**Implementation for big state spaces**

- Like binary decision diagrams (BDDs) can be used in implementing algorithms that use strong/weak preimages, there are data structures that can be used for implementing probabilistic algorithms for big state spaces.
- Problem: algorithms do not use just sets and relations, but value functions $v: S \rightarrow \mathbb{R}$ and non-binary transition matrices.
- Solution: Use a generalization of BDDs called algebraic decision diagrams (or MTBDDs: multi-terminal BDDs.)

**Algebraic decision diagrams**

- Graph representation of functions from $\{0,1\}^n \rightarrow \mathbb{R}$ that generalizes BDDs (BDDs are functions $\{0,1\}^n \rightarrow \{0,1\}$)
- Every BDD is an ADD.
- Canonicity: Two ADDs describe the same function if and only if they are the same ADD.
- Applications: Computations on very big matrices including computing steady-state probabilities of Markov chains; probabilistic verification; AI planning

**An algebraic decision diagram**

ADD represents a mapping $ABA'B' \rightarrow \mathbb{R}$

<table>
<thead>
<tr>
<th>$A'B'$</th>
<th>$A'B'$</th>
<th>$A'B'$</th>
<th>$A'B'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AB$</td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>00</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>01</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>10</td>
<td>0.8</td>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td>11</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Operations on ADDs: sum, product, maximum, ...**

Arithmetic operations as $(f \odot g)(x) = f(x) \odot g(x)$ for every $x$.

<table>
<thead>
<tr>
<th>$ABC$</th>
<th>$f$</th>
<th>$g$</th>
<th>$f + g$</th>
<th>$\max(f, g)$</th>
<th>$7 \cdot f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>010</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>011</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>101</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>110</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>111</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>21</td>
</tr>
</tbody>
</table>
Operations on ADDs: sum

\[
\begin{array}{c}
\begin{array}{c}
A \\
B \\
C
\end{array} \\
\begin{array}{c}
3 \\
2 \\
1
\end{array}
\end{array} +
\begin{array}{c}
\begin{array}{c}
A \\
B \\
C
\end{array} \\
\begin{array}{c}
3 \\
2 \\
1
\end{array}
\end{array} =
\begin{array}{c}
\begin{array}{c}
A \\
B \\
C
\end{array} \\
\begin{array}{c}
3 \\
2 \\
1
\end{array}
\end{array}
\]

Operations on ADDs: maximum

\[
\begin{array}{c}
\begin{array}{c}
A \\
B \\
C
\end{array} \\
\begin{array}{c}
3 \\
2 \\
1
\end{array}
\end{array} \max
\begin{array}{c}
\begin{array}{c}
A \\
B \\
C
\end{array} \\
\begin{array}{c}
3 \\
2 \\
1
\end{array}
\end{array} =
\begin{array}{c}
\begin{array}{c}
A \\
B \\
C
\end{array} \\
\begin{array}{c}
3 \\
2 \\
1
\end{array}
\end{array}
\]

Operations on ADDs: arithmetic \( \exists \) abstraction

\[(\exists p.f)(x) = (f[\top/p])(x) + (f[\bot/p])(x)\]

<table>
<thead>
<tr>
<th>ABC</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>1</td>
</tr>
<tr>
<td>011</td>
<td>2</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>101</td>
<td>2</td>
</tr>
<tr>
<td>110</td>
<td>2</td>
</tr>
<tr>
<td>111</td>
<td>3</td>
</tr>
</tbody>
</table>

\begin{array}{c}
\begin{array}{c}
A \\
B \\
C
\end{array}
\end{array} \exists C. f

Matrix multiplication with ADDs (I)

Consider matrices \( M_1 \) and \( M_2 \), represented as mappings:

\[
\begin{pmatrix}
1 & 2 \\
3 & 4
\end{pmatrix}
\begin{pmatrix}
1 & 2 \\
2 & 1
\end{pmatrix}
\]

\[
\begin{array}{c|c|c}
AA' & M_1 & AA'' M_2 \\
00 & 1 & 00 & 1 \\
01 & 2 & 01 & 2 \\
10 & 3 & 10 & 2 \\
11 & 4 & 11 & 1 \\
\end{array}
\]
**Matrix multiplication with ADDs (II)**

<table>
<thead>
<tr>
<th>$AA' A''$</th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_1 \cdot M_2$</th>
<th>$AA' \cdot (M_1 \cdot M_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>00</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>01</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>011</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>100</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>4</td>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

$\text{Translation of nondet. operators to ADDs}$

Operator $o = (c, e)$ in NF1 is translated to $T_o = c \land \text{PL}_P(e)$.

Nondeterministic choice and outermost conjunctions are by arithmetic sum and multiplication.

$$\text{PL}_B(e) = \begin{cases} \text{when } e \text{ is deterministic} \\ \text{translated like in Lecture 6, but restricted} \\ \text{to state variables in the set } B \end{cases}$$

$$\text{PL}_B(p_1e_1 \cdots | p_n e_n) = p_1 \text{PL}_B(e_1) + \cdots + p_n \text{PL}_B(e_n)$$

$$\text{PL}_B(e_1 \land \cdots \land e_n) = \text{PL}_{B \setminus (B_1 \cup \cdots \cup B_n)}(e_1) \cdot \text{PL}_B(e_2) \cdot \ldots \cdot \text{PL}_{B_n}(e_n)$$

where $B_i = \text{changes}(e_i)$ for all $i \in \{1, \ldots, n\}$

**Implementation of Value Iteration with ADDs**

- Start from $\langle P, I, O, R, \emptyset \rangle$.
- Propositions in ADDs $P$ and $P' = \{p'|p \in P\}$.
- Construct transition matrix ADDs from all $o \in O$ (next slide).
- Construct ADDs for representing reward functions $R(o), o \in O$.
- Functions $v^i$ are ADDs that map valuations of $P$ to $R$.
- All computation is for all states (one ADD) simultaneously: big speed-ups possible.

**Translation of reward functions to ADDs**

For $o = (c, e) \in O$ reward $R(o) = \{\langle \phi_1, r_1 \rangle, \ldots, \langle \phi_n, r_n \rangle\}$.

Reward ADD $R_o$ maps each state to a real number.

Construct the BDDs for $\phi_1, \ldots, \phi_n$ and multiply with the respective rewards:

$$R_o = r_1 \cdot \phi_1 + \cdots + r_n \cdot \phi_n = \infty \cdot \neg c$$
The Value Iteration algorithm: without ADDs

1. Assign \( n := 0 \) and (arbitrary) initial values to \( v^0(s) \) for all \( s \in S \).

2. 

\[
v^{n+1}(s) := \max_{a \in A(s)} \left( R(s, a) + \sum_{s' \in S} \lambda p(s'|s, a)v^n(s') \right)
\]

for every \( s \in S \)

If \( |v^{n+1}(s) - v^n(s)| < \frac{\epsilon(1-\lambda)}{2\lambda} \) for all \( s \in S \) then stop.
Otherwise, set \( n := n + 1 \) and repeat step 2.

---

The Value Iteration algorithm: with ADDs

Backup step for \( v^{n+1} \) as product of \( T_o \) and \( v^n \):

\[
\begin{pmatrix}
A'B' & A'B' & A'B' & A'B'\\
00 & 01 & 10 & 11
\end{pmatrix} \left( \begin{pmatrix}
A'B' \\
00 \\
01 \\
10 \\
11
\end{pmatrix} \right) = v^n
\]

Notice: The fact that the operator is not applicable in 11 is handled by having the immediate reward \(-\infty\) in that state.

---

The Value Iteration algorithm: with ADDs

1. Assign \( n := 0 \) and let \( v^n \) be an ADD that is constant 0.

2.

\[
v^{n+1} := \max_{(c,c') = o \in O} (R_o + \lambda \cdot \exists P'.(T_o \cdot (v^n[P'/P'])))
\]

(Unsatisfied preconditions are handled by the immediate rewards \(-\infty\).)

If all terminal nodes of ADD \( |v^{n+1} - v^n| \) are < \( \frac{\epsilon(1-\lambda)}{2\lambda} \) then stop.
Otherwise, set \( n := n + 1 \) and repeat step 2.