Quantified Boolean Logic

Extension of propositional logic with boolean quantifiers.

We write

$\exists A\ F$	as an abbreviation of	$F[A/0] \vee F[A/1]$
$\forall A\ F$	as an abbreviation of	$F[A/0] \wedge F[A/1]$

Abbreviations can be nested, and then they are “unfolded” inside-out:

	$\exists A \forall B\ (A \wedge B)$
abbreviates	$\exists A\ (A \vee 0) \wedge (A \vee 1)$
which abbreviates	$((0 \vee 0) \wedge (0 \vee 1)) \vee ((1 \vee 0) \wedge (1 \vee 1))$

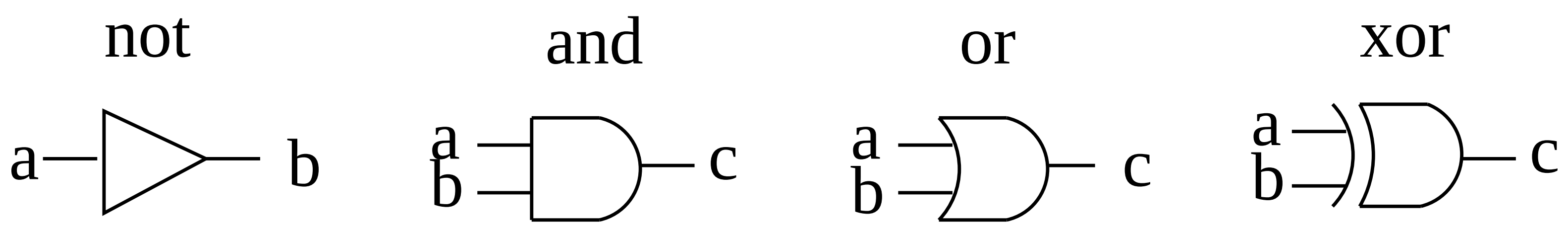
Intuitively, $\exists A \forall B\ (A \wedge B)$ “means”

> there exists a truth value for A such that for every truth value for B the formula $(A \wedge B)$ becomes true.

Modelling circuits with QBL

Gates as boolean formulas

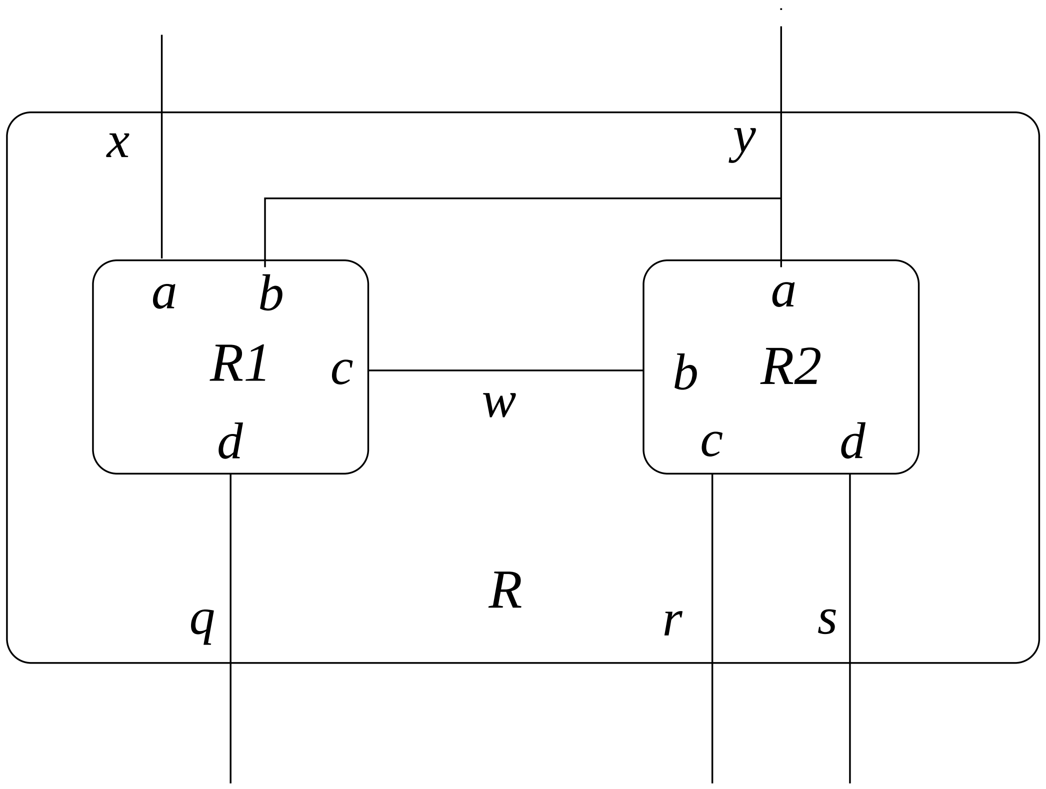
Stable states as satisfying truth assignments



$$\begin{aligned}
\textbf{not}(a,b) &\equiv \neg a \leftrightarrow b\\
\textbf{and}(a,b,c) &\equiv (a \wedge b) \leftrightarrow c\\
\textbf{or}(a,b,c) &\equiv (a \vee b) \leftrightarrow c\\
\textbf{xor}(a,b,c) &\equiv ((\neg a \wedge b) \vee (a \wedge \neg b)) \leftrightarrow c
\end{aligned}$$

Combining gates means combining formulas

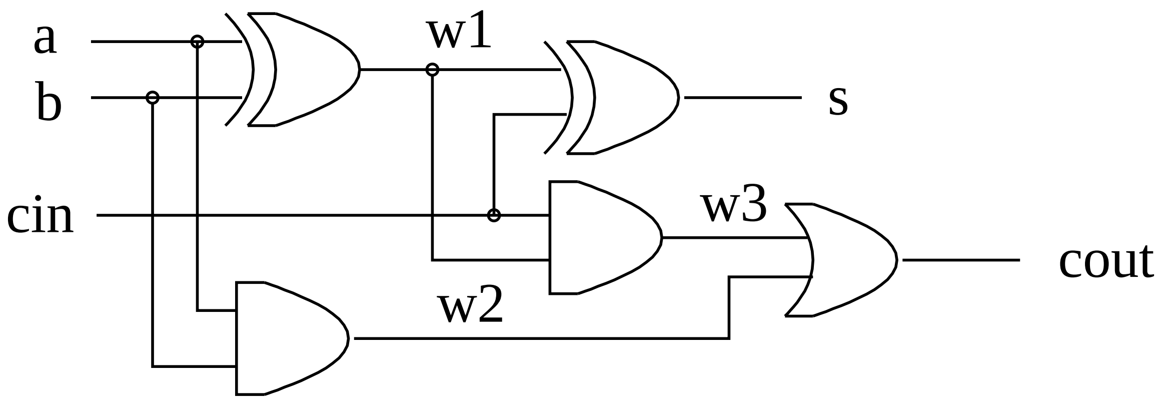
Combine gates with $\wedge$, $\exists$ (and renaming of atomic formulas)



$$R(x, y, q, r, s) = \exists w\, R_1(x, y, w, q) \wedge R_2(y, w, r, s)$$

A full adder

$$\begin{array}{cccc}
 & \color{red}{cout} & \color{red}{cin} & \ldots \\
 & \ldots & a & \ldots \\
+ & \ldots & b & \ldots \\
\hline
 & \ldots & s & \ldots
\end{array}$$

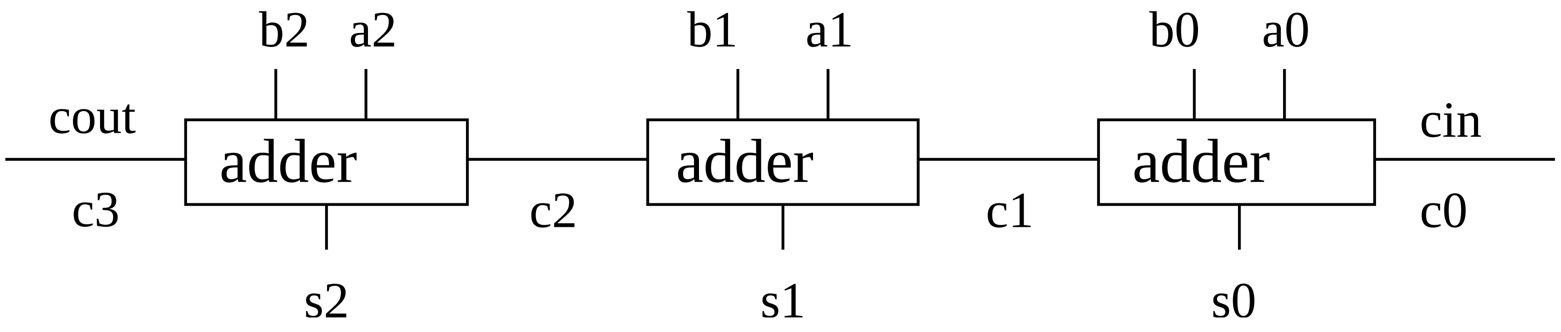


$$\mathbf{full\_adder}(a, b, s, cin, cout) \equiv$$
$$\exists w_1 \exists w_2 \exists w_3\ \mathbf{xor}(a, b, w_1) \wedge \mathbf{xor}(w_1, cin, s) \wedge \mathbf{and}(a, b, w_2) \wedge$$
$$\mathbf{and}(cin, w_1, w_3) \wedge \mathbf{or}(w_3, w_2, cout)$$

A n-bit ripple-carry adder

$$\begin{array}{ccccl} & c_2 & c_1 & cin & (=0) \\ & a_2 & a_1 & a_0 & \\ + & b_2 & b_1 & b_0 & \\ \hline cout & s_2 & s_1 & s_0 & \end{array}$$

Wire together n 1-bit adders where i-th carry-out is $i{+}1$-st carry-in, first carry is the carry-in and last is the carry-out.



We obtain the formula

$$\mathbf{adder}_n(a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}, s_0, \ldots, s_{n-1},\ cin, cout) \equiv$$
$$\exists c_0 \exists c_1 \ldots \exists c_n\, (c_0 \leftrightarrow cin) \wedge (c_n \leftrightarrow cout) \wedge$$
$$\bigwedge_{i=1}^{n-1} \mathbf{full\_adder}(a_i, b_i, s_i, c_i, c_{i+1}))$$

Problem: too slow. Each c_i can only be computed after all of $c_{i-1}, \ldots, c_0$ have been computed

Delay: $2n + 2$ time units for n-bit numbers

A carry-look-ahead n-adder

Compute all of $c_{n-1}, \ldots, c_0$ (and $cout$) concurrently

First step: given $a_{n-1} \ldots a_0$ and $b_{n-1} \ldots b_0$, identify the positions $i \in [0, n-1]$ that are

- Generating: $c_{i+1} \equiv 1$ independently of c_i.
  These are the positions such that $1 = g_i \equiv \mathbf{and}(a_i, b_i)$.
- Propagating: $c_{i+1} \equiv c_i$, i.e., c_i is ‘propagated’ to c_{i+1}.
  These are the positions such that $1 = p_i \equiv \mathbf{xor}(a_i, b_i)$

Observe: all g_i, p_i can be computed simultaneously

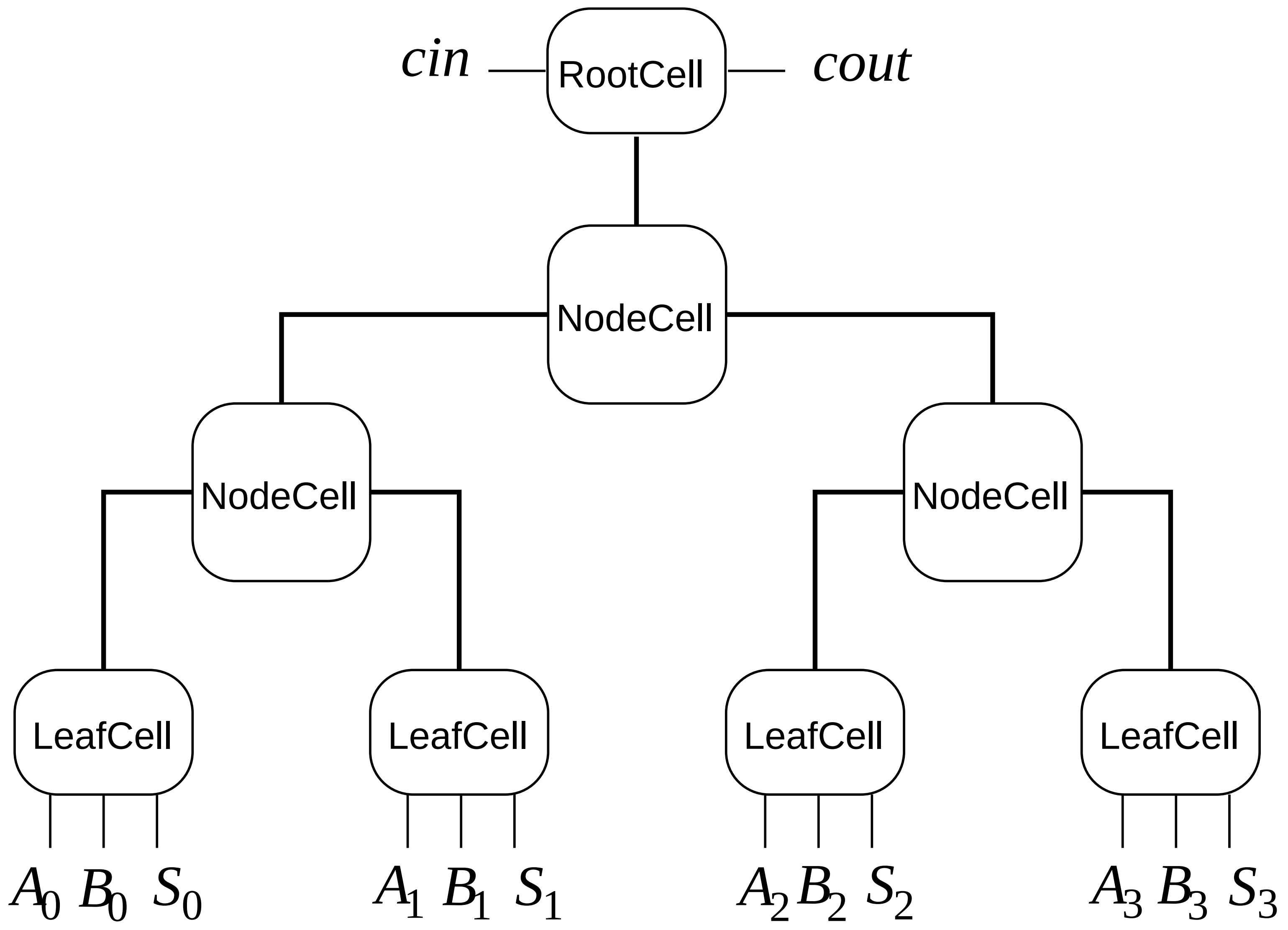
A carry-look-ahead n-adder

Second step: compute the c_i's by

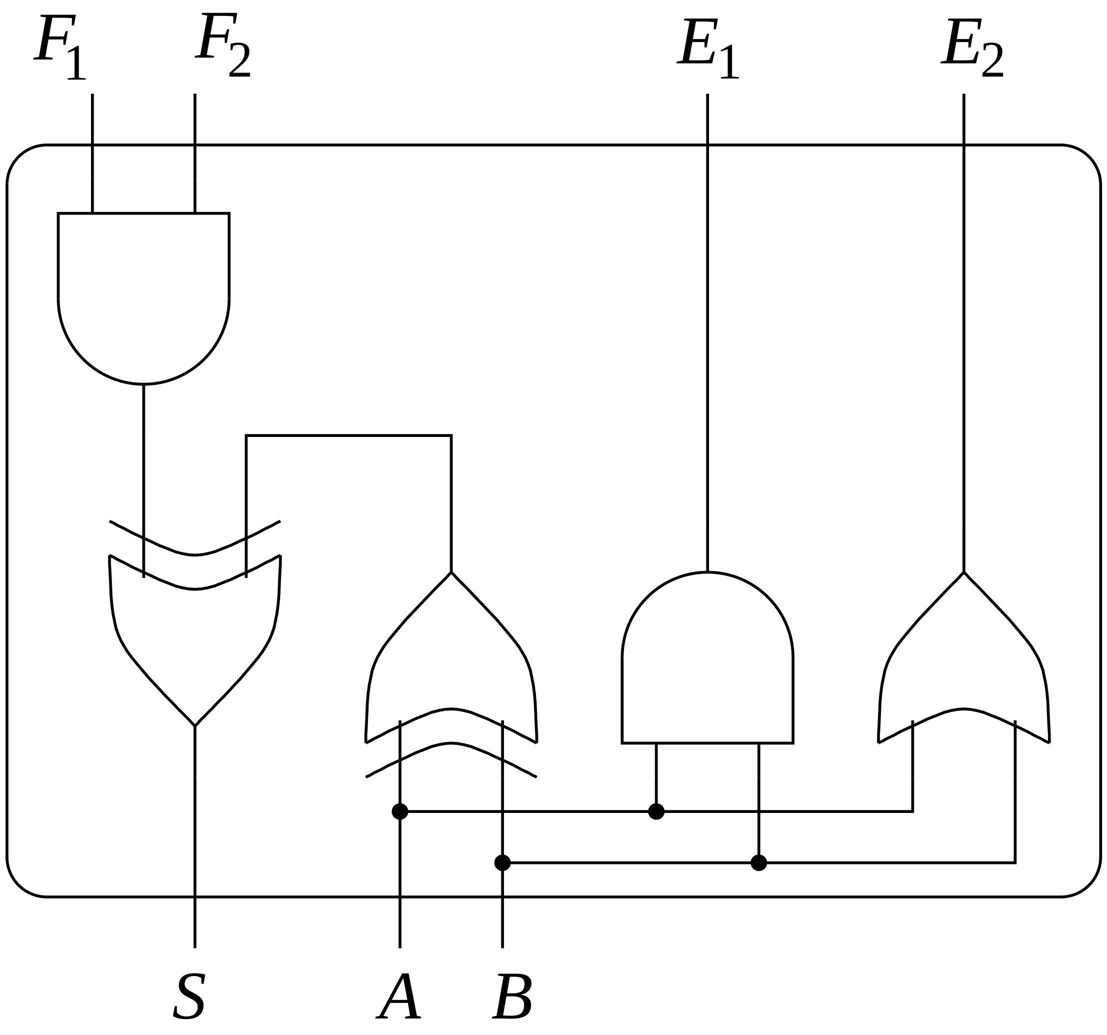
$$c_i \equiv g_i \vee (p_i \wedge g_{i-1}) \vee (p_i \wedge p_{i-1} \wedge g_{i-2}) \vee \ldots \vee (p_i \wedge p_{i-1} \wedge \ldots \wedge g_0)$$

Logarithmic delay in n using balanced $\vee$-trees and $\wedge$-trees.
Delay for 64 bits: 23-56 units (instead of 130)

Description of the circuit (for 4 bits)

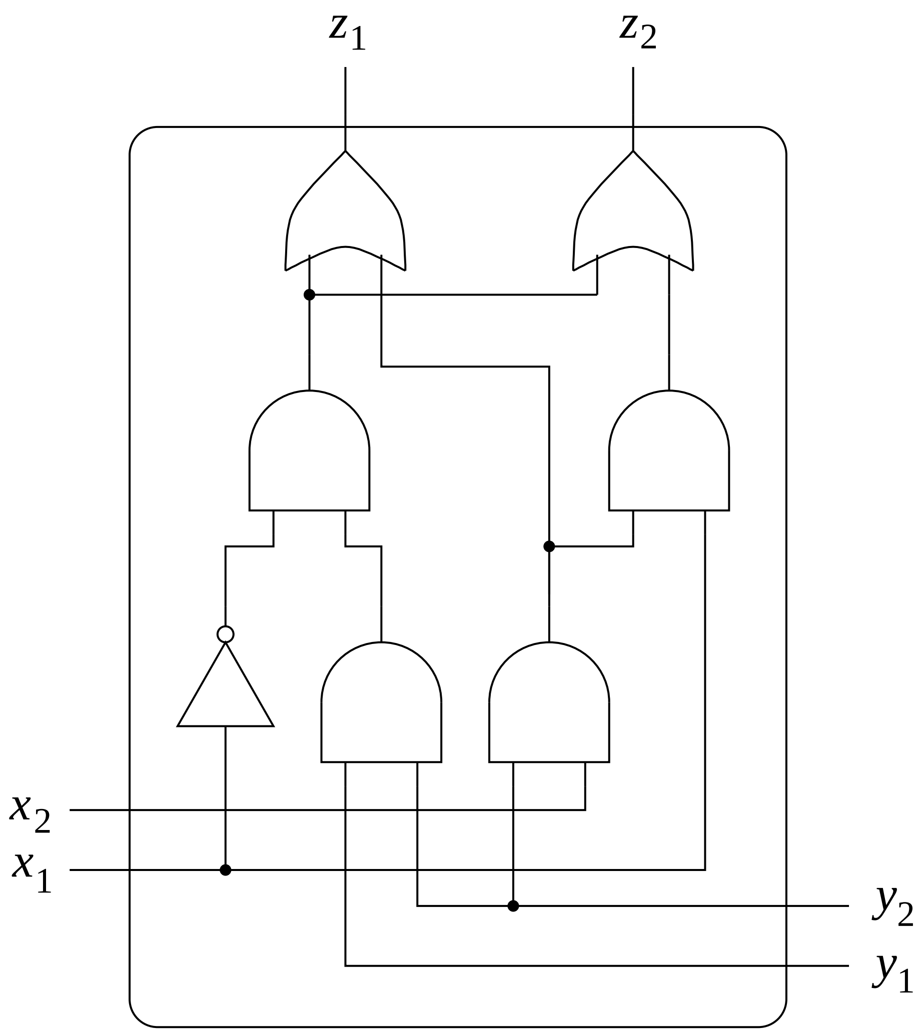


Description of the circuit II



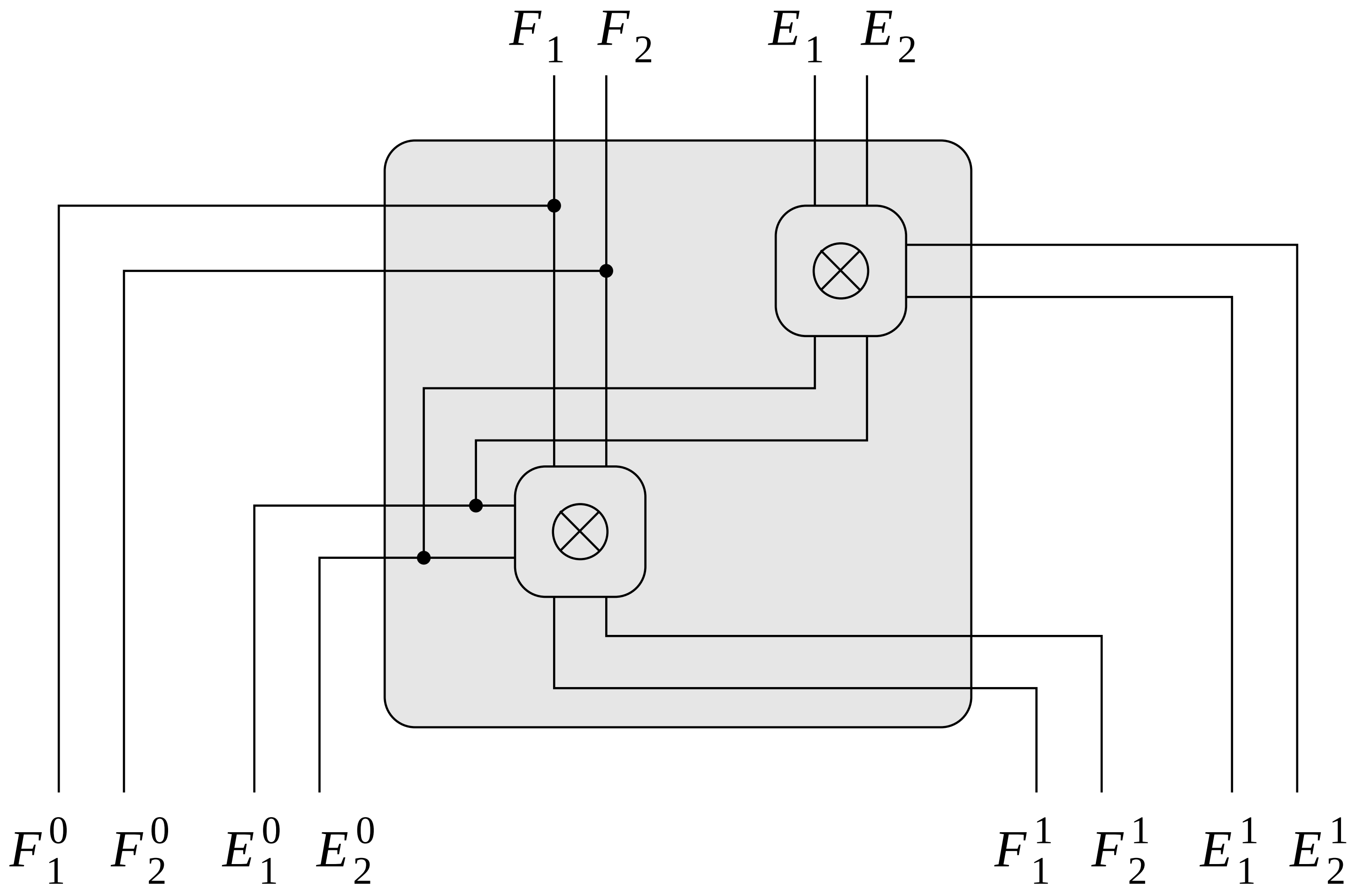
LeafCell circuit

Description of the circuit III



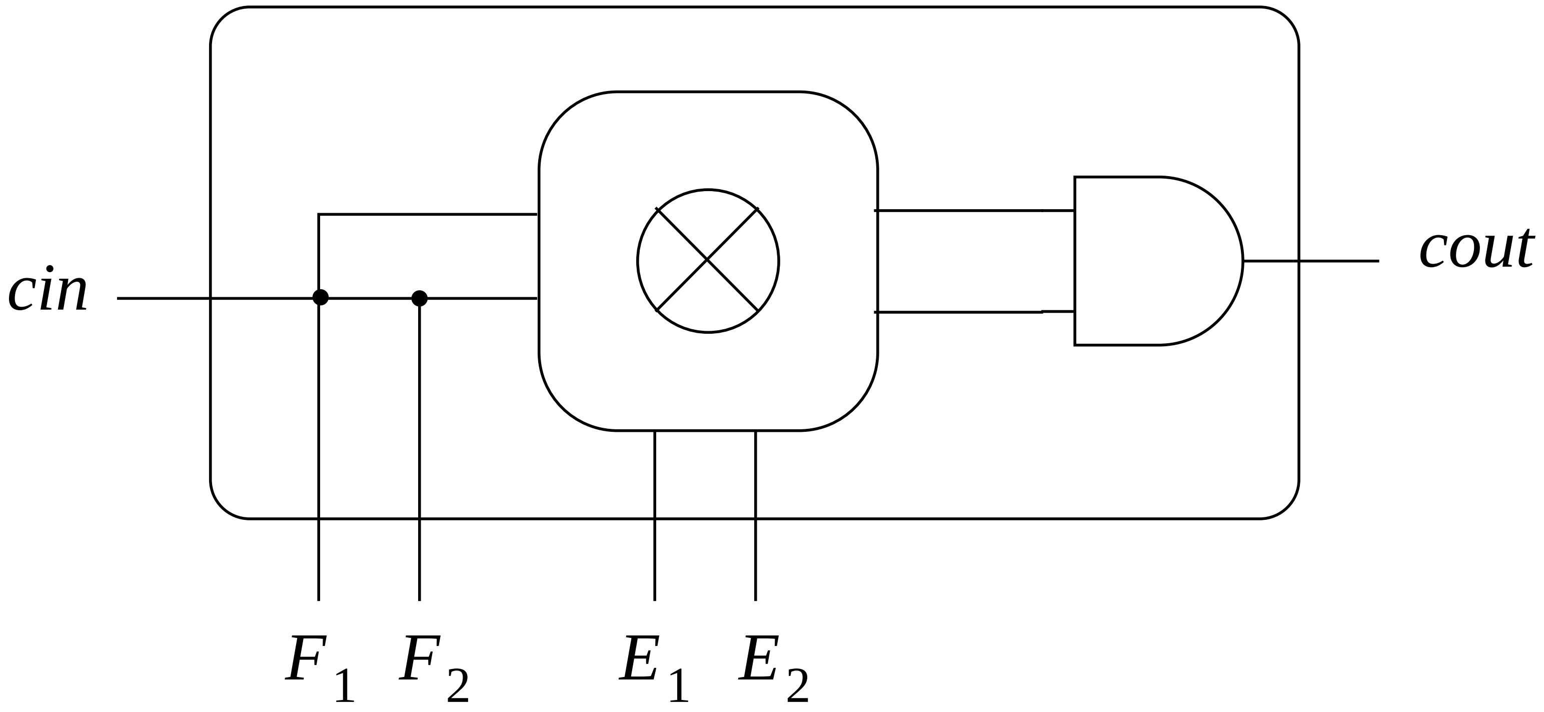
$\otimes$ circuit

Description of the circuit IV



NodeCell circuit

Description of the circuit V



RootCell circuit

Verification of the carry-look-ahead n-adder

Check validity of

$$\mathbf{adder}_n(a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}, s_0, \ldots, s_{n-1},\ cin, cout)$$
$$\Leftrightarrow$$
$$\mathbf{cla}_n(a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}, s_0, \ldots, s_{n-1},\ cin, cout)$$

Results of the SAT 2002 competition on a variant of this problem:

- Task was to compare 2, 4, 8, . . . , 256 bits adders (8 problems)
- From 26 variables and 70 3CNF clauses to 4584 variables and 13226 clauses
- Fastest solver (Zchaff) checked all 8 problems in 14 seconds
- More info at www.satcompetition.org

Rule-of-thumb: circuits with some hundreds of gates are routinely solved