Constraint-Based Synthesis

With slides by Armando Solar-Lezama

Synthesis as Constraint Solving

Synthesis Condition:

$$\exists \phi \; \forall in \in E \; Q(in, \phi)$$

where E = $\{x_1, x_2, \ldots, x_k\}$

Invention Pillar Question: How does Sketch work?

Semantics of expressions

e:= n | x | e_1 + e_2 | e_1 > e_2
c:= x := e | c_1 ; c_2 | if e then c_1 else c_2 | while e do c

What does an expression mean?

- An expression reads the state and produces a value
- The state is modeled as a map σ from vars to values
- $\mathcal{A}[\![\cdot]\!] : e \to \Sigma \to int$

Ex:

- $\mathcal{A}[\![x]\!] = \lambda\sigma.\sigma(x)$
- $\mathcal{A}[\![n]\!] = \lambda\sigma.n$
- $\mathcal{A}[\![e_1 + e_2]\!] = \lambda\sigma.\mathcal{A}[\![e_1]\!]\sigma + \mathcal{A}[\![e_2]\!]\sigma$
- $\mathcal{A}[\![e_1 > e_2]\!] = \lambda\sigma.\text{if}\,\mathcal{A}[\![e_1]\!]\sigma > \mathcal{A}[\![e_2]\!]\sigma\ \text{then 1 else 0}$

Semantics of commands

$e := n \mid x \mid e_1 + e_2 \mid e_1 > e_2$
$c := x := e \mid c_1 ; c_2 \mid \text{if } e \text{ then } c_1 \text{ else } c_2 \mid \text{while } e \text{ do } c$

What does a command mean?

- A command modifies the state
- $\mathcal{C}[\![\cdot]\!] : c \to \Sigma \to \Sigma$

Ex:

- $\mathcal{C}[\![x := e]\!] = \lambda\sigma.\sigma[x \to (\mathcal{A}[\![e]\!]\sigma)]$
- $\mathcal{C}[\![c_1; c_2]\!] = \lambda\sigma.\mathcal{C}[\![c_2]\!](\mathcal{C}[\![c_1]\!]\sigma)$
- $\mathcal{C}[\![\text{if } e \text{ then } c_1 \text{ else } c_2]\!] =$
  $\lambda\sigma.\text{if } \mathcal{A}[\![e]\!]\sigma = 1 \text{ then } (\mathcal{C}[\![c_1]\!]\sigma) \text{ else } (\mathcal{C}[\![c_2]\!]\sigma)$

What about loops?

e:= n | x | $e_1 + e_2$ | $e_1 > e_2$
c:= x := e | c_1 ; c_2 | if e then c_1 else c_2 | while e do c

Semantics of a while loop

- Let $W = \mathcal{C}[\![\text{while } e \text{ do } c]\!]$
- W satisfies the following equation:

$$W = \lambda\sigma.\,\text{if } \mathcal{A}[\![e]\!] \text{ then } \big(W(\mathcal{C}[\![c\,]\!]\sigma)\big) \text{ else } \sigma$$

- Equation can have many solutions
  - when loop doesn't terminate
- Rich theory for finding least fixed point solution
- We'll settle for a simpler strategy:

  unroll k times and then give up

Symbolic execution of sketches

Very similar to what we just saw

- But values are now parameterized by ϕ

$\Psi = \boldsymbol{\Phi} \rightarrow \mathbb{Z}$

$$\mathcal{A}[\![\circ]\!]^{\tau} : Aexp \rightarrow (\Sigma \rightarrow \Psi)$$

The denotation function will keep track of contexts

$$\mathcal{A}[\![x]\!]^{\tau}\sigma = \sigma(x)$$

$$\mathcal{A}[\![??_i]\!]^{\tau}\sigma = \lambda\phi.\phi(??_i, \tau)$$

$$\mathcal{A}[\![e_1 \ \star \ e_2]\!]^{\tau}\sigma = \lambda\phi.\ \mathcal{A}[\![e_1]\!]^{\tau}\sigma\phi \ \star \ \mathcal{A}[\![e_2]\!]^{\tau}\sigma\phi$$

Symbolic execution of commands

Commands have two roles

- Modify the symbolic state
- Generate constraints

$$\mathcal{C}[\![\circ]\!]^{\tau} : Command \to \big(\Sigma \times \mathcal{P}(\Phi) \to \Sigma \times \mathcal{P}(\Phi)\big)$$

Constraints represent sets of valid ϕ functions

Symbolic execution of commands

Assignments and Assertion

$$\mathcal{C}[\![x := e]\!]^{\tau}\langle\sigma\,,\Phi\rangle = \langle\sigma[x \mapsto \ \mathcal{A}[\![e]\!]^{\tau}\sigma]\,,\Phi\rangle$$

$$\mathcal{C}[\![\textsf{assert}\ \ e]\!]^{\tau}\langle\sigma\,,\Phi\rangle = \langle\sigma\,,\{\phi \in \Phi :\ \mathcal{A}[\![e]\!]^{\tau}\sigma\phi = 1\}\rangle$$

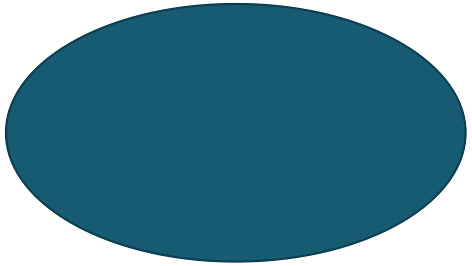
Symbolic execution of commands

If statement

$$\mathcal{C}[\![\,\textbf{if}\ \ e\ \ \textbf{then}\ \ c_1\ \ \textbf{else}\ \ c_2]\!]^\tau \langle \sigma\,,\, \Phi \rangle = \langle \sigma'\,,\, \Phi' \rangle$$

$$\begin{aligned}
\Phi_t &= \{\phi \in \Phi :\ \mathcal{A}[\![e]\!]^\tau \sigma\phi = true\} \\
\Phi_f &= \{\phi \in \Phi :\ \mathcal{A}[\![e]\!]^\tau \sigma\phi = false\} \\
\langle \sigma_1\,,\, \Phi_1 \rangle &= \ \mathcal{C}[\![c_1]\!]^\tau \langle \sigma\,,\, \Phi_t \rangle \\
\langle \sigma_2\,,\, \Phi_2 \rangle &= \ \mathcal{C}[\![c_2]\!]^\tau \langle \sigma\,,\, \Phi_f \rangle \\
\Phi' &= (\Phi_1) \cup (\Phi_2) \\
\sigma' &= \lambda x.\lambda\phi.\ \mathcal{A}[\![e]\!]^\tau \sigma\phi \ ?\ \sigma_1 x\phi\ :\ \sigma_2 x\phi
\end{aligned}$$

Conditionals



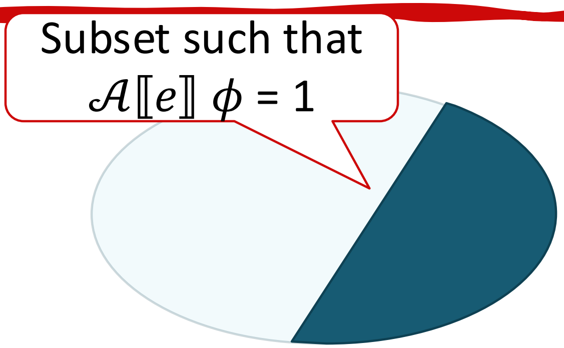
Initial set of viable ϕ

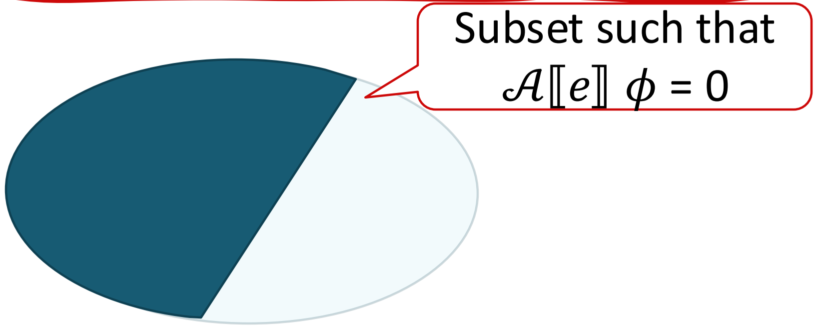
if e

then C1 **else** C2

Conditionals

if e





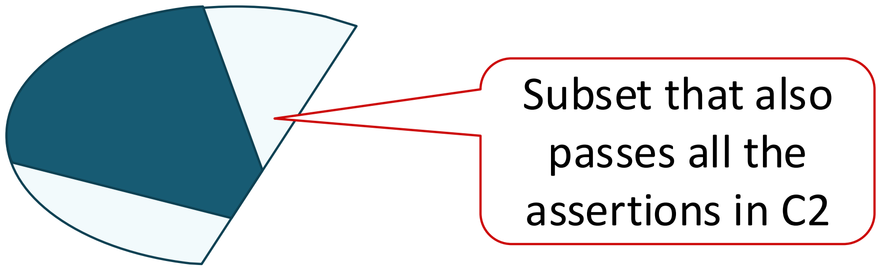
then C1 **else** C2

Conditionals

if e

then C1

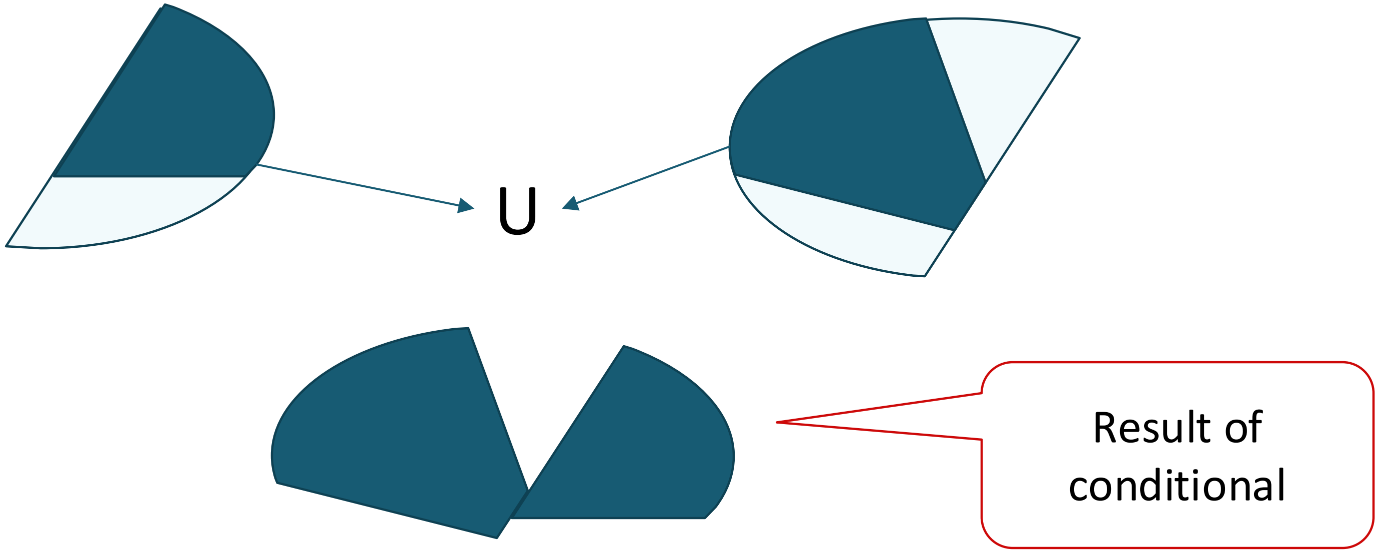
else C2



Conditionals

if e

then C1 **else** C2



Symbolic execution of commands

While loops

$$W(\langle \sigma , \Phi \rangle) = \mathcal{C}[\![\textbf{while } e \textbf{ do } c]\!]^{\tau} \langle \sigma , \Phi \rangle = \langle \sigma' , \Phi' \rangle$$

$$\Phi_t = \{\phi \in \Phi : \mathcal{A}[\![e]\!]^{\tau} \sigma \phi = true\}$$

$$\Phi_f = \{\phi \in \Phi : \mathcal{A}[\![e]\!]^{\tau} \sigma \phi = false\}$$

$$\langle \sigma_1 , \Phi_1 \rangle = W(\mathcal{C}[\![c]\!]^{\tau} \langle \sigma , \Phi_t \rangle)$$

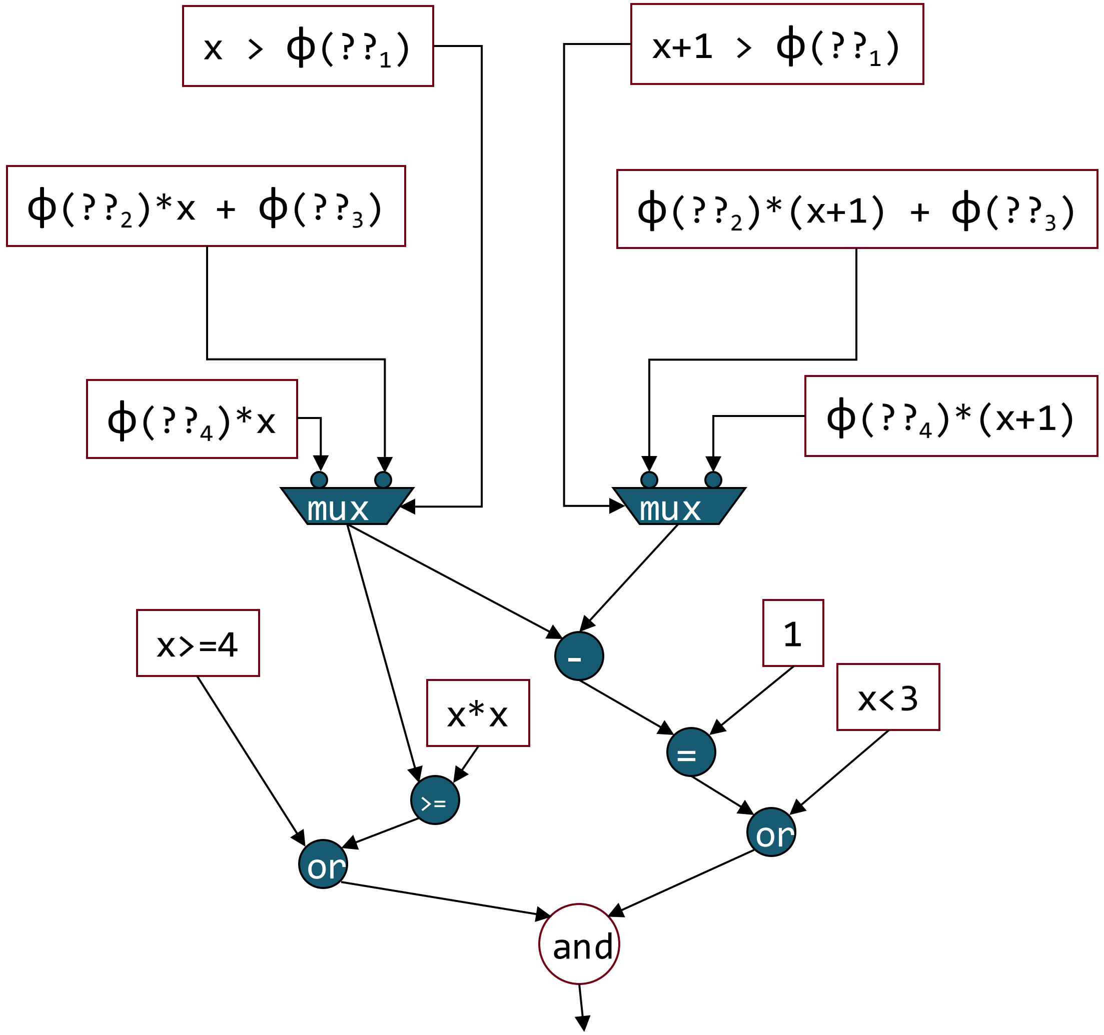
$$\Phi' = (\Phi_1) \cup (\Phi_f)$$

$$\sigma' = \lambda x.\lambda \phi.\ \mathcal{A}[\![e]\!]^{\tau} \sigma \phi \ ?\ \sigma_1 x \phi : \sigma x \phi$$

Building Constraints

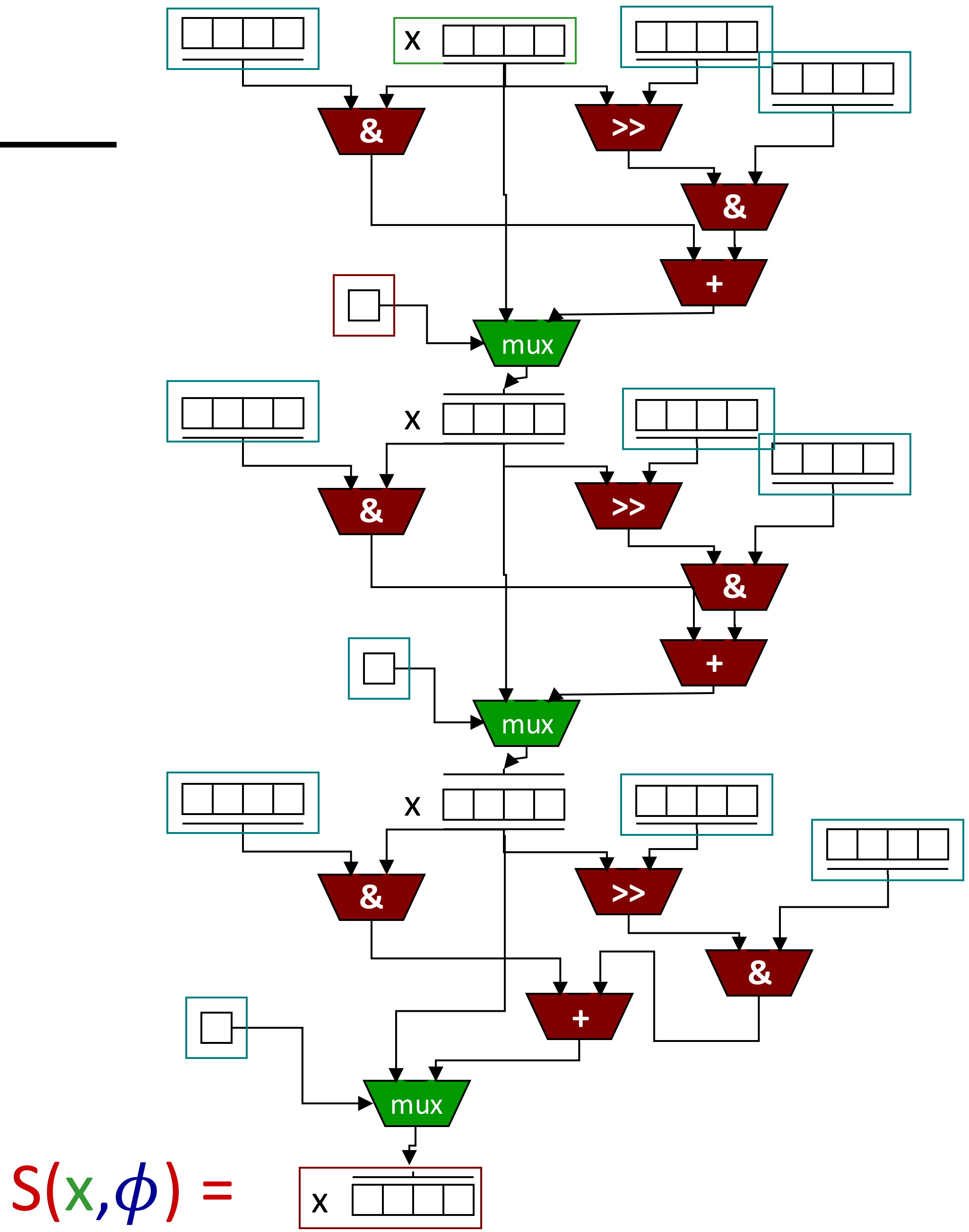
A sketch as a constraint system

```
int lin(int x){
   if(x > ϕ(??₁))
      return ϕ(??₂)*x + ϕ(??₃);
   else
      return ϕ(??₄)*x;
}

void main(int x){
   int t1 = lin(x);
   int t2 = lin(x+1);

   if(x<4) assert t1 >=  x*x;

   if(x>=3) assert t2-t1 == 1;
}
```



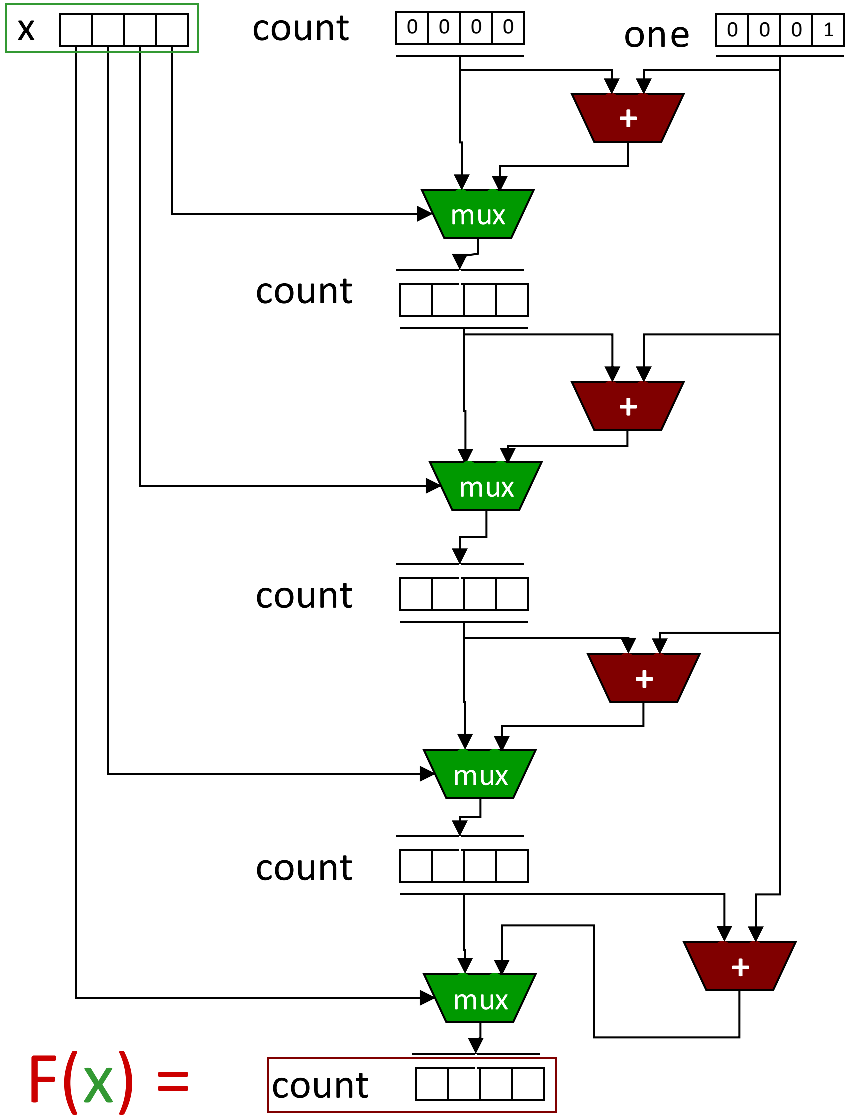
Symbolic Execution

```
int popSketched (bit[W] x)
    implements pop {
    repeat(??) {
            x = (x & ??)
            + ((x >> ??) & ??);
    }
    return x;
}
```



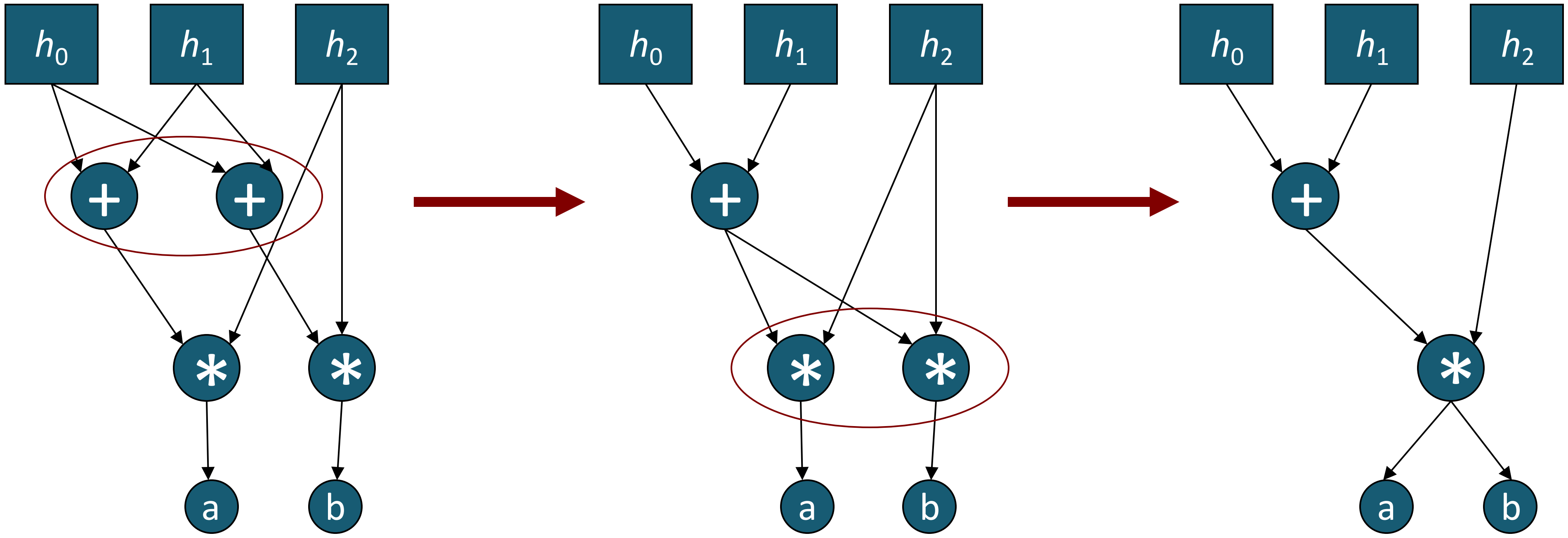
Ex : Population count. 0010 0110 → 3

```
int pop (bit[W] x)
{
    int count = 0;
    for (int i = 0; i < W; i++) {
        if (x[i]) count++;
    }
    return count;
}
```

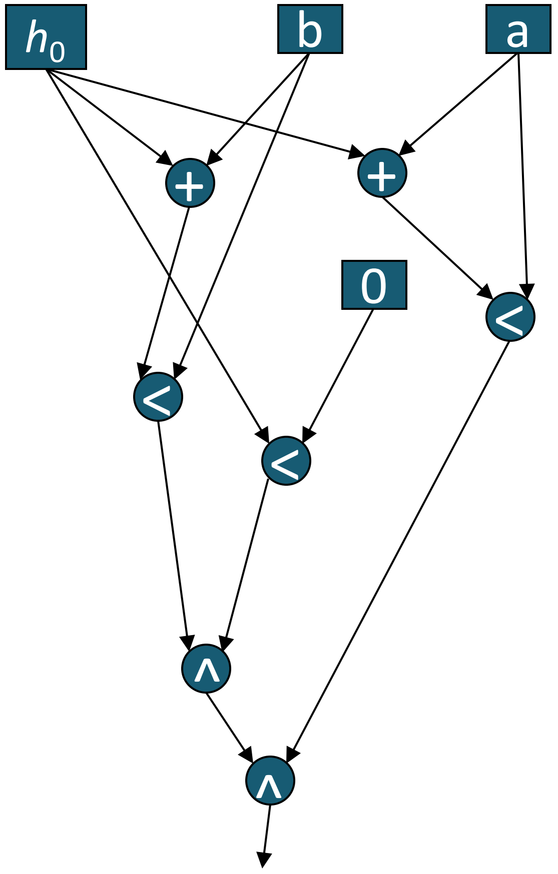


Simplification

Structural Hashing

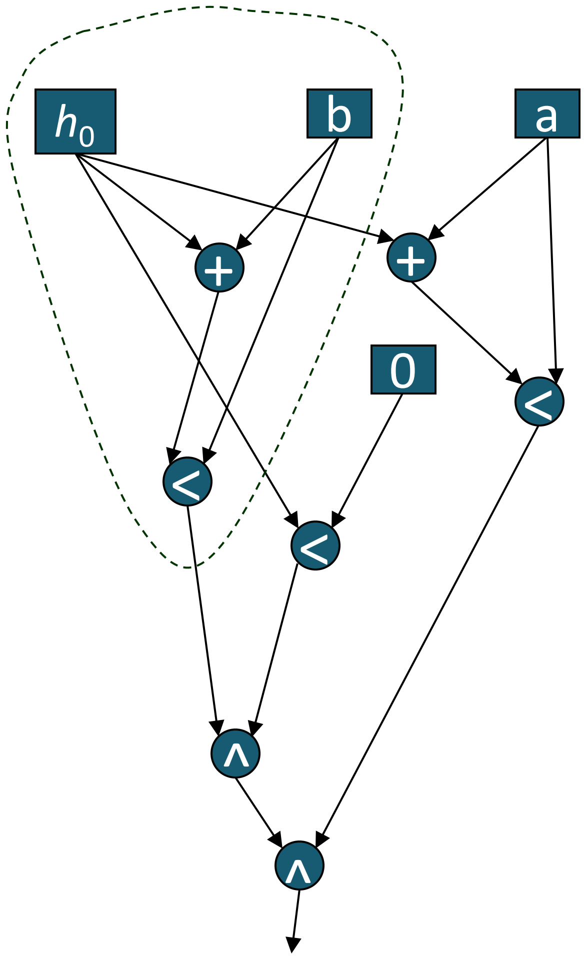


Structural Hashing + Rewriting



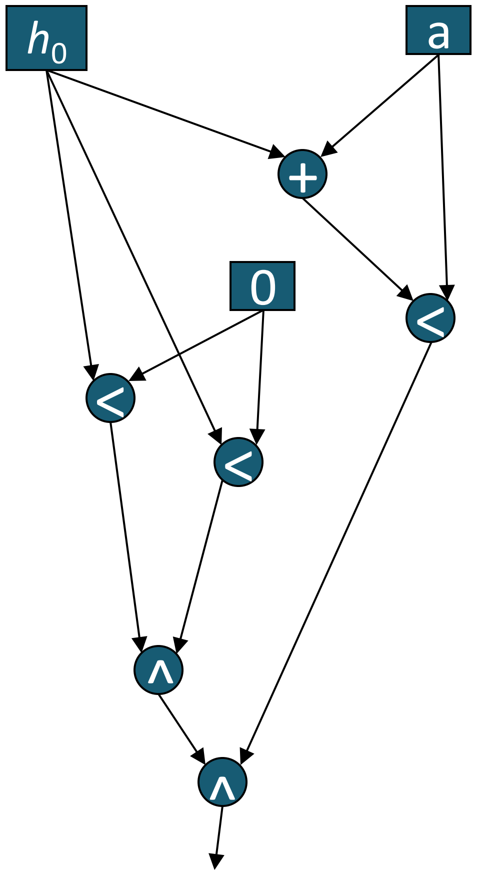
Structural Hashing + Rewriting

$X + b < b \;\rightarrow X < 0$



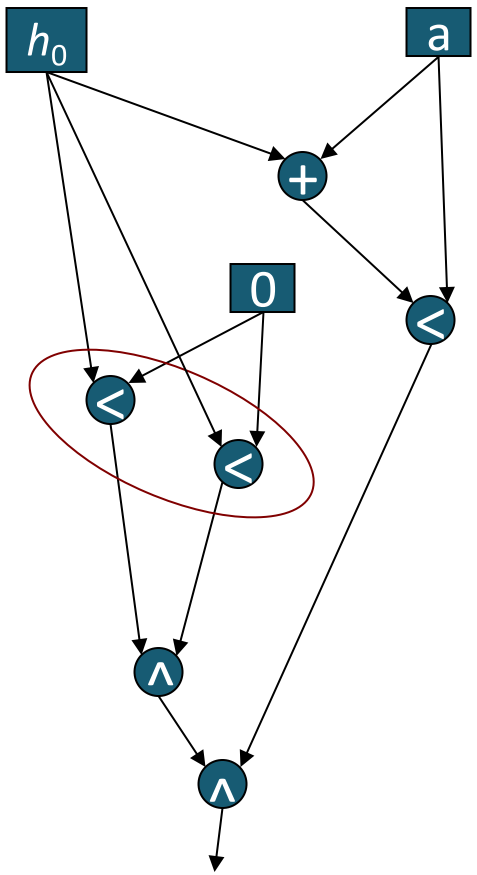
Structural Hashing + Rewriting

$X + b < b \ \rightarrow X < 0$



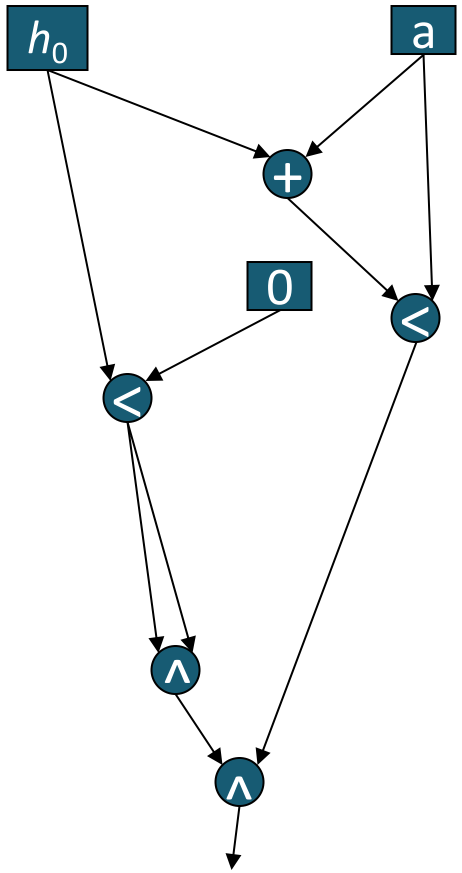
Structural Hashing + Rewriting

$X + b < b \ \rightarrow X < 0$



Structural Hashing + Rewriting

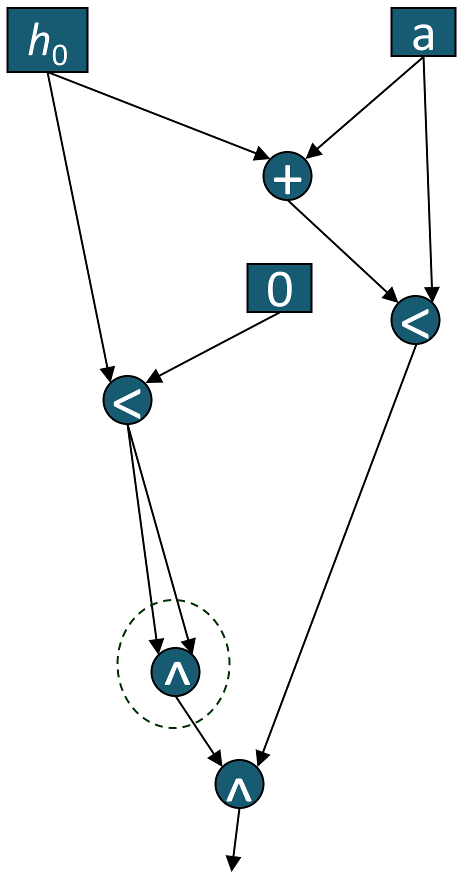
$X + b < b \ \rightarrow X < 0$



Structural Hashing + Rewriting

$X + b < b \ \rightarrow X < 0$

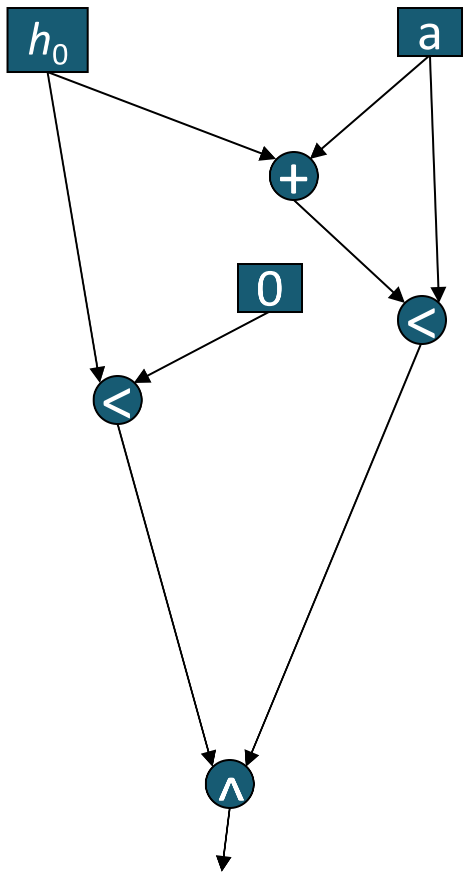
$X \wedge X \rightarrow X$



Structural Hashing + Rewriting

$X + b < b \rightarrow X < 0$

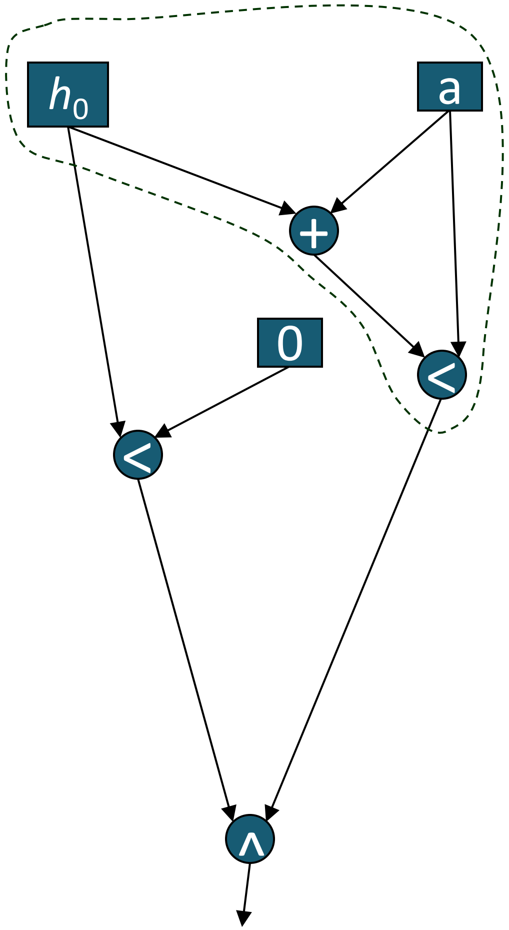
$X \wedge X \rightarrow X$



Structural Hashing + Rewriting

$X + b < b \ \rightarrow X < 0$

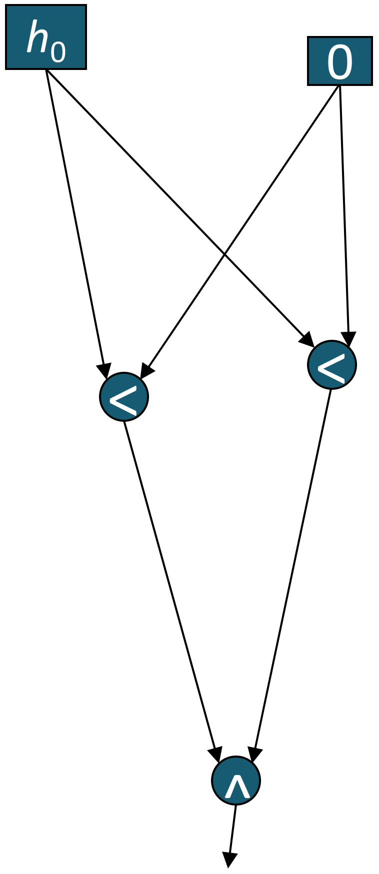
$X \wedge X \rightarrow X$



Structural Hashing + Rewriting

$$X + b < b \ \rightarrow X < 0$$

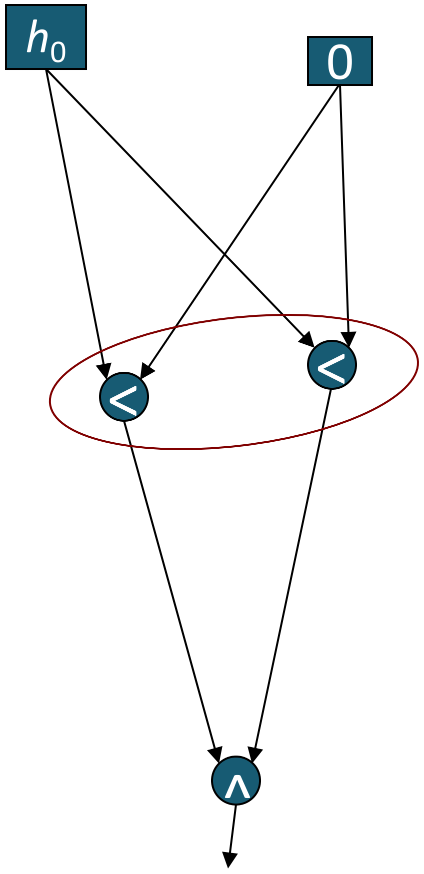
$$X \wedge X \rightarrow X$$



Structural Hashing + Rewriting

$X + b < b \;\rightarrow X < 0$

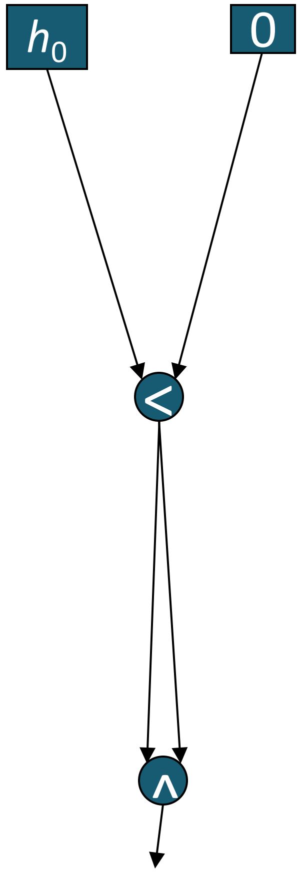
$X \wedge X \rightarrow X$



Structural Hashing + Rewriting

$X + b < b \rightarrow X < 0$

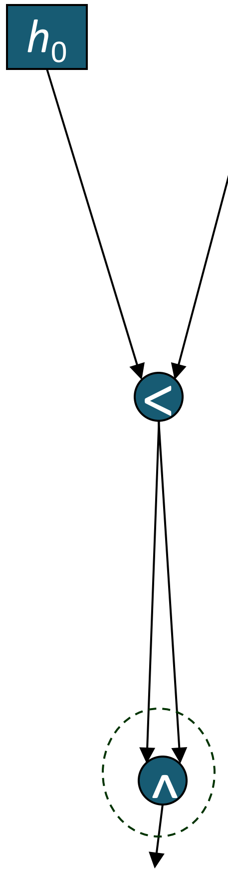
$X \land X \rightarrow X$



Structural Hashing + Rewriting

$X + b < b \rightarrow X < 0$

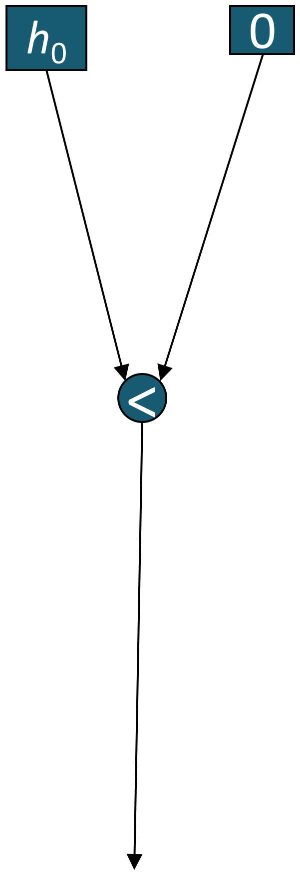
$X \wedge X \rightarrow X$



Structural Hashing + Rewriting

$X + b < b \;\rightarrow X < 0$

$X \wedge X \rightarrow X$



Symmetries

Multiple ways of representing the same problem

```
Expr := var*const
      | Expr + Expr
```

```
Expr := var*const
      | var*const + Expr
```

```
w*c1+(x*c2+(y*c3+z*c4))
```

- Grammar on the right has fewer symmetries
- Grammar on the left can produce all possible ways to parenthesize
- Can completely eliminate symmetries from the right by enforcing a variable ordering
    - Can't be done with a grammar, but it can with a generative model

```
Expr(vmin) := let v = var() in v*const  (assert v > vmin)
            | let v=var() in v*const + Expr(v) (assert v > vmin)
```

Symmetries

Do symmetries matter?

- It depends

Some methods are very sensitive to symmetries

- E.g. symbolic search

Others are largely oblivious to them

- E.g. sampling

How to solve the constraints?

How to solve the constraints?

In general, a Quantified Boolean Formula (QBF) Satisfiability problem:

$\exists \phi \in \{0,1\}^m \ \forall in \in \{0,1\}^n \ Q(in, \phi)$

- 2-QBF is Σ_2-complete
- Reduce to a sequence of SAT problems using the CEGIS loop (coming soon)

The SAT problem: How to check if a quantifier-free Boolean formula α is a tautology (or $\neg\alpha$ is satisfiable) ?

- Naïve algorithm: enumerate all possible models (exponentially many)
- The first known NP-complete problem (Cook 1971)
- At least as hard as *all* NP problems

CNF-SAT Solving

Conjunctive Normal Form (CNF)

- $\bigwedge_{i=1}^{m}(\bigvee_{j=1}^{n} l_{i,j})$
- E.g., $(p_1 \vee \mathrm{p}_2 \vee \neg p_3) \wedge (\neg p_1 \vee \mathrm{p}_2 \vee p_3)$
- Every $\bigvee_{j=1}^{n} l_{i,j}$ is called a clause/conjunct

Theorem: there is no polynomial blow-up translation from wff to CNF/DNF.

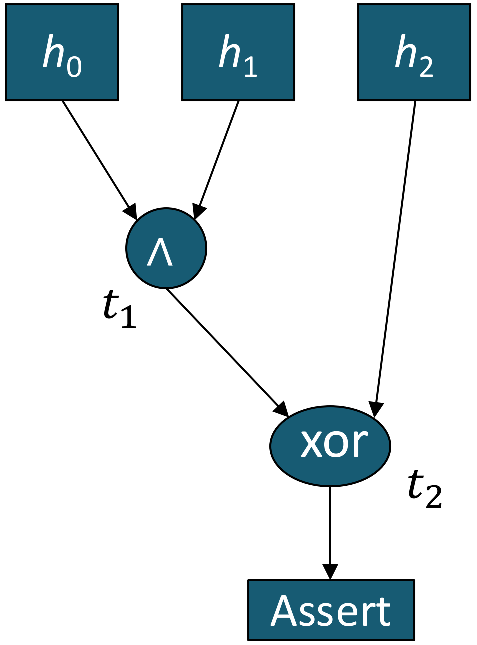
Theorem: SAT can be reduced to CNF-SAT in polynomial time.

- Idea: introduce a fresh variable for each subformula

Cook-Levin Theorem (1971): CNF-SAT is NP-complete.

- Proof: coming soon

Example



$$h_0 \wedge h_1 \Rightarrow t_1$$
$$t_1 \Rightarrow h_0$$
$$t_1 \Rightarrow h_1$$

$$t_1 \wedge h_2 \Rightarrow \overline{t_2}$$
$$\overline{t_1} \wedge \overline{h_2} \Rightarrow \overline{t_2}$$
$$t_1 \wedge \overline{h_2} \Rightarrow t_2$$
$$\overline{t_1} \wedge h_2 \Rightarrow t_2$$

Operation to CNF

- Sum (OR) of variables and their negation
- Equivalent to $\bigwedge_{i \in X} l_i \Rightarrow l_j$

Resolution Algorithm

Resolution: $$\frac{D \vee p \qquad D' \vee \neg p}{D \vee D'}$$

Apply resolution:

- If $D \vee p$ and $D' \vee \neg p$ are clauses, add $D \vee D'$ as a new clause
- Repeat until no more resolution can be done
- Resolution is *closed* if the empty clause is contained
- Return Unsatisfiable iff. Closed

Example

$(p \vee q) \wedge (\neg p \vee r) \wedge (\neg q \vee r) \wedge (\neg r)$

$\{\{p, q\}, \{\neg p, r\}, \{\neg q, r\}, \{\neg r\}\}$

$\{p, q\}$	(1)
$\{\neg p, r\}$	(2)
$\{\neg q, r\}$	(3)
$\{\neg r\}$	(4)
$\{\neg p\}$	(5) (resolvent of 2 and 4)
$\{q\}$	(6) (resolvent of 1 and 5)
$\{r\}$	(7) (resolvent of 3 and 6)
$\{\}$	(8) (resolvent of 4 and 7)

DPLL Algorithm

Backtracking based search

- Assign a value to a variable to simplify the CNF
- Stop if all variables are assigned
- Backtrack if unsatisfiable
- Variables are chosen heuristically

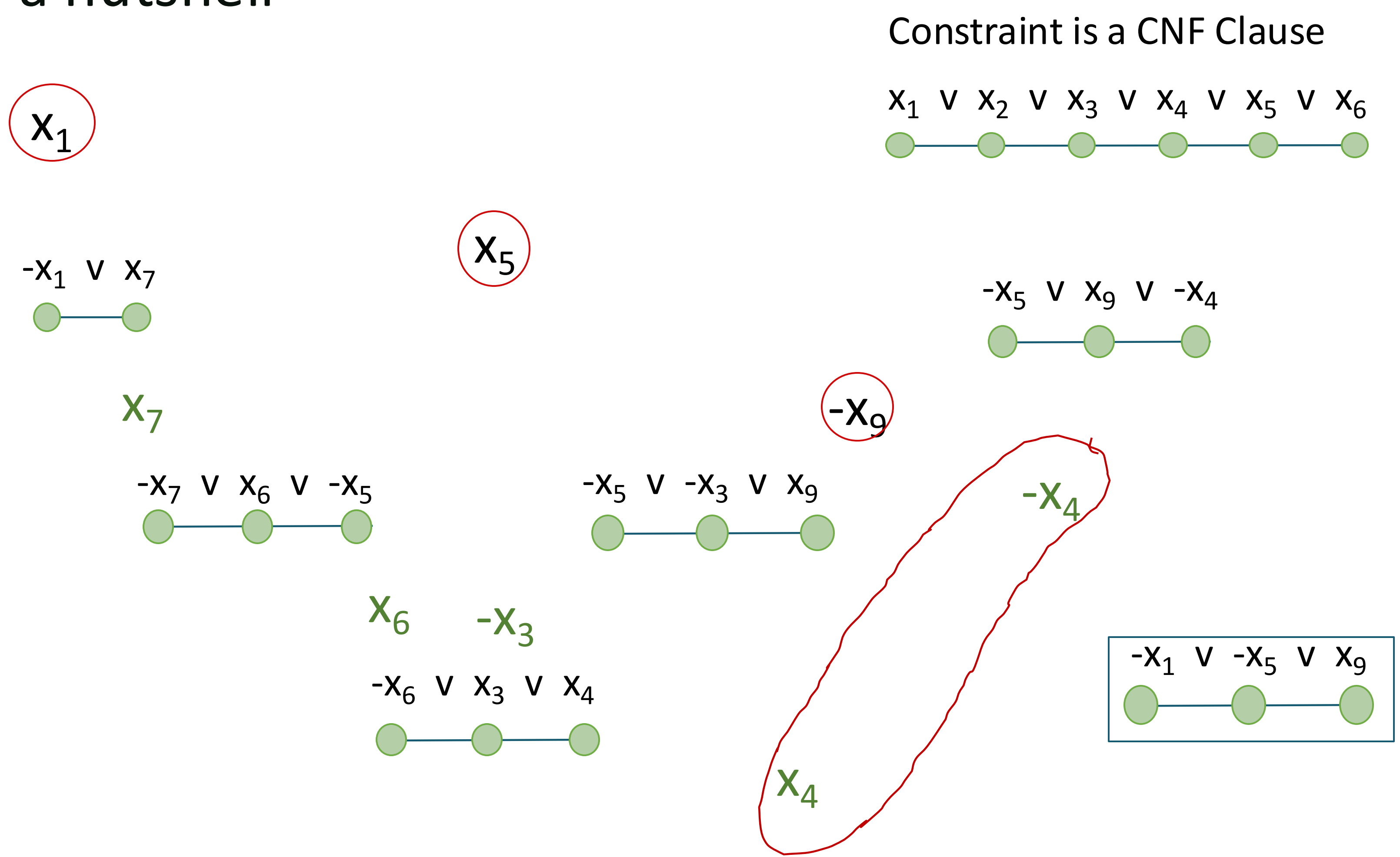
Most efficient SAT solving algorithm since 1960s

- Implementations: zChaff, Minisat, etc.

Example

DPLL in a nutshell

Constraint database



What about Arithmetic?

1) Bit-blast

2) Unary encoding

3) SMT