Verifying multipliers with *BMDs and a backward construction algorithm

Pierre Senellart

May 3rd 2002
Introduction

• BDDs (Bryant, 1986) very powerful tools for verifying arithmetic circuits.

• But exponential on multipliers.

• *BMDs (Bryant, 1994) give a polynomial algorithm but need high-level information.

• Backward construction algorithm (Hamaguchi et al, 1995)
Moment decomposition of a function

\[ f : \{0, 1\}^n \rightarrow \mathbb{N} \]

\[ f_{x_i} : \{0, 1\}^{n-1} \rightarrow \mathbb{N} \]
\[ (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \mapsto f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \]

\[ f_{x_i} : \{0, 1\}^{n-1} \rightarrow \mathbb{N} \]
\[ (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \mapsto f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \]
Moment decomposition of a function (continuing...)

\[ f_{x_i} = f_x - f_{\bar{x}} \]

\[ f(x_1, \ldots, x_n) = f_{\bar{x}}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) + x_i \left( f_{\bar{x}}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) - f_{\bar{x}}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \right) \]

\text{constant moment} \quad \text{linear moment}
BMDs and *BMDs

\[ f(x, y, z) = 8 - 20z + 2y + 4yz + 12x + 24xz + 15xy \]
Arithmetic operations

*BMDs of classical arithmetic operations are of linear size.*
**Backward construction algorithm - step 1**

**Beginning of the algorithm:** the cut crosses all the primary outputs. The *BMD of the word-level interpretation of the output is constructed.
Backward construction algorithm - step 2

A gate just left to the cut is chosen and its output is substituted in the *BMD by the corresponding function of its inputs.
Backward construction algorithm - step 3

At any time, the BMD expresses the word-level representation of the output as a function of the nets currently crossed by the cut.
Backward construction algorithm - step 4 (first try)

Problem: intermediary results must be kept!
Backward construction algorithm - step 4

Condition: A gate may be chosen only if its output is connected to only the input of the gates that have been already taken.
Backward construction algorithm - step 5

End of the algorithm: the cut crosses all primary outputs. The *BMD expresses the word-level representation of the output as a function of the inputs.
Add-step and carry-save multiplication

\[ \begin{array}{c}
\times \\
\hline
1 & 1 \\
+ & 1 & 1 \\
+ & 1 & 1 & 1 \\
\hline
1 & 0 & 0 & 1 \\
+ & 1 & 1 & 1 \\
\hline
1 & 0 & 1 & 0 & 1
\end{array} \]

Add-step multiplication

\[ \begin{array}{c}
\times \\
\hline
1 & 1 & 1 \\
+ & 1 & 1 & \cdot \\
+ & 1 & 1 & \cdot \\
\hline
\{110, 11\} \\
+ & 1 & 1 & \cdot \\
\hline
\{1001, 1100\} \\
\hline
1 & 0 & 1 & 0 & 1
\end{array} \]

Carry-save multiplication
Experimental results

<table>
<thead>
<tr>
<th>Number of bits</th>
<th>Time Add-step (s)</th>
<th>Time Carry-save (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>58</td>
</tr>
<tr>
<td>16</td>
<td>161</td>
<td>1115</td>
</tr>
<tr>
<td>32</td>
<td>$O(n^{3.7})$</td>
<td>$O(n^{4.3})$</td>
</tr>
</tbody>
</table>

(Lava, Hotlips)
What now?

• Backward Construction Algorithm: very efficient, in comparison with former methods

• Still, need of something better: $O(n^4)$ is too much!

• Completely different direction?