp Rümmer. Deciding Bit-Vector Formulas with mcSAT. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 249–266. Springer, 2016.