SAT-Based Assistance in Abstraction Refinement for Symbolic Trajectory Evaluation

Jan-Willem Roorda and Koen Claessen

Chalmers University of Technology, Sweden
{jwr, koen}@chalmers.se

Abstract. We present a SAT-based algorithm for assisting users of Symbolic Trajectory Evaluation (STE) in manual abstraction refinement. As a case study, we demonstrate the usefulness of the algorithm by showing how to refine and verify an STE specification of a CAM.

1 Introduction

Symbolic Trajectory Evaluation (STE) [12] is a well-known simulation-based model checking technique. It combines three-valued simulation (using the standard values 0 and 1 together with the extra value X, "unknown") with symbolic simulation (using symbolic expressions to drive inputs). STE has been extremely successful in verifying properties of circuits containing large data paths (such as memories, fifos, floating point units) that are beyond the reach of traditional symbolic model checking [1,11,7].

In STE, specifications are *assertions* of the form $A \Longrightarrow C$, where A is called the *antecedent* and C the *consequent*. Both A and C are formulas in a restrictive temporal logic, in which only statements about a finite number of time points can be made. The only variables in the logic are time-independent Boolean variables, called *symbolic variables*.

The power of STE comes from the use of *abstraction*. The abstraction is induced by the antecedent of the assertion; when the antecedent does not specify a value for a certain node, the value of the node is abstracted away by using the unknown value X. Thus, the antecedent plays two different roles in STE; it is the logical antecedent as well as a specification of what abstraction should be used in the verification. Because of the abstraction, the values of circuit nodes during simulation can be represented by BDDs in terms of the symbolic variables occurring in the assertion, providing an efficient means of checking an STE assertion.

A drawback of STE is that the user needs to spend time on finding the *right* abstraction. Often, just the right mix between symbolic variables and X's has to be used to make sure that the property holds in the abstraction induced, and the BDDs used in the verification do not blow up.

Abstraction Refinement. A common initial result in an STE verification attempt is that the model-checker cannot prove the assertion because the simulation using the antecedent yields X's at nodes that are required to have a particular Boolean value by

T. Ball and R.B. Jones (Eds.): CAV 2006, LNCS 4144, pp. 175–189, 2006.

the consequent. This indicates that the used abstraction was too coarse, leading to a so-called *spurious counter-model*. In contrast, a *real counter-model* is a simulation run that satisfies the antecedent but yields a 0 for a node for which the consequent requires a 1, or vice-versa. A *model* of an assertion is a simulation run that satisfies both the antecedent and the consequent.

When an STE model-checking run produces spurious counter-models but no real counter-models, we say that the result of the verification is *unknown*. In this case, the assertion must be refined (usually by introducing more symbolic variables in the antecedent) until the property is proved, or until a real counter-model is found. Often, a great deal of time is spent on such manual *abstraction refinement* [14,2].

Contribution. We have invented the concept of a *strengthening*, which is a particular piece of useful information that can help STE-users with manual abstraction refinement; given an STE assertion and a circuit, a strengthening indicates which extra inputs of the circuit need to be given a Boolean (non-X) value in order for relevant outputs to also get a Boolean value. We have also designed a SAT-based algorithm that calculates strengthenings, which we have implemented in a tool called STAR (**S**AT-based **T**ool for **A**bstraction **R**efinement in STE). STAR has two modes; the first mode calculates strengthenings that satisfy the assertion (corresponding to models), and the second mode calculates strengthenings that contradict the assertion (corresponding to real counter-models).

By inspecting a weakest satisfying strengthening, the user can gain intuition about how to refine the assertion by introducing a *minimal* number of extra symbolic variables. On the other hand, a weakest contradicting strengthening gives a *minimal* set of reasons for the failure of the assertion, which can be used to gain intuition about why the circuit does not satisfy the assertion. In the next section, we look at examples of satisfying and contradicting strengthenings in more detail.

Related Work. There exists a large body of work in the field of automatic abstraction refinement for model-checking techniques for hardware other than STE, for an overview see for example [5]. Most of these abstractions are state-based, focusing on how to represent the state space of a circuit, which is not applicable to STE. In [6] an algorithm providing an easy interface to abstraction in STE is described. The algorithm does, however, not help in finding a right abstraction.

In another paper [13] presented at this conference, the tool *AutoSTE* is described. This tool can automatically refine STE assertions that result in a spurious counter-model, until either the assertion is proved, a real counter-model is found (or resources are exhausted). We believe that STAR and AutoSTE are complementary, in the following sense. AutoSTE can automatically find certain refinements of a specific kind (namely where some nodes become driven by fresh symbolic variables under certain conditions). STAR assists the user in manually finding refinements of a much more general kind, for example when sophisticated symbolic indexing schemes [6,7] are needed. We show, for instance, in the next section, how the method can be used to derive a symbolic indexing scheme for the verification of Content-Addressable Memories.

2 A Case Study

Content-Addressable Memories (CAMs) are hardware implementations of lookup tables. A CAM stores a number of *tags*, each of which is linked to a specific *data-entry*. The basis of a CAM circuit consists usually of two memory blocks, one containing tag entries, and the other the same number of corresponding data entries, see Fig. 1. Given an input tag, the associative-read operation consists of searching all tags in the CAM to determine if there is a match to the input tag, and if so sending the associated data-entry to the output. Verifying this operation is non-trivial [7].

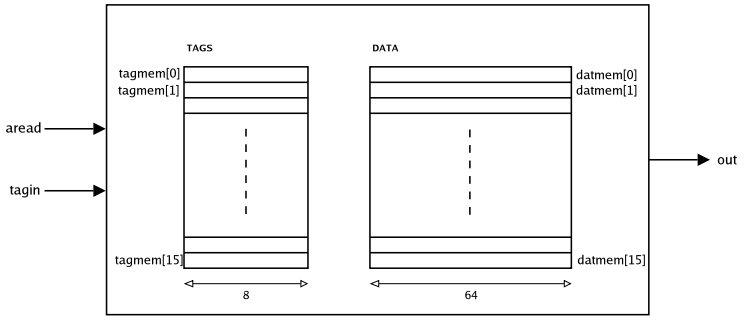

Fig. 1. A Content-Addressable Memory Circuit

What follows is a constructed, but realistic, account of how a verification engineer might use our method to derive an STE assertion for verifying the associative-read operation of a CAM. How to verify CAMs using STE is now well-known [7]. The STE assertion needed for the verification is however quite complex. We show how a user who is ignorant of the above mentioned work on CAM verification can derive the required assertion with help of the STAR-tool. We believe that this convincingly illustrates the usefulness of our method.

In the case-study, we assume that the verification engineer uses the BDD-based STE model-checker in Intel's verification toolkit Forte[4]. The CAM under verification is taken from Intel's GSTE tutorial.

An obvious way of verifying the associative-read operation using STE is to introduce symbolic variables for each tag- and data-entry. When doing so, the antecedent of the assertion specifies that each tag-entry $\mathsf{tagmem}[i]$ has symbolic value $tagmem_i$, and each data-entry $\mathsf{datmem}[i]$ has symbolic value $datmem_i$. The consequent checks that, for each i, when the input-tag is equal to $tagmem_i$ the output is equal to $datmem_i$.

$$\begin{array}{l} \quad (\mathsf{aread}\ \mathbf{is}\ 1)\ \ \mathbf{and}\ \ (\mathsf{tagin}\ \mathbf{is}\ tagin) \\ \mathbf{and}\ \ (\mathsf{tagmem}[0]\ \mathbf{is}\ tagmem_0)\ \ \mathbf{and}\ \ \ldots\ \ \mathbf{and}\ \ (\mathsf{tagmem}[15]\ \mathbf{is}\ tagmem_{15}) \\ \mathbf{and}\ \ (\mathsf{datmem}[0]\ \mathbf{is}\ datmem_0)\ \ \mathbf{and}\ \ \ldots\ \ \mathbf{and}\ \ (\mathsf{datmem}[15]\ \mathbf{is}\ datmem_{15}) \\ \qquad\qquad \Longrightarrow \\ \qquad\qquad ((tagin = tagmem_0)\ \ \rightarrow (\mathsf{out}\ \mathbf{is}\ tagmem_0)) \\ \qquad\qquad \vdots \\ \mathbf{and}\ \ ((tagin = tagmem_{15}) \rightarrow (\mathsf{out}\ \mathbf{is}\ tagmem_{15})) \end{array} \tag{1}$$

```
Warning: Consequent failure at time 0 on node out[63]
Current value:data[63] + X(!data[63])
Expected value:data[63]
Weak disagreement when:!data[63]
----WARNING: Some consequent errors not reported

data[16]&data[21]&data[61]&data[34]&data[2]&data[7]&data[47]&data[52]&data[20]&
data[60]&data[33]&data[38]&data[6]&data[46]&data[51]&data[19]&data[59]&data[56]&
data[24]&data[32]&data[29]&data[37]&data[5]&data[45]&data[13]&data[42]&data[53]&
data[10]&data[50]&data[18]&data[58]&data[15]&data[26]&data[55]&data[23]&data[63]&
data[31]&data[28]&data[39]&data[36]&data[43]&data[44]&data[40]&data[12]&data[41]&
data[27]&data[49]&data[17]&data[57]&data[14]&data[25]&data[54]&data[22]&data[62]&
data[30]&data[9]&data[35]&data[3]&data[4]&data[0]&data[11]&data[1]&data[8]&data[48]
```

Fig. 2. Forte Output for Assertion 2

This assertion, however, cannot be handled by a BDD-based STE-model checker. The large number of symbolic variables leads to an immediate BDD-blow up.

Suppose that, instead, the user tries to verify the operation by using symbolic indexing [6]. When doing so, a vector of symbolic variables, $index$, is created to index over the potentially matching tag-entries. The antecedent states that the indexed tag-entry has symbolic value $tagin$ and the indexed data-entry has value $data$. So, only variables for the content of the indexed data-entry and tag-entry are created, instead of variables for all tag- and data-entries. This greatly reduces the number of required symbolic variables. Using symbolic indexing, the user could arrive at the following assertion.

$$\begin{array}{ll} & (\mathsf{aread}\ \mathbf{is}\ 1)\ \ \mathbf{and}\ \ (\mathsf{tagin}\ \mathbf{is}\ tagin) \\ \mathbf{and} & ((index = 0000) \rightarrow ((\mathsf{tagmem}[0]\ \mathbf{is}\ tagin)\ \ \mathbf{and}\ \ (\mathsf{datmem}[0]\ \mathbf{is}\ data))) \\ \mathbf{and} & ((index = 0001) \rightarrow ((\mathsf{tagmem}[1]\ \mathbf{is}\ tagin)\ \ \mathbf{and}\ \ (\mathsf{datmem}[1]\ \mathbf{is}\ data))) \\ \vdots & \vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ \mathbf{and} & ((index = 1111) \rightarrow ((\mathsf{tagmem}[15]\ \mathbf{is}\ tagin)\ \ \mathbf{and}\ \ (\mathsf{datmem}[15]\ \mathbf{is}\ data))) \\ & \qquad\qquad\qquad\qquad \Longrightarrow \\ & \qquad\qquad\qquad\qquad \mathsf{out}\ \mathbf{is}\ data \end{array} \tag{2}$$

When the user tries to verify this assertion with the model-checker, the result is "unknown". The output of the model-checker is given in Fig. 2: the simulated value for node $\mathsf{out}[63]$ is $(data[63] + (\mathsf{X}\ \&\ \neg data[63]))$, while the required value is $data[63]$. When the symbolic variable $data[63]$ has value 0, the simulated value of $\mathsf{out}[63]$ evaluates to X, indicating a spurious counter-model. The expression $data[16]\&....\&\,data[48]$ indicates that only when the data-entry consists of only high bits no spurious counter-model exists. So, the STE model-checker does not give much help with refining the assertion. This is where our tool STAR comes in.

STAR can be used to calculate a weakest contradicting strengthening of Assertion 2, see Fig. 3. The table presents an assignment of the symbolic variables, and a weakest strengthening of the antecedent that together contradict the consequent. Here, only boldfaced values (**0** or **1**) in the table represent strengthened nodes. A normal-faced 0 or 1 represents a node that has received the value 0 or 1 because it was required by the (original) antecedent. For instance, $\mathsf{tagmem}[12]$ is required to have value 00000000 by the antecedent, but $\mathsf{tagmem}[1]$ is required to have the same value by the strengthening. To increase readability, X's are represented by a dash –; entries for which all values are X have been left out of the table completely. The table states that

Symbolic Variables	
$index$	= 1100
$tagin$	= 00000000
$data$	= 11111111111111111111111111111111111111111111111111111111111111100
Inputs at time 0	
aread	= 1
tagin	= 00000000
Initial Values	
tagmem[1]	= **00000000**
tagmem[12]	= 00000000
datmem[1]	= --**1**-
datmem[12]	= 11111111111111111111111111111111111111111111111111111111111111100
Outputs	
out	= 111111111111111111111111111111111111111111111111111111111111111-

Fig. 3. A Weakest Contradicting Strengthening of Assertion (2)

- the value of $index$ vector is 1100, so, tag- and data-entry 12 are indexed,
- not only the indexed tag 12 is equal to the input tag $tagin$ but also tag 1,
- data-entry 1 differs from the indexed data-entry 12, at the second-last position; data-entry 1 has value 1 at this position, while the indexed data-entry has value 0,
- the value of the output of the CAM at the second-last position is 1 instead of 0 as required by the consequent.

From this, the user can deduce that the assertion in fact does not hold for the circuit because the assertion does not consider the case in which two tag-entries are equal to the input tag. Also, the user can conclude that, apparently, the CAM contains a *bus* that, when given both a 0 and 1 value, chooses the 1 value over the 0 value.

An obvious way of circumventing this problem is to introduce symbolic variables for all tag-entries, and to add the constraint that there is at most one tag-entry equal to the input tag. To do so, many extra symbolic variables are needed; one for each bit of each tag-entry. Therefore, it is not surprising that the resulting assertion yields, again, a BDD blow-up.

To obtain an intuition on how to, instead, refine the assertion by introducing a very small number of extra symbolic variables, the user can calculate a weakest satisfying strengthening of the assertion. The user knows from the output of the model-checker that when all of the data-entries have value 1 no spurious counter-model exists. Therefore, the constraint that at least one of the data-entries has value 0 is given to STAR as well.

In Fig. 4, a weakest satisfying strengthening calculated by STAR is given. In this strengthening, for each non-indexed tag-entry *either* (1) the tag-entry differs at one position from the input tag, *or* (2) the tag-entry consist only of X′s (tag-entries 5 and 11), and the corresponding data-entry contains a zero at the position where the indexed data-entry has a zero, and X's at each of the positions where the indexed data-entry contains a 1.

This can be explained as follows. There are two ways of making sure that a non-indexed data-entry does not corrupt the output: (1) making the tag-entry differ at at-least one position from the input tag, or (2) as the bus in the CAM favors a 1 over a 0, for each tag that potentially matches, having a 0 in the data-entry at each position where the indexed data-entry contains a 0.

Symbolic Variables	
$index$	$= 0000$
$tagin$	$= 00000010$
$data$	$= 01000000000000000000000000000000000000000000000000000000000000000000$
Inputs at time 0	
aread	$= 1$
tagin	$= 00000010$
Initial Values	
tagmem[0]	$=$ 00000010
tagmem[1]	$=$ **1**-------
tagmem[2]	$=$ **1**-------
tagmem[3]	$=$ --**1**-----
tagmem[4]	$=$ ------**0**-
tagmem[5]	$=$ --------
tagmem[6]	$=$ -**1**------
tagmem[7]	$=$ --**1**-----
tagmem[8]	$=$ -**1**------
tagmem[9]	$=$ **1**-------
tagmem[10]	$=$ **1**-------
tagmem[11]	$=$ --------
tagmem[12]	$=$ ---**1**----
tagmem[13]	$=$ ----**1**---
tagmem[14]	$=$ ------**0**-
tagmem[15]	$=$ -**1**------
datmem[0]	$=$ 01000000000000000000000000000000000000000000000000000000000000000000
datmem[5]	$=$ **0**-**00000000000000000000000000000000000000000000000000000000000000000000**
datmem[11]	$=$ **0**-**00000000000000000000000000000000000000000000000000000000000000000000**

Fig. 4. A Weakest Satisfying Strengthening of Assertion (2)

As for the verification of the associative read property, no assumptions on the content of the data-entries in the CAM are wanted, the user can ask STAR to generate a weakest satisfying strengthening of Assertion (2) that does not strengthen the requirements on the values of data-entries. This strengthening, given in Fig. 5, makes each non-indexed tag-entry differ at one position from the input tag.

Inspired by this strengthening, the user can modify the assertion by introducing, for each tag-entry i, a vector of symbolic variables p_i that specifies at which position the tag-entry differs from the input tag when the tag-entry is not indexed. The formula expressing that tag i differs from the input tag $tagin$ at the position encoded by p_i is:

$$\begin{aligned}\mathrm{mismatch}(i) = \quad & ((p_i = 000) \rightarrow (\mathsf{tagmem}[i][0]\ \mathbf{is}\ \neg tagin[0])) \\ \mathbf{and}\ \ & ((p_i = 001) \rightarrow (\mathsf{tagmem}[i][1]\ \mathbf{is}\ \neg tagin[1])) \\ & \vdots \qquad\qquad \vdots \qquad\qquad\qquad \vdots \\ \mathbf{and}\ \ & ((p_i = 111) \rightarrow (\mathsf{tagmem}[i][7]\ \mathbf{is}\ \neg tagin[7]))\end{aligned}$$

The formula expressing that each of the non-indexed tag-entries differs at at-least one place from $tagin$ is:

$$\begin{aligned}A' = \quad & ((index \neq 0000) \rightarrow \mathrm{mismatch}(0)) \\ \mathbf{and}\ \ & ((index \neq 0001) \rightarrow \mathrm{mismatch}(1)) \\ & \vdots \qquad\qquad \vdots \qquad\qquad\qquad \vdots \\ \mathbf{and}\ \ & ((index \neq 1111) \rightarrow \mathrm{mismatch}(15)))\end{aligned}$$

Symbolic Variables	
$index$	= 1111
$tagin$	= 11111110
$data$	= 01000000000000000000000000000000000000000000000000000000000000000000
Inputs at time 0	
aread	= 1
tagin	= 11111110
Initial Values	
tagmem[0]	= -------**1**
tagmem[1]	= ------**0**-
tagmem[2]	= ---**0**----
tagmem[3]	= ---**0**----
tagmem[4]	= ------**0**-
tagmem[5]	= -------**1**
tagmem[6]	= -------**1**
tagmem[7]	= **0**-------
tagmem[8]	= ------**0**-
tagmem[9]	= --**0**-----
tagmem[10]	= ------**0**-
tagmem[11]	= --**0**-----
tagmem[12]	= -----**0**--
tagmem[13]	= ---**0**----
tagmem[14]	= **0**-------
tagmem[15]	= 11111110
datmem[15]	= 01000000000000000000000000000000000000000000000000000000000000000000

Fig. 5. A Weakest Satisfying Strengthening of Ass. (2) without extra assumptions on data-entries.

The assertion obtained by adding A' to the antecedent of assertion (2) is exactly the assertion described in [7] and is easily proved by an STE model-checker.

3 STE-Theory Revisited

Here, we briefly revisit the STE-theory needed to describe the algorithm in STAR.

Circuits and Values. A circuit is modeled by a set of node names $\mathcal{N}$ connected by logical gates and delay elements. $\mathcal{S} \subseteq \mathcal{N}$ is the set of output nodes of delay elements. In STE, we abstract away from specific Boolean values of a node taken from the set $\mathbb{B} = \{0, 1\}$, by using the value X, which stands for *unknown*. Furthermore, the *over-constrained value* T is introduced; a node assumes value T when it is both required to take on value 0 and value 1, leading to the set of *quaternary signal values*, denoted $\mathbb{V} = \{0, 1, \mathsf{X}, \mathsf{T}\}$. On this set an *information-ordering* $\leq$ is introduced. The unknown value X contains the least information, so $\mathsf{X} \leq 0$ and $\mathsf{X} \leq 1$, while 0 and 1 are incomparable. The overconstrained value T contains the most information, so $0 \leq \mathsf{T}$ and $1 \leq \mathsf{T}$. If $v \leq w$ it is said that v is *weaker* than w.

A *circuit state*, written $s : \mathbf{State}$, is a function from $\mathcal{N}$ to $\mathbb{V}$, assigning a value from $\mathbb{V}$ to each node in the circuit. A *sequence* $\sigma : \mathbb{N} \to \mathbf{State}$ is a function from points in time to circuit states, describing the behaviour of a circuit over time. The set of all sequences σ is written $\mathbf{Seq}$. The set of sequences that do not assign the overconstrained value T to any node at any time-point is written $\mathbf{Seq}_3$.

Trajectory Evaluation Logic. STE assertions have the form $A \Longrightarrow C$. Here A and C are formulas in *Trajectory Evaluation Logic* (TEL). The only variables in the logic are

time-independent Boolean variables taken from the set V of *symbolic variables*. The language is given by the following grammar:

$$f \ ::= \ n \,\mathbf{is}\, 0 \ \mid \ n \,\mathbf{is}\, 1 \ \mid \ f_1 \ \mathbf{and} \ f_2 \ \mid \ P \rightarrow f \ \mid \ \mathbf{N}f$$

where $n \in \mathcal{N}$ and P is a Boolean propositional formula over the set of symbolic variables V. The notation $n \,\mathbf{is}\, P$, where P is a Boolean formula over the set of symbolic variables V, is used to abbreviate the formula: $(\neg P \rightarrow n \,\mathbf{is}\, 0) \ \mathbf{and} \ (P \rightarrow n \,\mathbf{is}\, 1)$. The *depth* of a TEL-formula f is the maximal degree of nestings of $\mathbf{N}$ in f. The depth of an STE-assertion $A \Longrightarrow C$ is the maximum of the depth of A and the depth of C.

The meaning of a TEL formula is defined by a satisfaction relation that relates valuations of the symbolic variables and sequences to TEL formulas. Here, the following notation is used: The time shifting operator σ^1 is defined by $\sigma^1(t)(n) = \sigma(t+1)(n)$. Standard propositional satisfiability is denoted by $\models_{\mathrm{Prop}}$. Satisfaction of a TEL-formula f, by a sequence $\sigma \in \mathbf{Seq}$, and a valuation $\phi : V \rightarrow \mathbb{B}$ (written $\phi, \sigma \models f$) is defined by

$$\begin{array}{lcl}
\phi, \sigma \models n \,\mathbf{is}\, b & \equiv & b \leq \sigma(0)(n) \ \ , \ \ b \in \{0,1\} \\
\phi, \sigma \models f_1 \ \mathbf{and} \ f_2 & \equiv & \phi, \sigma \models f_1 \ \text{and} \ \phi, \sigma \models f_2 \\
\phi, \sigma \models P \rightarrow f & \equiv & \phi \models_{\mathrm{Prop}} P \ \text{implies} \ \phi, \sigma \models f \\
\phi, \sigma \models \mathbf{N}f & \equiv & \phi, \sigma^1 \models f
\end{array}$$

Trajectories. In STE, three abstractions are used: (1) the value X can be used to abstract from a specific Boolean value of a circuit node, (2) information is only propagated forwards through the circuit (i.e. from inputs to outputs of gates) and through time (i.e. from time t to time $t+1$), (3) the initial value of all delay elements is assumed to be X. Given a circuit c, a *trajectory* is a sequence that meets the constraints of the circuit c, taking these abstractions into account. How to obtain the set of trajectories of a circuit c is described, for instance, in [10,8,9,6,1,12].

A circuit c *satisfies* a trajectory assertion $A \Longrightarrow C$, written $c \models A \Longrightarrow C$ iff for every valuation $\phi \in V \rightarrow \mathbb{B}$ of the symbolic variables, and for every trajectory τ of c such that $\tau \in \mathbf{Seq}_3$, it holds that: $\phi, \tau \models A \ \ \Rightarrow \ \ \phi, \tau \models C$.

STE-Model Checking. The theory of STE guarantees that for every TEL-formula A, circuit c and valuation ϕ, there exists an unique weakest trajectory that satisfies A. This trajectory is called the *defining trajectory of A w.r.t. ϕ*, written ${}^{\phi}_{c}[\![\ A \ ]\!]$. Furthermore, for every TEL-formula C, and valuation ϕ there exists an unique weakest sequence that satisfies C. This sequence is called the *defining sequence of C w.r.t. ϕ*, written ${}^{\phi}[\, C \,]$.

The *Fundamental Theorem of STE* states that in order to check that an assertion is true, only the defining trajectories of the antecedent need to be considered (instead of all trajectories). That is, to check that $c \models A \Longrightarrow C$, we only need to check that for every valuation of the symbolic variables ϕ, such that ${}^{\phi}_{c}[\![\ A \ ]\!] \in \mathbf{Seq}_3$, holds ${}^{\phi}[\, C \,] \leq {}^{\phi}_{c}[\![\ A \ ]\!]$.

Given a circuit description and an STE-assertion, an STE-simulator calculates a symbolic representation of the set of defining trajectories of the antecedent of the assertion. In *BDD-based STE*, BDDs are used to represent the defining trajectories. In *SAT-based STE*, non-canonical *Boolean expressions* are used. In both cases a *dual-rail encoding* is used to encode a quaternary value by two Boolean values [12].

After simulation, it is checked whether the symbolic representation of the defining trajectories of the antecedent satisfies the requirements of the consequent. In BDD-

based STE this check is trivial because of the canonicity of BDDs. In SAT-based STE, a SAT-solver is called to perform this check.

4 Finding Satisfying and Contradicting Strengthenings

The job of the main algorithm in STAR is to, given a circuit and an STE-assertion, find a weakest satisfying strengthening (respectively weakest contradicting strengthening) of the assertion. In order to do so, the algorithm employs an STE-simulator on Boolean expressions. After simulation, a SAT-problem is generated whose solutions represent all satisfying (respectively contradicting) strengthenings of the assertion. Finally, an incremental SAT-solver [3] is iteratively called to find a weakest such strengthening. Before describing the algorithm in more detail, we make the concept of strengthenings more precise.

4.1 Satisfying Strengthenings

A strengthening of an STE-assertion gives extra Boolean requirements on nodes of the circuit over time. The set of the nodes and corresponding time-points that potentially can be strengthened is called the *set of strengthening candidates*, written $\mathcal{SC} \subseteq \mathbb{N} \times \mathcal{N}$. Given an assertion of depth d, the set of strengthening candidates commonly consists of the input nodes $\mathcal{I}$ of the circuit over time-points $\{0, \ldots, d\}$ and the initial values of delay elements. That is, in that case: $\mathcal{SC} = (\{0, \ldots, d\} \times \mathcal{I}) \cup (\{0\} \times \mathcal{S})$. However, sometimes, we might want to restrict the set of strengthening candidates as we did in the case-study.

Given a set of strengthening candidates, a *strengthening* is a function $\gamma : \mathcal{SC} \rightarrow \{0, 1, \mathsf{X}\}$ from nodes and time points to the values $0, 1$, and X, giving extra requirements on the nodes of a circuit. For example, if $\gamma(0, \mathsf{p}) = 1$, $\gamma(2, \mathsf{q}) = 0$, and $\gamma(t, n) = \mathsf{X}$ for all other t and n, then node p is strengthened to have value 1 at time-point 0, and node q is strengthened to value 0 at time-point 2.

A strengthening can easily be transformed into a TEL-formula with the same meaning, denoted by $\mathsf{TEL}(\gamma)$, which is defined to be the conjunction of all $\mathbf{N}^t(n \,\mathbf{is}\, \gamma(t, n))$ with $(t, n) \in \mathcal{SC}$ and for which $\gamma(t, n) \neq \mathsf{X}$. For example, if γ is defined as in the above example, then $\mathsf{TEL}(\gamma) = ((\mathsf{p}\,\mathbf{is}\,1)\ \mathbf{and}\ \mathbf{N}^2(\mathsf{q}\,\mathbf{is}\,0))$. The TEL-formula $(A\ \mathbf{and}\ \mathsf{TEL}(\gamma))$ is called the *strengthening of A w.r.t. γ*, and is written $\mathsf{Str}(A, \gamma)$.

Given a circuit c and an assignment of symbolic variables $\phi : V \rightarrow \{0, 1\}$, a *satisfying strengthening* of an assertion $A \Longrightarrow C$ is a strengthening γ such that simulating using γ and A does not yield overconstrained nodes and makes the consequent true, i.e. $\phi, {}^{\phi}_{c}[\![\ \mathsf{Str}(A, \gamma)\ ]\!] \in \mathbf{Seq}_3$ and $\phi, {}^{\phi}_{c}[\![\ \mathsf{Str}(A, \gamma)\ ]\!] \models C$.

Strengthenings can be compared by extending the information order $\leq$ point-wise to functions, arriving at the concept of a *weakest satisfying strengthening*, which is a satisfying strengthening weaker than all other satisfying strengthenings of an assertion. Note that weakest strengthenings are not unique; there can for example be several, but incomparable, weakest satisfying strengthenings.

4.2 Generation of the SAT-Problem

A SAT-problem consists of a set of *variables* W and a *Boolean formula* P. An *assignment* is a mapping $a : W \to \{0, 1\}$. A SAT-problem S is satisfied by an assignment a, written $a \models S$, if a makes P evaluate to 1.

For calculating a strengthening of an STE-assertion of depth d, only the first d time-points of the simulation matter. Therefore, the concept of a *truncated* sequence is introduced, which is a function from the time-points $\{0, .., d\}$ to circuit states.

We will define a *SAT-problem for all satisfying strengthenings*, written $\mathsf{SS}(A \Longrightarrow C, c, \mathcal{SC})$, whose solutions represent precisely those truncated sequences σ, valuations ϕ, and strengthenings γ such that γ is a satisfying strengthening of $A \Longrightarrow C$ w.r.t. ϕ.

For an STE-assertion of depth d, the SAT-problem contains a SAT-variable v for each variable v in the set of symbolic variables V. Furthermore, for each node n in the set of nodes $\mathcal{N}$ of the circuit c, and for each time point $0 \leq t \leq d$ *two* SAT-variables are introduced, written n_t^0 and n_t^1. The two variables n_t^0 and n_t^1 encode the value of node n at time t using a standard dual-rail encoding; the function mapping a dual-rail encoded quaternary value to the quaternary value itself, written quat, is defined by: $\mathsf{quat}(0, 0) = \mathsf{X}$, $\mathsf{quat}(1, 0) = 0$, $\mathsf{quat}(0, 1) = 1$, and $\mathsf{quat}(1, 1) = \mathsf{T}$.

Finally, for each time-point/node pair (t, n) in the set of strengthening candidates $\mathcal{SC}$, the SAT-problem contains a pair of SAT-variables $\hat{n}_t^0$ and $\hat{n}_t^1$ representing a possible requirement of a strengthening on node n at time t. Again, the dual-rail encoding is used; if $\hat{n}_t^0$ and $\hat{n}_t^1$ are both 0, there is no requirement on node n at time t, if $\hat{n}_t^0 = 1$ and $\hat{n}_t^1 = 0$ the node is required to have value 0, if $\hat{n}_t^0 = 0$ and $\hat{n}_t^1 = 1$ the node is required to have value 1. The SAT-problem is constructed such that $\hat{n}_t^0$ and $\hat{n}_t^1$ are not allowed to both have value 1.

A satisfying assignment a of the SAT-problem can thus be mapped to a assignment of symbolic variables ϕ_a defined by $\phi_a(v) = a(v)$, to a truncated sequence σ_a defined by $\sigma_a(t)(n) = \mathsf{quat}(a(n_t^0), a(n_t^1))$, and to a strengthening γ_a defined by $\gamma_a(t, n) = \mathsf{quat}(a(\hat{n}_t^0), a(\hat{n}_t^1))$.

Constructing the SAT-Problem. The SAT-problem for all satisfying strengthenings $\mathsf{SS}(A \Longrightarrow C, c, \mathcal{SC})$ is defined as the conjunction of two SAT-problems: (1) A SAT-problem that restricts the sequences σ, assignments ϕ and strengthenings γ such that σ is the defining trajectory of $\mathsf{Str}(A, \gamma)$ w.r.t. ϕ, and (2) A SAT-problem that restricts the sequences σ and assignments ϕ such that they together satisfy the consequent C. Below, we define both SAT-problems.

However, first we need to define the SAT-problem for the defining trajectory of a TEL-formula. It is well-known how to use an STE-simulator on Boolean expressions to generate a SAT-problem whose satisfying assignments correspond to the set of defining trajectories of the antecedent of the assertion [8,9,2,14]. We denote this SAT-problem by $\mathsf{DTA}(A, c, d)$, and we assume that its solutions represent exactly those valuations ϕ and truncated sequences σ such that $\sigma = {}^{\phi}_{c}[\![A]\!] \restriction \{0, .., d\}$ and $\sigma \in \mathbf{Seq}_3$.

SAT-Problem for the Antecedent. We now define the SAT-problem for *the defining trajectory of a symbolically strengthened antecedent*, written $\mathsf{DTSA}(A, c, d, \mathcal{SC})$ whose solutions represent precisely those truncated sequences σ, valuations ϕ, and strengthenings γ such that σ is the (truncated) defining trajectory of $\mathsf{Str}(A, \gamma)$ w.r.t. ϕ.

In order to do so, we first introduce the concept a *symbolically strengthened antecedent*, written $\mathsf{SymStr}(A, \mathcal{SC})$. The symbolically strengthened antecedent contains for each time-point/node pair in the set of strengthening candidates $\mathcal{SC}$ a pair of symbolic variables $\hat{n}_t^0$ and $\hat{n}_t^1$, representing a possible requirement of strengthening γ on node n at time t, and is defined by:

$$\mathsf{SymStr}(A, \mathcal{SC}) = A \ \mathbf{and} \ (\mathbf{and}_{(t,n)\in\mathcal{SC}} \ \mathbf{N}^t(\hat{n}_t^0 \rightarrow n \ \mathbf{is} \ 0 \ \mathbf{and} \ \hat{n}_t^1 \rightarrow n \ \mathbf{is} \ 1))$$

The SAT-problem for the defining trajectory of the symbolically strengthened antecedent is defined by: $\mathsf{DTSA}(A, c, d, \mathcal{SC}) = \mathsf{DTA}(\mathsf{SymStr}(A, \mathcal{SC}), c, d)$.

SAT-Problem for the Consequent. The SAT-problem for satisfaction of a consequent C, written $\mathsf{SAT}(C)$, is constructed such that its set of solutions contains precisely those sequences σ and assignments of the symbolic variables ϕ that together satisfy consequent C. (i.e. $\phi, \sigma \models C$).

In order to build this SAT-problem, we need to define the concept of *defining formula*. Given a consequent C, a node name n, a Boolean value $b \in \mathbb{B}$, and a time point t, we can construct a propositional formula that is true exactly when C requires the node n to have value b at time point t. This formula is called the *defining formula of* $n = b$ *at* t, and is denoted by $\langle C \rangle(t)(n = b)$.

For example, if the consequent C is defined as $(a \wedge b) \rightarrow \mathsf{p} \ \mathbf{is} \ 0$, then $\langle C \rangle(0)(\mathsf{p} = 0)$ is the formula $a \wedge b$, since only when $a \wedge b$ holds, does C require node p to be 0. However, $\langle C \rangle(0)(\mathsf{p} = 1)$ is the false formula 0, since C never requires the node p to be 1.

The *defining formula* is defined recursively as follows:

$$\begin{aligned}
\langle m \ \mathbf{is} \ b' \rangle(t)(n = b) &= \begin{cases} 1, & \text{if } m = n, b' = b \text{ and } t = 0 \\ 0, & \text{otherwise} \end{cases} \\
\langle f_1 \ \mathbf{and} \ f_2 \rangle(t)(n = b) &= \langle f_1 \rangle(t)(n = b) \vee \langle f_2 \rangle(t)(n = b) \\
\langle P \rightarrow f \rangle(t)(n = b) &= P \wedge \langle f \rangle(t)(n = b) \\
\langle \mathbf{N} f \rangle(t)(n = b) &= \begin{cases} \langle f \rangle(t-1)(n = b), & \text{if } t > 0 \\ 0, & \text{otherwise} \end{cases}
\end{aligned}$$

Note that for an antecedent of the form $f_1 \ \mathbf{and} \ f_2$ to require that a node n has a value b, it is enough that only one of the formulas f_1 or f_2 requires that n is b. The SAT-problem for the satisfaction of the consequent is now defined by:

$$\mathsf{SAT}(C) = \bigwedge_{(n,t)\in C} (\langle C \rangle(t)(n = 0) \rightarrow n_t^0) \ \wedge \ (\langle C \rangle(t)(n = 1) \rightarrow n_t^1)$$

Here, $(n, t) \in C$ means that C refers to node n at time-point t.

SAT-Problem for All Satisfying Strengthenings. Given an assertion $A \Longrightarrow C$ of depth d for a circuit c and a set of strengthening candidates $\mathcal{SC}$, the *SAT-problem for all satisfying strengthenings*, written $\mathsf{SS}(A \Longrightarrow C, c, \mathcal{SC})$, is defined by:

$$\mathsf{SS}(A \Longrightarrow C, c, \mathcal{SC}) = \mathsf{DTSA}(A, \mathcal{SC}, c, d) \wedge \mathsf{SAT}(C)$$

The solutions to the above SAT-problem represent exactly those valuations ϕ and strengthenings γ such that γ is a satisfying strengthening of $A \Longrightarrow C$ w.r.t. ϕ.

4.3 Finding a Weakest Assignment

Calling a SAT-solver on the SAT-problem for all satisfying strengthenings, $\mathsf{SS}(A \Longrightarrow C, c, \mathcal{SC})$, yields a satisfying strengthening (if one exists). This satisfying strengthening, however, is not necessarily a *weakest* satisfying strengthening. To find a weakest satisfying strengthening, iteratively "blocking constraints" are added to the SAT-problem that block the last found strengthening and allow only strictly weaker strengthenings. This process is repeated until the SAT-problem becomes unsatisfiable; the last found satisfying strengthening is then guaranteed to be a weakest satisfying strengthening. As said earlier, weakest strengthenings are not neccessarily unique; the result of this process is an arbitrary weakest satisfying strengthening.

Given a strengthening $\gamma : \mathcal{SC} \to \{0, 1\}$, the blocking constraint consists of four parts: (1) for every node n that is assigned value X at time t by γ, we require that it is assigned value X in any weaker strengthening, (2) any node that is assigned value 0 at time t is allowed to assume values 0 and X in a weaker strengthening, but not value 1, (3) any node that is assigned value 1 at time t is allowed to assume values 1 and X in a weaker strengthening, but not value 0, and (4) at least one of the nodes should change value. This yields the following blocking constraint $B(\gamma)$:

$$\begin{aligned} B(\gamma) = \quad & \Big(\bigwedge_{(t,n)\in\mathcal{SC},\gamma(t,n)=\mathsf{X}} (\neg\hat{n}_t^0 \;\&\; \neg\hat{n}_t^1) \Big) \\ \wedge \; & \Big(\bigwedge_{(t,n)\in\mathcal{SC},\gamma(t,n)=0} \neg\hat{n}_t^1 \Big) \\ \wedge \; & \Big(\bigwedge_{(t,n)\in\mathcal{SC},\gamma(t,n)=1} \neg\hat{n}_t^0 \Big) \\ \wedge \; & \Big(\big(\bigvee_{\gamma(t,n)=0} \neg\hat{n}_t^0 \big) \vee \big(\bigvee_{\gamma(t,n)=1} \neg\hat{n}_t^1 \big) \Big) \end{aligned}$$

The solutions to the SAT-problem $B(\gamma)$ represent exactly those strengthenings γ' such that $\gamma' < \gamma$. This finishes the description of the algorithm for finding a weakest satisfying strengthening.

4.4 Contradicting Strengthenings

Given a circuit c and an assignment of symbolic variables $\phi : V \to \{0, 1\}$, a *contradicting strengthening* of an assertion $A \Longrightarrow C$ is a strengthening γ such that such that there exists a node n, time-point t, and Boolean value b, such that simulating using γ and A yields b for n at time t (i.e. ${}^{\phi}_{c}[\![\, \mathsf{Str}(A, \gamma) \,]\!](t)(n) = b$), but the consequent requires n to be $\neg b$ (i.e. ${}^{\phi}[\, C \,](t)(n) = \neg b$). Again, we require that the strengthened antecedent does not yield overconstrained nodes, i.e. ${}^{\phi}_{c}[\![\, \mathsf{Str}(A, \gamma) \,]\!] \in \mathbf{Seq}_3$.

The SAT-problem for finding a weakest contradicting strengthening has the same structure as the SAT-problem for the weakest satisfying strengthening; one part for the antecedent, and one part for the consequent. The SAT-problem for the contradiction of a consequent C, written $\mathsf{CON}(C)$, is constructed such that at least one node differs in its Boolean value from what is required by C:

$$\mathsf{CON}(C) = \bigvee_{(n,t)\in C} (\langle C \rangle(t)(n = 0) \wedge n_t^1) \;\vee\; (\langle C \rangle(t)(n = 1) \wedge n_t^0)$$

The *SAT-problem for all contradicting strengthenings*, written $\mathsf{CS}(A \Longrightarrow C, c, \mathcal{SC})$ is defined by: $\mathsf{CS}(A \Longrightarrow C, c, \mathcal{SC}) = \mathsf{DTSA}(A, \mathcal{SC}, c, d) \wedge \mathsf{CON}(C)$. We find an actual *weakest* contradicting strengthening in exactly the same way as described in the previous subsection.

5 Discussion

We have introduced the novel concept of strengthenings, that can greatly assist in performing manual abstraction refinement for STE. Furthermore, we have developed a SAT-based algorithm for finding weakest strengthenings using an incremental SAT-solver to minimise the strengthening. We have implemented the algorithm in a tool called STAR, and have shown how it can be used to assist in abstraction refinement in a non-trivial case-study.

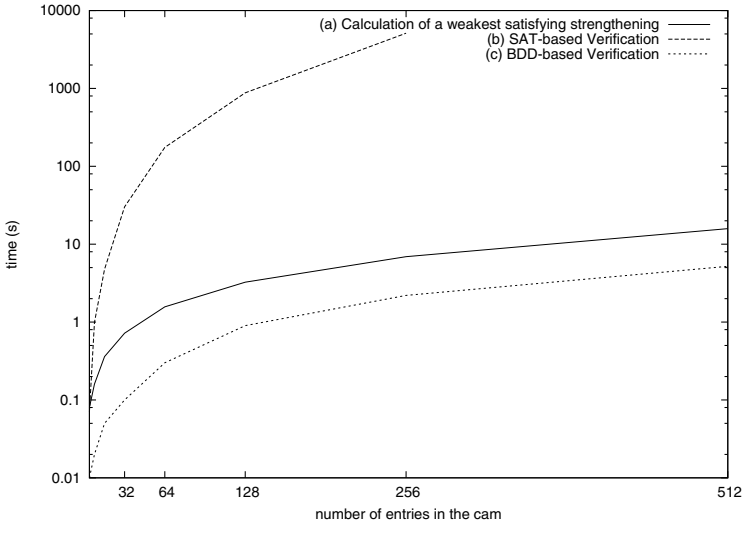

Fig. 6. Experimental results for CAMs with a tag-width of 16 bits, a data-width of 64 bits, and a varying number of entries, using a PC with a Pentium IV processor at 3GHz and 2GB of memory

As far as we believe, the information provided by our method cannot be calculated by BDD-based techniques, because too many BDD-variables would be needed.

Scalability. We believe that our method scales well. To illustrate this, we compare the running times[1] of three different experiments for CAMs with a varying number of entries[2] in Fig. 6: (a) finding a weakest satisfying strengthening of CAM assertion 2 using STAR, (b) proving the corrected assertion using a SAT-based STE model-checker (as described in [9]), and (c) proving the corrected assertion with BDDs using Forte.

As the figure shows, when the right abstraction has been found, BDD-based STE is superior over SAT-based STE for proving properties. As discussed before, finding

[1] For the SAT-based methods we only show the time spent by SAT-solving. Overhead in simulating the circuit is not counted since this was implemented inefficiently. Efficient symbolic simulators (like the one in Forte) can perform symbolic simulation with Boolean expressions in negligible time.

[2] We provide the netlists of the CAMs used at http://www.cs.chalmers.se/~jwr/CAV2006.

the right abstraction is, however, highly non-trivial. Here, STAR can help by finding weakest strengthenings. The graph shows that this can be done in reasonable time.

Another Application. In practical uses of STE, often the first step in a verification attempt is the *wiggling* phase [1]. The goal of this phase is to find out what minimal set of inputs and initial values of registers should be driven to make non-X values appear at designated circuit outputs. Commonly, wiggling is performed by using the STE-model checker as a scalar (that is, non-symbolic) simulator; the simulator is iteratively fed with vectors of Boolean values and X's, in the hope that, by trial-and-error, a minimal set of nodes to be driven can be found. Our method provides a more systematic approach to wiggling; the STAR tool can be asked to provide a weakest strengthening such that a given set of output nodes takes on non-X values. The adaption needed to the algorithm presented in the previous section is trivial. We have used this "wiggling"-mode of STAR on several different kinds of circuits (CAMs, memories, and arithmetic circuits), always quickly obtaining a weakest strengthening making a set of given outputs non-X.

Future Work. We would like to investigate whether we can use the presented technique for automatic discovery of symbolic indexing schemes [6].

Acknowledgements. Thanks to Mary Sheeran, Tom Melham, and the anonymous referees for giving valuable feedback. We are grateful for an equipment grant from Intel Corporation.

References

1. M. Aagaard, R. B. Jones, T. F. Melham, J. W. O'Leary, and C.-J. H. Seger. A methodology for large-scale hardware verification. In *FMCAD*, 2000.
2. P. Bjesse, T. Leonard, and A. Mokkedem. Finding bugs in an Alpha microprocessor using satisfiability solvers. In *CAV 2001*, volume 2102 of *LNCS*. Springer-Verlag, 2001.
3. N. Eén and N. Sörensson. An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT2003)*, 2003.
4. FORTE. http://www.intel.com/software/products/opensource/tools1/verification.
5. B. Li, C. Wang, and F. Somenzi. Abstraction refinement in symbolic model checking using satisfiability as the only decision procedure. *Journal on STTT*, 7(2):143–155, Apr. 2005.
6. T. F. Melham and R. B. Jones. Abstraction by symbolic indexing transformations. In *Formal Methods in Computer-Aided Design FMCAD*, volume 2517 of *LNCS*, 2002.
7. M. Pandey, R. Raimi, R. E. Bryant, and M. S. Abadir. Formal verification of content addressable memories using symbolic trajectory evaluation. In *DAC'97*, 1997.
8. J.-W. Roorda. Symbolic trajectory evaluation using a satisfiability solver. Licentiate thesis, Computing Science, Chalmers University of Technology, 2005.
9. J.-W. Roorda and K. Claessen. A new SAT-based Algorithm for Symbolic Trajectory Evaluation. In *Correct Hardware Design and Verification Methods (CHARME)*, 2005.
10. J.-W. Roorda and K. Claessen. Explaining Symbolic Trajectory Evaluation by Giving it a Faithful Semantics. In *International Computer Science Symposium in Russia (CSR)*, volume 3967 of *LNCS*, 2006.
11. T. Schubert. High level formal verification of next-generation microprocessors. In *Proceedings of the 40th conference on Design automation*, pages 1–6. ACM Press, 2003.

12. C.-J. H. Seger and R. E. Bryant. Formal verification by symbolic evaluation of partially-ordered trajectories. *Formal Methods in System Design*, 6(2), 1995.
13. R. Tzoref and O. Grumberg. Automatic Refinement and Vacuity Detection for Symbolic Trajectory Evaluation. In *Computer Aided Verification (CAV)*, 2006.
14. J. Yang, R. Gil, and E. Singerman. satGSTE: Combining the abstraction of GSTE with the capacity of a SAT solver. In *Designing Correct Circuits (DCC'04)*, 2004.