Theoretical and Practical Aspects of Bit-Vector Reasoning

Andreas Fröhlich
Institute for Formal Models and Verification
Johannes Kepler University

PhD Presentation
Thursday, April 28th, 2016
Linz, Austria
Research Overview

- Research Areas: Bit-Vectors, SAT, DQBF, SMT, Local Search, ...

- List of Contributions:
  - Total of 15 publications (14 peer-reviewed, 1 benchmark description)
  - 2 solvers (1 publicly available)
  - 2 translation tools (both publicly available)
  - Several challenging benchmark families (publicly available)

- Thesis “Theoretical and Practical Aspects of Bit-Vector Reasoning”
  - Consisting of 9 publications
  - Some additional unpublished complexity results
Outline

- Preliminaries

- Selected key contributions
  - Complexity of bit-vector logics
  - Reencoding of bit-vector formulas
  - DQBF solving
  - SLS for SMT

- Conclusion
Outline

- Preliminaries

- Selected key contributions
  - Complexity of bit-vector logics
  - Reencoding of bit-vector formulas
  - DQBF solving
  - SLS for SMT

- Conclusion
Bit-Vector Reasoning: Motivation

- Bit-Vector: String of bits \( \{0, 1\}^n \) of fixed length \( n \)

- Practical Applications
  - Hardware Verification
    - Natural representation of RTL specifications (e.g., VHDL, Verilog)
    - Equivalence checking or property checking (e.g., used by Intel)
  - Software Verification
    - Natural representation of datatypes
    - SAGE: Large-scale project at Microsoft
Complexity Classes

\[ P \subseteq NP \subseteq PSPACE \subseteq NEXPTIME \subseteq EXPSPACE \subseteq 2-NEXPTIME \subseteq \ldots \]

- Bounds in regard to the **input size**:
  - **P**: problems can be solved in polynomial time
  - **NP**: solutions can be checked in polynomial time
  - **PSPACE**: problems can be solved with polynomial space
  - **NEXPTIME**: solutions can be checked in exponential time

- **NEXPTIME**: more succinct representations than NP
  - Can be solved by NP algorithms after (exponential) expansion
Related Problems

Propositional domain \{0, 1\}:

- **SAT** \[ \exists x_1, x_2, x_3 . \ (x_1 \lor \neg x_2) \land (\neg x_1 \lor x_3) \land (x_2 \lor \neg x_3) \land (\neg x_1 \lor x_2) \]
  NP-complete

- **QBF** \[ \forall u_1 \exists e_1 \forall u_2 \exists e_2 . \ (u_2 \lor \neg e_1) \land (\neg u_1 \lor e_1) \land (u_1 \lor \neg e_2) \land (\neg u_2 \lor e_2) \]
  PSPACE-complete

- **DQBF** \[ \forall u_1, u_2 \exists e_1(u_1), e_2(u_2) . \ (u_2 \lor \neg e_1) \land (\neg u_1 \lor e_1) \land (u_1 \lor \neg e_2) \land (\neg u_2 \lor e_2) \]
  NEXPTIME-complete

First-order but no functions:

- **EPR** \[ \exists a, b \forall x, y . \ (p(a, x, y) \lor \neg q(y, x, b)) \land (q(x, b, y) \lor \neg p(y, a, x)) \]
  (Bernays-Schönfinkel class)
  NEXPTIME-complete
Bit-Vector Logics

- QF_BV: Included in SMT-LIB
- Bit-Vector Variables: $x^{[4]}$, $y^{[8]}$, $z^{[1]}$, ...
- Bit-Vector Constants: $1011^{[4]}$, $10011010^{[8]}$, $1^{[8]}$, ...
- Bit-Vector Operators:
  - Bitwise: $\sim$, $\&$, $|$ $\oplus$ ...
  - Arithmetic: $+$, $-$, $\cdot$, $/$ ...
  - Relational: $=$, $<$, $\leq$ ...
  - Shifts: $\ll$, $\gg$ ...
  - ...

Andreas Fröhlich, PhD Presentation, April 2016
Outline

- Preliminaries

- Selected key contributions
  - Complexity of bit-vector logics
  - Reencoding of bit-vector formulas
  - DQBF solving
  - SLS for SMT

- Conclusion
Solving QF_BV

- Running example:

\[(z = x + y) \land (z = x \ll 1) \land (x \neq y)\]

- With bit-vectors of fixed bit-width \(n\), e.g., \(n = 32\):

\[(z^{[32]} = x^{[32]} + y^{[32]}) \land (z^{[32]} = x^{[32]} \ll 1^{[32]}) \land (x^{[32]} \neq y^{[32]})\]

- Satisfiability: Are there bit-vectors, so that the formula evaluates to true?

  - Common solving approach:
    - **Bit-blasting** (encoding the bit-vector formula as a circuit) . . .
    - . . . and then using a **SAT-solver**

  - Often assumed to be NP-complete:

    “This paper addresses the satisfiability problem for bit-vector formulas: [...] It is easy to see that this problem is NP-complete.”
Complexity actually depends on the **encoding of bit-widths**

Consider the previous example, ...

\[
(z[n] = x[n] + y[n]) \land (z[n] = x[n] \ll 1[n]) \land (x[n] \neq y[n])
\]

... with large \( n \), e.g., \( n = 1,000,000 \).

In practice: **logarithmic encoding**, e.g., SMT-LIB format

```
(set-logic QF_BV)
(declare-fun x () (_ BitVec 1000000))
(declare-fun y () (_ BitVec 1000000))
(declare-fun z () (_ BitVec 1000000))
(assert (= z (bvadd x y)))
(assert (= z (bvshl x (_ bv1 1000000))))
(assert (distinct x y))
```
\( x^n \) can be “written down” using \( \log(n) \) bits, …

```
(set-logic QF_BV)
(declare-fun x () (_ BitVec 1000000))
(declare-fun y () (_ BitVec 1000000))
(declare-fun z () (_ BitVec 1000000))
(assert (= z (bvadd x y)))
(assert (= z (bvshl x (_ bv1 1000000)))))
(assert (distinct x y))
```

… but bit-blasting requires \( n \) separate variables \( x_0, x_1, \ldots, x_{n-1} \)

<table>
<thead>
<tr>
<th>bit-width</th>
<th>input size</th>
<th>bit-blasting</th>
<th>output size</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>223 Byte</td>
<td>0.0s</td>
<td>4.1 kB</td>
</tr>
<tr>
<td>100</td>
<td>227 Byte</td>
<td>0.0s</td>
<td>51.7 kB</td>
</tr>
<tr>
<td>1,000</td>
<td>231 Byte</td>
<td>0.0s</td>
<td>610.3 kB</td>
</tr>
<tr>
<td>10,000</td>
<td>235 Byte</td>
<td>0.9s</td>
<td>7.0 MB</td>
</tr>
<tr>
<td>100,000</td>
<td>239 Byte</td>
<td>14.1s</td>
<td>79.3 MB</td>
</tr>
<tr>
<td>1,000,000</td>
<td>243 Byte</td>
<td>167.9s</td>
<td>883.6 MB</td>
</tr>
<tr>
<td>10,000,000</td>
<td>247 Byte</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Satisfiability for QF\_BV is \text{NExpTime}-complete \[\text{[SMT'12]}\]

Hardness: reduction from DQBF to QF\_BV

- Use the so-called \textbf{binary magic numbers} (e.g., in Knuth—TAOCP)

\[
\forall u_0 u_1 u_2 \quad \rightarrow \quad U_0^{[8]} := \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \quad U_1^{[8]} := \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad U_2^{[8]} := \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}
\]

- Eliminate dependencies:

\[
\exists e(u_0, u_2) \quad \rightarrow \quad E^{[8]} \land \sim U_1^{[8]} = (E^{[8]} \ll 2^{[8]}) \land \sim U_1^{[8]}
\]
Complexity depends on the encoding . . .

...but also on the set of operators:

- $\text{QF}_{\text{BV}} \ll$ (only bitwise operations, equality, and left shift)
  
  $\text{QF}_{\text{BV}} \ll$ is $\text{NEXPTime}$-complete

- $\text{QF}_{\text{BV}} \ll_1$ (only bitwise operations, equality, and left shift by one)
  
  $\text{QF}_{\text{BV}} \ll_1$ is $\text{PSPACE}$-complete

- $\text{QF}_{\text{BV}_{bw}}$ (only bitwise operations and equality)
  
  $\text{QF}_{\text{BV}_{bw}}$ is $\text{NP}$-complete
Outline

- Preliminaries

- Selected key contributions
  - Complexity of bit-vector logics
  - Reencoding of bit-vector formulas
  - DQBF solving
  - SLS for SMT

- Further research

- Conclusion (Summary, Impact, Future Work)
Consider the previous example:

\[(z[n] = x[n] + y[n]) \land (z[n] = x[n] \ll 1[n]) \land (x[n] \neq y[n])\]

Can we do better than bit-blasting?

- + can be expressed by \(\oplus \mid \& = \ll_1\)

\[(z[n] = x[n] \oplus y[n] \oplus c_{in}[n]) \land (c_{out}[n] = (x[n] \& y[n]) \mid ((c_{in}[n] \ll 1[n]) \& (x[n] \mid y[n])))\]

- The example is in \(\text{QF}_\text{BV} \ll_1\)

\[\rightarrow\text{can be solved in PSPACE}\]

- **Polynomial** encoding as a **model checking** problem
SMV Encoding of Addition

init(counter_bit0) := FALSE;
next(counter_bit0) := counter_bit0 xor (TRUE);
init(counter_bit1) := FALSE;
next(counter_bit1) := counter_bit1 xor (counter_bit0);
...
init(counter_bit19) := FALSE;
next(counter_bit19) := counter_bit19 xor
  (counter_bit0 & ... & counter_bit18);

init(counter_gte_1000000) := FALSE;
next(counter_gte_1000000) := counter_gte_1000000 |
  (counter_bit0 & counter_bit1 & ... & counter_bit19);

init(atom_add) := TRUE;
next(atom_add) := case
  counter_gte_1000000 : atom_add;
  TRUE : atom_add & (z <-> (x xor y xor atom_cin));
esac;

init(atom_cin) := FALSE;
next(atom_cin) := case
  counter_gte_1000000 : atom_cin;
  TRUE : (x & y) | (x & atom_cin) | (y & atom_cin);
esac;

AG(!counter_gte_1000000 | !atom_add)
- **BV2SMV**: Polynomial translation from $\text{QF}_B\text{V} \ll_1$ to SMV

- SMV formulas can be solved with model checkers
  - BDD based model checkers are most efficient

- Application benchmarks by Intel
Outline

- Preliminaries
- Selected key contributions
  - Complexity of bit-vector logics
  - Reencoding of bit-vector formulas
  - DQBF solving
  - SLS for SMT
- Conclusion
DQBF: Motivation

- Interesting in the context of QF_BV
  - DQBF is \textit{N}EX\textsc{p}T\textsc{ime}-complete $\rightarrow$ possible target logic for QF_BV

- Succinct encodings of problems
  - Partial equivalence checking
  - Partial information games

- However: Not a lot of previous work
  - Mainly theoretic
  - No existing solver
DQBF: Solvers (1)

- **DQDPLL**
  - DPLL and QDPLL successful for SAT and QBF
  - Search-based approach
    - Requires **dependency constraints** to be respected
  - Many **techniques can be lifted** (bottom-up)
    - Unit Propagation, Pure Literal Reduction, Clause Learning
    - Universal Reduction, Cube Learning
  - Prototype: Not very efficient

- The first existing DQBF solver
DQBF: Solvers (2)

- **iDQ**
  - iProver successful for EPR
  - Techniques can be reused and refined (top-down)
    - SAT overapproximations
  - **CEGAR loop**
    - More efficient than iProver
    - Can compete with QBF solvers
  - First publicly available (complete) DQBF solver
Outline

- Preliminaries

- Selected key contributions
  - Complexity of bit-vector logics
  - Reencoding of bit-vector formulas
  - DQBF solving
  - SLS for SMT

- Conclusion
Search on the space of full assignments $\alpha \in \{0, 1\}^n$

- Starting from an initial assignment
- Local “improvement” in regard to a heuristic “score”
- Typical score for SAT: Number of unsatisfied clauses

Example:

- $F = (x_0 \lor x_1) \land (\neg x_0 \lor \neg x_2) \land (\neg x_1 \lor \neg x_2)$, with $\alpha = (0, 0, 0), F(\alpha) = 0 \land 1 \land 1$
  
  $\rightarrow \alpha(x_0) := \neg \alpha(x_0)$, with $\alpha = (1, 0, 0), F(\alpha) = 1 \land 0 \land 1$

- $\rightarrow \alpha(x_2) := \neg \alpha(x_2)$, with $\alpha = (1, 0, 1), F(\alpha) = 1 \land 1 \land 1$

- Stochastic: Probabilistic component in choosing the next move
Lifting SLS to SMT

- Stochastic local search for SAT
  - Lots of previous work, but bad on application benchmarks

- BV-SLS: Stochastic local search for bit-vectors [AAAI’15]
  - No bit-blasting
  - Works on the theory representation of the formula

- Idea: Combine techniques from SAT with QF_BV theory information
  - Many techniques from SAT can successfully be lifted
  - Theory information allows to deal with structure efficiently
BV-SLS: Results

<table>
<thead>
<tr>
<th></th>
<th>solved instances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QF_BV</td>
</tr>
<tr>
<td>CCAnr</td>
<td>5409</td>
</tr>
<tr>
<td>CCASat</td>
<td>4461</td>
</tr>
<tr>
<td>probSAT</td>
<td>3816</td>
</tr>
<tr>
<td>Sparrow</td>
<td>3805</td>
</tr>
<tr>
<td>VW2</td>
<td>2954</td>
</tr>
<tr>
<td>PAWS</td>
<td>3331</td>
</tr>
<tr>
<td>YalSAT</td>
<td>3756</td>
</tr>
<tr>
<td>Z3 (Default)</td>
<td>7173</td>
</tr>
<tr>
<td>Z3 BV-SLS</td>
<td>6172</td>
</tr>
</tbody>
</table>

Andreas Fröhlich, PhD Presentation, April 2016
Outline

- Preliminaries

- Selected key contributions
  - Complexity of bit-vector logics
  - Reencoding of bit-vector formulas
  - DQBF solving
  - SLS for SMT

- Conclusion
Conclusion

- Presented contributions:
  - Complexity of quantifier-free bit-vector logics \([SMT’12, CSR’13]\)
  - Reencoding of QF\(BV\ll_1\) to SMV \([SMT’13]\)
  - 2 decision procedures for DQBF \([POS’12, POS’14]\)
  - Lifting stochastic local search to the theory level \([AAAI’15]\)

- Further results:
  - Reencoding of QF\(BV\) to EPR \([CADE’13]\)
  - More on the complexity of bit-vector logics \([MFCS’14, TOCS’15, Thesis’16]\)
  - Improving state-of-the-art in SAT solving \([SAT’14a, SAT’14b, POS’15, SAT’15]\)
References (1)

- Andreas Fröhlich, Gergely Kovácsznai, Armin Biere. A DPLL Algorithm for Solving DQBF. [POS’12]
- Gergely Kovácsznai, Andreas Fröhlich, Armin Biere. On the Complexity of Fixed-Size Bit-Vector Logics with Binary Encoded Bit-Width. [SMT’12]
- Gergely Kovácsznai, Andreas Fröhlich, Armin Biere. BV2EPR: A Tool for Polynomially Translating Quantifier-free Bit-Vector Formulas into EPR. [CADE’13]
- Andreas Fröhlich, Gergely Kovácsznai, Armin Biere. Efficiently Solving Bit-Vector Problems Using Model Checkers. [SMT’13]
- Gergely Kovácsznai, Helmut Veith, Andreas Fröhlich, Armin Biere. On the Complexity of Symbolic Verification and Decision Problems in Bit-Vector Logic. [MFCS’14]
- Tomáš Balyo, Andreas Fröhlich, Marijn Heule, Armin Biere. Everything You Always Wanted to Know about Blocked Sets (But Were Afraid to Ask). [SAT’14a]
- Adrian Balint, Armin Biere, Andreas Fröhlich, Uwe Schöning. Improving implementation of SLS solvers for SAT and new heuristics for k-SAT with long clauses. [SAT’14b]

- Andreas Fröhlich, Gergely Kovásznai, Armin Biere, Helmut Veith. iDQ: Instantiation-Based DQBF Solving. [POS’14]

- Andreas Fröhlich, Armin Biere, Christoph M. Wintersteiger, Youssef Hamadi. Stochastic Local Search for Satisfiability Modulo Theories. [AAA1’15]


- Armin Biere, Andreas Fröhlich. Evaluating CDCL Variable Scoring Schemes. [SAT’15]

- Armin Biere, Andreas Fröhlich. Evaluating CDCL Restart Schemes. [POS’15]

- Andreas Fröhlich. Theoretical and Practical Aspects of Bit-Vector Reasoning. [Thesis’16]
- **BV2SMV**: Polynomial translation from QF\_BV\ll_1 to SMV
  
  - SMV formulas can be solved with model checkers
    
    - BDD based model checkers are most efficient
  
- Application benchmarks by Intel
Example: SLS for SMT

\[ x + 3 = \sim x, \]

where \( x \) is a bit-vector of \( n = 6 \). If we initialize the search:

\[ x = [0, 0, 0, 0, 0, 0] \]

\[ \rightarrow [0, 0, 0, 0, 1, 1] = [1, 1, 1, 1, 1, 1] \]

Best improvement by negating \( x \):

\[ x = [1, 1, 1, 1, 1, 1] \]

\[ \rightarrow [0, 0, 0, 0, 1, 0] = [0, 0, 0, 0, 0, 0] \]

Flipping the least significant bit is the only move that will further increase the score:

\[ x = [1, 1, 1, 1, 1, 0] \]

\[ \rightarrow [0, 0, 0, 0, 0, 1] = [0, 0, 0, 0, 0, 1] \]
Example: iDQ

\[ \psi = \forall u_1, u_2 \exists e_1(u_1, u_2), e_2(u_2) . (u_1 \lor e_1) \land (\overline{u_2} \lor \overline{e_1} \lor e_2) \]

Initial set of clause instances:

\[ (e_1)\overline{u_1} \land (\overline{e_1} \lor e_2)u_2 \]

Propositional abstraction:

\[ (x_1) \land (\overline{x_2} \lor x_3) \]

\[ \rightarrow \alpha = \{ x_1 \rightarrow 1, x_2 \rightarrow 0, x_3 \rightarrow 0 \} \]

Refinement:

\[ (e_1)\overline{u_1} \land (e_1)\overline{u_1}u_2 \land (\overline{e_1} \lor e_2)u_2 \land (\overline{e_1} \lor e_2)\overline{u_1}u_2 \]

Propositional abstraction:

\[ (x_1) \land (x_2) \land (\overline{x_3} \lor x_4) \land (\overline{x_2} \lor x_4) \]

\[ \rightarrow \alpha = \{ x_1 \rightarrow 1, x_2 \rightarrow 1, x_3 \rightarrow 0, x_4 \rightarrow 1 \} \]
Reduction: QF_BV to EPR

Bitwise: $z^{[2^n]} = x^{[2^n]} \mid y^{[2^n]}$

$$p_z(i_{n-1}, \ldots, i_0) \leftrightarrow p_x(i_{n-1}, \ldots, i_0) \lor p_y(i_{n-1}, \ldots, i_0)$$

Shift by one: $z^{[2^n]} = x^{[2^n]} \ll 1^{[2^n]}$

$$\text{succ}(i_{n-1}, \ldots, i_3, i_2, i_1, 0, i_{n-1}, \ldots, i_3, i_2, i_1, 1)$$

$$\text{succ}(i_{n-1}, \ldots, i_3, i_2, 0, 1, i_{n-1}, \ldots, i_3, i_2, 1, 0)$$

$$\text{succ}(i_{n-1}, \ldots, i_3, 0, 1, 1, i_{n-1}, \ldots, i_3, 1, 0, 0)$$

$$\vdots$$

$$\text{succ}(0, 1, \ldots, 1, 1, 0, \ldots, 0)$$

$$\neg p_z(0, \ldots, 0) \land (\text{succ}(i_{n-1}, \ldots, i_0, j_{n-1}, \ldots, j_0) \rightarrow (p_z(j_{n-1}, \ldots, j_0) \leftrightarrow p_x(i_{n-1}, \ldots, i_0)))$$
State-of-the-art solvers for QF_BV rely on **bit-blasting** and SAT solvers

- Bit-blasting can be exponential
- Is it possible to solve QF_BV without bit-blasting?
- Can we profit from knowing the complexity of certain bit-vector classes?

Some **alternative approaches** (and optimizations) exist

- Translation to EPR  
  [CADE’13]
- Translation to SMV  
  [SMT’13]
- Bit-width reduction (by Johannsen)  
- SLS for SMT  
  [AAAI’15]
- **BV2EPR**: Polynomial translation from QF_BV to EPR

- EPR formulas can be solved with iProver (by Korovin)
  - CEGAR approach

- Performance worse than bit-blasting for most instances
  - Beneficial on some instances (0.1s instead of T/O)
  - Less memory used (can be several orders of magnitude)
- For QF\textsubscript{BV}\textsubscript{bw}, bit-width reduction can be applied
  - There is a solution iff there is a solution with smaller bit-width, e.g.
    \[
    (X^{[32]} \neq Y^{[32]}|Z^{[32]}) \land (Y^{[32]} \neq Z^{[32]} \& X^{[32]})
    \]
    \[
    \rightarrow (X^{[2]} \neq Y^{[2]}|Z^{[2]}) \land (Y^{[2]} \neq Z^{[2]} \& X^{[2]})
    \]
  - Can be extended to allow certain cases of other operators

- Existing work for RTL Property Checking (by Johannsen)
  - Reduces size of design model to up to 30%
  - Reduces runtimes to up to 5%
Upgrading Theorem: If a problem is complete for a complexity class $C$, it is complete for a $\nu$-exponentially harder complexity class than $C$ when succinctly encoded by bit-vectors with $\nu$-logarithmic scalars. [MFCS’14]

Implication: Word-Level Model Checking and Reachability for bit-vectors with binary encoded bit-widths are \text{EXPSPACE}-complete.
- **Upgrading SAT**: Satisfiability for quantifier-free bit-vector formulas with \( \nu \)-logarithmic encoded scalars is \( \nu \)-\text{NExpTime}-complete. [Thesis’16]
  - Proof: Reduction from Turing machines or domino tiling problems.
Complexity of Bit-Vector Logics

<table>
<thead>
<tr>
<th>encoding</th>
<th>quantifiers</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>uninterpreted functions</td>
<td>NP</td>
<td>NP</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>binary</td>
<td>NExPTIME</td>
<td>NExPTIME</td>
</tr>
</tbody>
</table>
The head initially is at position 0:

$$H^N \land lo^N = 1^N \ll mid^N$$

$M$ initially is in state $s$:

$$Q_s^N \land lo^N = 1^N \ll mid^N$$

In each computation step, there is at most one symbol per tape cell, i.e., $\forall \sigma, \sigma' \in \Sigma$, with $\sigma \neq \sigma'$, we add:

$$\neg T_\sigma^N \lor \neg T_{\sigma'}^N = \neg 0^N$$

In each computation step, there is at least one symbol per tape cell:

$$\bigvee_{\sigma \in \Sigma} T_\sigma^N = \neg 0^N$$
- In each computation step, there is at most one state at a time, i.e., $\forall q, q' \in Q$, with $q \neq q'$, we add:

$$\neg Q_q^{[N]} \vee \neg Q_{q'}^{[N]} = \neg 0^{[N]}$$

- The bits of the state variables can only be set at the head positions:

$$\bigvee_{q \in Q} Q_q^{[N]} \land \neg H^{[N]} = 0^{[N]}$$

- The tape does not change at positions different from those of the head, i.e., $\forall \sigma \in \Sigma$, we add:

$$(T_\sigma^{[N]} \ll size^{[N]} \leftrightarrow T_\sigma^{[N]}) \lor (H^{[N]} \ll size^{[N]}) \lor lo^{[N]} = \neg 0^{[N]}$$
- The transition relation, i.e., \( \forall q \in Q, \sigma \in \Sigma \), we add:

\[
(H^{[N]} \land Q_q^{[N]} \land T_{\sigma}^{[N]}) \ll size^{[N]} \rightarrow \\
\bigvee_{(q, \sigma, q', \sigma', d) \in \delta} (H^{[N]} \circ d 1^{[N]} \land Q_{q'}^{[N]} \land T_{\sigma'}^{[N]}) = \neg 0^{[N]}
\]

- \( M \) must reach a final state at one point:

\[
\bigvee_{q \in F} Q_q^{[N]} \land H^{[N]} \neq 0^{[N]}
\]

- Helper variables: \( size^{[N]} = 2 \cdot \exp_{\nu}(n) + 1 \), \( mid^{[N]} = \exp_{\nu}(n) \),

\[
hi^{[N]} = \neg 0^{[N]} \ll size^{[N]} , \quad lo^{[N]} = \neg(\neg 0^{[N]} \ll size^{[N]})
\]
If the head is in a certain position in a computation step, it cannot be at any position other than left or right of the current one in the next step:

\[ lo^{[N]} \lor \neg H^{[N]} \lor H^{[N]} \ll (\text{size} + 1)^{[N]} \lor H^{[N]} \ll (\text{size} - 1)^{[N]} = \neg 0^{[N]} \]

If the head is in a certain position in a computation step, it has to be at position left or right (non-exclusive) of the current one in the next step:

\[ \neg (H^{[N]} \ll \text{size}^{[N]}) \lor H^{[N]} \ll 1^{[N]} \lor H^{[N]} \gg_u 1^{[N]} = \neg 0^{[N]} \]

In any computation step, the head will never be at two distinct positions exactly two indices apart from each other:

\[ H^{[N]} \ll 1^{[N]} \land H^{[N]} \gg_u 1^{[N]} = 0^{[N]} \]