# Indexing Transformations for Relational Symbolic Trajectory Evaluation

**Tang Yu Han, Brandon**

A dissertation submitted for the degree of
*Computer Science (Part B)*
Trinity 2025

Word Count: 4995

# Abstract

Symbolic Trajectory Evaluation (STE) is a symbolic model checking method for hardware verification. It makes verification more tractable by symbolically simulating families of related initial states over an abstracted circuit, a technique called symbolic indexing. Relational STE (rSTE) is a significant reformulation of the original STE logic and method to allow for arbitrary relational properties to be verified.

This project reinvents the theory of symbolic indexing transformations, previously limited to STE, to allow rSTE to exploit symbolic indexing abstractions systematically. We formulate a new theory for efficient indexing transformations on a subclass of abstractions called partitioned abstractions in rSTE, going beyond the original theory for STE to handle so-called symbolic constants. We also discuss methods for dealing with environmental constraints, a significant weakness of the original STE method.

These procedures are implemented in the industry standard verification tool Jasper and our experiments show them to be more efficient on uniform circuits compared to vanilla symbolic simulation without symbolic indexing.

# Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Thomas Melham, for his invaluable guidance, insight, and encouragement throughout this project. His support and thoughtful feedback have been instrumental at every stage of this work.

I am also deeply grateful to my collaborator, Julia Irvine, for her dedication and contributions to the companion project on the automatic generation of indexing relations for rSTE.

# Contents

# Chapter 1
# Introduction

Formal verification is the process of proving that a system satisfies a given specification or set of properties.

Symbolic Trajectory Evaluation (STE) [1], [2], [3] is a symbolic model checking technique [4] for performing hardware formal verification on circuits. It allows the verification of functional properties while incorporating circuit abstraction by grouping families of related input cases together and simulating them symbolically. STE has had particular success in the verification of memory circuits [5], [6], which have highly regular structures that can be exploited by abstraction.

The core idea of STE is performing circuit simulation over a 3-valued domain of “true”, “false” and “don’t know” (X) that is ordered by information content [7]. On any concrete set of inputs over the 3-valued domain, a simulator can compute the output of the circuit, processing each logical gate based on a modified truth table to account for the X value, such as in Table 1.

| AND | 0 | 1 | X |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | X |
| X | 0 | X | X |

| D-Latch | 0 | 1 | X |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| X | X | X | X |

Table 1: 3-valued Excitation Function for AND Gate and D-Latch

We can use this 3-valued simulation to observe behaviours of circuits on groups of related inputs. For example, by simulating an AND gate on the inputs 0 and $X$, we can observe that the output is 0 and thus conclude that $0 \wedge 0 = 1 \wedge 0 = 0$. This has allowed us to merge the analysis of two cases that we would have to simulate separately in a conventional 2-valued Boolean simulation.

Furthermore, rather than doing simulation of only one concrete instance of 3-valued inputs at once, we do multiple 3-valued simulations at once via symbolic simulation. In the case of basic Boolean simulation, we could start by assigning a variable to each input and then propagate these values through the circuit, building up a propositional formula at each node that represents the truth value of the circuit node.

For our 3-valued simulation, we do this symbolic simulation using a pair of Boolean formulae over "indexing variables" on each circuit node. For each node, these are called the "high" and "low" expressions and have the following semantics.[1]

| | $H_s = 0$ | $H_s = 1$ |
|---|---|---|
| $L_s = 0$ | $s$ is X | $s$ is 1 |
| $L_s = 1$ | $s$ is 0 | s is $\top$ |

Table 2: Semantics of High and Low Expressions

where $\top$ represents inconsistent information.

The symbolic simulator propagates these high and low expressions through the circuit. For example, for an AND gate, on inputs with expressions $(h_A, l_A)$ and $(h_B, l_B)$, the output produced would be $(h_A \wedge h_B, l_A \vee l_B)$.

Symbolic simulators generally represent these expressions as reduced, ordered binary decision diagrams (BDDs) [8] that allow for efficient manipulation and thus simulation of the circuit. We term these pairs of BDDs as dual-rail BDDs.

As an example, consider verification of a 3-input AND gate. We would like to prove that $o = a \wedge b \wedge c$ for inputs $a, b, c$. To use STE to verify the AND gate, observe that rather than simulating all $2^3$ different Boolean input assignments to the circuit, it is sufficient to simulate 4 cases under the 3-valued domain:

$$p \wedge q : a = b = c = 1$$
$$\overline{p} \wedge \overline{q} : a = 0, b = c = X$$
$$\overline{p} \wedge q : b = 0, a = c = X$$
$$p \wedge \overline{q} : c = 0, a = b = X$$

[1]This implementation strategy is the modern one implemented in Jasper. The legacy literature[3] had different names and interpretations for the two expressions.

We can enumerate these cases symbolically by assigning them propositional formulae in terms of indexing variables $p$ and $q$. Using propositional formulae to do case-splitting this way is termed as symbolic indexing.

To apply this symbolic indexing scheme, we will set the dual rail inputs as follows:

$$h_a := p \wedge q, l_a := \overline{p} \wedge \overline{q}$$
$$h_b := p \wedge q, l_b := \overline{p} \wedge q$$
$$h_c := p \wedge q, l_c := p \wedge \overline{q}$$

We then run the symbolic simulator, getting output $(h_o, l_o) = (p \wedge q, \overline{p} \vee \overline{q})$. We can then conclude that the AND gate outputs true if and only if $a = b = c = 1$ and false otherwise, which is the desired property.

## 1.1. Starting Point for this Research

Legacy literature on STE [1] provided a theory of writing stimuli (antecedent) to combinational circuits using a linear temporal logic, specifying an intended functional output (consequent) and then using the 3-valued symbolic simulator to verify that the circuit would satisfy the consequent under the given antecedent. But, it was difficult in this setting to write stimuli in a manner that covered all relevant input cases and to interpret the results of the symbolic simulation to check properties. The theory of indexing relation transformations [9] was therefore developed to simplify usage of STE.

To use STE with indexing transformations, we first set up an antecedent and consequent without any abstraction from Xs. We then apply a transformation on the antecedent and consequent based on an indexing relation that mapped groups of target assignments to corresponding indexing variable assignments. We then check that the transformed antecedent leads to the transformed consequent holding on the circuit. Assuming a technical coverage condition held for the indexing relation, this would imply that the original property held on the circuit. Further work was later done to automate the process of generating effective indexing relations via an automatic abstraction algorithm [10].

This theory and its implementation had promise, but there were two impediments to it being used across the semiconductor industry. First, the tools that implemented these methods were proprietary to Intel [11], although academic prototypes also existed [12]. Second, the abstraction method proposed in these papers did not incorporate environmental constraints, which are virtually essential for real-world use.

Since then, a variant of STE known as relational STE (rSTE) [13] that allows for specifications with arbitrary relational properties has been described, but not formalized. Furthermore, a dual-rail BDD symbolic simulator has been developed within the industry standard formal verification tool Jasper [14].

## 1.2. Contributions

This project reinvents the theory of symbolic indexing transformations for rSTE and brings it to the modern day in Jasper.

- We provide a procedure for incorporating symbolic indexing into the rSTE workflow via indexing transformations. We provide proofs of soundness for the procedure of indexing transformations and interpretations of the simulation outputs to derive proofs and counterexamples.

- We provide new proofs and formal descriptions for efficient preimage computations on a subclass of indexing relations known as partitioned indexing relations that extend the existing technique by incorporating symbolic constants mentioned in [10].

- We formulate theory for dealing with environmental constraints in rSTE using ideas from [9] but updated to consider partitioned indexing relations with symbolic constants.

- We implement the theoretical procedures to perform rSTE with symbolic indexing in Jasper, including procedures for efficient indexing transformations of partitioned indexing relations, running the symbolic simulator with transformed antecedent, and checking transformed output.

- We evaluate the efficiency of rSTE on two scalable example circuits: a Content-Addressable Memory (CAM) and a multi-input maximum circuit. For each of these, we evaluate the technique with the use of a manually constructed indexing relation and an automatically generated one when compared to symbolic simulation without symbolic indexing. We used a manual indexing of the CAM based on [6] and crafted an original one for the maximum circuit.

## 1.3. Relation to Companion Project

While this project focuses on the process of using indexing relations to perform rSTE with symbolic indexing, the companion project focuses on the automatic generation of indexing relations for rSTE. The two projects are complementary and can be used together to perform symbolically indexed rSTE with minimal user input.

# Chapter 2
# Theory for Relational STE

## 2.1. Design Under Test, Specifications and Relational Properties

We make use of Jasper to process a circuit design under test (DUT) written in SystemVerilog at Register Transfer Level (RTL).

### 2.1.1. Relational Properties with SVA

Traditionally STE properties were functional properties that were written in a linear temporal logic that allowed you to assert that specific signals would be high or low depending on the specific stimulus on the input of the circuit.

However, rSTE allows for more general properties through the use of a bound specification circuit that implements properties written as SystemVerilog Assertions (SVA). When the DUT is elaborated, the specification circuit will be instantiated together with it. The specification takes as input, the inputs to the DUT, along with internal/output signals from the DUT that the are relevant to the properties checked.

The key to checking relational properties is the insight that SVA properties can be expressed as a circuit that produces a signal that is high if the property holds in that clock cycle and low otherwise. This is done as part of Jasper's internal symbolic simulation model construction. We call the signal that represents the property being held as the "property wire".

When using the symbolic simulator without symbolic indexing, we will do a Boolean symbolic simulation and check that the property wire is high under any assignment of inputs.

### 2.1.2. Output Constraints

To incorporate symbolic indexing via indexing transformations, we build an additional layer of output constraints on top of the SVA properties that we want to check. These output constraints will mirror the LTL properties from STE but are written in a 5-tuple form as implemented in VOSSII [15]:

```
Output Constraint: [(signal, tickStart, tickEnd, highExpr, lowExpr)]
```

Each output constraint consists of two parts:

1. Positive output constraint: When `highExpr` is true, the signal should be high.
2. Negative output constraint: When `lowExpr` is true, the signal should be low.

If neither are true, then we don't assert anything about the signal. `highExpr` and `lowExpr` are called the guards of the output constraint. The `tickStart` and `tickEnd` form the range of clock cycles that the output constraint should hold for.

Before doing the symbolic indexing, for each SVA property we wish to check, we will build a corresponding output constraint of the form:

```
Output Constraint: [(propertyWire, tickStart, tickEnd, True, False)]
```

This asserts that the SVA property should always hold. Such output constraints are termed "unrestricted output constraints".

### 2.1.3. Antecedents

The antecedent (stimulus) of the circuit will also be written in the 5-tuple form. The tuple `(signal, tickStart, tickEnd, highExpr, lowExpr)` in the stimulus, means that the signal is high when `highExpr` is true, low when `lowExpr` is true and X or inconsistent otherwise.

## 2.2. Indexing Relations and Transformations

Indexing relations are used to symbolic represent input case-splitting to perform symbolic indexing and are the basis of the indexing transformation procedure.

Indexing relations $R[X, C, T]$ are Boolean formulae over propositional variables where

- $X$ are the indexing propositional variables that index cases
- $T$ are the target propositional variables that are indexed. These variables correspond to input wires in the circuit and assignments to them correspond to concrete inputs.
- $C$ are the symbolic constants that correspond to input wires that should not be indexed.

The indexing relation $R[X, C, T]$ is interpreted as:

- For each $X, C$, we cover the cases $T, C$ where $\exists T R[X, C, T]$.

Note that the indexing relation $R'[(X, C'), \emptyset, (C, T)] = R[X, C, T] \wedge \bigwedge(C = C')$ that doesn't use symbolic constants is equivalent to $R[X, C, T]$ when doing indexing transformations.

Symbolic constants don't improve the expressive power of indexing relations, but rather simplify them to achieve higher efficiency. When using the automatic abstraction algorithm in the companion paper, symbolic constants also provide a crude way to prevent over-abstraction since the algorithm will not include them in abstraction.

### 2.2.1. Preimage Operations

Suppose

- $P[C, T]$ represents some set of input cases
- $R[X, C, T]$ is an indexing relation

The **weak preimage** of $P$ under $R$,

$$P_R[X, C] = \exists T(R[X, C, T] \wedge P[C, T])$$

is the set of $(X, C)$ that cover at least one $(T, C)$ case in $P$

The **strong preimage** of $P$ under $R$,

$$P^R[X, C] = P_R \wedge \neg\exists T(R[X, C, T] \wedge \neg P[C, T])$$

is the set of $(X, C)$ that cover at least one case in $P$ and don't cover any cases not in $P$.

### 2.2.2. Image Operation

The image of $H[X, C]$

$$\mathrm{im}(H, R)[C, T] = \exists X(R[X, C, T] \wedge H[X, C])$$

is the set of cases indexed by $H[X, C]$ under $R$.

### 2.2.3. Relationship between Preimages

For any fixed $R$ and any given $P$ and $(X, C)$ one of the following holds:

- $X, C$ indexes cases only in $P$
- $X, C$ indexes cases only in $\neg P$
- $X, C$ indexes cases in both $P$ and $\neg P$
- $X, C$ indexes cases in neither $P$ nor $\neg P$

Any $X, C$ that fall into the 4th category must correspond to empty indexing cases, i.e. don't map onto any $(C, T)$. This is independent of $P$, thus, we term the union of the first 3 cases as the domain of $R$.

$$\mathrm{dom}(R)[X, C] = \exists T(R[X, C, T])$$

Observe that

- The weak preimage is the set of $X, C$ that fall into the 1st and 3rd categories.
- The strong preimage is the set of $X, C$ that fall into the 1st category.

The following are equivalent definitions of the strong preimage operation:

$$\begin{aligned} P^R[X, C] &= \operatorname{dom}(R) \wedge \overline{\overline{P}_R} \\ &= \operatorname{dom}(R) \wedge (\forall T(R[X, C, T] \to P[C, T])) \end{aligned}$$

### 2.2.4. Indexing Coverage Condition

For each output constraint $i$ that we wish to check, we let the guards be $P_i$ and $Q_i$. The total space of inputs we need to check is then $B[C, T] = \bigvee_i (P_i \vee Q_i)$. In order for our abstraction to fully cover this space, we require that

$$\forall T \forall C (B[T, C] \to \exists X (R[X, C, T]))$$

This is the coverage condition, and it is required for the indexing transformation procedure to be sound.

For unrestricted output constraints, we will have $P_i = \text{True}, Q = \text{False}$. In this case, $B[C, T] \equiv \text{True}$ so the coverage condition takes the simpler form:

$$\forall T \forall C \exists X (R[X, C, T])$$

### 2.2.5. Constructing Indexing Relations

To formulate an indexing relation, we will need to encode the case-splitting for the different input cases that elicit different outputs from the circuit. This is done manually for specific circuits in the examples of Section 5.1 and Section 5.2.

The companion project reinvents the automatic abstraction algorithm from [10] and implements it within Jasper. It produces an indexing relation that satisfies the coverage condition by construction.

## 2.3. rSTE Model Checking Procedure

Suppose we are given a circuit, an indexing relation $R$, and an output constraint list `cout`. We will describe the procedure to prove that the output constraints hold.

We first construct unabstracted antecedents for each of the input variables and time steps

```
antv = [(signal, t, t, s, not s)]
```

where $s$ is a bdd variable for the signal at time $t$.

We then apply the strong preimage operations on the high and low expressions of the antecedent tuples as follows:

```
antv = [(s, t, t, s^R, (not s)^R)]
```

Next we run the symbolic simulator with the transformed antecedent. This produces an evaluation sequence which will contain `highExpr` and `lowExpr` BDD expressions for when a signal at a certain time step is known to be high or low.

From this evaluation sequence, we can then check the output constraints. To check an output constraint of the form `(signal, tickStart, tickEnd, P, Q)` , we will transform the output constraint to

```
(signal, tickStart, tickEnd, P_R, Q_R)
```

where $P_R$ and $Q_R$ are the weak preimages of $P$ and $Q$.

Then we analyse the `highExpr` and `lowExpr` BDD expressions from the evaluation sequence for the signal at each of the ticks in the tick range. Suppose that at a given tick $t$, for the property wire the high expression is $H[X, C]$ and the low expression is $L[X, C]$.

If $(P_R \to H)[X, C] \equiv \text{True}$ and $(Q_R \to L)[X, C] \equiv \text{True}$, then the output constraint is satisfied.

If $(P^R \wedge L) \neq \text{False}$, we have disproven the positive output constraint. $\text{im}(P^R \wedge L, R)$ symbolically represents the counterexamples found in terms of the original circuit inputs.

Similarly, if $(Q^R \wedge H) \neq \text{False}$, we have disproven the negative part of the property.

It is possible to not prove the property but also have no counterexamples. This occurs in cases where the antecedent does not provide enough information to prove or disprove the property. This is called a weak disagreement [3] and requires refining the indexing relation.

## 2.4. Simplified Checking for Unrestricted Output Constraints

A benefit of the unrestricted output constraint form is that the property guards are simple and thus have nice properties related to the indexing transformation.

To check the properties, we need to know the weak/strong preimage images of the guards. However, we have the following:

$$\text{True}_R = \text{True}^R = \text{dom}(R)[X, C]$$
$$\text{False}_R = \text{False}^R = \text{False}$$

This means that we only need to take a single preimage operation to get the domain and use that for checking all the properties written in this form. In fact, computing the domain of a partitioned indexing relation (Section 3) is a simple operation that will be described later.

Furthermore, analysis of counterexamples is also simplified.

$$\text{True}^R \wedge L = \text{dom}(R)[X, C] \wedge L = L$$

The 2nd equality is since we use strong preimages of target variables and their negations to transform the antecedents, so for any signal and time step, $H$ and $L$ don't include any cases that are not in the domain of $R$.

## 2.5. Soundness of Model Checking Procedure

### 2.5.1. Symbolic Simulation Invariants

Let $\text{st}(C, T)$ be true iff the signal and time pair st is high under the untransformed antecedent under some assignment of the variables in $C, T$.

Consider the low and high expressions in any signal timestep pair. We have that

$$H_{\text{st}}[X, C] \wedge R[X, C, T] \rightarrow \text{st}(C, T)$$
$$L_{\text{st}}[X, C] \wedge R[X, C, T] \rightarrow \overline{\text{st}(C, T)}$$

(For all $X, C, T$)

This can be proved via induction on the fan-in of st.

The main base case is where $s$ is an input variable corresponding to a certain time step, with `highExpr` $P$ and `lowExpr` $\overline{P}$. In this case, since we apply the strong preimage to the antecedents, we have that

$$H = P^R = \text{dom}(R) \wedge \forall T(R[X, C, T] \rightarrow P[C, T])$$
$$L = \overline{P}^R = \text{dom}(R) \wedge \forall T\Big(R[X, C, T] \rightarrow \overline{P[C, T]}\Big)$$

This directly gives us that for all $X, C, T$ where $H[X, C] \wedge R[X, C, T]$, we have $P[C, T]$. But $\text{st}(C, T) = P[C, T]$ since $P$ will be a target variable / symbolic constant corresponding to the stimulus to the signal at the time step in the untransformed antecedent, so we are done.[2] The proof for $L[X, C] \wedge R[X, C, T] \to \overline{P[C, T]}$ is analogous.

The subtle base case is circuit nodes with state, such as a latch, before any stimulus from the antecedent reaches it. In this case, the high and low expressions will be false so the invariants hold trivially.

An inductive case example:

Suppose $s = s_1 \wedge s_2$. Since the AND gate is combinational, we fix any time step. Let $\left(H_{s_1}, L_{s_1}\right)$ and $\left(H_{s_2}, L_{s_2}\right)$ be the high and low expressions for $s_1$ and $s_2$ respectively. The symbolic simulator will give us $(H_s, L_s) = \left(H_{s_1} \wedge H_{s_2}, L_{s_1} \vee L_{s_2}\right)$. Under any fixed $X, C, T$, Figure 1 gives us a proof that $H_s[X, C] \wedge R[X, C, T] \to s(C, T)$ and $L_s[X, C] \wedge R[X, C, T] \to \neg s(C, T)$.

Other circuit components can be proven in a similar manner.

[2]Note that we don't use the domain part for this. It just serves to remove useless/inconsistent indexing cases, which is useful for counterexample analysis.

| | | | |
|---|---|---|---|
| | **1.** | $H_s[X,C]$ | **Assume** |
| | **2.** | $R[X,C,T]$ | **Assume** |
| *1* | **3.** | $H_{s_1}[X,C] \wedge H_{s_2}[X,C]$ | $H_s = H_{s_1} \wedge H_{s_2}$ |
| | **4.** | $H_{s_1}[X,C] \wedge R[X,C,T] \to s_1(C,T)$ | **Inductive Hypothesis** |
| | **5.** | $H_{s_2}[X,C] \wedge R[X,C,T] \to s_2(C,T)$ | **Inductive Hypothesis** |
| *2, 3, 4* | **6.** | $s_1$ | |
| *2, 3, 5* | **7.** | $s_2$ | |
| *6, 7* | **8.** | $s$ | $s = s_1 \wedge s_2$ |
| *1, 2, 8* | **9.** | $H_s[X,C] \wedge R[X,C,T] \to s(C,T)$ | **Conclusion** |

| | | | |
|---|---|---|---|
| | **1.** | $L_s[X,C]$ | **Assume** |
| | **2.** | $R[X,C,T]$ | **Assume** |
| *1* | **3.** | $L_{s_1}[X,C] \vee L_{s_2}[X,C]$ | $L_s = L_{s_1} \vee L_{s_2}$ |
| | **4.** | $L_{s_1}[X,C] \wedge R[X,C,T] \to \neg s_1(C,T)$ | **Inductive Hypothesis** |
| | **5.** | $\quad \vert\ L_{s_1}[X,C]$ | **Assume** |
| *2, 4, 5* | **6.** | $\quad \vert\ \neg s_1(C,T)$ | |
| *6* | **7.** | $\quad \vert\ \neg s$ | $s = s_1 \wedge s_2$ |
| | **8.** | $L_{s_2}[X,C] \wedge R[X,C,T] \to \neg s_2(C,T)$ | **Inductive Hypothesis** |
| | **9.** | $\quad \vert\ L_{s_2}[X,C]$ | **Assume** |
| *2, 8, 9* | **10.** | $\quad \vert\ \neg s_2(C,T)$ | |
| *10* | **11.** | $\quad \vert\ \neg s$ | |
| *3, 5-7, 9-11* | **12.** | $\neg s$ | |
| *1, 2, 12* | **13.** | $L_s[X,C] \wedge R[X,C,T] \to \neg s(C,T)$ | **Conclusion** |

Figure 1: Inductive Case on AND Gate

### 2.5.2. Output Constraint Checking

Positive output constraints are of the form:

$$\forall C \forall T (P[C,T] \to s(C,T))$$

If $P_R[X,C] \to H[X,C] \equiv \text{True}$ and the coverage condition $\forall T \forall C (B[C,T] \to \exists X (R[X,C,T]))$ holds, then we know the output constraint will hold as shown in Figure 2.

| | | | |
|---|---|---|---|
| | **1.** | **Fresh X, C, T** | |
| | **2.** | $P[C, T]$ | **Assume** |
| | **3.** | $R[X, C, T]$ | **Assume** |
| *2, 3* | **4.** | $P[C, T] \wedge R[X, C, T]$ | |
| *4* | **5.** | $\exists T(P[C, T] \wedge R[X, C, T])$ | |
| *5* | **6.** | $P_R[X, C]$ | **Definition of $P_R$** |
| *1, 6* | **7.** | $\forall X, C, T\ (P[C, T] \wedge R[X, C, T] \to P_R[X, C])$ | **Since $X, C, T$ were arbitrary. Lemma** |

| | | | |
|---|---|---|---|
| | **1.** | **Fresh** $C, T$ | |
| | **2.** | $\forall X(P_R[X, C] \to H[X, C])$ | **Premise** |
| | **3.** | $B[C, T] \to \exists X(R[X, C, T])$ | **Premise** |
| | **4.** | $P[C, T]$ | **Assume** |
| | **5.** | $\forall X, C, T\ (P[C, T] \wedge R[X, C, T] \to P_R[X, C])$ | **Lemma above** |
| *4* | **6.** | $B[C, T]$ | **Definition of $B[C, T]$** |
| *3, 6* | **7.** | **Fresh** $X^*$ **s.t.** $R[X^*, C, T]$ | |
| *4, 5, 7* | **8.** | $P_R[X^*, C]$ | |
| *2, 8* | **9.** | $H[X^*, C]$ | |
| *9* | **10.** | $s(C, T)$ | **Symbolic Simulation Invariants** |
| *4, 10* | **11.** | $P[C, T] \to s(C, T)$ | |
| *1, 11* | **12.** | $\forall C \forall T(P[T, C] \to s(C, T))$ | **Since $C, T$ were arbitrary** |

Figure 2: Proof of Positive Output Constraint Check

We can prove the check for negative output constraints of the form

$$\forall C \forall T(P[C, T] \to \neg s(C, T))$$

in a similar manner.

### 2.5.3. Alternative Checking by Reversing the Indexing

Another way to do the check is to take the image of $H[X, C]$ under the indexing relation, $\mathrm{im}(H, R)[C, T]$. This will symbolically represent all the cases for which we know the property will hold.

$$\begin{aligned} C, T \in \mathrm{im}(H, R) &\Rightarrow \exists X(H[X, C] \wedge R[X, C, T]) \\ &\Rightarrow H[X^*, C] \wedge R[X^*, C, T] \text{ for some } X^* \\ &\Rightarrow s(C, T) \text{ by symbolic simulation invariant} \end{aligned}$$

We can then check that $P \to \mathrm{im}(H, R) \equiv \mathrm{True}$. If so, then the property is true. On some circuits and indexing relations, this method can avoid weak disagreements compared to the preimage method. This is particularly true if some cases in $P$ are indexed by multiple $X, C$. If the number of indexing variables is typically much less than the number of target variables, this method can result in large BDDs that may be infeasible to compute.

### 2.5.4. Counterexample Analysis

Suppose that $P^R \wedge L \neq \mathrm{False}$. Let $(X, C)$ be such that $(P^R \wedge L)[X, C]$ is true. We show that $(X, C)$ is a counterexample to the property $\forall C \forall T(P[C, T] \to s(C, T))$.

| | | | |
|---|---|---|---|
| | **1.** | $P^R[X, C] \wedge L[X, C]$ | **Premise** |
| *1* | **2.** | $\mathrm{dom}(R)[X, C] \wedge \forall T(R[X, C, T] \to P[C, T])$ | **Definition of** $P^R$ |
| *2* | **3.** | $\exists T(R[X, C, T])$ | **Definition of** $\mathrm{dom}(R)$ |
| *3* | **4.** | **Fresh** $T^*$ **s.t.** $R[X, C, T^*]$ | |
| *2, 4* | **5.** | $P[C, T^*]$ | |
| | **6.** | $L[X, C] \wedge R[X, C, T^*] \to \neg s(C, T^*)$ | **Symbolic Simulation Invariants** |
| *1, 4, 6* | **7.** | $\neg s(C, T^*)$ | |
| *5, 7* | **8.** | $P[C, T^*] \to \neg s(C, T^*)$ | |
| *8* | **9.** | $\exists C \exists T(P[C, T] \to \neg s(C, T))$ | |
| *9* | **10.** | $\neg \forall C \forall T(P[C, T] \to s(C, T))$ | **Property disproven** |

Figure 3: Proof for Positive Property Counterexample Check

It is practical to note that the set of counterexamples we find is

$$\{(C, T) \mid \exists X(R[X, C, T] \wedge L[X, C] \wedge P^R[X, C])\}$$

which is described by the image operation on $P^R \wedge L$, $\mathrm{im}(P^R \wedge L, R)$.

We can prove that the counterexample analysis for negative properties is correct in a similar manner.

# Chapter 3

# Partitioned Abstraction Relations

An important subclass of indexing relations that we consider is the partitioned indexing relation. A partitioned indexing relation is one that can be expressed in the following form:

$$R = \bigwedge(\text{hexpr} \to \text{expr} \wedge \text{lexpr} \to \overline{\text{expr}})$$

Where `expr` is either some **target variable** or an **expression of symbolic constants** while `hexpr` and `lexpr` are in terms of only the indexing variables.

In the implementation, a partitioned abstraction relation is represented as a list of tuples of the form:

```
[(expr = targVar / Cexpr, hexpr, lexpr)]
```

The indexing relations produced by the automatic abstraction algorithm and the manually constructed indexing relations that we use in our scalable examples are partitioned abstractions.

Partitioned abstractions allow computation of preimages which are critical for performing indexing transformations at scale.

## 3.1. Efficient Weak Preimage Computation

We first normalise $R$ into

$$R = S[X, C] \wedge U[X, T]$$

where $U[X, T] = \bigwedge_{t_i \in \text{ TargVars}}(h_i \to t_i \wedge l_i \to \overline{t_i})$

We can then make several observations.

Firstly,

$$\text{dom}(R)[X, C] = S \wedge \bigwedge_i \overline{h_i \wedge l_i}$$

Proof:

$$\begin{aligned}
\mathrm{dom}(R)[X,C] &= \exists T R[X,C,T] \\
&= S[X,C] \wedge \exists T U[X,T] \\
&= S[X,C] \wedge \bigwedge_i (\exists t_i (h_i \to t_i \wedge l_i \to \overline{t_i})) \\
&= S[X,C] \wedge \bigwedge_i (((h_i \to 1) \wedge (l_i \to 0)) \vee ((h_i \to 0 \wedge l_i \to 1))) \\
&= S[X,C] \wedge \bigwedge_I (\neg l_i \vee \neg h_i) \\
&= S \wedge \bigwedge_i \overline{h_i \wedge l_i}
\end{aligned}$$

Thus we are able to compute the domain of the indexing relation easily.

Secondly, to compute the preimage of some guard $P[C,T]$, we let $R \downarrow P$ denote that the part of the indexing relation that mentions target variables present in $P$. Specifically, if $\mathcal{F} = \mathrm{FreeTargVars}(P)$ then

$$R \downarrow P = S[X,C] \wedge \bigwedge_{t_i \in \mathcal{F}} (h_i \to t_i \wedge l_i \to \overline{t_i})$$

We have that $P_R = \mathrm{dom}(R) \wedge P_{R\downarrow P}$

Proof:

$$
\begin{aligned}
P_R[X,C] &= \exists T(R[X,C,T] \wedge P[C,T]) \\
&= \exists T\left(S[X,C] \wedge \bigwedge_{t_i \notin \mathcal{F}} (h_i \to t_i \wedge l_i \to \overline{t_i}) \wedge \bigwedge_{t_i \in \mathcal{F}} (h_i \to t_i \wedge l_i \to \overline{t_i}) \wedge P[C,T_{\mathcal{F}}]\right) \\
&= S[X,C] \wedge \exists T_{\neg\mathcal{F}}\left(\bigwedge_{t_i \notin \mathcal{F}} (h_i \to t_i \wedge l_i \to \overline{t_i})\right) \\
&\quad \wedge \exists T_{\mathcal{F}}\left(\bigwedge_{t_i \in \mathcal{F}} (h_i \to t_i \wedge l_i \to \overline{t_i}) \wedge P[C,T_{\mathcal{F}}]\right) \\
&= S[X,C] \wedge \exists T_{\neg\mathcal{F}}\left(\bigwedge_{t_i \notin \mathcal{F}} (h_i \to t_i \wedge l_i \to \overline{t_i})\right) \wedge \exists T_{\mathcal{F}}\left(\bigwedge_{t_i \in \mathcal{F}} (h_i \to t_i \wedge l_i \to \overline{t_i})\right) \\
&\quad \wedge S[X,C] \wedge \exists T_{\mathcal{F}}\left(\bigwedge_{t_i \in \mathcal{F}} (h_i \to t_i \wedge l_i \to \overline{t_i}) \wedge P[C,T_{\mathcal{F}}]\right) \\
&= S[X,C] \wedge \exists T\left(\bigwedge_{t_i \in \text{ targVars}} (h_i \to t_i \wedge l_i \to \overline{t_i})\right) \\
&\quad \wedge \exists T_{\mathcal{F}}\left(S[X,C] \wedge \bigwedge_{t_i \in \mathcal{F}} (h_i \to t_i \wedge l_i \to \overline{t_i}) \wedge P[C,T_{\mathcal{F}}]\right) \\
&= S[X,C] \wedge \bigwedge_i \left(\overline{h_i \wedge l_i}\right) \wedge P_{R\downarrow P} \\
&= \text{dom}(R) \wedge P_{R\downarrow P}
\end{aligned}
$$

We can further consider common cases of $P$ that we will be constructing preimages for.

$$
P_R = \begin{cases}
\text{dom}(R) \wedge P & \text{if } \mathcal{F} \cap \text{TargVars} = \emptyset \\
\text{dom}(R) \wedge \overline{l_i} & \text{if } P = t_i \\
\text{dom}(R) \wedge \overline{h_i} & \text{if } P = \overline{t_i} \\
\text{dom}(R) \wedge P_{R\downarrow P} & \text{otherwise}
\end{cases}
$$

Proof:

If $\mathcal{F} \cap \text{TargVars} = \emptyset$, $R \downarrow P = \text{True}$,

$$P_R = \text{dom}(R) \wedge \exists T(\text{True} \wedge P[C]) = \text{dom}(R) \wedge P[C]$$

If $P = t_i$, $R \downarrow P = (h_i \to t_i) \wedge (l_i \to \overline{t_i})$,

$$P_R = \text{dom}(R) \wedge \exists t_i((h_i \to t_i) \wedge (l_i \to \overline{t_i}) \wedge t_i) = \text{dom}(R) \wedge \overline{l_i}$$

If $P = \overline{t_i}$, $R \downarrow P = (h_i \to t_i) \land (l_i \to \overline{t_i})$,

$$P_R = \text{dom}(R) \land \exists t_i((h_i \to t_i) \land (l_i \to \overline{t_i}) \land \overline{t_i}) = \text{dom}(R) \land \overline{h_i}$$

## 3.2. Efficient Strong Preimage Computation

Observe that

$$\begin{aligned} P^R &= \text{dom}(R) \land \overline{\overline{P}_R} \\ &= \text{dom}(R) \land \neg\Big(\text{dom}(R) \land P_{R\downarrow\overline{P}}\Big) \\ &= \text{dom}(R) \land \neg P_{R\downarrow\overline{P}} \end{aligned}$$

Thus, the final conjunction with $\text{dom}(R)$ when computing a preimage is unnecessary if we are going to immediately be using the preimage to compute a strong preimage.

This represents an important speed-up since we will be doing many strong preimage computations where computing $P_{R\downarrow\overline{P}}$ is easy (such as when $P = t_i$) but conjuncting it with $\text{dom}(R)$ is expensive since $\text{dom}(R)$ can be complex.

# Chapter 4
# Verification Under Environmental Constraints

Environmental constraints are constraints on the inputs to the circuit. They are of the form $J[C, T]$ to denote that we only need a constraint to hold if $J[C, T]$ is true. We call such a constraint a “care predicate”.

## 4.1. Indexing Relation Restriction

An easy way to include environmental constraints is to just add them to the guards of the output constraint. We can then check for the property holding under the environmental constraint as described above.

However, this can lead to weak disagreements for abstractions that merge cases in $J$ and $\neg J$, since we cannot assume that $J$ holds for the $X, C$ indexing these cases.

It is thus desirable to find indexing relations that exactly index cases in $J$ and not any in $\neg J$, i.e.

$$\forall C \forall T \ ((\exists X R[X, C, T]) \rightarrow J[C, T])$$

We can get such an indexing relation by conjuncting the original indexing relation and the input constraint to make a new indexing relation.

$$R'[X, C, T] = R[X, C, T] \wedge J[C, T]$$

This will ensure that the indexing relation only ever indexes target variable assignments satisfying the care predicate. However, this can still lead to weak disagreements from the indexing relation being too coarse.

An additional way to mitigate the weak disagreements is to use the alternative checking method involving reversing the indexing (Section 2.5.3).

### 4.1.1. Preserve Partitioned Indexing Relation by Conditioning Antecedent

The indexing relation restriction unfortunately destroys the partitioned structure of partitioned indexing relations, meaning we cannot apply the efficient preimage computations from Section 3.

However, we can employ an equivalent strategy where we first modify each BDD expression $E$ in the antecedent to the form $E' := (E \land J) \lor \overline{J} \equiv J \rightarrow E$. We then do the strong preimage computations, allowing us to only consider the indexing cases we know $E$ to be true when the care predicate is true . We modify the guard of the output constraint to include $J$ and do the checking as described above.

This approach will potentially cause some signals to be $\top$, but only on indexing variable assignments that exclusively index $\overline{J}$, such as the blue assignment in Figure 6. This is fine since $J_R$ and $J^R$ not will contain such cases, thus not affecting output checking procedures for neither correctness nor counterexample analysis. Since the only indexing cases that we consider during analysis are those that index at least one case in $J$, this is equivalent to the restriction method.

Since the indexing relation is preserved, we can use the efficient partitioned abstraction preimage operations.

## 4.2. Parametric Encoding

An alternative approach is to use a parametric encoding [16] of the input constraints. A parametric encoding uses the `param` function to compute a substitution of the input signals with functions of new parameterisation variables.

`param` takes a list of input constraints and a list of signals $s_1, s_2, ..., s_n$ and computes Boolean functions $f_1, f_2, ..., f_n$ from new parameterisation variables $\boldsymbol{p} = \{p_1, ..., p_k\}$ where $k \leq n$ for the purpose of substituting $s_i := f_i(\boldsymbol{p})$.

These functions satisfy the following conditions:

- (Soundness): $\forall \boldsymbol{p}, \langle s_i := f_i(\boldsymbol{p}) \mid i \in 1..n \rangle$ satisfies the input constraints
- (Completeness): $\forall \langle s_1, s_2, ..., s_n \rangle$ that satisfy the input constraints, $\exists \boldsymbol{p}\ s_i = f_i(\boldsymbol{p})$

An efficient algorithm to compute `param` is described in [16].

The intention behind symbolic constants is that they should not be abstracted or replaced. Thus, in our analysis of parametric encoding, we will assume that the symbolic constants are not part of the environmental constraint and thus not passed into `param`.

There are two ways to apply this to deal with environmental constraints.

### 4.2.1. Parametric Encoding of Indexing Relations

The first strategy is to apply the parametric encoding to transform an independently computed indexing relation. This was suggested in [9] but not proven sound.

```
Input: circuit, antecedent, input constraints, output constraints
1. Compute param on the input constraints
2. Substitute each BDD variable in the unrestricted antecedent with the corresponding function from the parametric encoding
3. Compute an indexing relation, possibly via the automatic abstraction algorithm
4. Substitute each BDD variable in the automatic abstraction result with the corresponding function from the parametric encoding
5. Perform indexing transformation on the parameterised antecedent, and output constraints
6. Perform symbolic simulation
7. Check whether the output constraint holds or a counterexample exists

Output: proven, disproven (with counterexample) or unproven
```

Figure 4: Parametric Encoding of Indexing Relation

We prove this to be sound.

Parameterising the antecedent is equivalent to binding the parameterisation functions onto each input to form a larger circuit. For each $C, T$ such that $J[C, T]$ holds, we will have some $C, T'$ that maps to it by the completeness of `param`.

The main base case for the symbolic simulation invariants still holds even though our input signals are now functions of the parameterisation variables. The input signals $s$ are now replaced with the param functions $f_s$ and our base case argument will still hold. This means that the symbolic simulation invariants will still hold.

Furthermore, assuming that the indexing relation $R[X, C, T]$ used satisfies the coverage condition $\forall T \forall C (J[C, T] \to \exists X R[X, C, T])$ then the parameterised indexing relation $R'[X, C, T']$ will also satisfy the coverage condition, in terms of the new parameterisation variables, i.e. $\forall T' \forall C \exists X R'[X, C, T']$ where $T'$ is the new parameterised input signals.

| | | |
|---|---|---|
| | **1.** **Fresh** $C, T'$ | |
| *1* | **2.** **Let** $T = f(T')$ | **Where** $f$ **is the parametric encoding** |
| *2* | **3.** $J[C, T]$ | **Soundness of param** |
| | **4.** $\forall T \forall C (J[C, T] \rightarrow \exists X R[X, C, T])$ | **Premise** |
| *3, 4* | **5.** $\exists X R[X, C, T]$ | |
| *5* | **6.** **Fresh** $X*$ **s.t.** $R[X*, C, T]$ | |
| | **7.** $R' = R[T/f(T)]$ | **Definition of** $R'$ |
| *2, 6, 7* | **8.** $R'[X*, C, T']$ | |
| *1, 8* | **9.** $\forall T' \forall C \exists X R'[X, C, T']$ | **Since** $C, T'$ **were arbitrary** |

Figure 5: Proof of Coverage Condition Satisfaction

The symbolic simulation invariants together with the coverage condition are sufficient premises for our proof of the output checking procedure to hold.

However, this approach also destroys the partitioned structure of the partitioned indexing relations.

#### 4.2.1.1. Equivalence to Indexing Relation Restriction

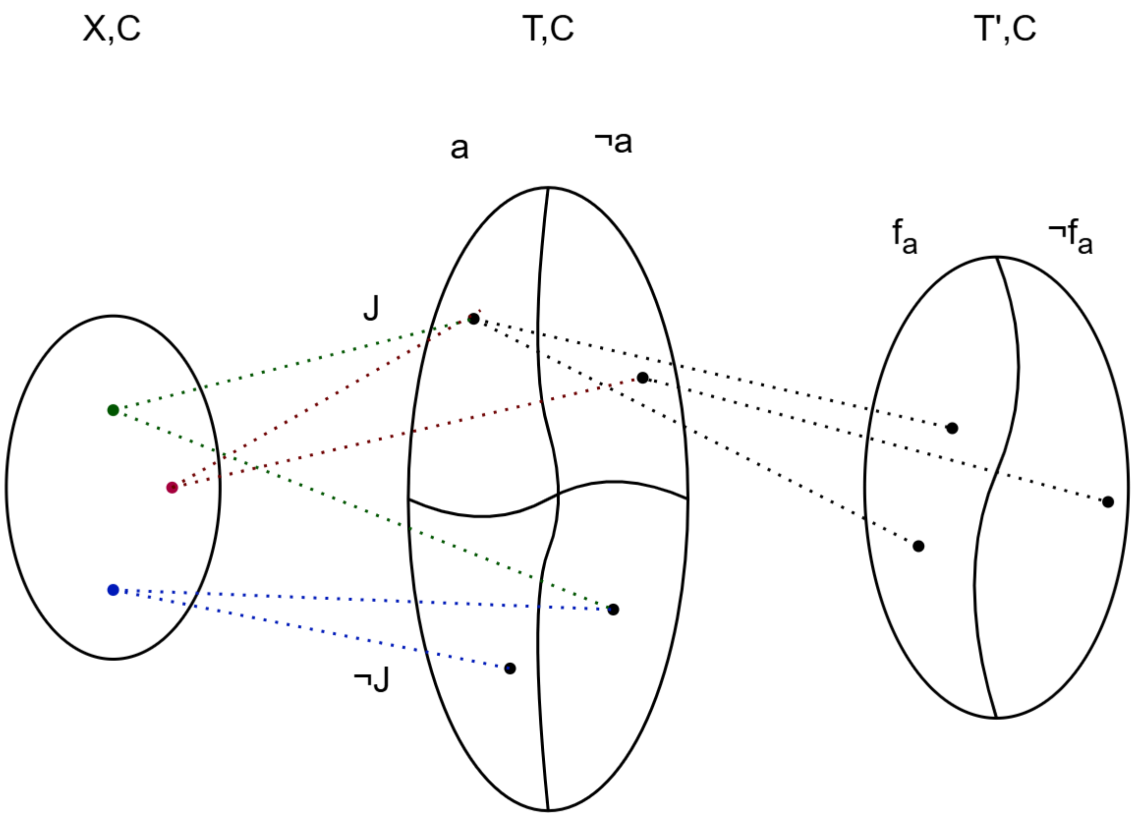

Figure 6: Illustration of Indexing Relation and Parameterisation

We see that the indexing cases that are included in the strong preimage operation on the antecedent are the same whether we are parameterising the indexing relation or restricting the indexing relation to the environmental constraint. Suppose we are taking the strong preimage of target variable $a$, we have that $a^R = (f_a)^{R'}$. In either case, we only consider the indexing cases that at least map to one target variable assignment that satisfies $J \wedge a$, and doesn't index any cases that satisfy $J \wedge \overline{a}$. Analysis is symmetric for taking the strong preimage of the negation of a target variable.

Furthermore,

$$\mathrm{dom}(R') = \{X, C \mid \exists T' R'[X, C, T']\} = \{X, C \mid \exists T (P[C, T] \wedge R[X, C, T])\} = J_R$$

Both of them are the set of cases that index into at least one case in $J$. As such the output checks will be the same for both methods.

This means that both methods are equivalent.

Since the indexing relation restriction is equivalent to the more efficient antecedent conditioning, this parametric method is also equivalent to that. Given the better efficiency of the method of conditioning the antecedent, that is a preferable method.

### 4.2.2. Parameterise Before Abstraction

While parameterising the indexing relation is not more effective than the restriction methods, our second strategy of using `param` before abstraction is more likely to be more effective.

Input: circuit, antecedent, input constraints, output constraints

1. Compute `param` on the input constraints
2. Viewing each parameterisation function as a circuit, bind the functions to the inputs of the original circuit to form a larger parameterised circuit.
3. Compute an indexing relation on the larger circuit, possibly via the automatic abstraction algorithm
4. Substitute each BDD variable in the unrestricted antecedent of the original circuit with the corresponding function from the parametric encoding
5. Perform indexing transformations on the parameterised antecedent, and output constraints
6. Perform symbolic simulation
7. Check whether the output constraint holds or a counterexample exists

Output: proven, disproven (with counterexample) or unproven

Figure 7: Model Checking with Parameterisation Before Abstraction

This approach will compute a new indexing relation that is not easily found as a modification of the non-parameterised indexing relation.

Soundness is proven similarly to Section 4.2.1.

# Chapter 5
# Experiments and Evaluation

## 5.1. Content-Addressable Memory (CAM)

We illustrate the effectiveness of symbolic simulation with symbolic indexing on a CAM.

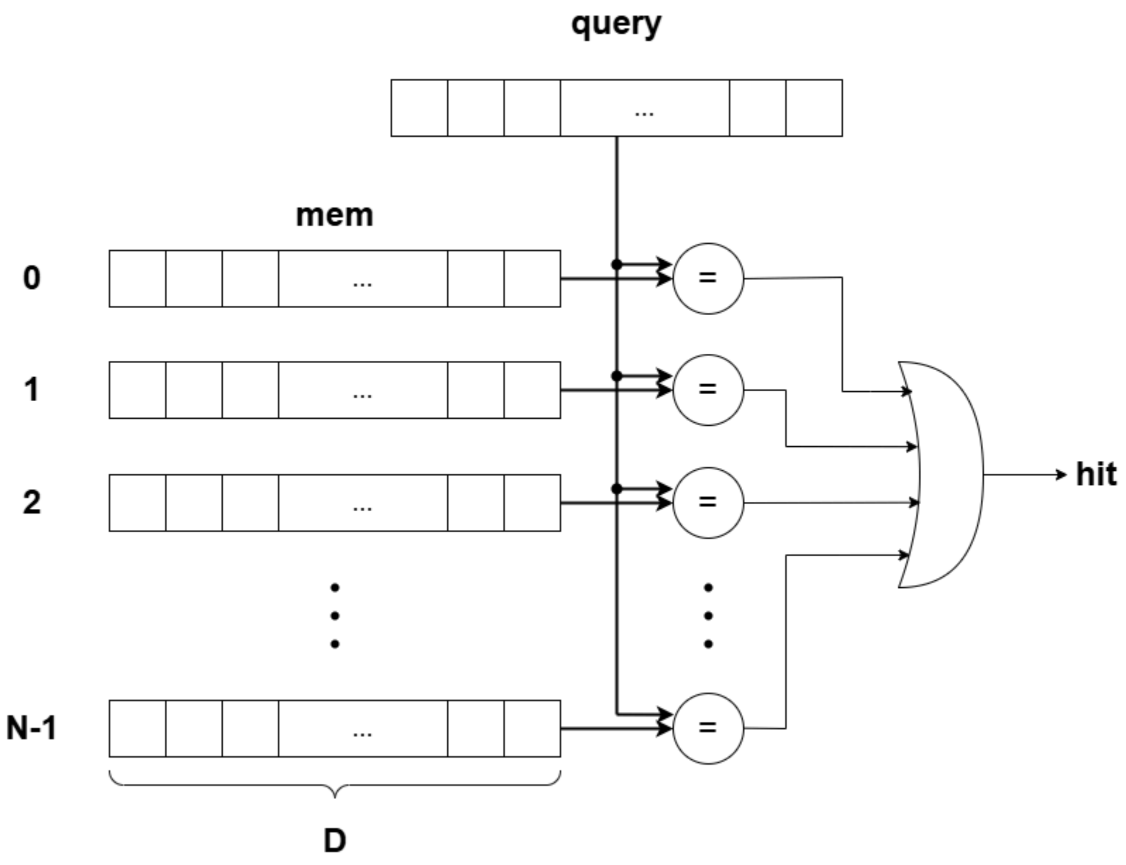

Figure 8: Content-Addressable Memory

The CAM stores *N* entries, each of *D* bits. It takes a query of *D* bits on each clock cycle and outputs on a `hit` whether the query matches any of its entries.

On each cycle our CAM will compare the query with each of the entries in parallel and take a logical OR of all the comparisons to get the `hit` value. The specification of our

CAM compares each entry with the query in sequence, doing many binary ORs in series to check for a hit.

Our property is that the specification and the circuit agree on the `hit` value.

### 5.1.1. Results

We prove the correctness of different sizes of the CAM with manual abstraction based on [6], automatic abstraction and no abstraction. We measure the time taken for the different steps of running rSTE as follows:

| **Timing** | **Description** |
|---|---|
| **Abstraction Time** | Time taken to produce a partitioned indexing relation. For manual indexing relations, this is the time taken to construct the relevant BDDs for the indexing relation. For automatic abstraction, this is the time taken to run the automatic abstraction algorithm.<br><br>Not included for unabstracted tests. |
| **Transformation Time** | Time taken to compute the domain and the strong preimage of each antecedent tuple. For these examples, the efficient preimage algorithms for partitioned indexing relations are used.<br><br>Not included for unabstracted tests. |
| **Evaluation Time** | Time taken to run the symbolic simulator with the transformed antecedent. |
| **Check Time** | Time taken to verify the output constraint is correct after the simulation. |

Table 3: Measured Times for rSTE Tests

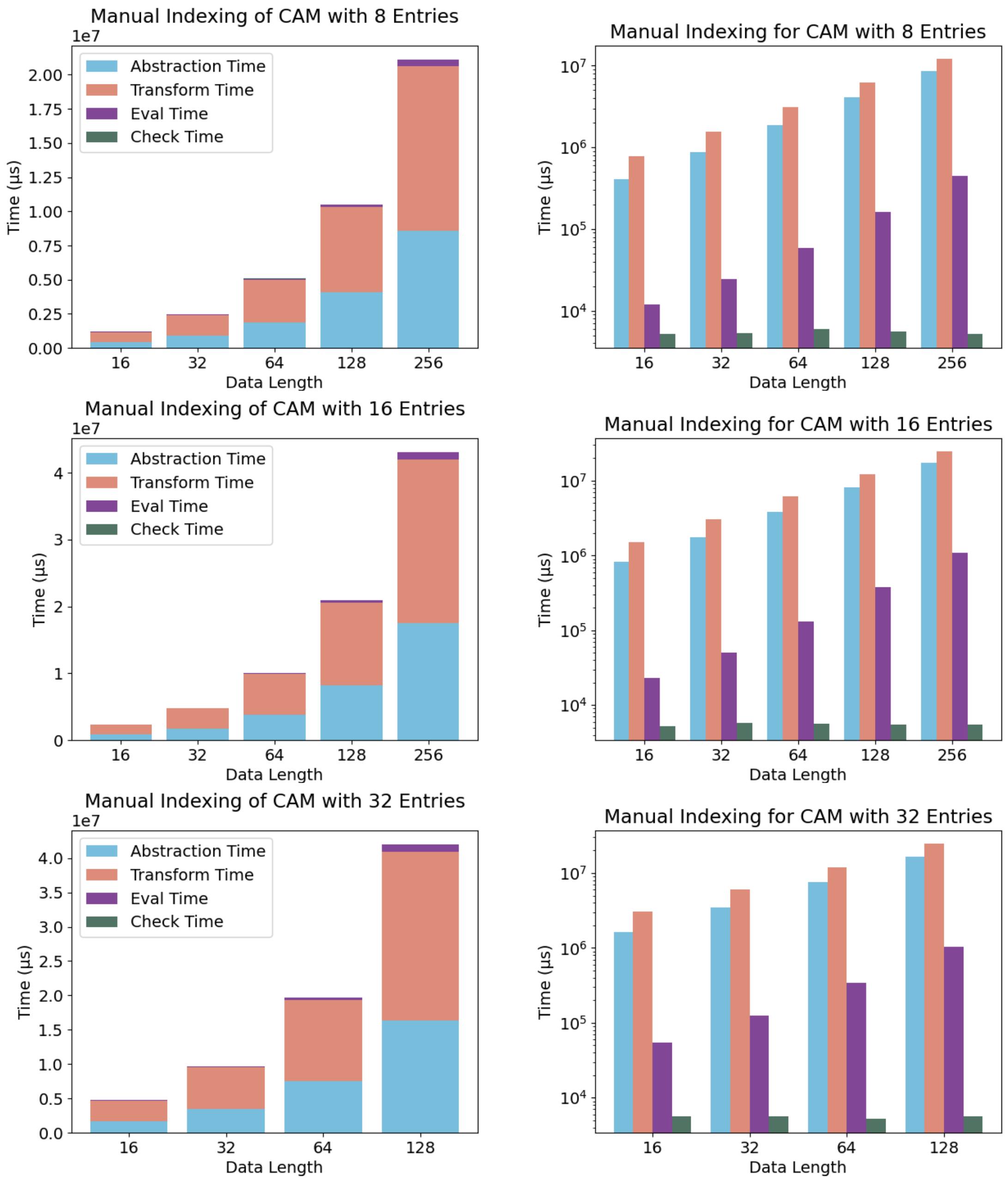

Figure 9: Symbolic Simulation on CAM with Manual Indexing

With manual abstraction, the time taken to perform each step of the proof grows linearly with the length of the data entries in the CAM, as is expected based of the indexing relation constructed.

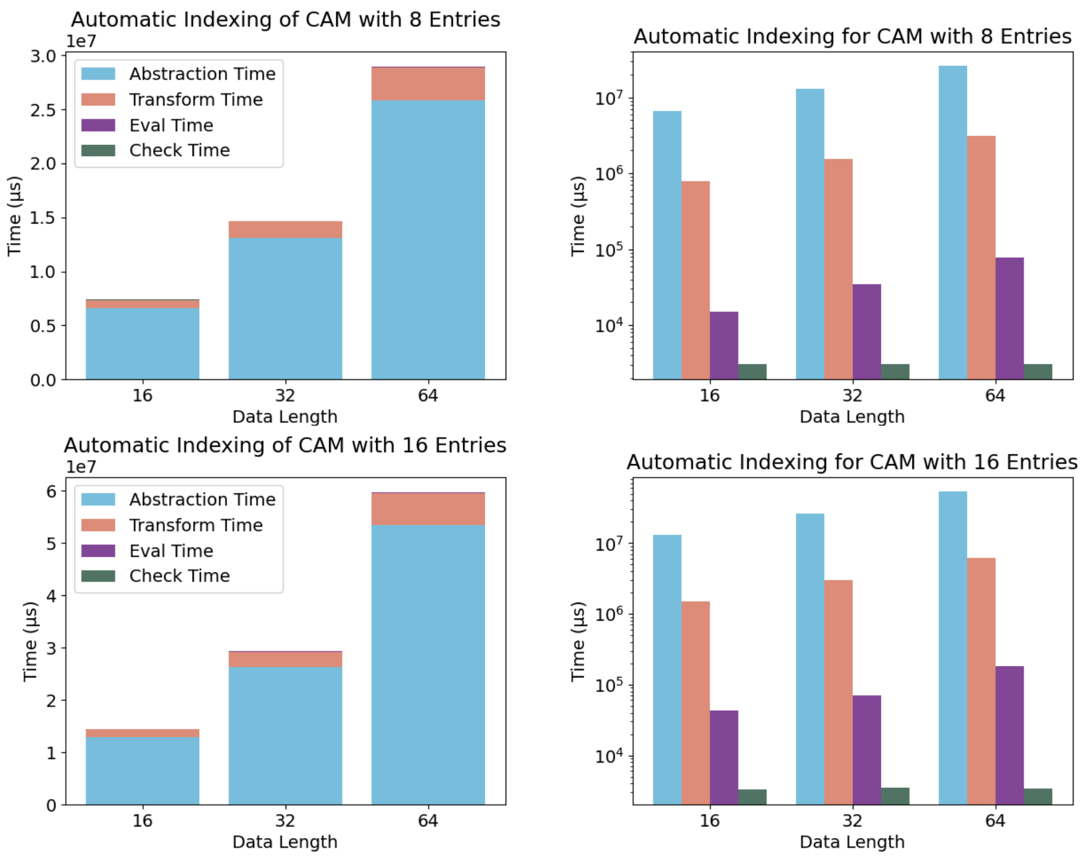

Figure 10: Symbolic Simulation on CAM with Automatic Abstraction

With automatic abstraction, we still see the linear growth in time taken to perform the proof, but the time taken is longer compared to manual abstraction. With the manually constructed indexing relation, the rate of increase in evaluation time is the fastest. With automatic abstraction, this is the abstraction time.

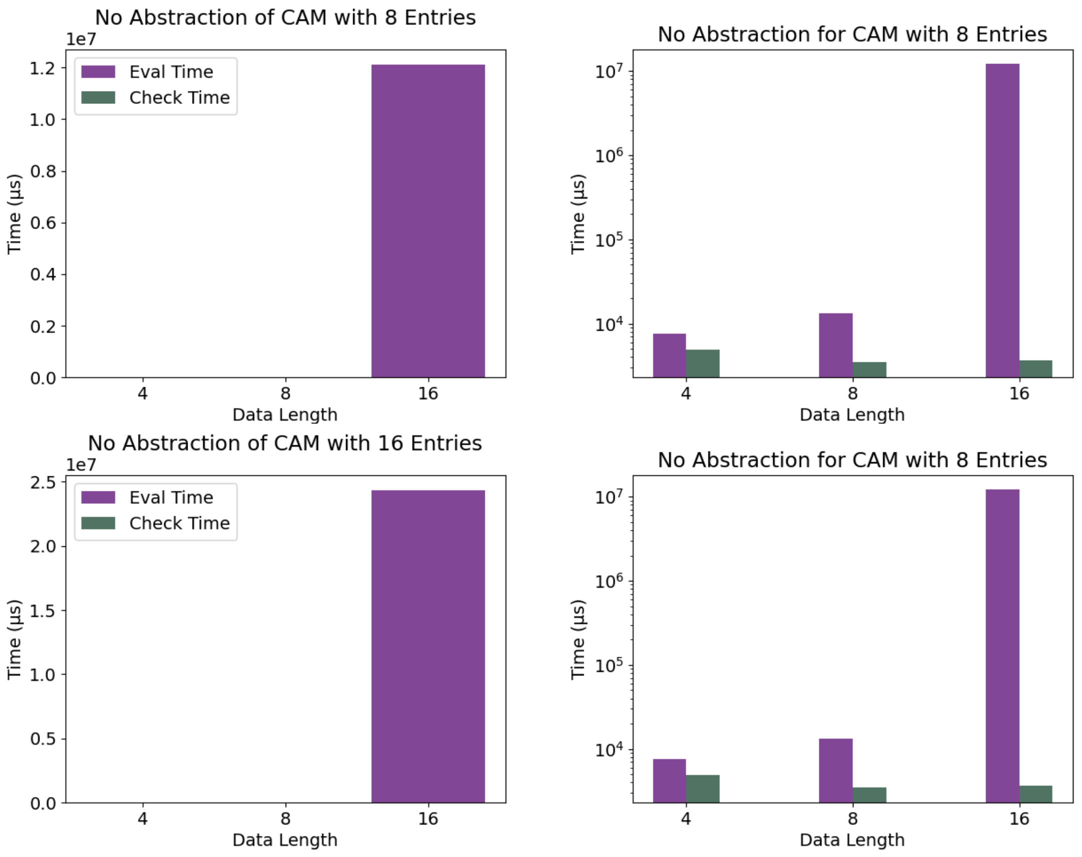

Figure 11: Symbolic Simulation on CAM with No Abstraction

Comparatively, when we run the symbolic simulation without abstraction, the time taken to perform the proof grows exponentially with the length of the data entries in the CAM and quickly becomes intractable.

## 5.2. Multi-Input Maximum Circuit

We also showcase the method on a maximum circuit. The maximum circuit takes $N$ inputs, each of length $D$, and every clock cycle, will output the maximum of all the inputs, treating them as binary integers. This is implemented as a binary tree of 2-input maximum operations between tree nodes, with the output at the root.

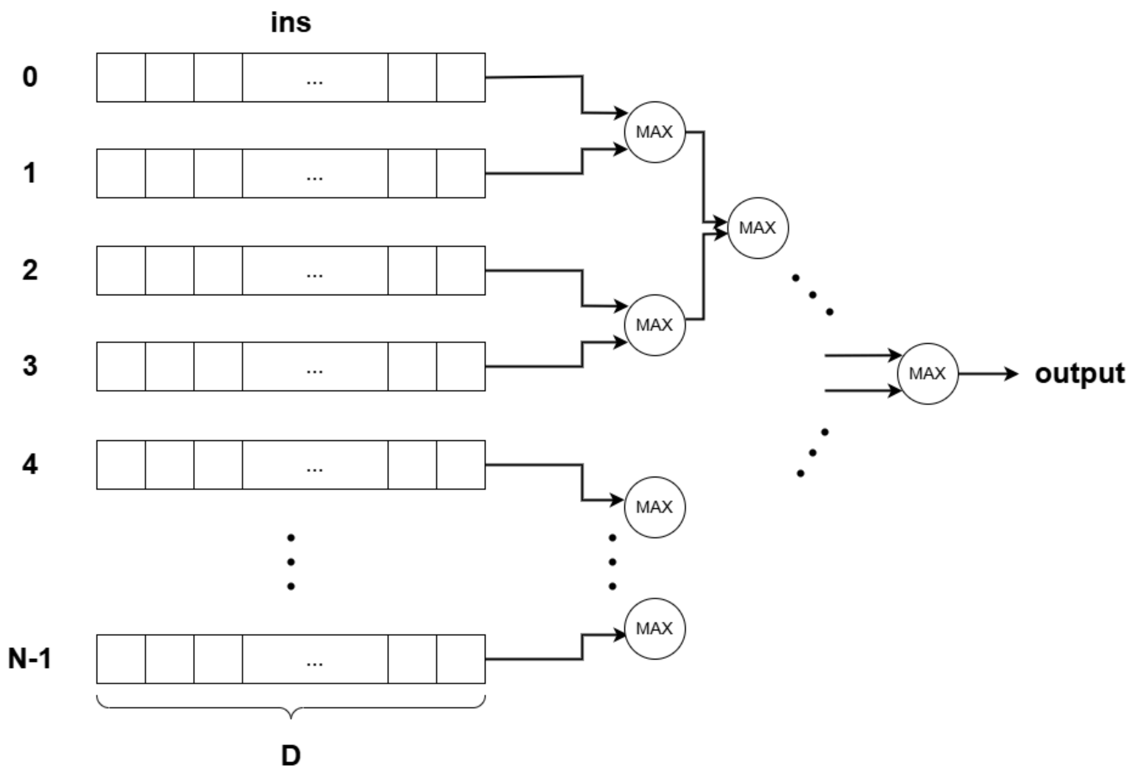

Figure 12: Maximum Circuit

Rather than using a specification that does the same computation in a different way, this circuit has a natural relational specification that consists of two properties:

- Contained Property: The output is one of the input values
- Bounded Property: The output is at least as large as all the input values

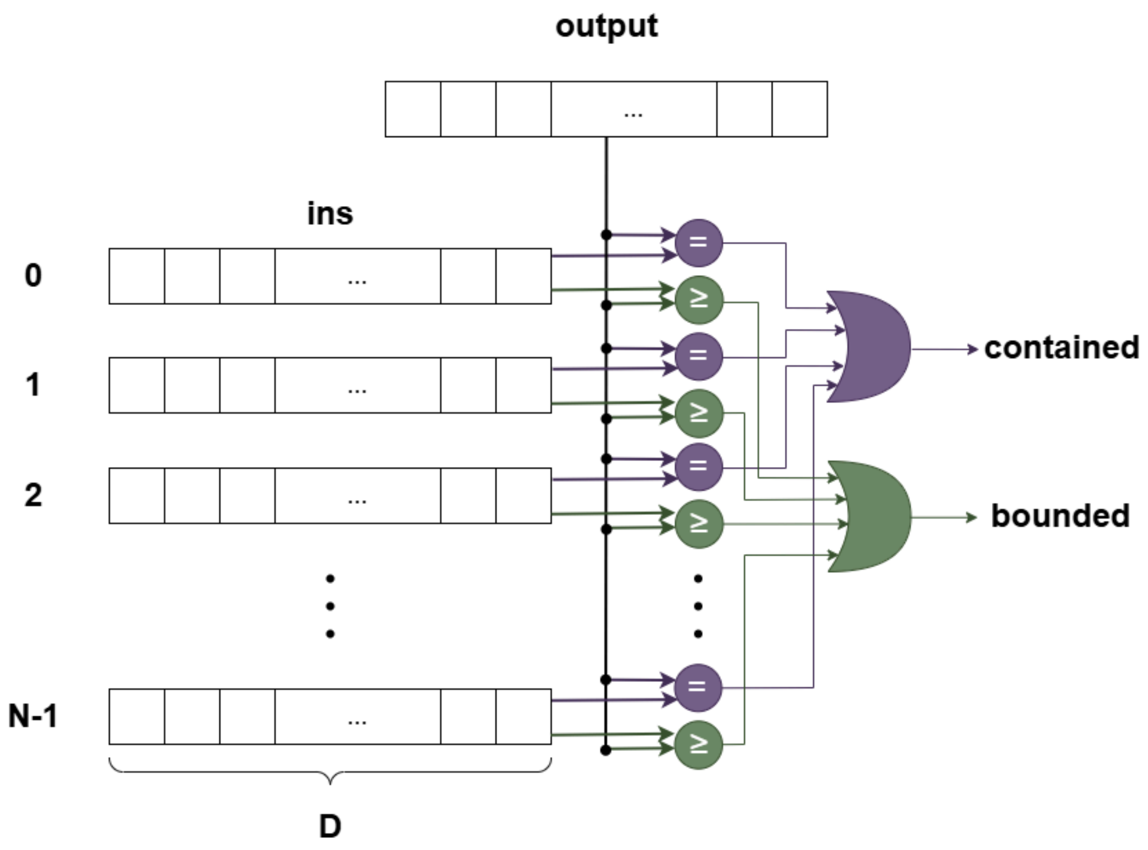

Figure 13: Maximum Circuit Specification Circuit

### 5.2.1. Manual Indexing Relation

We construct a partitioned indexing relation that exponentially reduces the number of BDD variables needed for symbolic simulation.

We have the following indexing variables:

| Variables | Type | Purpose |
|---|---|---|
| $\boldsymbol{t}$ | Vector of $D$ Boolean variables | $\boldsymbol{t}$ represents the target output of the maximum circuit. All cases that lead to the circuit producing $x$ will be covered by some indexing cases where $\boldsymbol{t} = x$ |
| $\boldsymbol{d}$ | Matrix of size $N$ by $\log_2 D + 1$ Boolean variables | $\boldsymbol{d}_i : \boldsymbol{d}_i[0], ..., \boldsymbol{d}_i[\log_2 D]$ encodes an integer that represents the number of most significant bits of the $i$th input will be the same as the corresponding bits of $\boldsymbol{t}$ before the critical bit of $i$ which will be low in $\mathbf{ins}[i]$ but high in $\boldsymbol{t}$ |

Table 4: Variables for Max Circuit Manual Indexing

These form conjuncts that we merge together to form the indexing relation:

| **Conjunct** | **Purpose** |
|---|---|
| $\bigvee_{i=0}^{n-1} (\boldsymbol{d}_i = D)$ | Ensures that at least one entry will completely match the target output |
| $\forall i \forall j : (D - j \leq \boldsymbol{d}_i) \Rightarrow (\textbf{ins}[i][j] = \boldsymbol{t}[j])$ | Ensures the most significant bits of each input match with the target output |
| $\forall i \forall j : (D - 1 - j = \boldsymbol{d}_i) \Rightarrow (\textbf{ins}[i][j] = 0) \wedge (t[j] = 1)$ | Ensures the critical bit for each entry is high in the target output but low in the input |

Table 5: Conjuncts for Max Circuit Manual Indexing

where $0 \leq i < N$, $0 \leq j < D$.

Abstraction is achieved by the lack of restriction of the bits less significant than the critical bits in each input since their value will not affect the output.

This takes $D + N \log_2 D + N$ variables, exponentially less than the $ND$ variables used for symbolic simulation without symbolic indexing.

### 5.2.2. Results

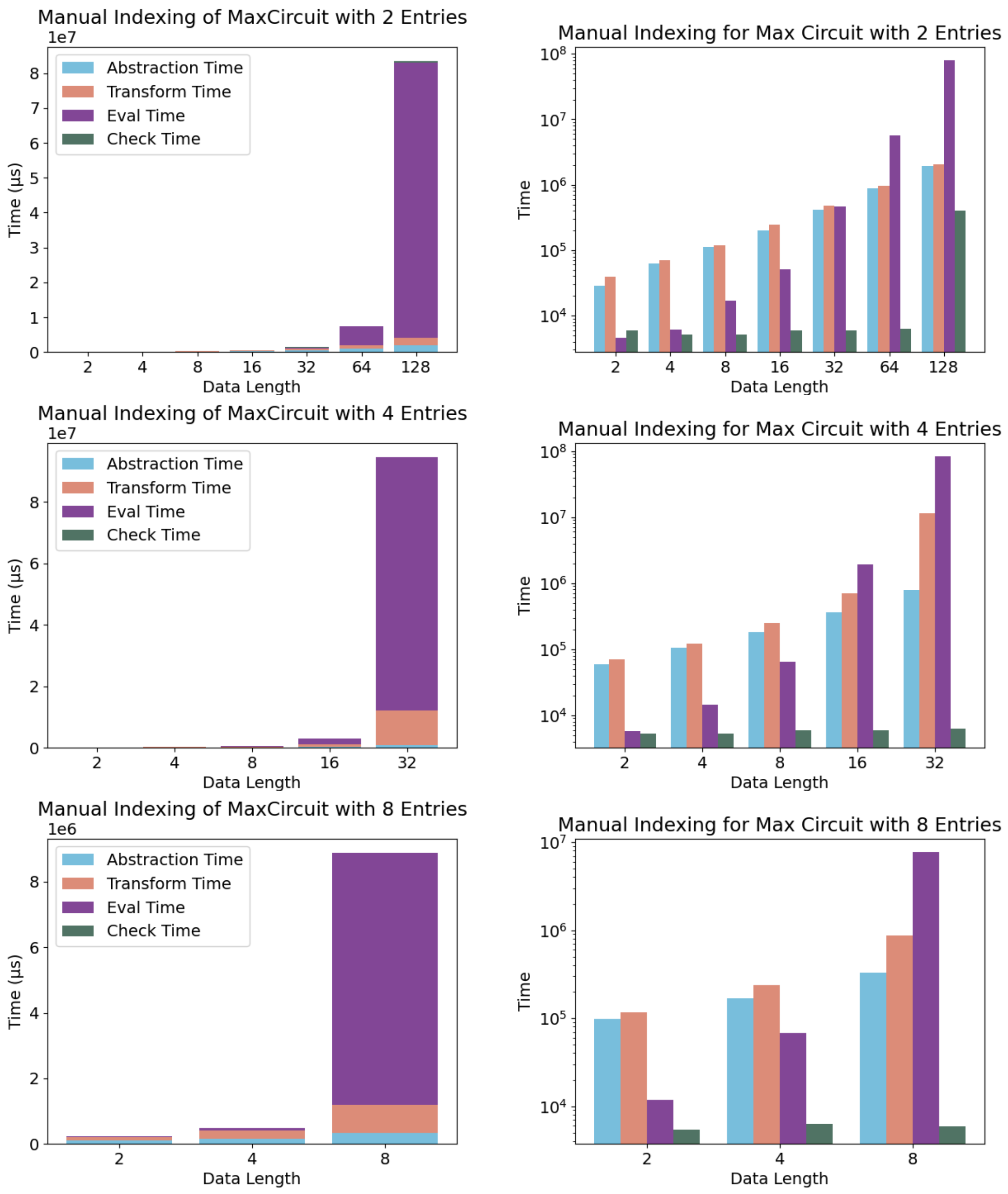

Figure 14: Symbolic Simulation of Max Circuit with Manual Indexing

With manual indexing, the abstraction time mostly grows linearly with size of the circuit. However, the transformation time grows super-linearly, probably due to the larger BDDs that must be manipulated, particularly the domain of the abstraction relation.

The evaluation time grows super-linearly at an even faster rate. This seems to be because, compared to the CAM, the maximum circuit has its entire comparison tree take $O(D)$ input and $O(D)$ output rather than comparing single wires.

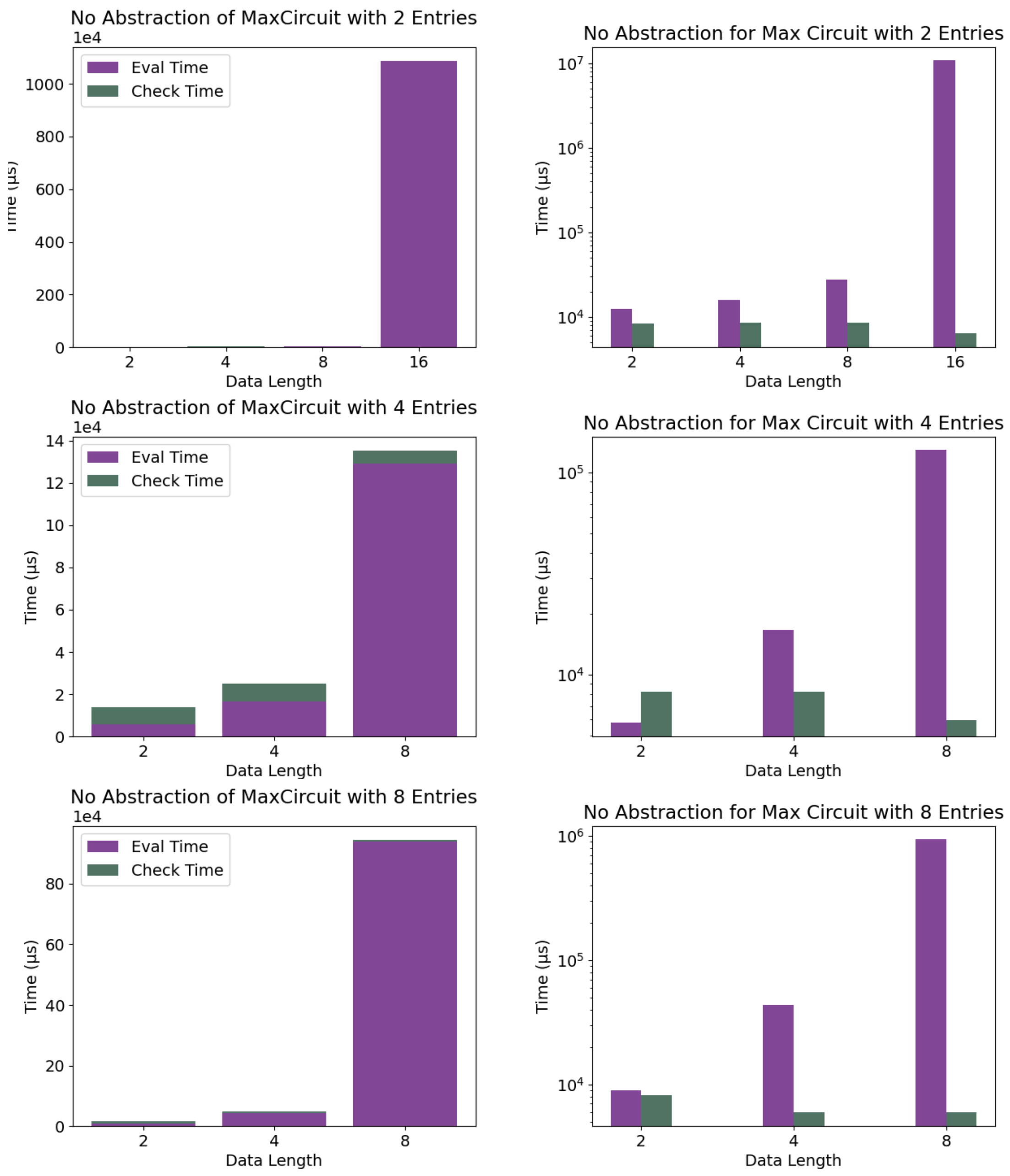

Figure 15: Symbolic Simulation of Max Circuit without Abstraction

Super-linear scaling is also seen in the no abstraction case, but the rate in increase is much faster, resulting in the proof quickly becoming infeasible.

Unfortunately we are not able to prove either of the max circuit properties with the automatic abstraction algorithm. This is likely due to over-abstraction. Unlike for the CAM, here there are no obvious candidates for symbolic constants, which was our primary defence against over-abstraction.

# Chapter 6
# Conclusion

We have reinvented the theory of symbolic indexing transformations for rSTE and implemented it in Jasper. We have shown that the method can be more effective than running symbolic simulation without symbolic indexing.

## 6.1. Future Work

Automatic abstraction is critical to be able to apply rSTE widely. The speed of the automatic abstraction algorithm developed in the companion project is currently a limiting factor in efficient rSTE proofs. Furthermore, over-abstraction has proven to be a problem. Thus, work on counterexample guided abstraction refinement [17] of the automatically discovered indexing relations such as in [18] would make rSTE much more practical.

While we described methods to deal with environmental constraints, we have yet to implement or evaluate these methods. This would be a good area of further work.

STE and rSTE work on the level of individual bits which could limit the level of abstraction that can be achieved. Existing work has been done to extend STE to work on the level of data values [19], [20]. An area of further work is to extend indexing transformations and automatic abstraction to work on these variants.

# Bibliography

[1] C.-j. Seger and R. Bryant, “Formal Verification by Symbolic Evaluation of Partially-Ordered Trajectories,” *Formal Methods in System Design*, vol. 6, p. , 1994, doi: 10.1007/BF01383966.

[2] S. Hazelhurst and C.-J. H. Seger, “Symbolic trajectory evaluation,” in *Formal Hardware Verification: Methods and Systems in Comparison*, T. Kropf, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 3–78. doi: 10.1007/3-540-63475-4_1.

[3] T. Melham, “Symbolic Trajectory Evaluation,” in *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds., Cham: Springer International Publishing, 2018, pp. 831–870. doi: 10.1007/978-3-319-10575-8_25.

[4] R. E. Bryant, “Symbolic simulation—techniques and applications,” in *Proceedings of the 27th ACM/IEEE Design Automation Conference*, in DAC '90. Orlando, Florida, USA: Association for Computing Machinery, 1991, pp. 517–521. doi: 10.1145/123186.128296.

[5] M. Pandey and R. Bryant, “Formal verification of memory arrays using symbolic trajectory evaluation,” in *Proceedings. International Workshop on Memory Technology, Design and Testing (Cat. NO.97TB100159)*, 1997, pp. 42–49. doi: 10.1109/MTDT.1997.619393.

[6] M. Pandey, R. Raimi, R. Bryant, and M. Abadir, “Formal Verification Of Content Addressable Memories Using Symbolic Trajectory Evaluation,” in *Proceedings of the 34th Design Automation Conference*, 1997, pp. 167–172. doi: 10.1109/DAC.1997.597138.

[7] R. E. Bryant and C.-J. H. Seger, “Formal Verification of Digital Circuits Using Symbolic Ternary System Models,” in *Proceedings of the 2nd International Workshop on Computer Aided Verification*, in CAV '90. Berlin, Heidelberg: Springer-Verlag, 1990, pp. 33–43.

[8] R. E. Bryant, “Symbolic Boolean manipulation with ordered binary-decision diagrams,” *ACM Comput. Surv.*, vol. 24, no. 3, pp. 293–318, Sept. 1992, doi: 10.1145/136035.136043.

[9] T. F. Melham and R. B. Jones, “Abstraction by Symbolic Indexing Transformations,” in *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design*, in FMCAD '02. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 1–18.

[10] S. Adams, M. Bjork, T. Melham, and C.-J. Seger, “Automatic Abstraction in Symbolic Trajectory Evaluation,” in *Formal Methods in Computer Aided Design (FMCAD'07)*, 2007, pp. 127–135. doi: 10.1109/FAMCAD.2007.27.

[11] C.-J. Seger *et al.*, “An industrially effective environment for formal hardware verification,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 9, pp. 1381–1405, 2005, doi: 10.1109/TCAD.2005.850814.

[12] C.-J. H. Seger, “VOSS - A Formal Hardware Verification System User”s Guide,” 1993. [Online]. Available: https://api.semanticscholar.org/CorpusID:61246413

[13] J. O'Leary, R. Kaivola, and T. Melham, “Relational STE and theorem proving for formal verification of industrial circuit designs,” in *2013 Formal Methods in Computer-Aided Design*, 2013, pp. 97–104. doi: 10.1109/FMCAD.2013.6679397.

[14] “Jasper RTL Apps.” [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-verification-platform.html

[15] “Voss II.” [Online]. Available: https://github.com/TeamVoss/VossII

[16] M. D. Aagaard, R. B. Jones, and C.-J. H. Serger, “Formal verification using parametric representations of Boolean constraints,” in *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, in DAC '99. New Orleans, Louisiana, USA: Association for Computing Machinery, 1999, pp. 402–407. doi: 10.1145/309847.309968.

[17] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, “Counterexample-Guided Abstraction Refinement ,” in *Computer Aided Verification*, E. A. Emerson and A. P. Sistla, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 154–169.

[18] S. E. Adams, “Abstraction discovery and refinement for model checking by symbolic trajectory evaluation,” Doctoral dissertation, 2014. [Online]. Available: https://www.cs.ox.ac.uk/people/tom.melham/phd/Adams-2013-ADR.pdf

[19] S. Chakraborty *et al.*, “Word-level Symbolic Trajectory Evaluation.” [Online]. Available: https://arxiv.org/abs/1505.07916

[20] D. Li, O. Ait-Mohamed, and S. Abed, “Towards First-Order Symbolic Trajectory Evaluation,” in *37th International Symposium on Multiple-Valued Logic (ISMVL'07)*, 2007, p. 53. doi: 10.1109/ISMVL.2007.57.