BAR ARCHITECTURE

William Bricken

July 1995

A possible silicon architecture for a dgraph would have a column of inputs, with two state-bits for the value of each literal (i.e. a on/off, not-a on/off). The value of each bit is determined by the variable's depth of nesting in the parens structure. There are four possibilities:

1. no occurrences (when register is off)	no logic
2. all occurrences at an odd depth	unate logic
3. all occurrences at an even depth	unate logic
4. occurrences at both odd and even depths	binate logic

Unate variables are quite simple to handle, they distribute upwards. Binate variables require NP algorithms.

In the MAJORITY example

```
((a b)(a c)(b c))
```

All variables are unate and at the same depth. Functionally equivalent forms maintain a unate/binate invariance.

```
((a ((b)(c))) (b c))
```

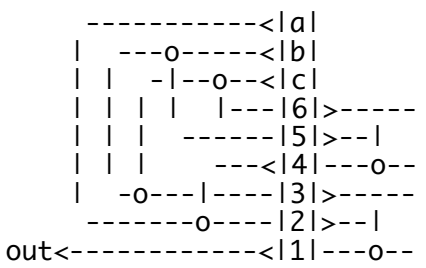
The bar-circuit is formed by marking a column of variables on the left when the variable is at an even depth, and on the right when the variable occurs at an odd depth:

```
((a ((b)(c))) (b c))
12  45  6     3
```

even		odd
1	a	0
1	b	0
1	c	0
0	6	1
0	5	1
1	4	0
0	3	1
0	2	1
1	1	0

The wiring mimics connectivity in the dgraph. The bar isolates the variables as well as representing an inverter when crossed. Numbers in the bar are

individual dnode ids. Wires on the left are from variables at even depth; wires on the right are odd depth variables. Wires join at an intersection "o".



This diagram is simply a distinction graph contorted so that there is only one distinction. In this model, computation occurs by dynamic routing at O-intersections. To get the connections to dynamically change, the crossover points must have some simple reconfigurable internal structure

Variable names are input sources. When a variable is bound, its value is propagated along the connective lines. When the value crosses the bar, it flips, or changes state.

The rule of connection reduction is occlusion.

Initialization Logic

Positive variable values propagate. Negative values disconnect.

The ROUTING-LOGIC is:

> If a 0 enters, disconnect the source
> If a 1 enters, disconnect the source and propagate
> (in all other possible directions).

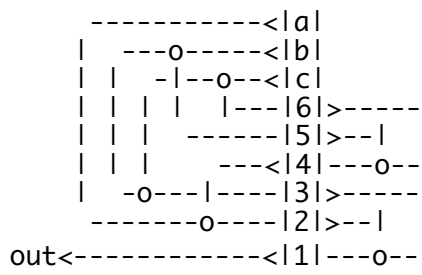
The BAR-LOGIC is:

> If any value enters,
> toggle the value for right-side leads (not for left side),
> propagate in all cases, and
> disconnect the variable source.
>
> If a 0 enters the bar (from a propagation),
> disconnect the source.
>
> If a 1 enters the bar (from a propagation),
> disconnect the source and propagate 0.

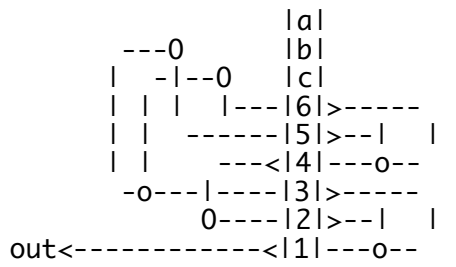
Example

Consider the binding (a 0)(b 1)(c 1)

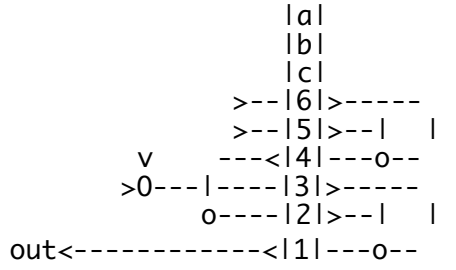
```
                ((a ((b)(c))) (b c))
                12  45  6     3
```



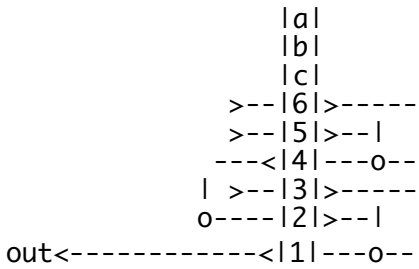
A 1 value comes out of the rhs of the bar from each of b and c. They disconnect the (former) variable while propagating. The 0-value lead from a is disconnected at the first 0-intersection (highlighted). Disconnection is represented by erasure.



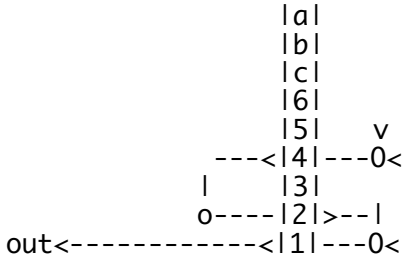
The first 0-nodes reached by the b and c propagation get the 1 input (also highlighted). Each disconnects and propagates the 1 value. Below, a pair of propagated 1-values converge on the highlighted node.



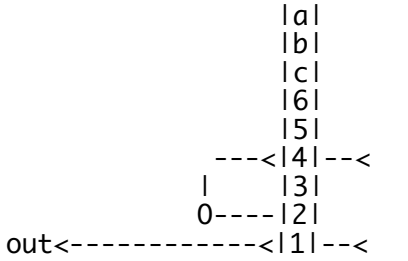
It too disconnects and propagates.



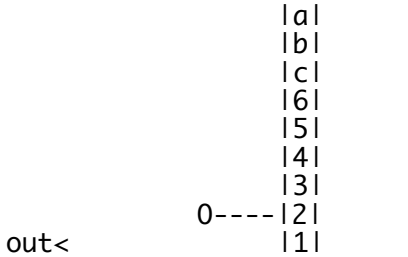
Now dnodes {3,5,6} are each receiving a 1 value at the bar. Since the entry is from the left, the values are not toggled. The right-hand-side of each propagates the 1 value until each reaches another 0-intersection (highlighted).



These disconnect and propagate the 1-value.



The 1-values entering the bar at 1 and 4 are from the right; they toggle. Exiting the bar as 0s, their left-side connections are erased.



The left-out gets 0 and disconnects from the computation.