Hardware & Software Verification

John Wickerson

Lecture 5: SAT and SMT solving

Automatic proof

- We often rely on automatic provers:
  - e.g. in Dafny, to show that `invariant P` is preserved,
  - e.g. in Isabelle methods like `by auto`.
- How do these automatic provers work?

SAT queries

- Simple case: proofs about Boolean statements.

SAT queries

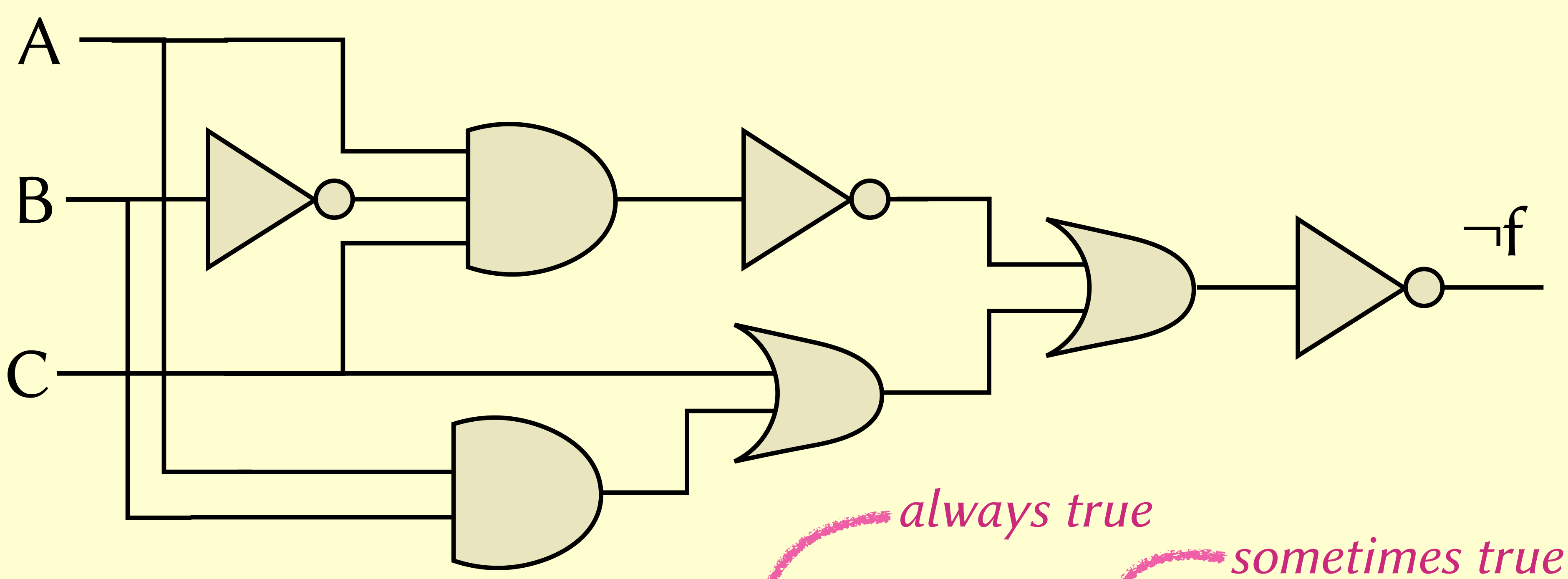
- Simple case: proofs about Boolean statements.
  - $f = ((A \land \neg B \land C) \implies (C \lor (B \land A)))$

SAT queries

- Simple case: proofs about Boolean statements.
    - $f = (\neg(A \wedge \neg B \wedge C) \vee (C \vee (B \wedge A)))$

SAT queries

- Simple case: proofs about Boolean statements.
  - $\neg f \;=\; \neg(\neg(A \wedge \neg B \wedge C) \vee (C \vee (B \wedge A)))$



A	B	C	¬f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

always true *sometimes true*

A formula can be VALID, SATISFIABLE, UNSATISFIABLE, or INVALID.

always false *sometimes false*

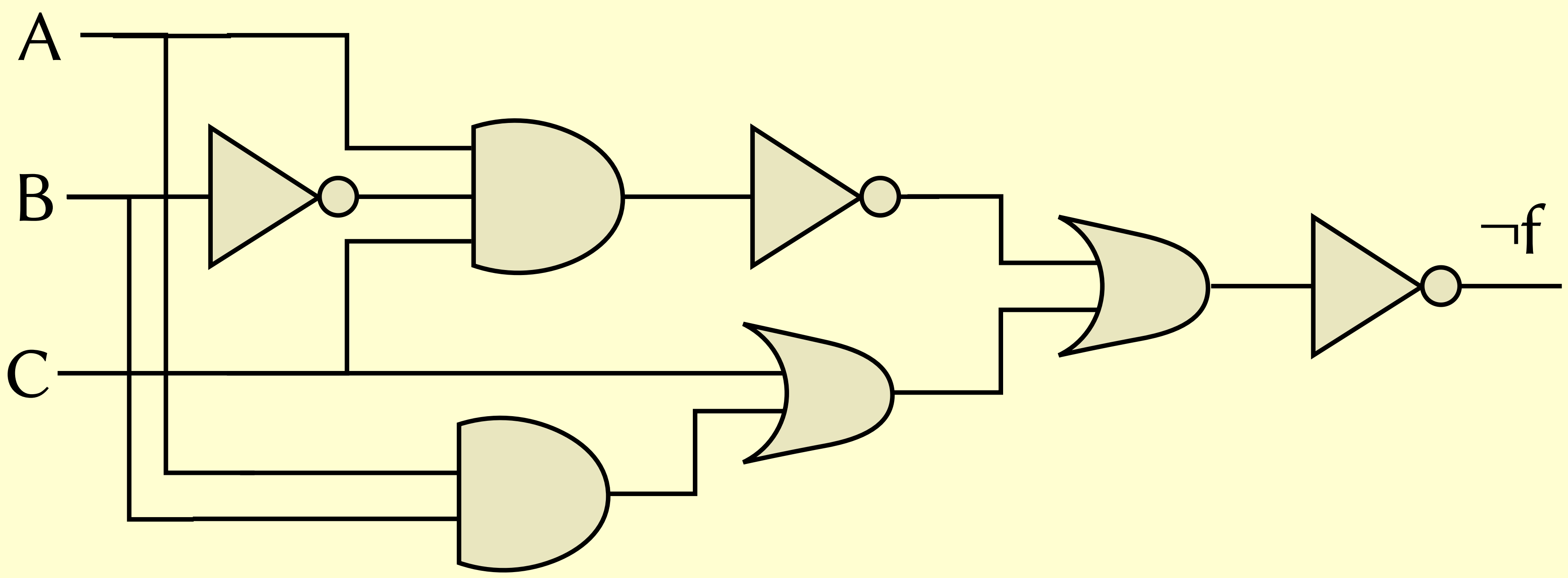
SAT solving

- A simple algorithm:

```
for A in {0, 1}:
  for B in {0, 1}:
    for C in {0, 1}:
      if ¬f(A,B,C) = 1:
        return ("SAT", [A,B,C])
return ("UNSAT")
```

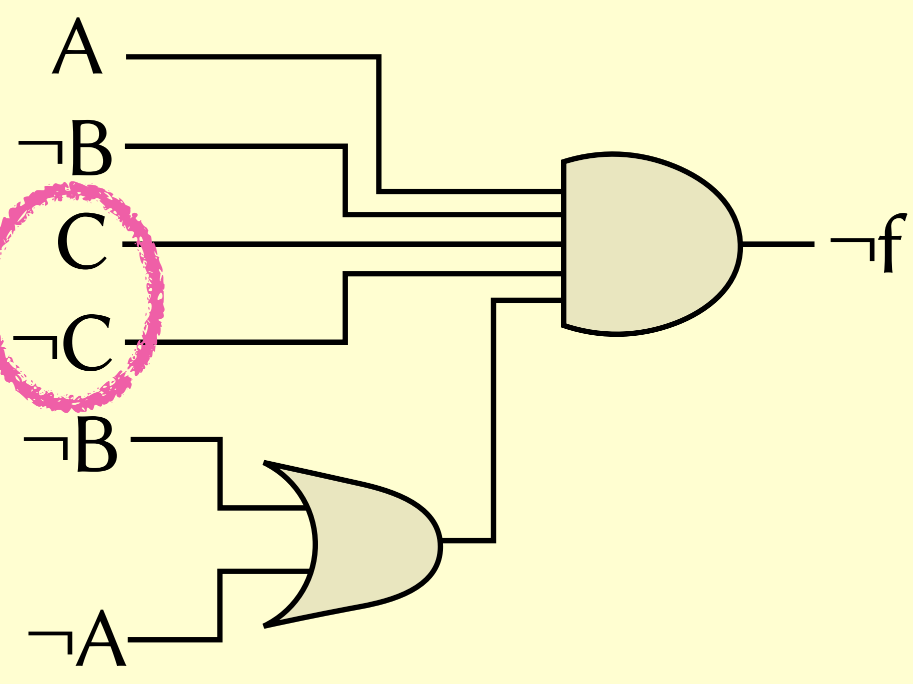
SAT solving

- A cleverer way: use de Morgan's rules to convert the formula to *conjunctive normal form* (CNF).

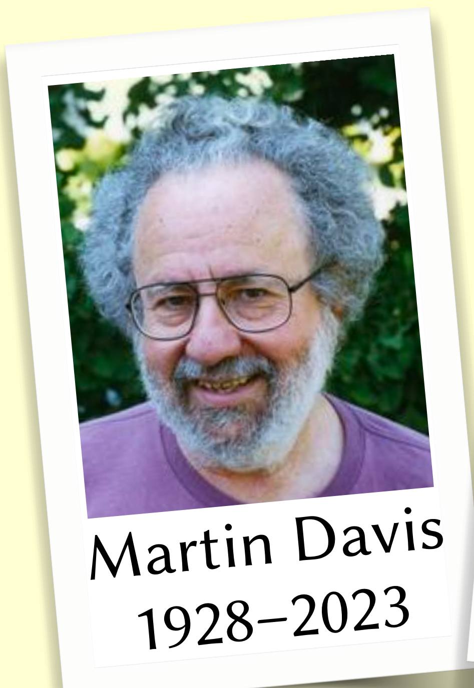


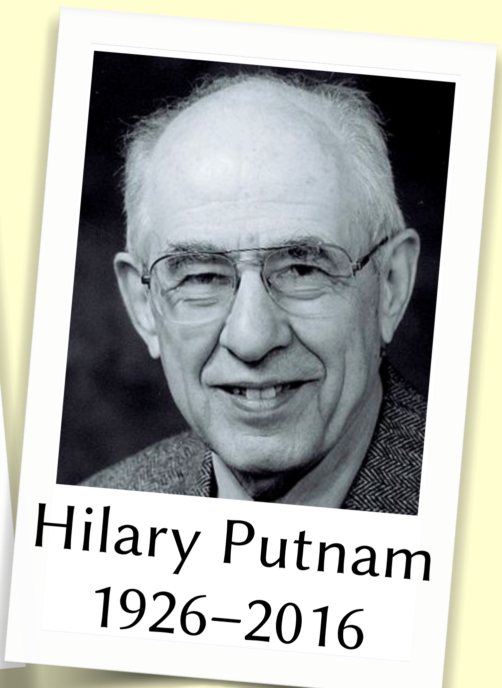
SAT solving

- A cleverer way: use de Morgan's rules to convert the formula to *conjunctive normal form* (CNF).



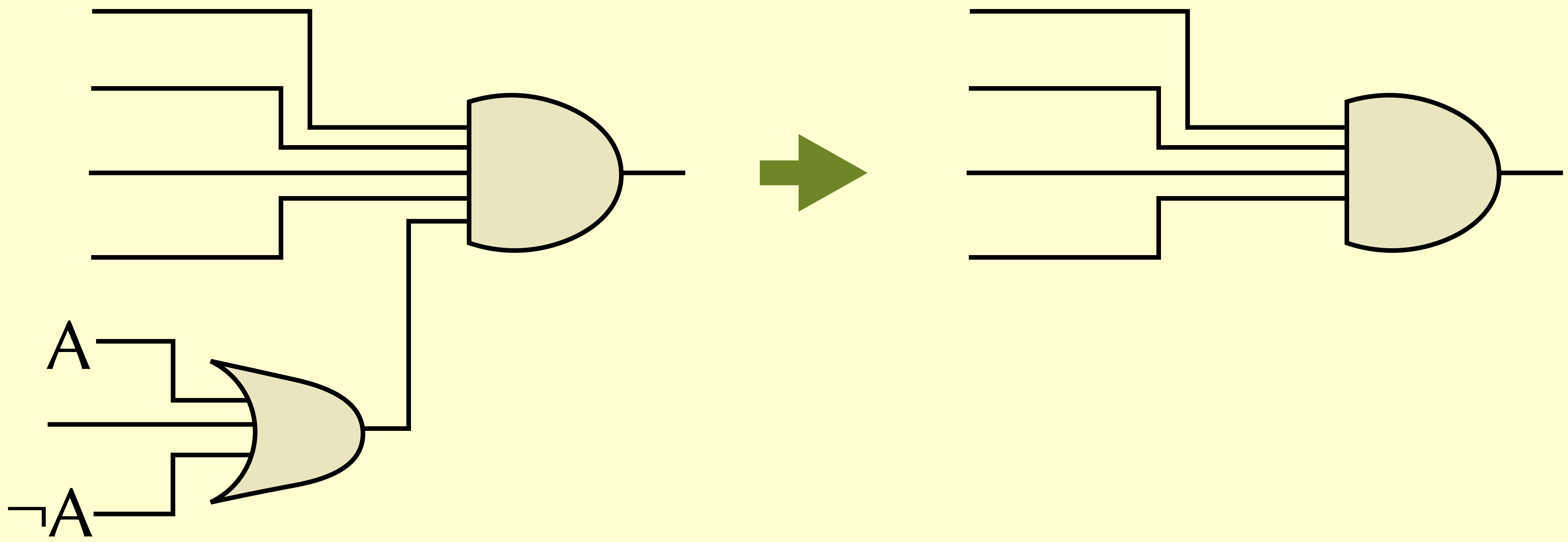
- It may then become obvious that ¬f is `UNSAT`.
- If not, we can use the Davis–Putnam algorithm...


Martin Davis
1928–2023


Hilary Putnam
1926–2016

The DP method

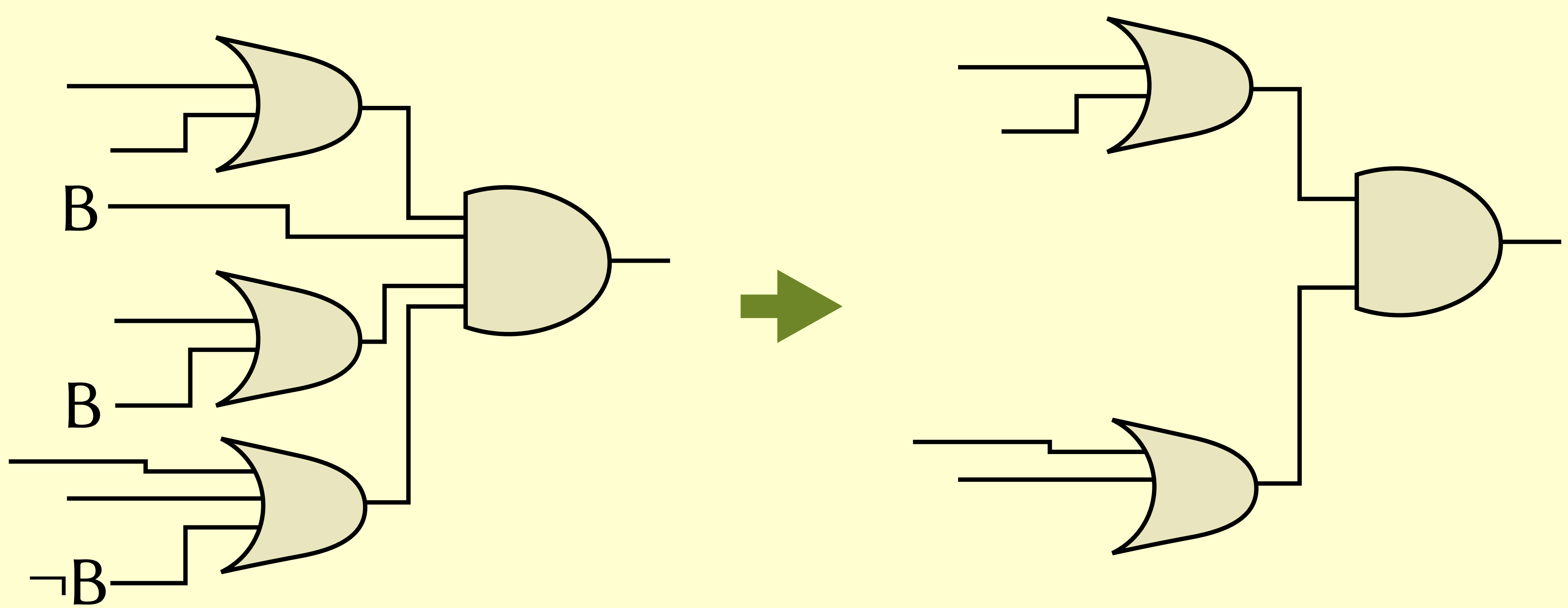
1. If an OR-gate takes both L and ¬L, delete it.



The DP method

1. If an OR-gate takes both L and ¬L, delete it.

2. If L is connected directly to the AND-gate, delete it, delete all OR-gates that take L, and delete any connections to ¬L.
(The solution, if it exists, will surely involve setting L=1.)

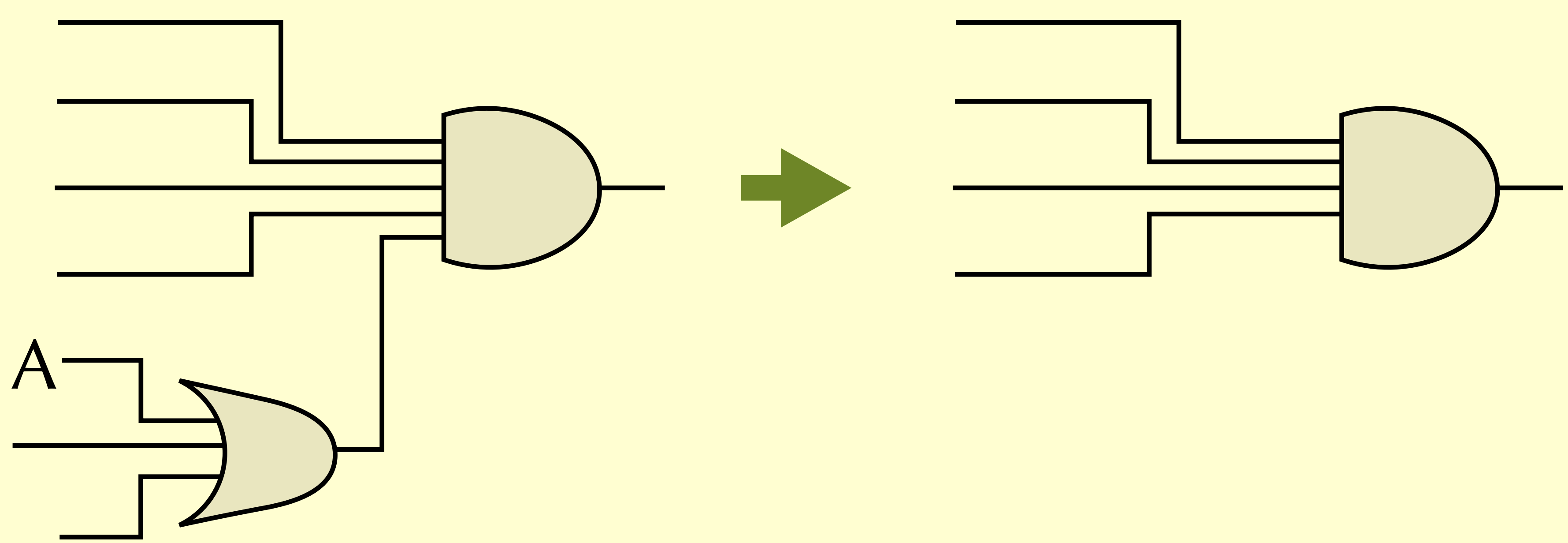


The DP method

1. If an OR-gate takes both L and ¬L, delete it.

2. If L is connected directly to the AND-gate, delete it, delete all OR-gates that take L, and delete any connections to ¬L.
(The solution, if it exists, will surely involve setting L=1.)

3. If L is unused, delete all OR-gates that take ¬L.
(The solution, if it exists, will surely involve setting L=0.)



The DP method

1. If an OR-gate takes both L and ¬L, delete it.

2. If L is connected directly to the AND-gate, delete it, delete all OR-gates that take L, and delete any connections to ¬L.
   (The solution, if it exists, will surely involve setting L=1.)

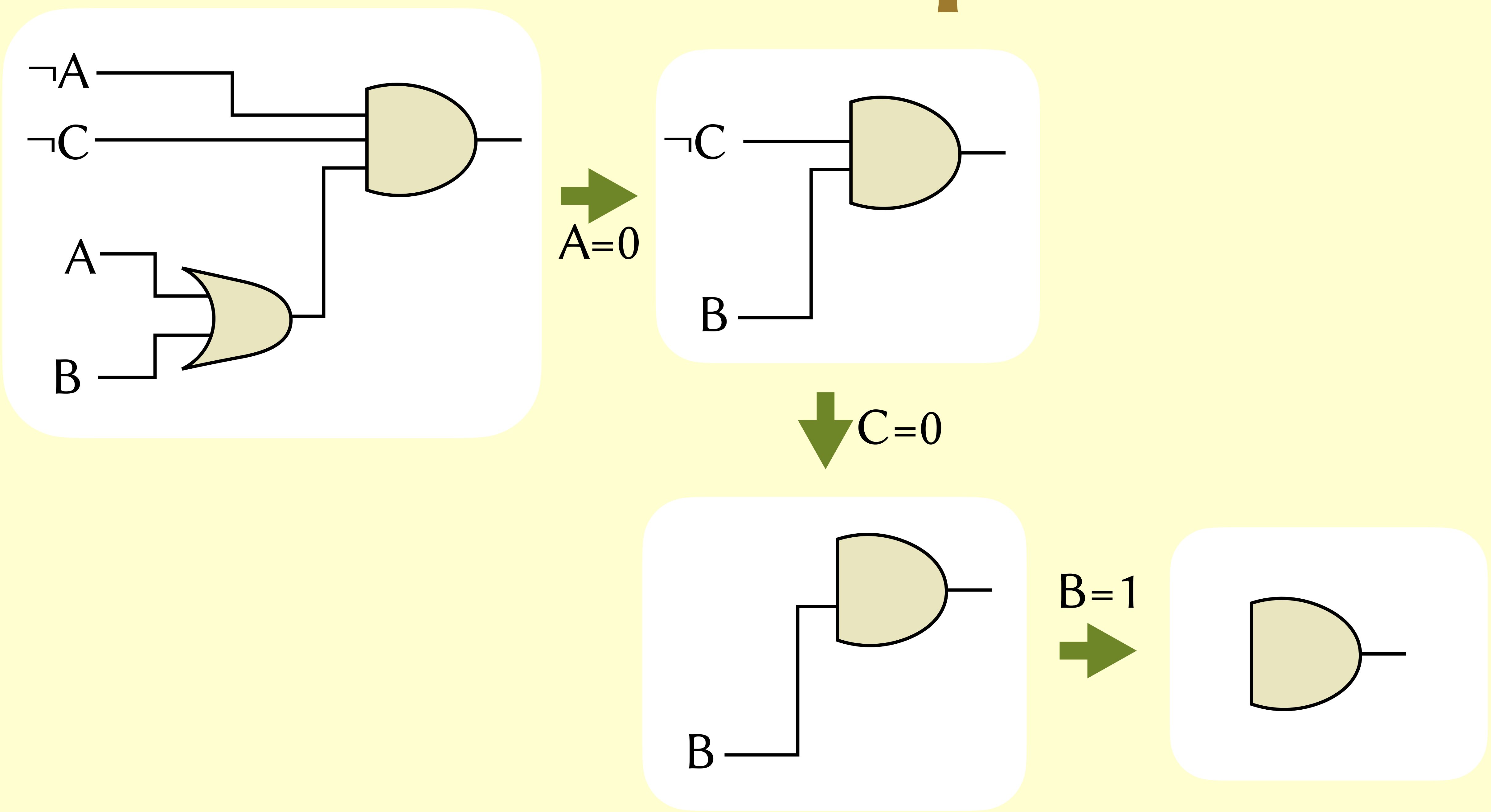
3. If L is unused, delete all OR-gates that take ¬L.
   (The solution, if it exists, will surely involve setting L=0.)

4. If any OR-gate has no inputs, the formula is false.

5. If the AND-gate has no inputs, the formula is true.

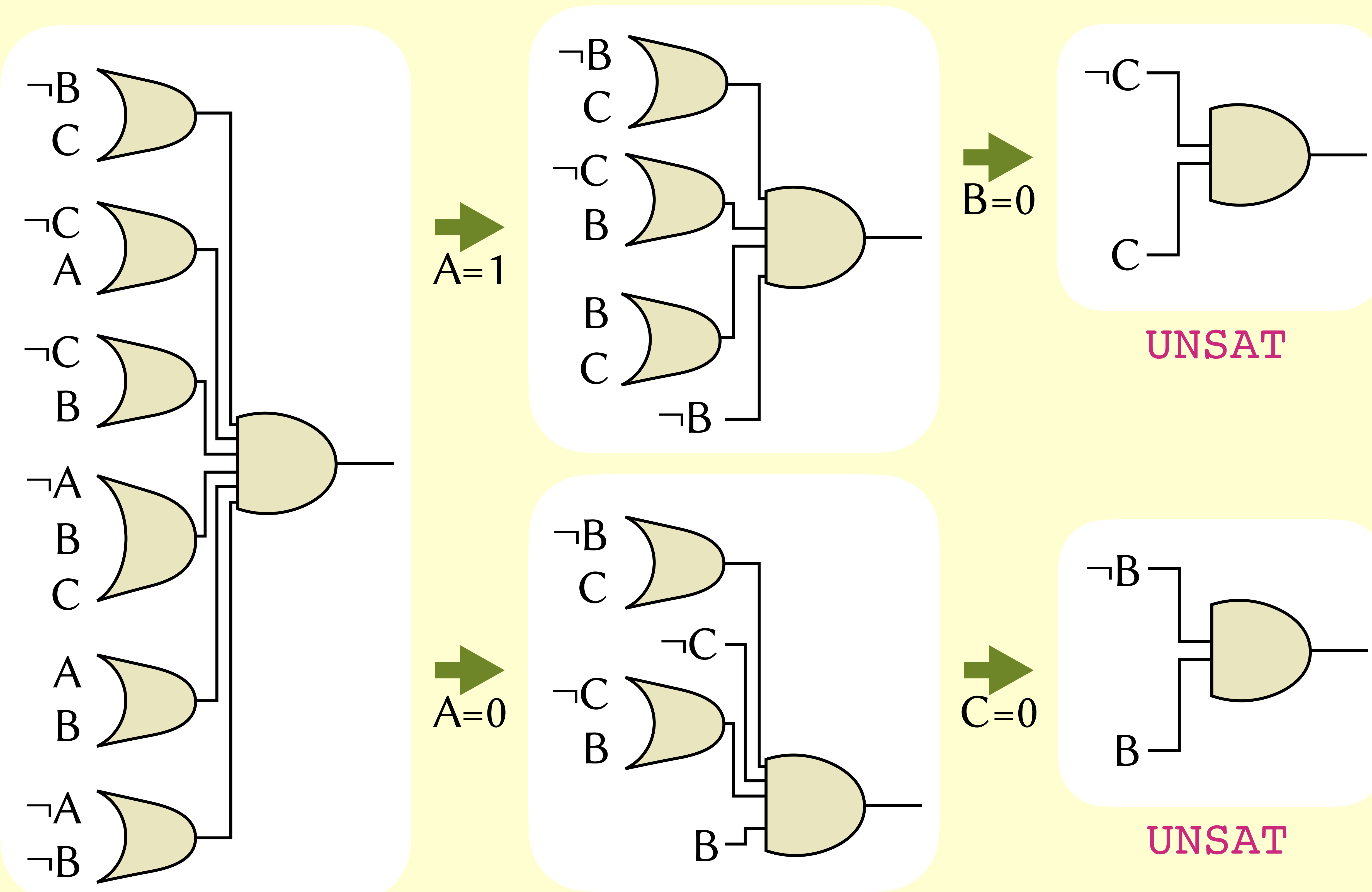
6. Pick a literal L and repeat the above for the cases L=0 and L=1.

DP example 1



SAT, A=0, B=1, C=0

DP example 2



Schöning's method

- The DP method is *complete*: it will always eventually return `SAT` or `UNSAT`.

- Schöning's method is *incomplete*: it will either return `SAT` or `UNKNOWN`, but if the input is a satisfiable formula, it may be quicker.

Uwe Schöning
1955–

Schöning's method

- Input: a formula in k-CNF with n variables.

each OR gate takes at most k inputs

- Guess an initial assignment.
- Repeat $3n$ times:
    - If the formula is satisfied: stop and accept.
    - Otherwise, randomly pick one of the unsatisfied clauses.
    - Randomly pick one of the $\leq k$ literals in the clause, and update the assignment by flipping its value.

The Tseitin transformation

- Converting a formula to CNF can make it explode in size, e.g.: $(A \land B) \lor (C \land D) \Rightarrow (A \lor C) \land (A \lor D) \land (B \lor C) \land (B \lor D)$.

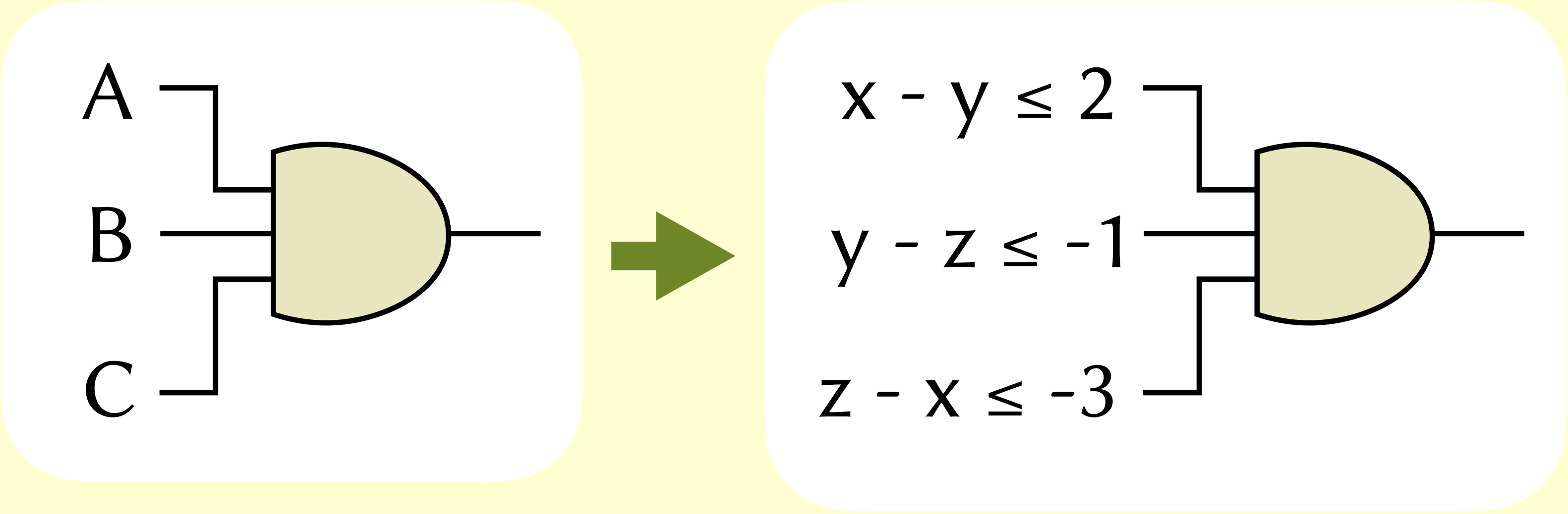
- The Tseitin transformation keeps formulas small when converting to CNF, at the cost of more variables.

$$\underbrace{\underbrace{(A \land B)}_{E} \lor \underbrace{(C \land D)}_{F}}_{G}$$

$$\begin{array}{l} G \\ \land\ (G \leftrightarrow E \lor F) \\ \land\ (E \leftrightarrow A \land B) \\ \land\ (F \leftrightarrow C \land D) \end{array} \Rightarrow \begin{array}{l} G \\ \land\ (G \lor \neg E) \land (G \lor \neg F) \land (\neg G \lor E \lor F) \\ \land\ (\neg E \lor A) \land (\neg E \lor B) \land (E \lor \neg A \lor \neg B) \\ \land\ (\neg F \lor C) \land (\neg F \lor D) \land (F \lor \neg C \lor \neg D) \end{array}$$

Towards SMT solving

- We can now prove basic Boolean formulas. But what about proving something like A × (B + C) = A × B + A × C?
- If these are 32-bit integers, we could make this a SAT problem by treating each variable as 32 Boolean variables and encoding the rules of Boolean arithmetic.
- Or we can move up to SMT: *satisfiability modulo theories.*



Some theories

- **Equality and uninterpreted functions**, which knows that x=y and y=z implies x=z, and that x=y implies f(x)=f(y).
- **Difference logic**, where statements take the form $x - y \leq c$.
- **Presburger arithmetic**, which allows statements about naturals containing +, 0, 1, and =. For instance, n is a McNugget number if $\exists x\ y\ z.\ n = 6x + 9y + 20z$.

Mojżesz Presburger
1904–c.1943

Some theories

- **Equality and uninterpreted functions**, which knows that x=y and y=z implies x=z, and that x=y implies f(x)=f(y).
- **Difference logic**, where statements take the form x - y ≤ c.
- **Presburger arithmetic**, which allows statements about naturals containing +, 0, 1, and =. For instance, n is a McNugget number if ∃x y z. n = 6x + 9y + 20z.
- **Non-linear arithmetic**, which allows queries like:

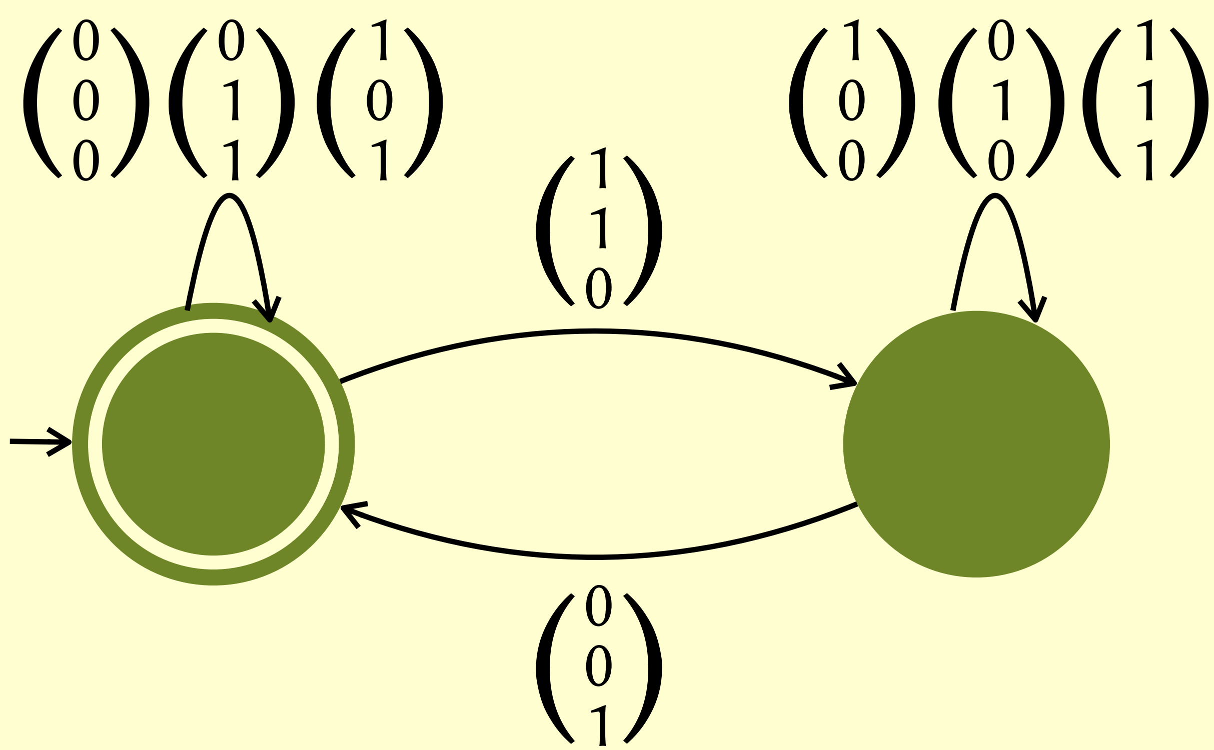
$$(\sin(x)^3 = \cos(\log(y) \cdot x) \vee b \vee -x^2 \geq 2.3y) \wedge \left(\neg b \vee y < -34.4 \vee \exp(x) > \frac{y}{x}\right)$$

- **Theory of arrays**, **theory of bit-vectors**, etc.

Decidability of Presburger

$x + y = z$

	1	2	4	8	16	32	64
$x =$	0	1	0	0	1	0	0
$y =$	0	1	0	1	0	1	0
$z =$	0	0	1	1	1	1	0



Julius Richard Büchi
1924–1984

Adding multiplication

- If we add multiplication, we can write a statement representing the Collatz conjecture: does there exist an infinite sequence of positive integers x_0, x_1, x_2, ... such that

  $2 \times x_{i+1} = x_i$ if x_i is even
  $x_{i+1} = 3 \times x_i + 1$ if x_i is odd

- So **if** arithmetic with multiplication were decidable, we could solve the Collatz conjecture automatically!

Automatic proof

- We often rely on automatic provers:
  - e.g. in Dafny, to show that **`invariant`** `P` is preserved,
  - e.g. in Isabelle methods like **`by`** `auto`.
- How do these automatic provers work?