Probabilistic Model Checking

Marta Kwiatkowska
Gethin Norman
Dave Parker

University of Oxford

Part 10 – Implementation of Probabilistic Model Checking
Overview

• Implementation of probabilistic model checking
  – overview, key operations, symbolic vs. explicit

• Binary decision diagrams (BDDs)
  – introduction, operations, sets, transition relations, …

• Multi-terminal BDDs (MTBDDs)
  – introduction, operations, vectors, matrices, performance, …
Implementation overview

• Overview of the probabilistic model checking process
  – two distinct phases: model construction, model checking
  – three different models, two different logics, various methods
  – but... all these processes have much in common
Model construction

High-level model

PRISM language description

Model construction

Translation from high-level language

Reachability: building set of reachable states

Model

matrix manipulation
graph-based algorithm

 DTMC, MDP or CTMC
Model checking

Two distinct classes of techniques:
- graph-based algorithms
- iterative numerical computation
Underlying operations

• Key objects/operations for probabilistic model checking

• Graph–based algorithms
  – underlying transition relation of DTMC/MDP/CTMC
  – manipulation of transition relation and state sets

• Iterative numerical computation
  – transition matrix of DTMC/MDP/CTMC, real–valued vectors
  – manipulation of real–valued matrices and vectors
  – in particular: matrix–vector multiplication
State-space explosion

• Models of real-life systems are typically huge
  – familiar problem for verification/model checking techniques

• State-space explosion problem
  – linear increase in size of system can result in an exponential increase in the size of the model
  – e.g. \( n \) parallel components of size \( m \), can give up to \( m^n \) states

• Need efficient ways of storing models, sets of states, etc.
  – and efficient ways of constructing, manipulating them

• Here, we will focus on symbolic approaches
Symbolic data structures

- Distinguish between explicit and symbolic storage
- Symbolic data structures
  - usually based on binary decision diagrams (BDDs) or variants
  - avoid explicit enumeration of data by exploiting regularity
  - potentially very compact storage (but not always)
- Sets of states:
  - explicit: bit vectors, symbolic: BDDs
- Real-valued vectors:
  - explicit: arrays of reals (in practice, doubles/floats)
  - symbolic: multi-terminal BDDs (MTBDDs)
- Real-valued matrices:
  - explicit: sparse matrices
  - symbolic: MTBDDs
Overview

• Implementation of probabilistic model checking
  – overview, key operations, symbolic vs. explicit

• Binary decision diagrams (BDDs)
  – introduction, operations, sets, transition relations, ...

• Multi-terminal BDDs (MTBDDs)
  – introduction, operations, vectors, matrices, performance, ...
Representations of Boolean formulas

- Propositional formula: $f = (x_1 \lor x_2) \land x_3$

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth table

Binary decision tree

Binary decision diagram
Binary decision trees

- Graphical representation of Boolean functions
  - \( f(x_1, \ldots, x_n) : \{0,1\}^n \rightarrow \{0,1\} \)
- Binary tree with two types of nodes
  - Non-terminal nodes
    - labelled with a Boolean variable \( x_i \)
    - two children: 1 (“then”, solid line) and 0 (“else”, dotted line)
  - Terminal nodes (or “leaf” nodes)
    - labelled with 0 or 1
- To read the value of \( f(x_1, \ldots, x_n) \)
  - start at root (top) node
  - take “then” edge if \( x_i = 1 \)
  - take “else” edge if \( x_i = 0 \)
  - result given by leaf node
Binary decision diagrams

- **Binary decision diagrams (BDDs)** [Bry86]
  - based on binary decision trees, but reduced and ordered
  - sometimes called reduced ordered BDDs (ROBDDs)
  - actually directed acyclic graphs (DAGs), not trees
  - compact, canonical representation for Boolean functions

- **Variable ordering**
  - a BDD assumes a fixed total ordering over its set of Boolean variables
  - e.g. $x_1 < x_2 < x_3$
  - along any path through the BDD, variables appear at most once each and always in the correct order
BDD reduction rule 1

- Rule 1: Merge identical terminal nodes

- Example:
BDD reduction rule 2

• Rule 2: Merge isomorphic nodes, redirect incoming nodes

• Example:
BDD reduction rule 3

- Rule 3: Remove redundant nodes (with identical children)

- Example:
Canonicity

- BDDs are a canonical representation for Boolean functions
  - two Boolean functions are equivalent if and only if the BDDs which represent them are isomorphic
  - uniqueness relies on: reduced BDDs, fixed variable ordered

- Important implications for implementation efficiency
  - can be tested in linear (or even constant) time
BDD variable ordering

- BDD size can be very sensitive to the variable ordering
- example: \( f = (x_1 \land y_1) \lor (x_2 \land y_2) \lor (x_3 \land y_3) \)

Diagram 1: \( x_1 < y_1 < x_2 < y_2 < x_3 < y_3 \)
- \( 2n+2 \) nodes

Diagram 2: \( x_1 < x_2 < x_3 < y_1 < y_2 < y_3 \)
- \( 2^{n+1} \) nodes
BDDs – Some notation

• **Boolean functions**
  – for a BDD $A$, the function represented by $A$ is denoted $f_A$

• **Restriction**
  – for a BDD $A$, Boolean variable $x$ in $A$, and Boolean value $b$
  – $A|_{x=b}$ denotes the BDD representing the function $f_A$ restricted to the case where $x=b$
  – extends easily to multiple variables
  – $A|_{x_1=b_1,x_2=b_2} = (A|_{x_1=b_1})|_{x_2=b_2}$

• **Shannon’s Law: recursive expansion of BDDs**
  – let $x$ be the top–most Boolean variable in a BDD $A$
  – $f_A = \neg x \land f_{A|_{x=0}} \lor x \land f_{A|_{x=1}}$
Manipulating BDDs

- **Need efficient ways to manipulate Boolean functions**
  - while they are represented as BDDs
  - i.e. algorithms which are applied directly to the BDDs

- **Basic operations on Boolean functions:**
  - negation ($\neg$), conjunction ($\land$), disjunction ($\lor$), etc.
  - can all be applied directly to BDDs

- **Key operation on BDDs: Apply(op, A, B)**
  - where A and B are BDDs and op is a binary operator over Boolean values, e.g. $\land$, $\lor$, etc.
  - Apply(op, A, B) returns the BDD representing function $f_A \text{ op } f_B$
  - often just use infix notation, e.g. $\text{Apply}(\land, A, B) = A \land B$
The Apply operation

- **Apply**(op, A, B): recursive depth-first traversal of A and B
  - let x be the top-most variable in the two BDDs
  - reusing Shannon’s Law: we have the following as a basis:

\[
 f_A \text{ op } f_B = \neg x \land (f_{A|x=0} \text{ op } f_{B|x=0}) \lor x \land (f_{A|x=1} \text{ op } f_{B|x=1})
\]
Apply – Example

• Example: \( \text{Apply}(\lor, A, B) \)

Argument BDDs, with node labels:

Recursive calls to Apply:
Apply – Example

• Example: Apply(∨, A, B)
  – recursive call structure implicitly defines resulting BDD
Apply – Example

- **Example: Apply(∨, A, B)**
  - but the resulting BDD needs to be reduced
  - in fact, we can do this as part of the recursive Apply operation, implementing reduction rules bottom-up
More on BDD operations

• **Complexity for the Apply operator**
  - \( C = \text{Apply}(\text{op}, A, B) \)
  - \(|C| = \text{size of BDD } C = \text{number of nodes} = O( |A| \cdot |B| )\)
  - since at most one recursive call for each pair of nodes
  - for a good implementation, time complexity is also \(|A| \cdot |B|\)

• **Quantification (\( \exists, \forall \)) over Boolean variables**
  - can be computed in terms of restriction
  - for Boolean variable \( x \) and BDD \( A \):
    \[ \exists x.A \equiv A|_{x=0} \lor A|_{x=1} \]
  - extends easily to multi-variable quantification
  - \[ \exists(x_1,x_2,\ldots,x_n).A \equiv \exists x_1.(\exists x_2.\ldots(\exists x_n.A)) \]
Implementation of BDDs

- **Store all BDDs currently in use as one multi-rooted BDD**
  - no duplicate BDD subtrees, even across multiple BDDs
  - every time a new node is created, check for existence first
  - sometimes called the “**unique table**”
  - implemented as set of **hash tables**, one per Boolean variable
  - need: node referencing/dereferencing, garbage collection

- **Efficiency implications**
  - very **significant memory savings**
  - trivial checking of BDD equality (pointer comparison)

- **Caching of BDD operation results for reuse**
  - store result of every BDD operation (memory dependent)
  - applied at every step of recursive BDD operations
  - relies on fast check for BDD equality
BDDs to represent sets of states

• Consider a state space $S$ and some subset $S' \subseteq S$

• We can represent $S'$ by its characteristic function $\chi_{S'}$.
  – $\chi_{S'} : S \rightarrow \{0,1\}$ where $\chi_{S'}(s) = 1$ if and only if $s \in S'$

• Assume we have an encoding of $S$ into $n$ Boolean variables
  – this is always possible for a finite set $S$
  – e.g. enumerate the elements of $S$ and use a binary encoding
  – (note: there may be more efficient encodings though)

• So $\chi_{S'}$ can be seen as a function $\chi_{S'}(x_1, \ldots x_n) : \{0,1\}^n \rightarrow \{0,1\}$
  – which is simply a Boolean function
  – which can therefore be represented as a BDD
BDD and sets of states – Example

- State space $S$: \{0, 1, 2, 3\}
- Encoding of $S$: \{000, 001, 010, 011, 100, 101, 110, 111\}
- Subset $S' \subseteq S$: \{011, 101, 111\}

Truth table:

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

BDD:
Set operations with BDDs

- Set operations can be expressed in terms of Boolean operations on the characteristic functions of sets
  - for sets A and B, represented by BDDs A and B

- Set union: $A \cup B$, in BDDs: $A \lor B$
  - $x_{A\cup B}(s) = x_A(s) \lor x_B(s)$

- Set intersection: $A \cap B$, in BDDs: $A \land B$
  - $x_{A\cap B}(s) = x_A(s) \land x_B(s)$

- Set complement: $S \setminus A$, in BDDs: $\neg A$
  - $x_{S\setminus A}(s) = \neg x_A(s)$
BDDs and transition relations

• Transition relations can also be represented by their characteristic function, but over pairs of states
  – relation: \( R \subseteq S \times S \)
  – characteristic function: \( \chi_R : S \times S \rightarrow \{0,1\} \)

• For an encoding of state space \( S \) into \( n \) Boolean variables
  – we have Boolean function \( f_R(x_1,...,x_n,y_1,...,y_n) : \{0,1\}^{2n} \rightarrow \{0,1\} \)
  – which can be represented by a BDD

• Row and column variables
  – for efficiency reasons, we interleave the row variables \( x_1,..,x_n \) and column variables \( y_1,...,y_n \)
  – i.e. we use function \( f_R(x_1,y_1,...,x_n,y_n) : \{0,1\}^{2n} \rightarrow \{0,1\} \)
BDDs and transition relations

- Example:
  - 4 states: 0, 1, 2, 3
  - Encoding: 0→00, 1→01, 2→10, 3→11

<table>
<thead>
<tr>
<th>Transition</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$x_1y_1x_2y_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>(0,2)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0100</td>
</tr>
<tr>
<td>(1,0)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0010</td>
</tr>
<tr>
<td>(2,3)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1101</td>
</tr>
<tr>
<td>(3,1)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1011</td>
</tr>
<tr>
<td>(3,2)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1110</td>
</tr>
</tbody>
</table>
Forward image

• Fundamental operation for model checking
  – for set of states $S$, transition relation $R \subseteq S \times S$, subset $T \subseteq S$, $\text{Image}(T)$ is the set of states that can be reached from $T$ in one step
  
• Express in terms of Boolean functions over states
  – $T : S \rightarrow \{0,1\}$, $R : S \times S \rightarrow \{0,1\}$, $\text{Image}_T : S \rightarrow \{0,1\}$
  – $\text{Image}_T(s') = \exists s . T(s) \land R(s,s')$

• For an encoding of state space $S$ into $n$ Boolean variables
  – express in terms of Boolean functions over Boolean variables
  – row variables $x_1,..,x_n$ and column variables $y_1,..,y_n$
  – $\text{Image}_T(y_1,..,y_n) = \exists(x_1,..,x_n) . T(x_1,..,x_n) \land R(x_1,..,x_n, y_1,..,y_n)$

• Translate directly into BDDs
  – $\text{Image}_T = \exists(x_1,..,x_n).T \land R$
Reachability

• Basic breadth-first search algorithm to compute the set of reachable states
  – inputs: initial state $s_{init}$, transition relation $R$ (in fact, Image)
  – output: set $T$ of all states reachable from $s_{init}$ in $R$

```
done = false
T = \{ s_{init} \}
while (done == false)
    T' = T \cup \text{Image}(T)
    if (T' == T) done = true
    T = T'
endwhile
return T
```
Reachability with BDDs

- Translate directly into BDD operations:
  - inputs: BDD $\text{init}$ for set $\{s_{\text{init}}\}$, BDD $R$ for transition relation
  - output: BDD $T$ representing all reachable states

```plaintext
done = false
T = init
while (done == false)
  T’ = T ∨ ∃(x_1,..,x_n).T ∧ R
  if (T’ == T) done = true
  T = T’
endwhile
return T
```

Easy thanks to canonicity of BDDs
Overview

- Implementation of probabilistic model checking
  - overview, key operations, symbolic vs. explicit

- Binary decision diagrams (BDDs)
  - introduction, operations, sets, transition relations, ...

- Multi-terminal BDDs (MTBDDs)
  - introduction, operations, vectors, matrices, performance, ...
Multi-terminal binary decision diagrams

- Multi-terminal BDDs (MTBDDs), sometimes called ADDs
  - extension of BDDs to represent real-valued functions
  - like BDDs, an MTBDD \( M \) is associated with \( n \) Boolean variables
  - MTBDD \( M \) represents a function \( f_M(x_1, \ldots, x_n) : \{0,1\}^n \rightarrow \mathbb{R} \)

For clarity, we omit the zero terminal node and any incoming edges.

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 3 \\
0 & 1 & 0 & 9 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 4 \\
1 & 0 & 1 & 4 \\
1 & 1 & 0 & 9 \\
1 & 1 & 1 & 0 \\
\end{array}
\]
Operations on MTBDDs

• The BDD operation Apply extends easily to MTBDDs

• For MTBDDs A, B and binary operation op over the reals:
  – Apply(op, A, B) returns the MTBDD representing $f_A \text{ op } f_B$
  – examples for op: $+, -, \times, \text{min, max, ...}$
  – often just use infix notation, e.g. $\text{Apply}(+, A, B) = A + B$

• BDDs are just an instance of MTBDDs
  – in this case, can use Boolean ops too, e.g. $\text{Apply}(\lor, A, B)$

• The recursive algorithm for implementing Apply on BDDs
  – can be reused for Apply on MTBDDs
Some other MTBDD operations

- **Threshold**($A$, $\sim$, $c$)
  - for MTBDD $A$, relational operator $op$ and bound $c \in \mathbb{R}$
  - converts MTBDD to BDD based on threshold $\sim c$
  - i.e. builds BDD representing function $f_A \sim c$
  - e.g. computing the underlying transition relation from the probability matrix of a DTMC: $R = \text{Threshold}(P, >, 0)$

- **Abstract**($op$, $\{x_1, \ldots, x_n\}$, $A$)
  - for MTBDD $A$, variables $\{x_1, \ldots, x_n\}$ and commutative/associative binary operator over reals $op$
  - analogue of existential/universal quantification for BDDs
  - e.g. $\text{Abstract}(+, \{x\}, A)$ constructs the MTBDD representing the function $f_{A|\cdot|x=0} + f_{A|\cdot|x=1}$
  - e.g. for BDD $A$: $\exists(x_1, \ldots, x_n).A \equiv \text{Abstract}(\lor, \{x_1, \ldots, x_n\}, A)$
MTBDDs to represent vectors

• In the same way that BDDs can represent sets of states...
  – MTBDDs can represent real-valued vectors over states $S$
  – e.g. a vector of probabilities $\text{Prob}(s, \psi)$ for each state $s \in S$
  – assume we have an encoding of $S$ into $n$ Boolean variables
  – then vector $\mathbf{v} : S \rightarrow \mathbb{R}$ is a function $f_{\mathbf{v}}(x_1,\ldots,x_n) : \{0,1\}^n \rightarrow \mathbb{R}$

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$i$</th>
<th>$f_{\mathbf{v}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

Vector $\mathbf{v}$

$[0,3,9,0,4,4,9,0]$
MTBDDs to represent matrices

- MTBDDs can be used to represent **real-valued matrices** indexed over a set of states $S$
  - e.g. the **transition probability/rate matrix** of a DTMC/CTMC

- **For an encoding of state space $S$ into $n$ Boolean variables**
  - a vector $v : S \rightarrow \mathbb{R}$ is a function $f_v(x_1,..,x_n) : \{0,1\}^n \rightarrow \mathbb{R}$
  - a matrix $M$ maps pairs of states to reals i.e. $M : S \times S \rightarrow \mathbb{R}$
  - this becomes: $f_M(x_1,..,x_n,y_1,..,y_n) : \{0,1\}^{2n} \rightarrow \mathbb{R}$

- **Row and column variables**
  - for efficiency reasons, we **interleave** the row variables $x_1,..,x_n$
    and column variables $y_1,..,y_n$
  - i.e. we use function $f_M(x_1,y_1,..,x_n,y_n) : \{0,1\}^{2n} \rightarrow \mathbb{R}$
Matrices and MTBDDs – Example

Matrix M

\[
\begin{bmatrix}
0 & 8 & 0 & 5 \\
2 & 0 & 0 & 5 \\
0 & 0 & 0 & 5 \\
0 & 0 & 2 & 0 \\
\end{bmatrix}
\]

Entry in M                               |  \(x_1\) |  \(x_2\) |  \(y_1\) |  \(y_2\) |  \(x_1y_1x_2y_2\) |  \(f_M\)  \\
---|---|---|---|---|---|---|---
(0,1) = 8                               | 0 | 0 | 0 | 1 | 0001 | 8 \\
(1,0) = 2                               | 0 | 1 | 0 | 0 | 0010 | 2 \\
(0,3) = 5                               | 0 | 0 | 1 | 1 | 0101 | 5 \\
(1,3) = 5                               | 0 | 1 | 1 | 1 | 0111 | 5 \\
(2,3) = 5                               | 1 | 0 | 1 | 1 | 1101 | 5 \\
(3,2) = 2                               | 1 | 1 | 1 | 0 | 1110 | 2 \\

MTBDD M
Matrices and MTBDDs – Recursion

- Descending one level in the MTBDD (i.e. setting $x_i = b$)
  - splits the matrix represented by the MTBDD in half
  - row variables ($x_i$) give horizontal split
  - column variables ($y_i$) give vertical split
Matrices and MTBDDs – Recursion

Matrix M

<table>
<thead>
<tr>
<th>Entry in M</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$x_1y_1x_2y_2$</th>
<th>$f_M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1) = 8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0001</td>
<td>8</td>
</tr>
<tr>
<td>(1,0) = 2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>(0,3) = 5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>(1,3) = 5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0111</td>
<td>5</td>
</tr>
<tr>
<td>(2,3) = 5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1101</td>
<td>5</td>
</tr>
<tr>
<td>(3,2) = 2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1110</td>
<td>2</td>
</tr>
</tbody>
</table>
Matrices and MTBDDs – Regularity

Matrix M

\[
\begin{bmatrix}
0 & 8 & 0 & 5 \\
2 & 0 & 0 & 5 \\
0 & 0 & 0 & 5 \\
0 & 0 & 2 & 0 \\
\end{bmatrix}
\]

Repeated submatrices

MTBDD M

Repeated submatrices

Shared MTBDD node

<table>
<thead>
<tr>
<th>Entry in M</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$x_1y_1x_2y_2$</th>
<th>$f_M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1) = 8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0001</td>
<td>8</td>
</tr>
<tr>
<td>(1,0) = 2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>(0,3) = 5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>(1,3) = 5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0111</td>
<td>5</td>
</tr>
<tr>
<td>(2,3) = 5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1101</td>
<td>5</td>
</tr>
<tr>
<td>(3,2) = 2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1110</td>
<td>2</td>
</tr>
</tbody>
</table>
Matrices and MTBDDs – Regularity

Matrix $M$

\[
\begin{bmatrix}
0 & 8 & 0 & 5 \\
2 & 0 & 0 & 5 \\
0 & 0 & 0 & 5 \\
0 & 0 & 2 & 0
\end{bmatrix}
\]

MTBDD $M$

<table>
<thead>
<tr>
<th>Entry in $M$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$x_1y_1x_2y_2$</th>
<th>$f_M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1) = 8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0001</td>
<td>8</td>
</tr>
<tr>
<td>(1,0) = 2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>(0,3) = 5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>(1,3) = 5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0111</td>
<td>5</td>
</tr>
<tr>
<td>(2,3) = 5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1101</td>
<td>5</td>
</tr>
<tr>
<td>(3,2) = 2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1110</td>
<td>2</td>
</tr>
</tbody>
</table>

Identical adjacent submatrices

MTBDD node removed
Matrices and MTBDDs – Sparseness

Matrix $M$

<table>
<thead>
<tr>
<th>Entry in $M$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$x_1y_1x_2y_2$</th>
<th>$f_M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1) = 8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0001</td>
<td>8</td>
</tr>
<tr>
<td>(1,0) = 2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>(0,3) = 5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>(1,3) = 5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0111</td>
<td>5</td>
</tr>
<tr>
<td>(2,3) = 5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1101</td>
<td>5</td>
</tr>
<tr>
<td>(3,2) = 2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1110</td>
<td>2</td>
</tr>
</tbody>
</table>

MTBDD $M$

Blocks of zeros

Edge goes straight to zero node
MTBDD matrix/vector operations

- **Pointwise addition/multiplication and scalar multiplication**
  - can be implemented with the *Apply operator*
  - Matrices: \( A + B \), MTBDDs: \( \text{Apply}(+, A, B) \)

- **Matrix–matrix multiplication \( A \cdot B \)**
  - can be expressed recursively based on 4-way matrix splits
    \[
    \begin{bmatrix}
    A_1 & A_2 \\
    A_3 & A_4
    \end{bmatrix} = \begin{bmatrix}
    B_1 & B_2 \\
    B_3 & B_4
    \end{bmatrix} \cdot \begin{bmatrix}
    C_1 & C_2 \\
    C_3 & C_4
    \end{bmatrix}
    \]
  - \( A_1 = B_1 \cdot C_1 + B_2 \cdot C_3 \), etc.
  - which forms the basis of an MTBDD implementation
  - various optimisations are possible

- **Matrix–matrix multiplication \( A \cdot v \) is done in similar fashion**
Sparse matrices

- Explicit data structure for matrices with many zero entries
  - assume a matrix $P$ of size $n \times n$ with $\text{nnz}$ non-zero elements
  - store three arrays: $\text{val}$ and $\text{col}$ (of size $\text{nnz}$) and $\text{row}$ (of size $n$)
  - for each matrix entry $(r,c)=v$, $c$ and $v$ are stored in $\text{col}/\text{val}$
  - entries are grouped by row, with pointers stored in $\text{row}$
  - also possible to group by column

\[
\begin{bmatrix}
\cdot & 0.5 & \cdot & 0.5 \\
\cdot & 0.3 & \cdot & 1 \\
0.3 & \cdot & \cdot & 0.7 \\
1 & \cdot & \cdot & \cdot
\end{bmatrix}
\]

<table>
<thead>
<tr>
<th>val</th>
<th>0.5</th>
<th>0.5</th>
<th>1</th>
<th>0.3</th>
<th>0.7</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>row</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
Sparse matrices

• **Advantages**
  – *compact storage* (proportional to number of non-zero entries)
  – *fast access* to matrix entries
  – especially if usually need an entire row at once
  – (which is the case for e.g. matrix–vector multiplication)

• **Disadvantage**
  – less efficient to manipulate (i.e. add/delete matrix entries)

• **Storage requirements**
  – for a matrix of size $n \times n$ with $nnz$ non-zero elements
  – assume reals are 8 byte doubles, indices are 4 byte integers
  – we need $8 \cdot nnz + 4 \cdot nnz + 4 \cdot n = 12 \cdot nnz + 4 \cdot n$ bytes
Sparse matrices vs. MTBDDs

- Storage requirements
  - MTBDDs: each node is 20 bytes
  - sparse matrices: $12 \cdot \text{nnz} + 4 \cdot n$ bytes ($n$ states, $\text{nnz}$ transitions)

- Case study: Kanban manufacturing system, $N$ jobs
  - store transition rate matrix $R$ of the corresponding CTMCs

<table>
<thead>
<tr>
<th>$N$</th>
<th>States ($n$)</th>
<th>Transitions ($\text{nnz}$)</th>
<th>MTBDD (KB)</th>
<th>Sparse matrix (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>58,400</td>
<td>446,400</td>
<td>48</td>
<td>5,459</td>
</tr>
<tr>
<td>4</td>
<td>454,475</td>
<td>3,979,850</td>
<td>96</td>
<td>48,414</td>
</tr>
<tr>
<td>5</td>
<td>2,546,432</td>
<td>24,460,016</td>
<td>123</td>
<td>296,588</td>
</tr>
<tr>
<td>6</td>
<td>11,261,376</td>
<td>115,708,992</td>
<td>154</td>
<td>1,399,955</td>
</tr>
<tr>
<td>7</td>
<td>41,644,800</td>
<td>450,455,040</td>
<td>186</td>
<td>5,441,445</td>
</tr>
<tr>
<td>8</td>
<td>133,865,325</td>
<td>1,507,898,700</td>
<td>287</td>
<td>13,193,599</td>
</tr>
</tbody>
</table>
Implementation in PRISM

• PRISM is a **symbolic** probabilistic model checker
  – the key underlying data structures are MTBDDs (and BDDs)

• In fact, has multiple numerical computation engines
  – **MTBDDs**: storage/analysis of very large models (given structure/regularity), numerical computation can blow up
  – **Sparse matrices**: fastest solution for smaller models (<10^6 states), prohibitive memory consumption for larger models
  – **Hybrid**: combine MTBDD storage with explicit storage, ten-fold increase in analysable model size (~10^7 states)
Summing up…

- **Implementation of probabilistic model checking**
  - graph-based algorithms, e.g. reachability, precomputation
  - manipulation of sets of states, transition relations
  - iterative numerical computation
  - key operation: matrix-vector multiplication

- **Binary decision diagrams (BDDs)**
  - representation for Boolean functions
  - efficient storage/manipulation of sets, transition relations

- **Multi-terminal BDDs (MTBDDs)**
  - extension of BDDs to real-valued functions
  - efficient storage/manipulation of real-valued vectors, matrices (assuming structure and regularity)
  - can be much more compact than (explicit) sparse matrices