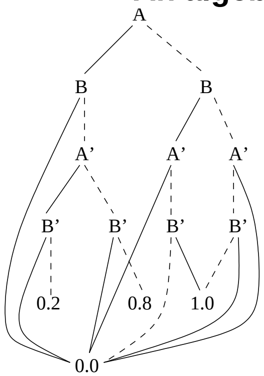
Implementation for big state spaces

- 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 $v : S \rightarrow \mathcal{R}$ 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 $\{0,1\}^n \rightarrow \mathcal{R}$ 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'$\rightarrow \mathcal{R}$

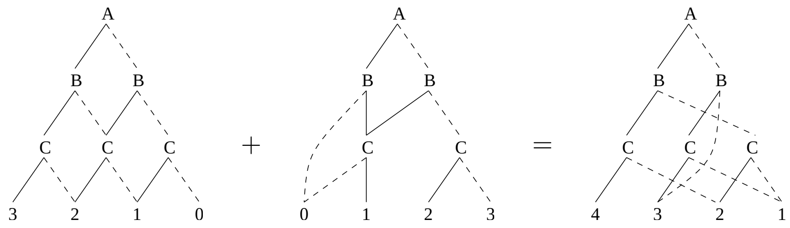
	$A'B'$	$A'B'$	$A'B'$	$A'B'$
AB	00	01	10	11
00	1.0	0	0	0
01	0	1.0	0	0
10	0.8	0	0.2	0
11	0	0	0	0

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

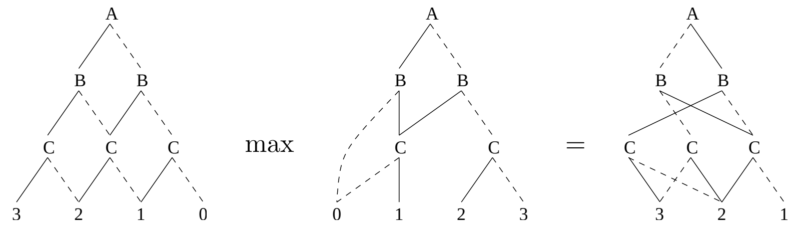
Arithmetic operations as $(f \circledcirc g)(x) = f(x) \circledcirc g(x)$ for every x.

ABC	f	g	$f+g$	$\max(f,g)$	$7 \cdot f$
000	0	3	3	3	0
001	1	2	3	2	7
010	1	0	1	1	7
011	2	1	3	2	14
100	1	0	1	1	7
101	2	0	2	2	14
110	2	0	2	2	14
111	3	1	4	3	21

Operations on ADDs: sum



Operations on ADDs: maximum



Operations on ADDs: arithmetic ∃ abstraction

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

ABC	f
000	0
001	1
010	1
011	2
100	1
101	2
110	2
111	3

$\exists C.f$ is obtained by summing $f(x)$ and $f(x')$ when x and x' differ only on C:

AB	$\exists C.f$
00	1
01	3
10	3
11	5

Matrix multiplication with ADDs (I)

Consider matrices M_1 and M_2, represented as mappings:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

AA'	M_1
00	1
01	2
10	3
11	4

$A'A''$	M_2
00	1
01	2
10	2
11	1

Matrix multiplication with ADDs (II)

$AA'A''$	M_1	M_2	$M_1 \cdot M_2$
000	1	1	1
001	1	2	2
010	2	2	4
011	2	1	2
100	3	1	3
101	3	2	6
110	4	2	8
111	4	1	4

AA''	$\exists A'.(M_1 \cdot M_2)$
00	5
01	4
10	11
11	10

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 5 & 4 \\ 11 & 10 \end{pmatrix}$$

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 $\mathcal{R}$.
- All computation is for all states (one ADD) simultaneously: big speed-ups possible.

Translation of nondet. operators to ADDs

Operator $o = \langle c, e \rangle$ in NF1 is translated to $T_o = c \wedge \mathrm{PL}_P(e)$.

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

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

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

$\mathrm{PL}_B(e_1 \wedge \cdots \wedge e_n) = \mathrm{PL}_{B \setminus (B_2 \cup \cdots \cup B_n)}(e_1) \cdot \mathrm{PL}_{B_2}(e_2) \cdot \ldots \cdot \mathrm{PL}_{B_n}(e_n)$ where $B_i = \mathit{changes}(e_i)$ for all $i \in \{1, \ldots, n\}$

Translation of reward functions to ADDs

For $o = \langle c, e \rangle \in O$ 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. $$v^{n+1}(s) := \max_{a \in A(s)} \left(R(s,a) + \sum_{s' \in S} \lambda p(s'|s,a) v^n(s') \right) \text{ for every } s \in S$$

   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:

AB \ $A'B'$	00	01	10	11
00	**1.0**	**0**	**0**	**0**
01	**0**	**1.0**	**0**	**0**
10	**0.2**	**0**	**0.8**	**0**
11	**0**	**0**	**0**	**0**

$A'B'$	v^n
00	**−5.1**
01	**2.8**
10	**10.2**
11	**3.7**

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

   (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.