A Theory of Abstraction for Arrays

Steven German
IBM T.J. Watson Research Center

October 2011
The Problem of Verifying Systems with Arrays

- Large arrays are often a barrier to verifying hardware designs
- Many previous approaches to abstracting arrays
- Abstracting arrays over a bounded time interval
- Prefer methods that:
  - Build unbounded-time sequential models
  - Are fully automatic
- Most directly related previous approach by Bjesse [FMCAD 2008]
- Limitations of previous approach
  - No reduction when latency from array read to output is unbounded
  - Clock gating introduces unbounded latency
New Results of This Paper

• New mathematical principle for abstraction of arrays
  – New principle allows unbounded latency from array read to output
  – Based on Small Model Theorem for a word-level logic with arrays
  – Previous approaches are based on principle of overapproximating behavior

• Automatic algorithm for constructing abstract models
  – Algorithm can build small abstract models for complex industrial designs

• Abstract models are sound and complete for safety properties

• To obtain these results, need to develop mathematical theory

• Details are in a longer version of paper, available from author
Traditional Abstract Models of Arrays

- Modeled address: Normal array semantics
- Unmodeled address: Nondeterministic value

1. Replace array with smaller array that overapproximates
   • Sound for safety properties

2. Restrict safety property to cases where modeled addresses are read

\[ p \xrightarrow{\text{modeled}} p \]
Unbounded Latency

• Bjesse 2008 shows how to define $modeled(k)$ to mean
  
  “$k$ cycles in past, a modeled address was read”
  
  – Example: $modeled(2) \land modeled(3) \rightarrow p$
  
  – Solution for bounded latency

• For unbounded latency, not helpful to use
  
  “Array reads at all times in past were to modeled addresses”

  – Only true in unabstracted model

• New idea: Define a formula that means
  
  “Output at current time does not depend on reading unmodeled array addresses at any time in past”
A New Approach to Array Abstraction

- Read, write to modeled addresses have normal semantics
- Choose modeled addresses nondeterministically (as in Bjesse 2008)
- Read to unmodeled addresses returns special value ⊥
- Value ⊥ propagates according to semantic rules
- Property $p \implies p \neq ⊥ \implies p = \text{true}$
- Sound provided:
  At all times, For all inputs,
  Number of array addresses $p$ depends on $\leq$ Number of modeled addresses
- If there is a counterexample to safety property $p$, some nondeterministic choice of modeled addresses finds the counterexample
- Goal of talk is to make these ideas more clear
Steps to Realize New Approach

1. Define mathematical meaning of dependence of a signal on an array address
2. Give automatic method for determining that at all times, for all inputs,
   the signal $p$ depends on $\leq n$ array addresses
3. Show that the proof method is sound

- Mathematics is different from traditional approach, where soundness follows easily from overapproximate behavior on unmodeled addresses
A Term Logic with Arrays

Two kinds of expressions: \textit{signal expressions} and \textit{array expressions}.

- Signal expressions
  1. Signal variable
     - Represents word level signal
  2. $\text{op}(e_1, \ldots, e_k)$, where $e_1, \ldots, e_k$ are signal expressions
     - Represents block of combinational logic
  3. $\text{mux}(\text{control}, \text{data}_1, \text{data}_2)$, where \text{control}, \text{data}_1, \text{data}_2 are signal expressions. Use data forwarding properties in abstract models.
  4. $a[addr]$, where $a$ is an array expression and $addr$ is a signal expression.

- Array expressions
  1. Array variable
  2. $\text{write}(a, addr, value)$, where $a$ is an array expression and $addr$, $value$ are signal expressions
Signal and Array Values

- Finite set of signal values (word-level), $V$
- Bottom value, $\perp \notin V$, represents subscripting array out of range
- Extended set of signal values, $V^+ = V \cup \{\perp\}$
- Set of array values, $V \rightarrow V^+$
States

A state $\sigma$ is a function mapping all signal and array variables to values.

- For signal variable $s$, $\sigma(s) \in V$
- For array variable $a$, $\sigma(a) \in (V \rightarrow V)$
- States are used to represent initial conditions of systems
Semantics of Expressions

The semantics of expressions maps a state and an expression to a value.

• For signal expression $se$, $\sigma[se] \in V^+$

• For array expression $ae$, $\sigma[ae] \in (V \rightarrow V^+)$

• Purpose of semantics is to allow reasoning about system with reduced arrays

• Reading an array outside its domain produces bottom value $\bot$

• Writing an array to an address in $V$ outside domain of array, does not change value of array

• Writing an array with address $\bot$ causes all elements of array to be $\bot$

• Operator expression $op(e_1, \ldots, e_n)$ produces output $\bot$ if any input is $\bot$

• Multiplexor $mux(e_1, e_2, e_3)$ produces output $\bot$ if control input $e_1$ is $\bot$ or selected input $e_2, e_3$ is $\bot$
Operational Semantics

• A system $M$ is defined by state variables and next-state expressions

  $\mathcal{N}(s)$ is the next-state expression for state variable $s$

• Define $s^k$ to be an expression for state variable $s$ at time $k$

  $s^0 = s$

  $s^k$ is $k^{th}$ expansion of $\mathcal{N}(s)$

• Value of $s$ at time $k$ in initial state $\sigma$ is $\sigma[s^k]$
Checking Safety Properties

- System $\mathcal{M}$
- Safety property represented by output signal $p$ ($p = 1$ iff property is true)
- Let $\mathcal{T}$ be a set of states
- Safety property $p$ holds over all initial states in $\mathcal{T}$ iff

$$\forall \sigma \in \mathcal{T}, \forall k \geq 0 : \sigma[p^k] = 1$$

- This check corresponds to model checking the design on arrays of original size
  - Construct circuit representation of $\sigma[p^k]$ using the next-state expressions
- We will show how to check safety properties over arrays of a smaller size
Essential Array Indices

Depending on the state, some indices of an array do not need to be evaluated

- Example: Let $E$ be the expression $\text{write} (\text{write}(a, e1, a[1]), e2, a[2]) [f]$
  
  If $\sigma[f] = \sigma[e2] \implies \{f, 2\}$
  
  If $\sigma[f] \neq \sigma[e2] \land \sigma[f] = \sigma[e1] \implies \{f, 1\}$
  
  If $\sigma[f] \neq \sigma[e2] \land \sigma[f] \neq \sigma[e1] \implies \{f\}$

  In every state, set of needed index expressions is an element of the set $S = \{\{f\}, \{f, 1\}, \{f, 2\}\}$

  For general case, we can define a function

  - Essential Indices, $\text{eindx}(\text{exp}, \sigma, \text{array}\_\text{variable}) \mapsto \{\text{array}\_\text{indices}\} \subseteq V$
    
    - Array indices that must be read from $\text{array}\_\text{variable}$ to evaluate $\text{exp}$ in $\sigma$

  - Idea of Small Model Theorem

    For any state $\sigma$, no matter how large the array $a$ in $\sigma$, there exists a state $\sigma'$ where $a$ has size 2, and $\sigma'[E] = \sigma[E]$
Small Model Using Essential Indices

The semantics $\sigma[exp]$ and the function $\text{eindx}(exp, \sigma, a)$ have the following relationship:

**Lemma.** For all $exp$, $\sigma$, $a$, there exists a state $\sigma'$ such that

- $\sigma' \leq \sigma$
- For all array variables $a$, $\text{dom}(\sigma'(a)) = \text{eindx}(exp, \sigma, a)$
- $\sigma'[exp] = \sigma[exp]$

- The state $\sigma'$ is a small model for the value of expression $exp$ in state $\sigma$

**Definition.** A state $\sigma'$ is called a *substate* of $\sigma$, written $\sigma' \leq \sigma$ iff

- For all signal variables $s$, $\sigma'(s) = \sigma(s)$, and
- For all array variables $a$, $\sigma'(a) \subseteq \sigma(a)$
Checking Safety Properties with Small Arrays

• Let $T$ be a set of states and $a$ an array variable such that $a$ has size $n$ for all states in $T$

• Let $m$ be

$$m = \max_{\sigma \in T} \max_{k \geq 0} |\text{eindx}(p^k, \sigma, a)| \leq n$$

\forall \sigma \in T, \forall k \geq 0, there is a state $\sigma'$ where $a$ has size $m$ and $\sigma'[p^k] = \sigma[p^k]$

• Let $T'$ be the set of substates of states in $T$ where $a$ has size $m$

• Assume for all initial states in $T$, that $p$ is evaluated without subscript errors

• Then, $(p = 1)$ is always true in executions from initial states in $T$

  \textbf{iff} $(p = 1 \lor p = \bot)$ is always true in executions from initial states in $T'$

• Model where array $a$ has size $m$ is sound and complete for safety property $p$

• See conference paper for proof
Size of the Abstract Model

• The function \( \max_{k \geq 0} \max_{\sigma} |\text{eindx}(p^k, \sigma, a)| \) is difficult to compute!

• Case splitting overapproximates \( \max_{\sigma} |\text{eindx}(p^k, \sigma, a)| \), for a fixed \( k \)

• Example: Let \( E \) be the expression \( \text{write}(\text{write}(a, e_1, a[1]), e_2, a[2]) [f] \)
  
  If \( \sigma[f] = \sigma[e_2] \) \( \implies \) \( \{f, 2\} \)
  
  If \( \sigma[f] \neq \sigma[e_2] \) \( \land \) \( \sigma[f] = \sigma[e_1] \) \( \implies \) \( \{f, 1\} \)
  
  If \( \sigma[f] \neq \sigma[e_2] \) \( \land \) \( \sigma[f] \neq \sigma[e_1] \) \( \implies \) \( \{f\} \)

  In every state, set of index expressions is an element of the two-level set
  \( S = \{\{f\}, \{f, 1\}, \{f, 2\}\} \)

• The set \( S \) overapproximates \( \text{eindx} \) \( \forall \sigma \exists s \in S : \text{eindx}(E, \sigma, a) \subseteq \sigma(s) \)

• Recursive algorithm constructs the two-level set for any expression

• A fixed point computation can find a set of expressions that overapproximates
  the largest set of index expressions over the sequence \( p^0, p^1, p^2, \ldots \)
Industrial Examples

• Implementation is in development

• Preliminary results with algorithm show reduction in cases that could not be reduced by previous methods

• Set of 255 examples not solvable in 24 hours by other methods
  – Reduced some arrays in 85 examples (33%)
  – Completely solved 33 examples in $\leq 2$ hours
Sequential Equivalence of Systems with Arrays

- Due to physical limits, designers may split large array into smaller arrays
- In simple cases, new design has arrays with same number of rows, fewer columns
- Harder case is when new design has array with different number of rows
Original Model: 32912 registers
Reduced Model: 401 registers
Summary

• New theory of array abstraction based on Small Model Theorem
• Reduced size of arrays is computed automatically by static analysis
• Early experimental results are encouraging
• Planned Improvements
  – Improve the accuracy of the array size estimate
• Longer version of paper is available
Extra Slides
Automatic Array Abstraction [Bjesse 2008]

- Define $modeled(k)$ to mean
  
  “$k$ clock cycles ago, a modeled address read was read from array”

- Use abstraction-refinement to decide values of $k$ needed to prove property $p$

- The modeled addresses are chosen nondeterministically at start of each run

- Limitations
  
  - Many designs have unbounded latency from array read to output
  - Abstraction-refinement uses long runtimes in many examples
Semantics

1. $\sigma[v] = \sigma(v)$, where $v$ is a signal variable.

2. $\sigma[\text{op}(e_1, \ldots, e_n)] =$
   \[
   \begin{cases} 
   OP(\sigma[e_1], \ldots, \sigma[e_n]), & \text{if } \sigma[e_i] \neq \bot, \text{ for } i = 1, \ldots, n, \\
   \bot & \text{if for some } i, \sigma[e_i] = \bot
   \end{cases}
   \]
   where $OP$ is the interpretation of $\text{op}$

3. $\sigma[\text{mux}(e_1, e_2, e_3)] =$
   \[
   \begin{cases} 
   \sigma[e_2] & \text{if } \sigma[e_1] = 0 \\
   \sigma[e_3] & \text{if } \sigma[e_1] = 1 \\
   \bot & \text{if } \sigma[e_1] \notin \{0, 1\}
   \end{cases}
   \]

4. $\sigma[a[e]] =$
   \[
   \begin{cases} 
   (\sigma[a])(\sigma[e]) & \text{if } \sigma[e] \in D(a, \sigma) \\
   \bot & \text{if } \sigma[e] \notin D(a, \sigma)
   \end{cases}
   \]

5. $\sigma[a] = \sigma(a)$, where $a$ is an array variable.

6. $\sigma[\text{write}(a, e_1, e_2)] =$
   \[
   \begin{cases} 
   \sigma[a][\sigma[e_1] \leftarrow \sigma[e_2]] & \text{if } \sigma[e_1] \in D(a, \sigma) \\
   \sigma[a] & \text{if } \sigma[e_1] \in V - D(a, \sigma) \\
   \text{bottom}(a, \sigma) & \text{if } \sigma[e_1] = \bot
   \end{cases}
   \]
Substates

Definition. A state $\sigma'$ is called a substate of $\sigma$, written $\sigma' \leq \sigma$ iff

- For all signal variables $s$, $\sigma'(s) = \sigma(s)$, and
- For all array variables $a$, $\sigma'(a) \subseteq \sigma(a)$
A system $\mathcal{M}$ has the form $(\mathcal{S}, \mathcal{I}, \mathcal{N}, \mathcal{O}, \mathcal{E})$

- $\mathcal{S}$ set of state variables
- $\mathcal{I}$ set of input variables
- $\mathcal{N}$ next-state expressions $\mathcal{N} : \mathcal{S} \rightarrow \text{expressions}$
- $\mathcal{O}$ set of output variables
- $\mathcal{E}$ output expressions
Approximating Over All States

• Want to compute an overapproximate value for $\max_\sigma |\text{eindx}(e, \sigma, a)|$

• Define a function $\phi(expression, array\_variable) \rightarrow \{s_1, \ldots, s_n\}$,
  where the $s_i$ are sets of expressions.

• We call $S = \{s_1, \ldots, s_n\}$ a two-level set.

• Each $s_i \in \phi(e, a)$ is a set of possible expressions for the values of $\text{eindx}(e, \sigma, a)$

• For all $\sigma$, $\exists s_i \in \phi(e, a) : \text{eindx}(e, \sigma, a) \subseteq \sigma(s_i)$

• $\forall \sigma : |\text{eindx}(e, \sigma, a)| \leq \|\phi(e, a)\|$, where $\|\{s_1, \ldots, s_n\}\| = \max_i |s_i|$, maximum size of element in $\{s_1, \ldots, s_n\}$
Definition of $\phi$

Define $X \uplus Y = \{ x \uplus y \mid x \in X, y \in Y \}$

\[
\begin{align*}
\phi(v, a) &= \{\emptyset\}, \text{ if } v \text{ is a signal variable or an array variable} \\
\phi(c, a) &= \{\emptyset\}, \text{ if } c \text{ is a constant} \\
\phi(b[e], a) &= \begin{cases} 
\phi(b, a) \uplus \phi(e, a) \uplus \{\{e\}\} & \text{ if } \text{root}(b) = a \\
\phi(b, a) \uplus \phi(e, a) & \text{ otherwise}
\end{cases} \\
\phi(op(e_1, \ldots, e_n), a) &= \phi(e_1, a) \uplus \ldots \uplus \phi(e_n, a) \\
\phi(mux(e_1, e_2, e_3), a) &= (\phi(e_1, a) \uplus \phi(e_2, a)) \uplus (\phi(e_1, a) \uplus \phi(e_3, a)) \\
\phi(write(b, e_1, e_2), a) &= (\phi(e_1, a) \uplus \phi(e_2, a)) \uplus (\phi(e_1, a) \uplus \phi(b, a))
\end{align*}
\]
Building Abstract Model

• Original design over word-level values $V \rightarrow$ Design over $V \cup \{\bot\}$

• Add boolean $v$ field to each signal

\[
\begin{array}{c}
\text{value} \\
\rightarrow \\
\text{v} \\
\rightarrow \\
\text{value}
\end{array}
\]

• $v = \text{true}$ represents values in $V$; $v = \text{false}$ represents $\bot$

• Concern about adding many bits to model
  – Work with word level values

• Replace blocks of combinational logic and $\text{mux}$ with versions over $V \cup \{\bot\}$
  – Abstract models do not need to have $\bot$ version of each gate

• Safety property $p$

\[
p \rightarrow p.v \rightarrow p.\text{value}
\]
Abstract Arrays

• Each row of abstract array has address field and v field

<table>
<thead>
<tr>
<th>value</th>
<th>address</th>
<th>v</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td></td>
<td></td>
<td>value</td>
</tr>
<tr>
<td>value</td>
<td></td>
<td></td>
<td>value</td>
</tr>
<tr>
<td>value</td>
<td></td>
<td></td>
<td>value</td>
</tr>
</tbody>
</table>

• Address field is set nondeterministically in initial state

• Read and write operations search the address field
Early Results on Industrial Examples

- Reductions on 401 industrial examples.
- Algorithm reduced arrays in 187 examples.
- Implementation in development – some examples not fully processed.

<table>
<thead>
<tr>
<th>Original Rows</th>
<th>Reduced Number of Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>144</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>32</td>
<td>37</td>
</tr>
<tr>
<td>39</td>
<td>24</td>
</tr>
<tr>
<td>48</td>
<td>24</td>
</tr>
<tr>
<td>64</td>
<td>46</td>
</tr>
<tr>
<td>128</td>
<td>4</td>
</tr>
<tr>
<td>256</td>
<td>3</td>
</tr>
<tr>
<td>1024</td>
<td>3</td>
</tr>
</tbody>
</table>
Reconfigured Arrays Example

- Reconfigured large array into two smaller arrays
- Problem is to verify sequential equivalence
- Original design has array with 1024 rows × 16 columns
- New design has two arrays, each 128 rows × 64 columns
- Array addressing, data alignment and staging logic substantially redesigned
- Design uses clock gating, so method of Bjesse does not reduce arrays