$$
\begin{array}{rcl}
\mathit{formula} & : & \mathit{formula} \lor \mathit{formula} \mid \neg\mathit{formula} \mid \mathit{atom} \\
\mathit{atom} & : & \mathit{term}\ \mathit{rel}\ \mathit{term} \mid \mathit{Boolean\text{-}Identifier} \mid \mathit{term}[\,\mathit{constant}\,] \\
\mathit{rel} & : & = \ \mid\ < \\
\mathit{term} & : & \mathit{term}\ \mathit{op}\ \mathit{term} \mid \mathit{identifier} \mid \sim \mathit{term} \mid \mathit{constant} \mid \\
 & & \mathit{atom}\,?\,\mathit{term}:\mathit{term} \mid \\
 & & \mathit{term}[\,\mathit{constant} : \mathit{constant}\,] \mid \mathit{ext}(\,\mathit{term}\,) \\
\mathit{op} & : & + \mid - \mid \cdot \mid / \mid << \mid >> \mid \& \mid \shortmid \mid \oplus \mid \circ
\end{array}
$$

- $\sim x$: bit-wise negation of x
- $ext(x)$: sign- or zero-extension of x
- $x << d$: left shift with distance d
- $x \circ y$: concatenation of x and y

- Transform Bit-Vector Logic to Propositional Logic
- Most commonly used decision procedure
- Also called '*bit-blasting*'

- Transform Bit-Vector Logic to Propositional Logic
- Most commonly used decision procedure
- Also called '*bit-blasting*'

Bit-Vector Flattening

1. Convert propositional part as before
2. Add a *Boolean variable for each bit* of each sub-expression (term)
3. Add *constraint* for each sub-expression

We denote the new Boolean variable for i of term t by t_i

What constraints do we generate for a given term?

What constraints do we generate for a given term?

- This is easy for the bit-wise operators.
- Example for *t=a | b*

$$\bigwedge_{i=0}^{l-1} t_i = (a_i \vee b_i))$$

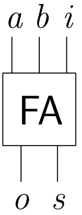
What about x=y

How to flatten $s=a+b$

Flattening bit-vector arithmetic

How to flatten *s=a+b*

$\longrightarrow$ we can build a *circuit* that adds them!



Full Adder

$$s \equiv (a+b+i) \bmod 2 \equiv a \oplus b \oplus i$$

$$o \equiv (a+b+i) \operatorname{div} 2 \equiv a \cdot b + a \cdot i + b \cdot i$$

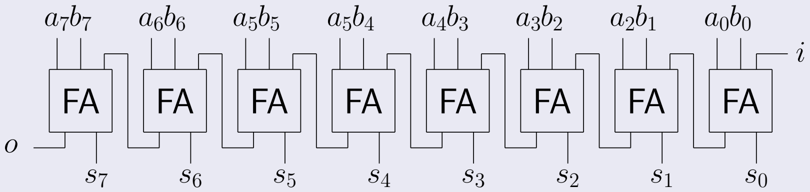
The full adder in CNF:

$$(a \vee b \vee \neg o) \wedge (a \vee \neg b \vee i \vee \neg o) \wedge (a \vee \neg b \vee \neg i \vee o) \wedge$$
$$(\neg a \vee b \vee i \vee \neg o) \wedge (\neg a \vee b \vee \neg i \vee o) \wedge (\neg a \vee \neg b \vee o)$$

Ok, this is good for one bit! How about more?

Ok, this is good for one bit! How about more?

8-Bit ripple carry adder (RCA)



- Also called *carry chain adder*
- Adds l variables
- Adds 10* l clauses 6 for o , 4 for s

- Multipliers result in very hard formulas
- Example:

$$a \cdot b = c \wedge b \cdot a \neq c \wedge x < y \wedge x > y$$

CNF: About 11000 variables, unsolvable for current SAT solvers

- Similar problems with division, modulo

- Multipliers result in very hard formulas
- Example:

$$a \cdot b = c \wedge b \cdot a \neq c \wedge x < y \wedge x > y$$

CNF: About 11000 variables, unsolvable for current SAT solvers

- Similar problems with division, modulo
- Q: How do we fix this?