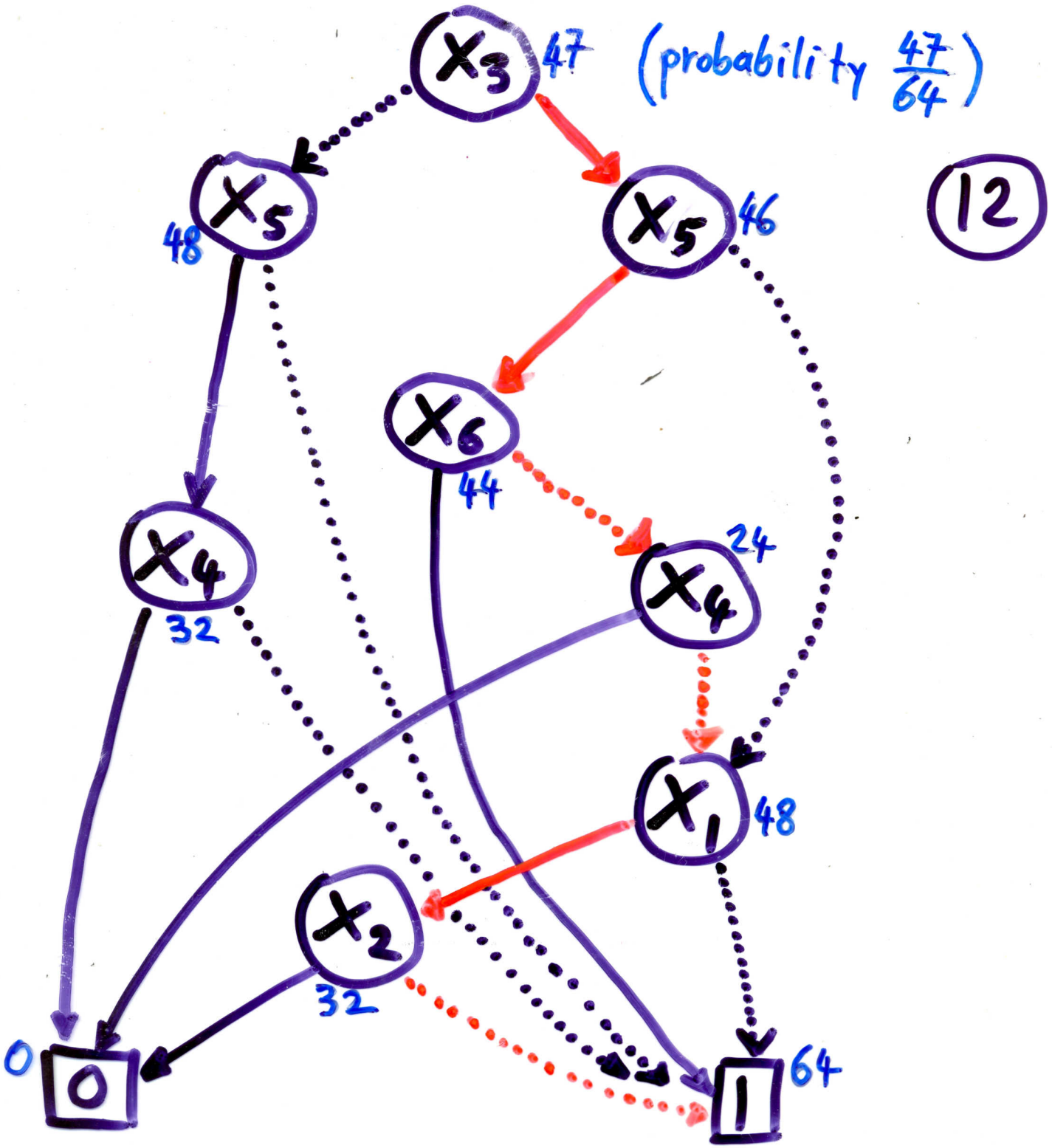
C) Binary decision diagrams

Binary decision diagrams (BDD's, Bryant 1986) are a kind of Boolean decision trees which have significant advantages (e.g. are more easily merged) upon previous related approaches. The underlie the MATHEMATICA 8.0 command SatisfiabilityCount which the author has used extensively. Donald Knuth has fallen in love with BDD's recently.

Here comes a BDD that handles the Boolean function $b: \{0,1\}^6 \to \{0,1\}$ which is the conjunction of the two implications previously considered (Ganter's algorithm, impl. n-algorithm):



$$b(x) := (x_1 \wedge x_2 \wedge x_3 \rightarrow x_5 \wedge x_6) \wedge (x_4 \wedge x_5 \rightarrow x_3 \wedge x_6)$$

$$b(1,0,1,0,1,0) = 1 \quad \checkmark$$

Pros and cons of BDD's (13)

The number N of models of $b(x)$ can be calculated in time linear in the size of the BDD (indep. of N!).

One can also generate all models of $b(x)$. However this

a) results in a less compact representation than with the implication n-alg. (or POE) since only $\{0,1,2\}$-valued rows arise.

b) is a detour compared to the POE where $b(x)$ is translated directly into a disjoint union of multivalued rows.

The biggest drawback of BDD's is the following. The only way (at least for MATHEMATICA) to set up $b: \{0,1\}^9 \to \{0,1\}$ which has as models exactly the bitstrings of weight 3 (say) is to put

$$b(x) ::= (x_1 \wedge x_2 \wedge x_3 \wedge \bar{x}_4 \wedge \bar{x}_5 \wedge \cdots \wedge \bar{x}_9)$$
$$\vee (x_1 \wedge x_2 \wedge x_4 \wedge \bar{x}_3 \wedge \bar{x}_5 \wedge \cdots \wedge \bar{x}_9)$$
$$\vee \cdots \vee (\bar{x}_1 \wedge \bar{x}_2 \wedge \cdots \wedge \bar{x}_6 \wedge x_7 \wedge x_8 \wedge x_9)$$

This is a very clumsy way to impose cardinality constraints, and that's why BDD's cannot compete with POE in this regard. For instance, consider calculating the number of all, resp. fixed-cardinality, transversals of random (w,h,d)-hypergraphs:

(w,h,d)	K	e-algorithm K-time	time	SatisfiabilityCount K-time	time
30,100,15	5	34	97	17	0.3
1000	5	68	30264	68	12
100000	5	67	—	5402	1294
60,30000,3	0	—	23906	—	—
5000,30, 4800	0	—	122	—	125
5000,40, 4800	0	—	455	—	—

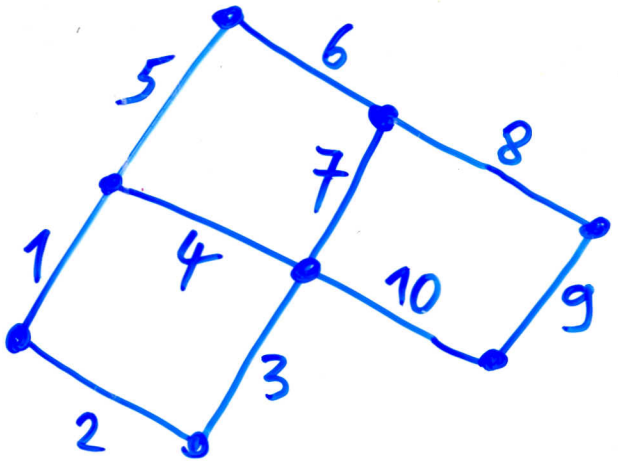
Six applications and variations of the implication n-algorithm

In the previous Theorem Horn formulae were presented as $\Sigma \cup \Theta$. In most applications one either deals exclusively with Σ or Θ. Here we encounter three of each kind.

I) Trees from circuits

For a connected graph G let W be its edge set and Θ its set of circuits (their edge sets).

For instance, consider

$G =$ 

The largest of the six circuits in $\ominus$ is $\{1,2,3,10,9,8,6,5\}^*$. Applying the noncover n-algorithm yields all $\ominus$-noncovers, i.e. all forests:

	1	2	3	4	5	6	7	8	9	10
$r_1 =$	n_1	n_1	n_1	1	n_2	n_2	0	n_2	n_2	n_2
$r_2 =$	n_1	n_1	n_1	1	n_2	n_2	1	n_3	n_3	n_3
$r_3 =$	n	n	n	0	n	n	0	n	n	n
$r_4 =$	n_1	n_1	n_1	0	n_1	n_1	1	n_2	n_2	n_2

The number of forests is

$$|r_1| + |r_2| + |r_3| + |r_4| = 7 \cdot 3 \cdot 7 + 7 \cdot 3 \cdot 7 + 255 + 3 \cdot 7 \cdot 3$$

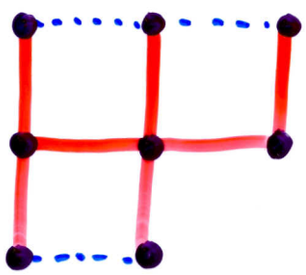
$$= \underline{\underline{696}}$$

The trees among the forests are (17
exactly the 7-element sets in each row.
They number

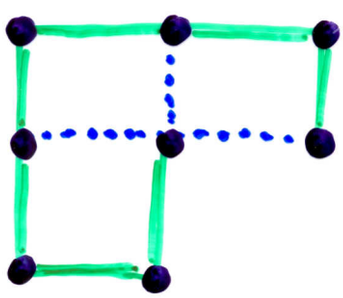
$$3\cdot 5 + 3\cdot 2\cdot 3 + 8 + 3\cdot 2\cdot 3 = \underline{\underline{59}}$$

Two concrete examples:

1	2	3	4	5	6	7	8	9	10	
n_1	n_1	n_1	1	n_2	n_2	1	n_3	n_3	n_3	$= r_2$
1	0	1	1	1	0	1	0	1	1	



1	2	3	4	5	6	7	8	9	10	
n	n	n	0	n	n	0	n	n	n	$= r_3$
1	1	1	0	1	1	0	1	1	0	



II) The anticliques of a graph

A clique in a graph G is a set of vertices any two of which are adjacent. The other extreme is an anticlique which is so that no two vertices are adjacent.

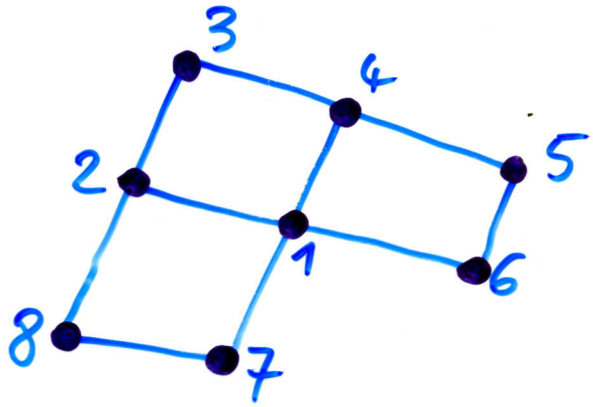
Let W be the vertex set of G and let Θ consist of certain 2-element subsets of W, i.e. the edges of G. Then the Θ-noncovers are exactly the anticliques X of G.

Rather than applying the noncover n-algorithm to all edges (the members of Θ), it would be more efficient to somehow process the much fewer vertices α. For this purpose let star(α) be the set of vertices β ($\neq \alpha$) which are adjacent to α. Each anticlique X that happens to contain α must not contain any $\beta \in$ star(α).

In other words, X must satisfy the "anti-implication" (19

$$\{\alpha\} \longrightarrow (\text{not star}(\alpha))$$

for each vertex α of G. To fix ideas, take G from section I but now label the vertices:



Let us sketch the $(\alpha, \bar{\beta})$-algorithm that generates all anticliques by imposing all anti-implications:

1	2	3	4	5	6	7	8	
α	$\bar{\beta}$	2	$\bar{\beta}$	2	$\bar{\beta}$	$\bar{\beta}$	2	impose $\{1\} \to \{\bar{2}, \bar{4}, \bar{6}, \bar{7}\}$
α	0	2	$\bar{\beta}$	2	$\bar{\beta}$	$\bar{\beta}$	2	impose $\{2\} \to \{\bar{1}, \bar{3}, \bar{8}\}$
0	1	0	2	2	2	2	0	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Counting all k-element anticliques of G yields the independence polynomial of G. The $(\alpha,\overline{\beta})$-algorithm easily adapts to find one (or all) maximum anticliques.

III) Calculating the subalgebra lattice from the given Cayley-tables

For simplicity consider groupoids $(W,*)$ but mutatis mutandis the sequel works for any finite universal algebra.
The subgroupoids of $(W,*)$ are exactly the Σ-closed subsets of W, where

$$\Sigma := \{\{a,b\} \to \{a*b, b*a\} : a,b \in W\}$$

For instance, a random commutative idempotent groupoid $(W,*)$ with $|W|=50$ had 101 subgroupoids (25 sec).

If all algebra operations are unary or binary, one can speed up things as follows.

Since $\Sigma_1 := \{ \{a\} \to \{a*a\} : a \in W \}$ is easily handled, and for ease of notation, we only focus on $\Sigma_2 := \Sigma \setminus \Sigma_1$.

Furthermore we suppose $(W,*)$ is commutative. Under these provisos partition Σ_2 as

$$\Sigma_2 = \dot{\bigcup} \{ \Sigma[c] : c \in W \}$$

where $\Sigma[c]$ consists of all $\{a,b\} \to \{a*b\}$ in Σ_2 with $a*b = c$.

By definition the graph $G[c]$ has as edges the premises of the implications in $\Sigma[c]$; its vertex set $V[c]$ is the union of these edges.

Fix $c \in W$. It is clear that a subset $X \subseteq W$ is $\trianglelefteq[c]$-closed if and only if one of these cases occurs:

i) $c \in X$

ii) $c \notin X$ and $X \cap V[c]$ is an anticlique of $G[c]$

Hence the fast (α, β)-algorithm from section II can be applied.