Experiences in the Research and Development of a Non-clausal SAT Solver

John Franco, Sean Weaver
ECECS, University of Cincinnati
United States Department of Defense

Abridged Sample Input (from CMU webpage)

```
MODULE module-name-changed
INPUT
  ID_EX_RegWrite, ID_EX_MemToReg, _Taken_Branch_1_1, EX_MEM_Jump, ...
OUTPUT  _temp_1252;
STRUCTURE
  _squash_1_1 = or(_Taken_Branch_1_1, EX_MEM_Jump);
  _squash_bar_1_1 = not(_squash_1_1);
  _EX_Jump_1_1 = and(_squash_bar_1_1, ID_EX_Jump);
  _Taken_Branch_9_1 = and(_squash_bar_1_1, ID_EX_Branch, TakeBranchALU_0);
  _Reg2Used_1_1 = or(IF_ID_UseData2, IF_ID_Branch, IF_ID_MemWrite, IF_ID_MemToReg);
  _temp_967 = and(_Reg2Used_1_1, e_2_1);
  ...
  _temp_976 = ite(_temp_969, IF_ID_Jump, Jump_0);
  ...
  _temp_1249 = and(_temp_1038, _temp_1066, _temp_1072, _temp_1189, _temp_1246);
  ...
  true_value = new_int_leaf(1);
  are_equal(_temp_1252, true_value); % 1
ENDMODULE
```

Courtesy M.N. Velev, Superscalar Suite 1.0. Available from: http://www.ece.cmu.edu/˜ mvelev.

Translation to CNF

Expression: $v_3 = ite(v_0, v_1, v_2);$

Karnaugh Map:

	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	0	1	1	0
10	1	0	0	1

CNF:

$$(v_0 \vee v_2 \vee \bar{v}_3)$$
$$(v_0 \vee \bar{v}_2 \vee v_3)$$
$$(\bar{v}_0 \vee \bar{v}_1 \vee v_3)$$
$$(\bar{v}_0 \vee v_1 \vee \bar{v}_3)$$

Translation to CNF

Expression: $v_3 = ite(v_0, v_1, v_2);$

Karnaugh Map:

	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	0	1	1	0
10	1	0	0	1

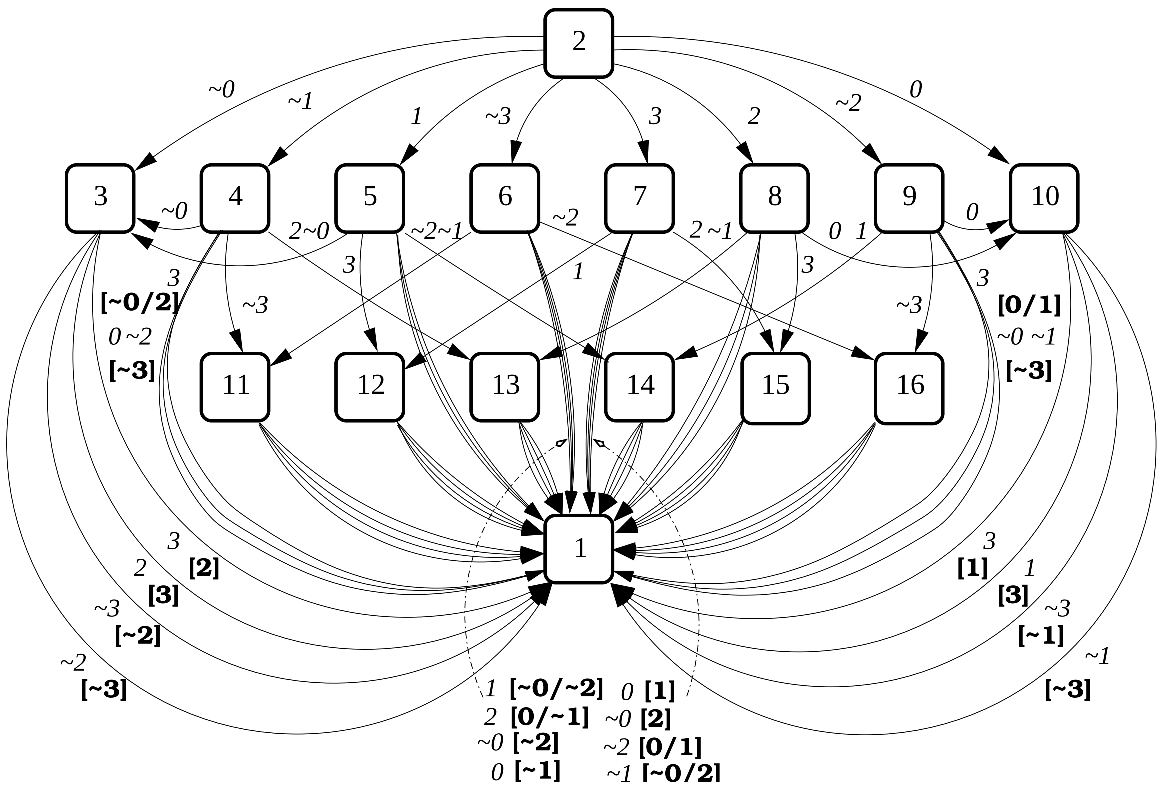
CNF:

$$(v_0 \vee v_2 \vee \bar{v}_3)$$
$$(v_0 \vee \bar{v}_2 \vee v_3)$$
$$(\bar{v}_0 \vee \bar{v}_1 \vee v_3)$$
$$(\bar{v}_0 \vee v_1 \vee \bar{v}_3)$$

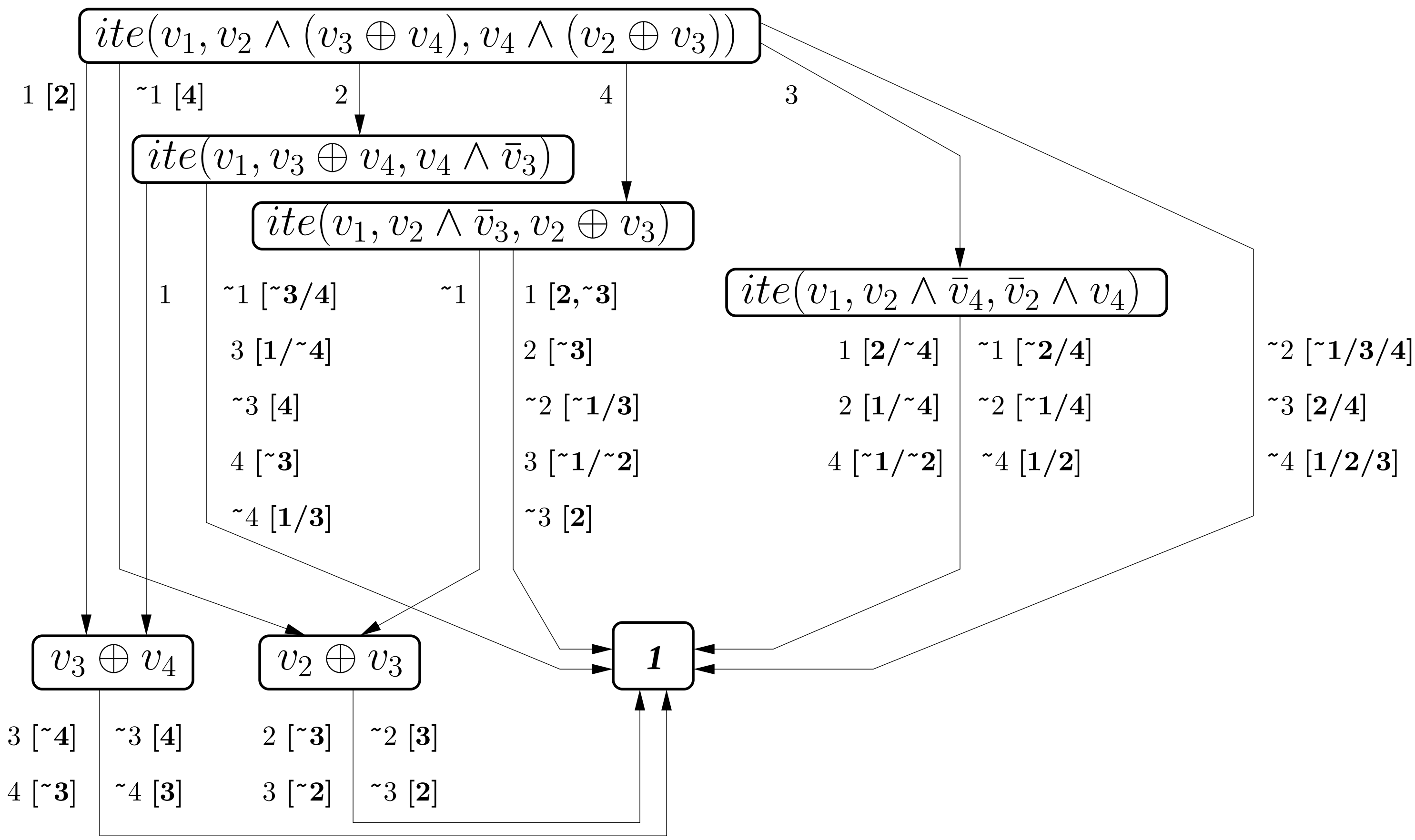
What Heuristics Like:

$$(v_0 \wedge v_1) \rightarrow v_3 \qquad (v_0 \wedge v_3) \rightarrow v_1$$
$$(\bar{v}_0 \wedge v_2) \rightarrow v_3 \qquad (\bar{v}_0 \wedge v_3) \rightarrow v_2$$
$$(v_0 \wedge \bar{v}_1) \rightarrow \bar{v}_3 \qquad (v_0 \wedge \bar{v}_3) \rightarrow \bar{v}_1$$
$$(\bar{v}_0 \wedge \bar{v}_2) \rightarrow \bar{v}_3 \qquad (\bar{v}_0 \wedge \bar{v}_3) \rightarrow \bar{v}_2$$
$$(v_1 \wedge v_2) \rightarrow v_3 \qquad (\bar{v}_1 \wedge \bar{v}_2) \rightarrow \bar{v}_3$$
$$(v_1 \wedge \bar{v}_3) \rightarrow \bar{v}_0, \bar{v}_2 \qquad (v_2 \wedge \bar{v}_3) \rightarrow v_0, \bar{v}_1$$
$$(\bar{v}_1 \wedge v_3) \rightarrow \bar{v}_0, v_2 \qquad (\bar{v}_2 \wedge v_3) \rightarrow v_0, v_1$$
$$(v_1 \wedge \bar{v}_2) \rightarrow v_0 = v_3$$
$$(\bar{v}_1 \wedge v_2) \rightarrow \bar{v}_0 = v_3$$

State Machine Used to Represent Functions



States of a SMURF representing $v_3 = ite(v_0, v_1, v_2)$


$ite(v_1, v_2 \wedge (v_3 \oplus v_4), v_4 \wedge (v_2 \oplus v_3))$
1 [2]
~1 [4]
2
4
3
$ite(v_1, v_3 \oplus v_4, v_4 \wedge \bar{v}_3)$
$ite(v_1, v_2 \wedge \bar{v}_3, v_2 \oplus v_3)$
$ite(v_1, v_2 \wedge \bar{v}_4, \bar{v}_2 \wedge v_4)$
1
~1 [~3/4]
3 [1/~4]
~3 [4]
4 [~3]
~4 [1/3]
~1
1 [2,~3]
2 [~3]
~2 [~1/3]
3 [~1/~2]
~3 [2]
1 [2/~4]
2 [1/~4]
4 [~1/~2]
~1 [~2/4]
~2 [~1/4]
~4 [1/2]
~2 [~1/3/4]
~3 [2/4]
~4 [1/2/3]
$v_3 \oplus v_4$
$v_2 \oplus v_3$
1
3 [~4]
4 [~3]
~3 [4]
~4 [3]
2 [~3]
3 [~2]
~2 [3]
~3 [2]

Locally Skewed, Globally Balanced Heuristics

Weight of terminal state is 0

If state s has p successors $\{s_1, s_2, \ldots, s_p\}$, weight of s is

$$\frac{\sum_{i=1}^{p} weightOf(s_i) + numberInferencesMadeEnrouteTo(s_i)}{K * p}$$

Every state transition gets a weight:

inferences to destination state plus weight of that state

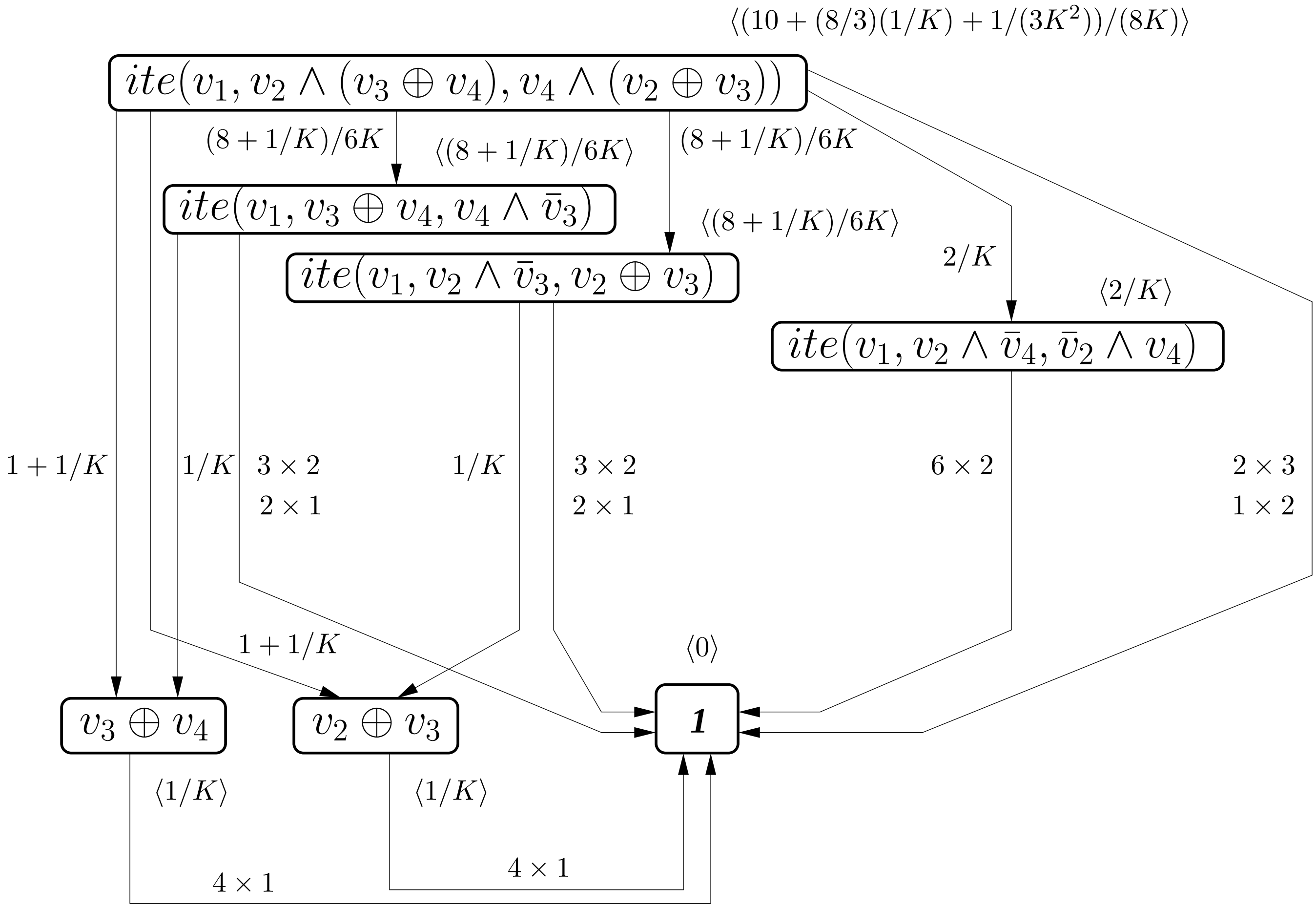
Every literal gets a score:

Sum of transition weights for that literal across SMURFs

Every variable v gets a score:

$(score(v) + \epsilon) * (score(\bar{v}) + \epsilon)$

Branch on highest scoring variable



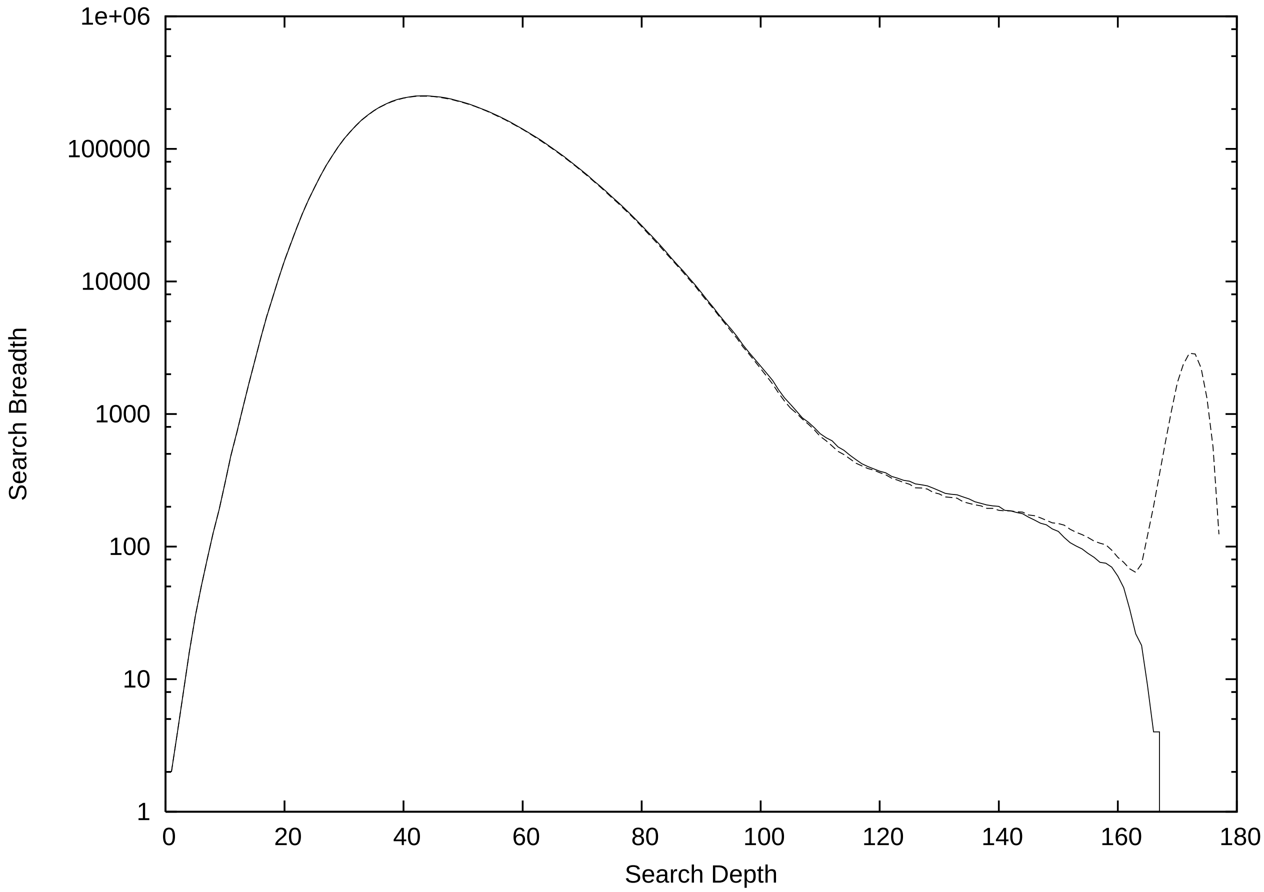
⟨(10 + (8/3)(1/K) + 1/(3K^2))/(8K)⟩
ite(v1, v2 ∧ (v3 ⊕ v4), v4 ∧ (v2 ⊕ v3))
(8 + 1/K)/6K
⟨(8 + 1/K)/6K⟩
(8 + 1/K)/6K
ite(v1, v3 ⊕ v4, v4 ∧ v̄3)
⟨(8 + 1/K)/6K⟩
ite(v1, v2 ∧ v̄3, v2 ⊕ v3)
2/K
⟨2/K⟩
ite(v1, v2 ∧ v̄4, v̄2 ∧ v4)
1 + 1/K
1/K
3 × 2
2 × 1
1/K
3 × 2
2 × 1
6 × 2
2 × 3
1 × 2
1 + 1/K
⟨0⟩
v3 ⊕ v4
v2 ⊕ v3
1
⟨1/K⟩
⟨1/K⟩
4 × 1
4 × 1

Require Pre-processing

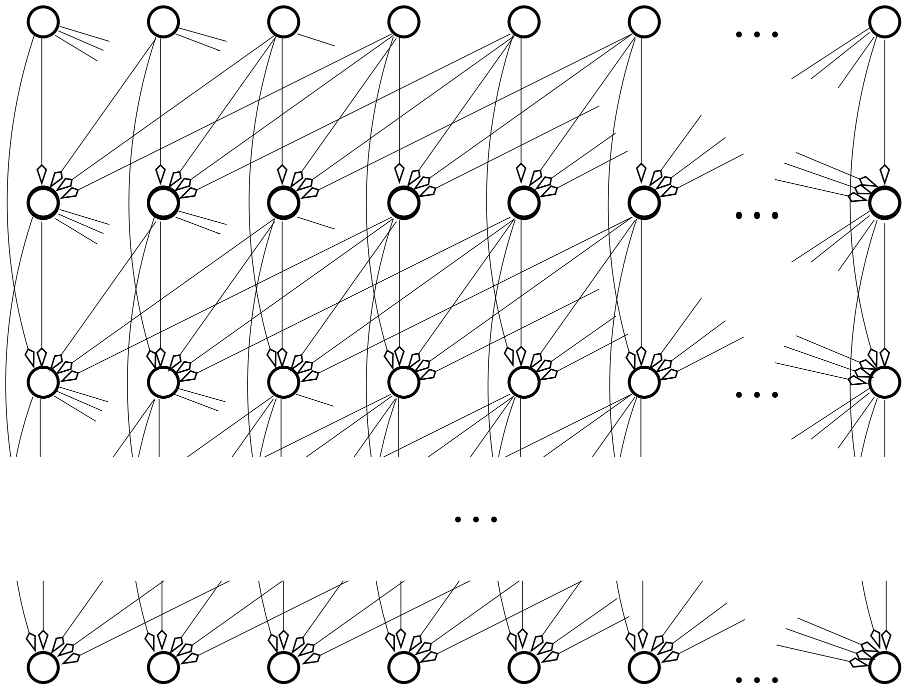
Intuitively, we desire:

- Fewer State Machines
- Smaller State Machines
- Redundancies Removed Across State Machines
- Inferences Revealed and Assigned as Early as Possible
- Safe Assignments Revealed and Assigned Early

Search Space Profile for Hard Problems



Hard Problems That Fit This Profile

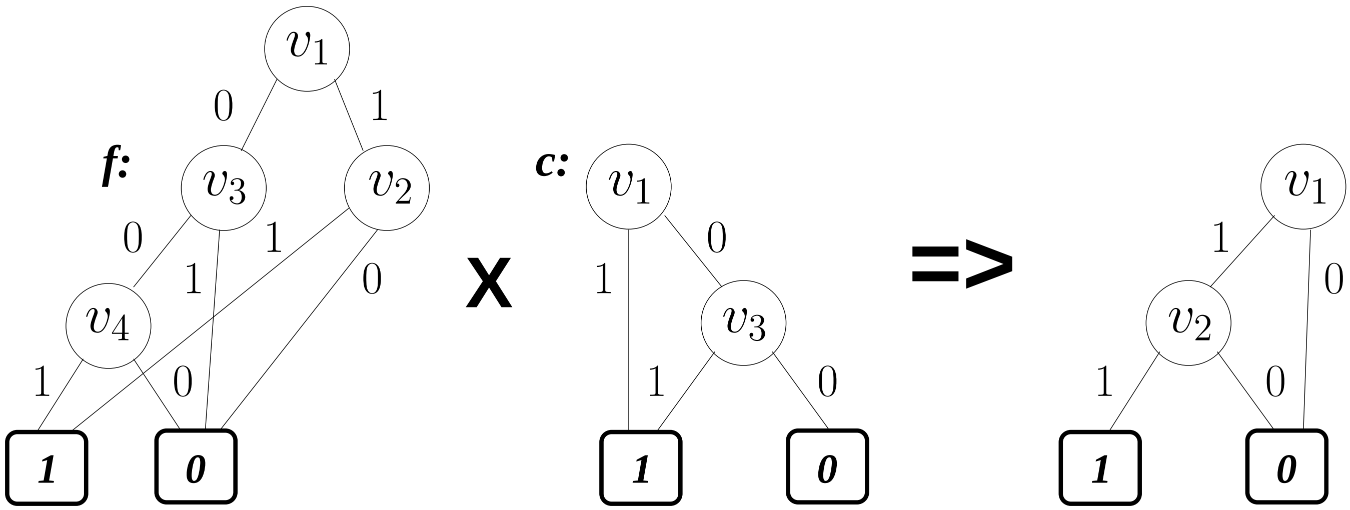


For example, Bounded Model Checking problems

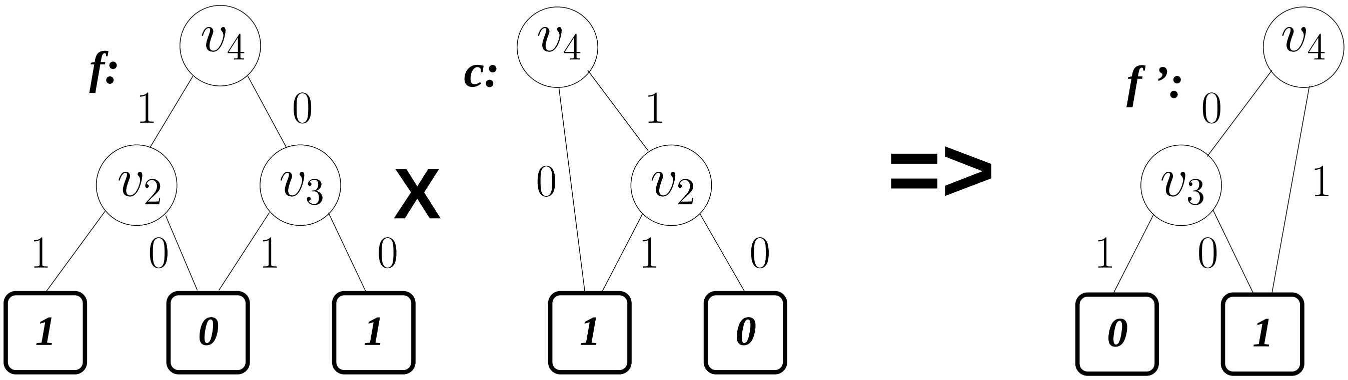
Tools for Pre-processing

- Restriction (eliminate redundancies, find inferences)
- Strengthening (find inferences missed by restriction)
- Generalized co-factor (eliminate functions)
- Cluster some functions (conjoin them)
- Existential Quantification (eliminate variables)
- Assign uninferred but safe values (reductions)
- Add uninferred and unsafe constraints (tunnel)

Restrict



Restrict

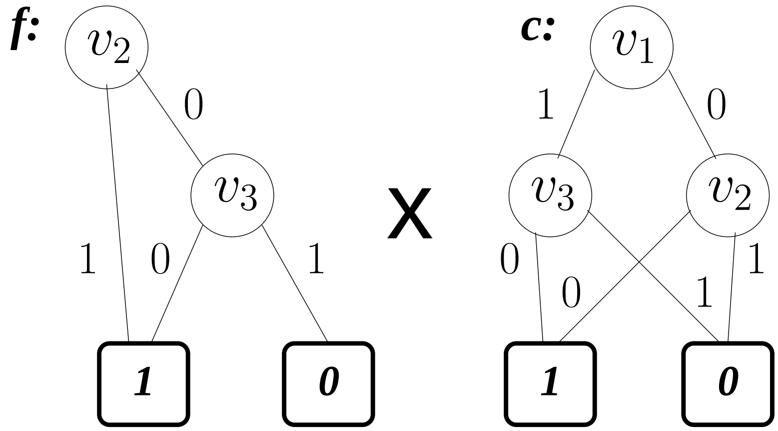


Spreading an inference from one function to another.

If $v_2 = 0$ in f then $v_3 = v_4 = 0$ is inferred.

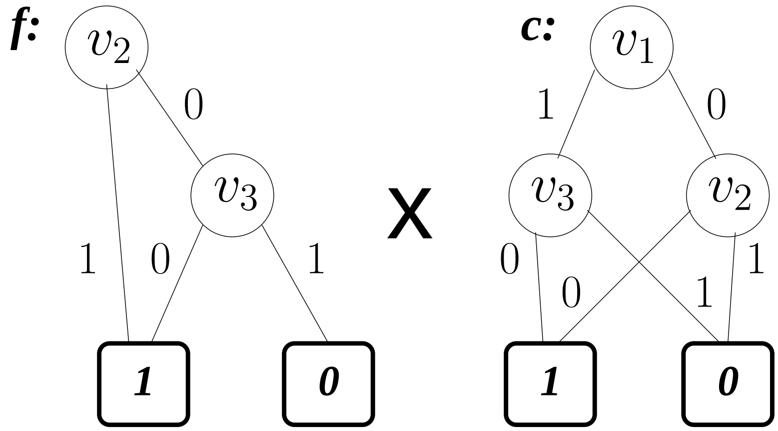
Replacing f with f', gives inference $v_4 = 0$ from c (if $v_2 = 0$) and then inference $v_3 = 0$ from f'.

Strengthening

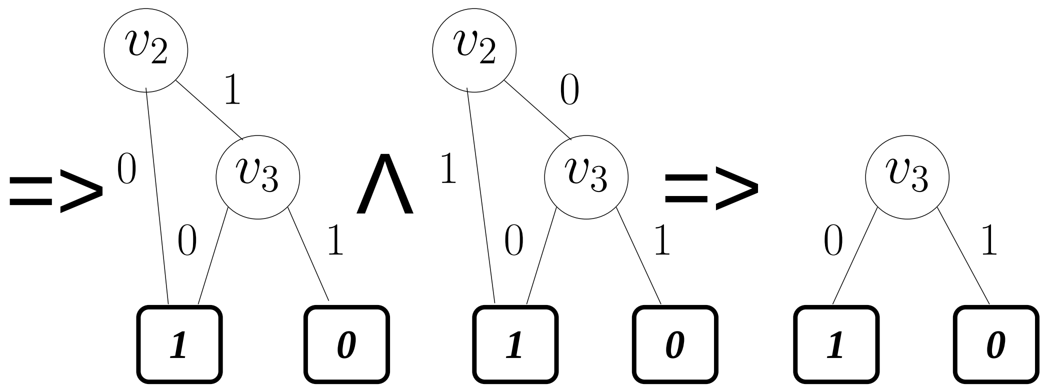


Existentially quantify away v_1 from f, then ...

Strengthening



Existentially quantify away v_1 from f, then ...



conjoin f and c to reveal inference $v_3 = 0$.

Clustering and Existential Quantification

$$\exists v(f_1 \wedge \ldots \wedge f_m) \equiv (f_1 \wedge \ldots \wedge f_m)|_{v=0} \vee (f_1 \wedge \ldots \wedge f_m)|_{v=1}$$

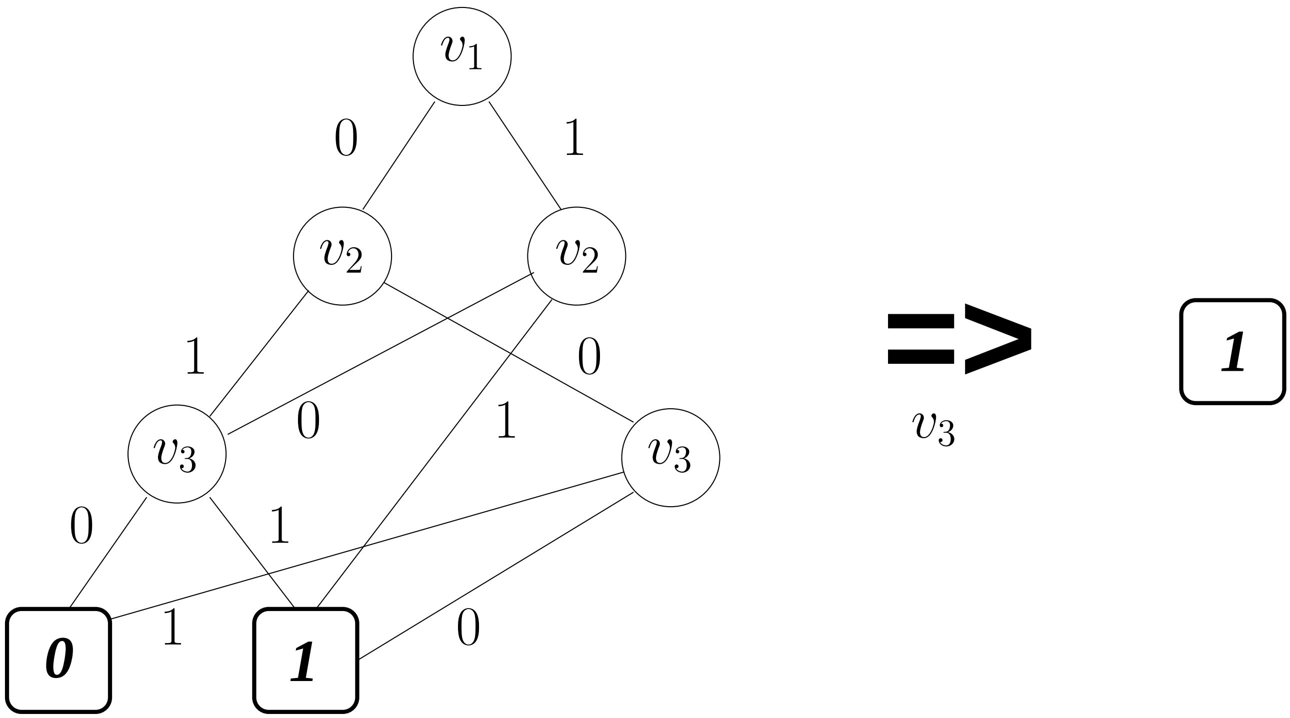
Clustering and Existential Quantification

$$\exists v(f_1 \wedge \ldots \wedge f_m) \equiv (f_1 \wedge \ldots \wedge f_m)|_{v=0} \vee (f_1 \wedge \ldots \wedge f_m)|_{v=1}$$

But this is what we really want:

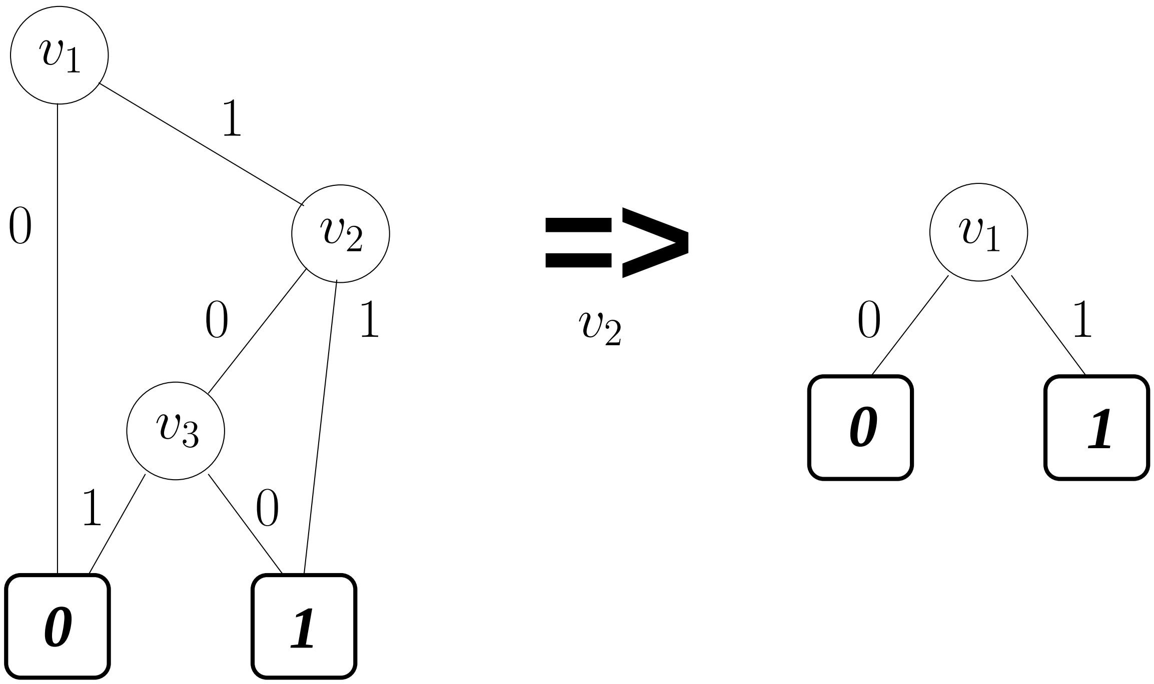
$$\exists v(f_1 \wedge \ldots \wedge f_m) \equiv (f_1|_{v=0} \vee f_1|_{v=1}) \wedge \ldots \wedge (f_m|_{v=0} \vee f_m|_{v=1})$$

Existential Quantification



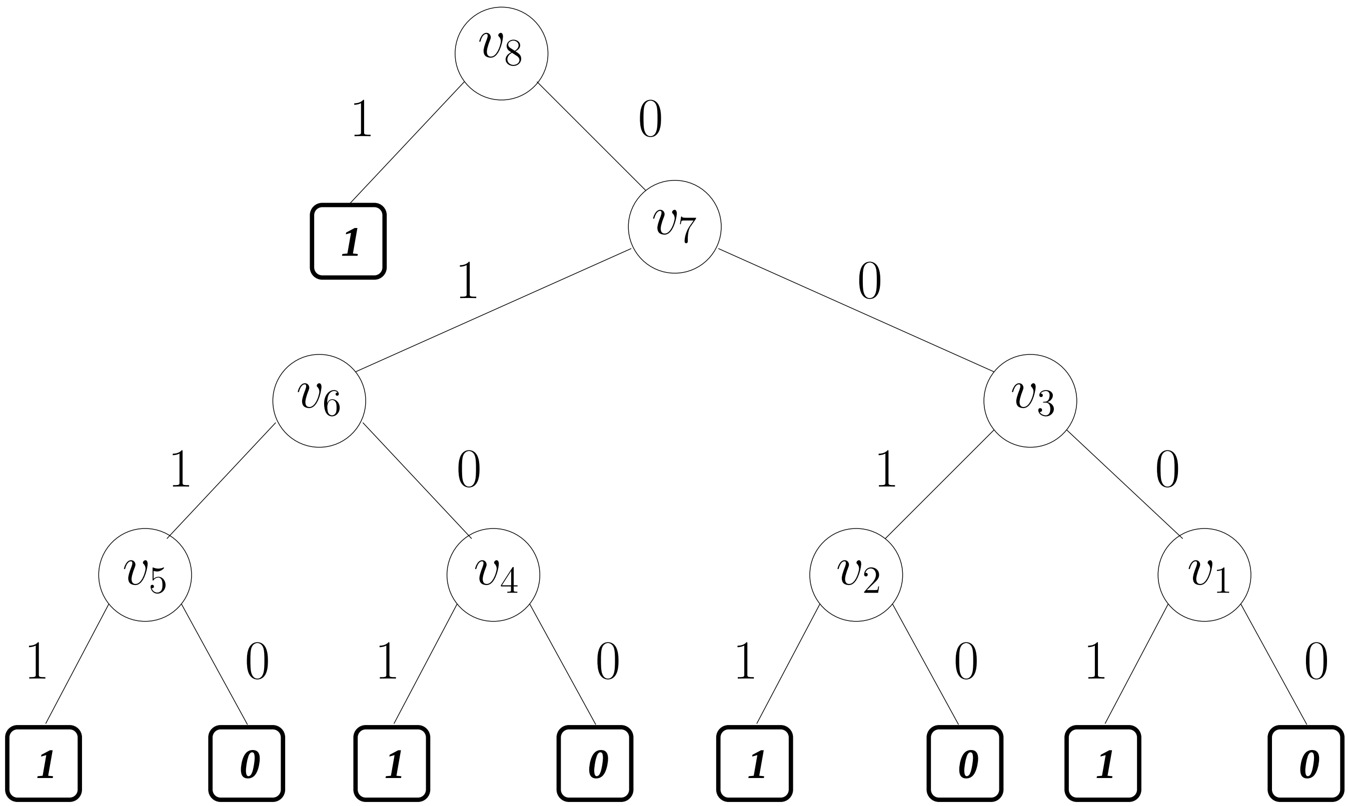
Replace f with $(f|_{v=0} \vee f|_{v=1})$

Existential Quantification



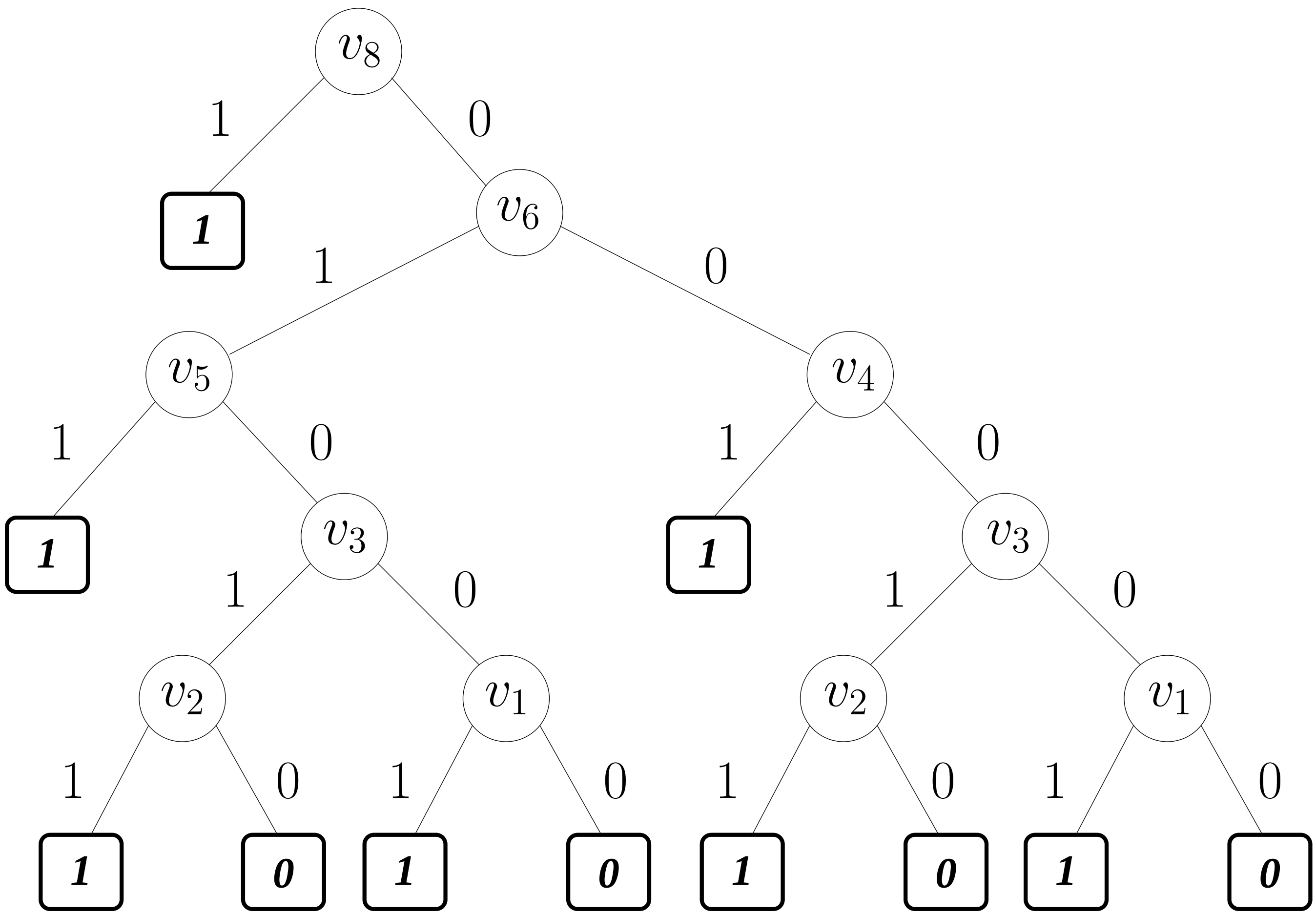
Replace f with $(f|_{v=0} \vee f|_{v=1})$

Existential Quantification



Choose v_7 to separate v_1, v_2, v_3 from v_4, v_5, v_6

Existential Quantification



Splitting An Expression Is Desirable

$$\underbrace{f_1 \wedge f_2 \wedge \ldots \wedge f_i} \wedge \underbrace{f_{i+1} \wedge \ldots \wedge f_{m-1} \wedge f_m}$$

If no variables
in f_1 to f_i
are in f_{i+1} to f_m

and no variables
in f_{i+1} to f_m
are in f_1 to f_i

then $\phi_1 = f_1 \wedge \ldots \wedge f_i$ and $\phi_2 = f_{i+1} \wedge \ldots \wedge f_m$
can be solved independently

Autark Assignments Come Close

$$\underbrace{f_1 \wedge f_2 \wedge \ldots \wedge f_i}_{\text{If there is a subset } V' \text{ of variables in } f_1 \text{ to } f_i \text{ and an assignment } t_{V'} \text{ of values to } V' \text{ that satisfies } f_1 \text{ to } f_i} \wedge \underbrace{f_{i+1} \wedge \ldots \wedge f_{m-1} \wedge f_m}_{\text{and none of the variables of } V' \text{ is in } f_{i+1} \text{ to } f_m}$$

then satisfy $\phi_1 = f_1 \wedge \ldots \wedge f_i$ with partial assignment $t_{V'}$ and solve $\phi_2 = f_{i+1} \wedge \ldots \wedge f_m$ independently

Safe Assignments

$$\text{If } (f_1 \wedge \ldots \wedge f_m)|_{v=0} \equiv (f_1 \wedge \ldots \wedge f_m)|_{v=0} \vee (f_1 \wedge \ldots \wedge f_m)|_{v=1}$$

$$\text{then } v = 0 \text{ is safe}$$

$$\text{If } (f_1 \wedge \ldots \wedge f_m)|_{v=1} \equiv (f_1 \wedge \ldots \wedge f_m)|_{v=0} \vee (f_1 \wedge \ldots \wedge f_m)|_{v=1}$$

$$\text{then } v = 1 \text{ is safe}$$

Example: $v = 1$ is a safe assignment but is not autark:

$$f = (v \vee a) \wedge (v \vee \bar{a} \vee b) \wedge (\bar{v} \vee \bar{a} \vee b \vee c) \wedge (\bar{b} \vee \bar{c}) \ldots$$

Because

$$f|_{v=0} = (a) \wedge (\bar{a} \vee b) \wedge (\bar{b} \vee \bar{c}) \rightarrow$$
$$f|_{v=1} = (\bar{a} \vee b \vee c) \wedge (\bar{b} \vee \bar{c})$$

Safe Assignments: Checking isn't too bad

But conjoining functions to find safe assignments can be expensive

Luckily, the computational effort can be distributed:

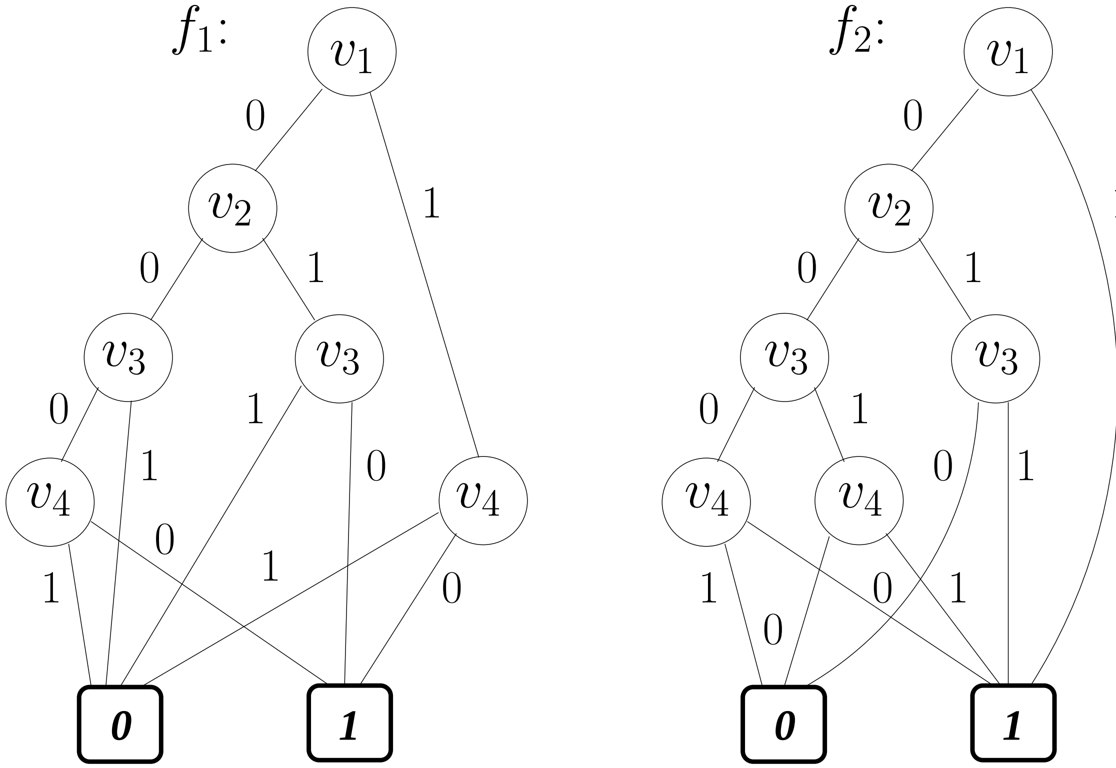
If $\left(\overline{f_i|_v} \wedge f_i|_{\bar{v}}\right) \equiv 0$ for every i such that v is in f_i then $v = 1$ is safe

If $\left(f_i|_v \wedge \overline{f_i|_{\bar{v}}}\right) \equiv 0$ for every i such that v is in f_i then $v = 0$ is safe

But the check may fail on some safe assignments

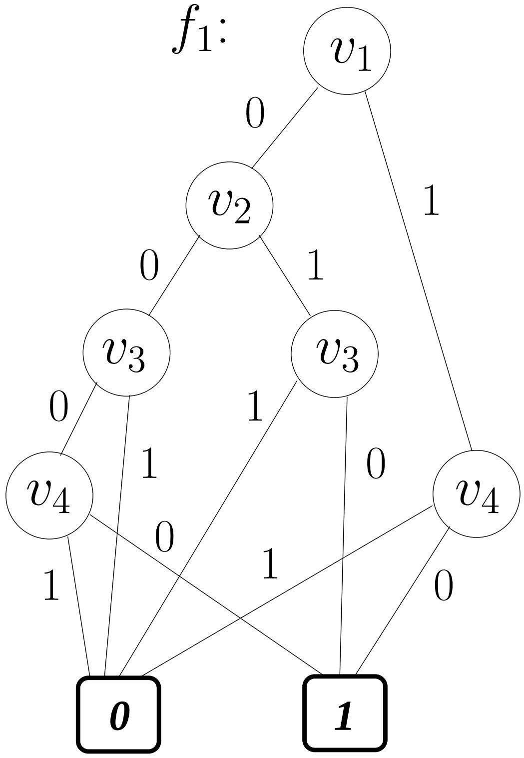
The idea extends to groups of variables

Safe Assignments: Example, Single Variable



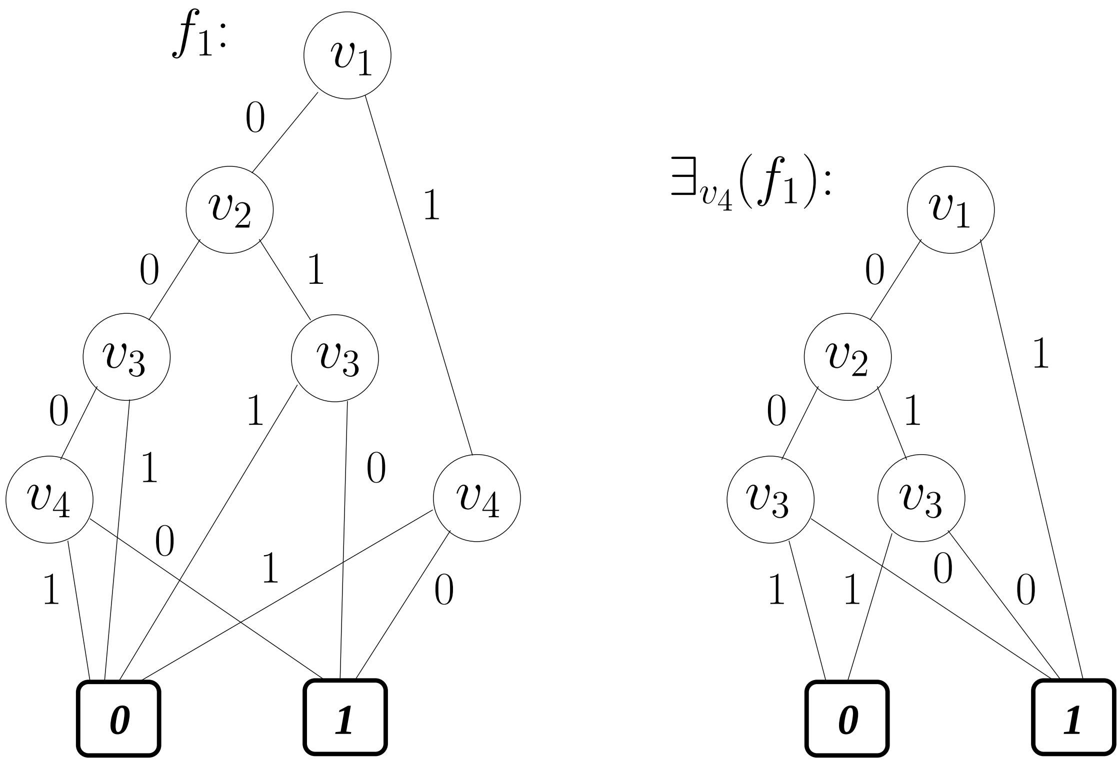
Consider two functions f_1 and f_2

Safe Assignments: Example, Single Variable



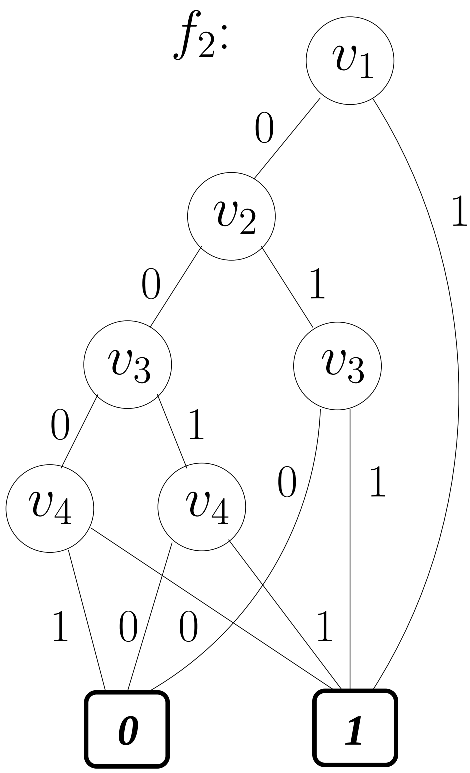
Finding safe assignment for v_4 in f_1 alone:

Safe Assignments: Example, Single Variable



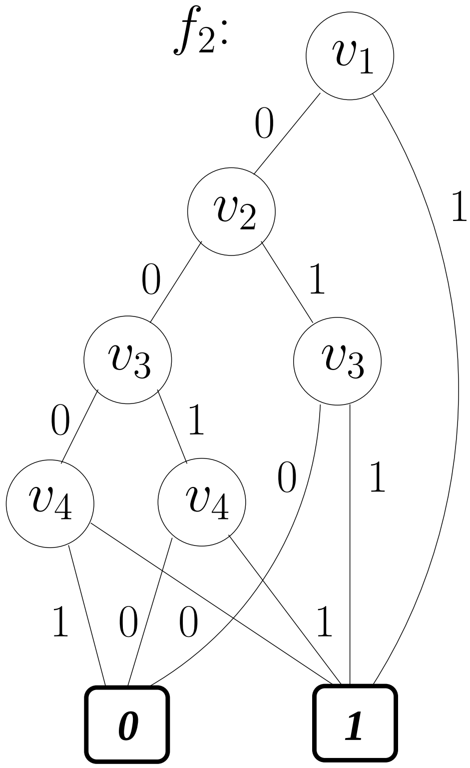
Finding safe assignment for v_4 in f_1 alone: $v_4 = 0$ only

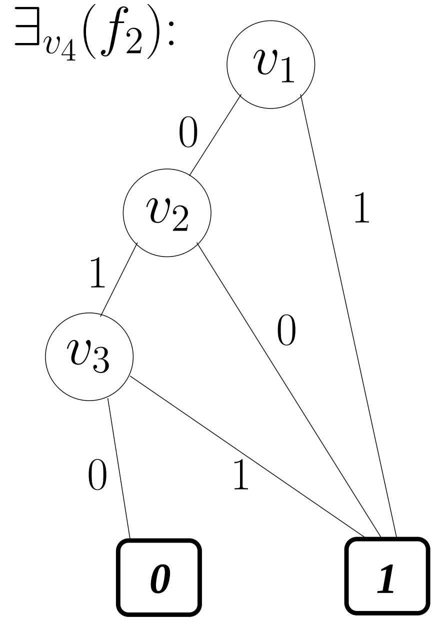
Safe Assignments: Example, Single Variable



Finding safe assignment for v_4 in f_2 alone:

Safe Assignments: Example, Single Variable

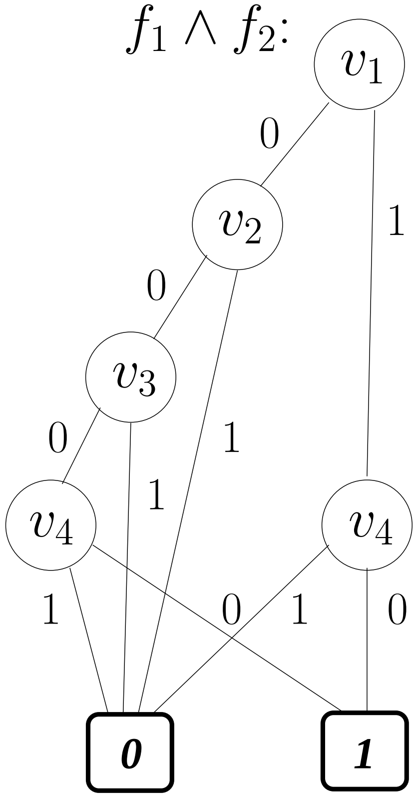




Finding safe assignment for v_4 in f_2 alone: neither one

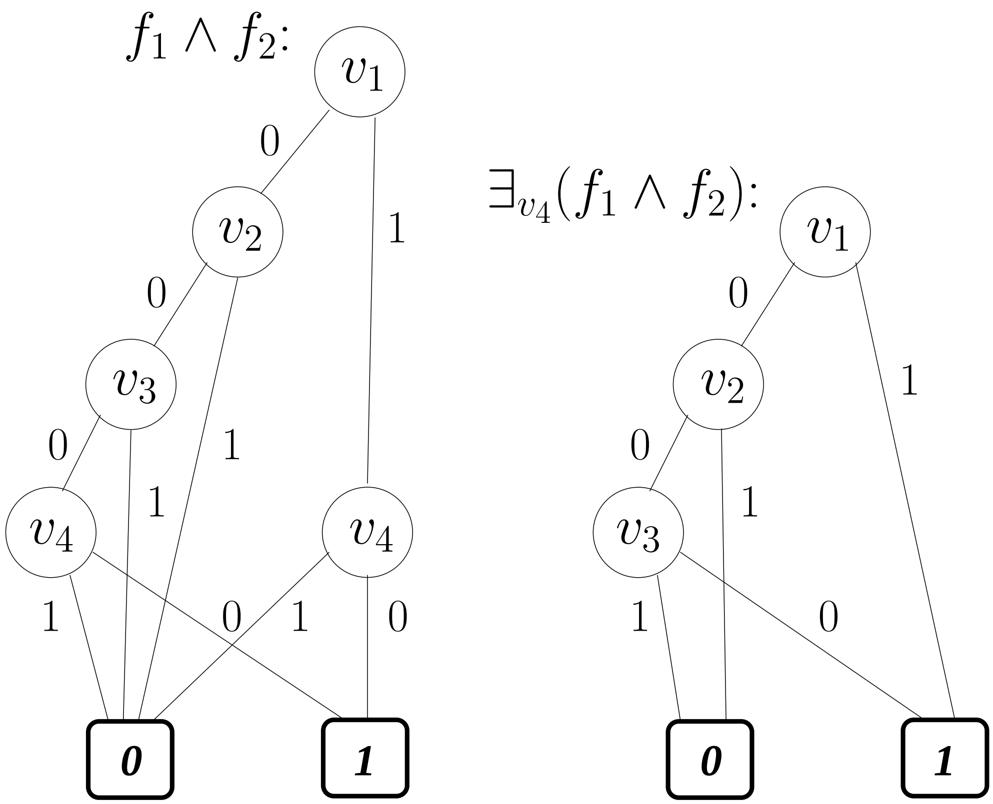
Safe Assignments: Some Are Missed

$f_1 \wedge f_2$:



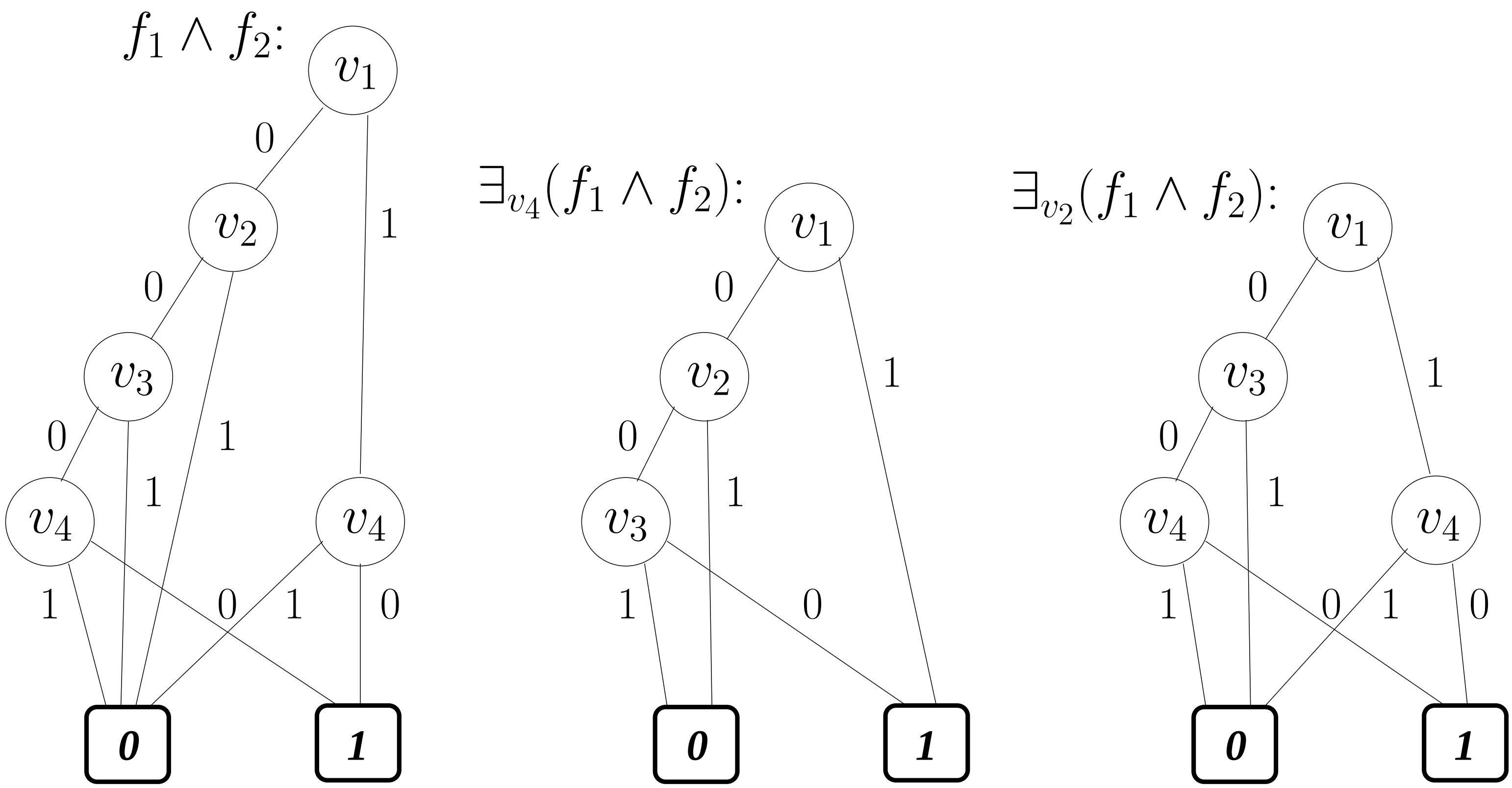
Conjoin the two functions

Safe Assignments: Some Are Missed



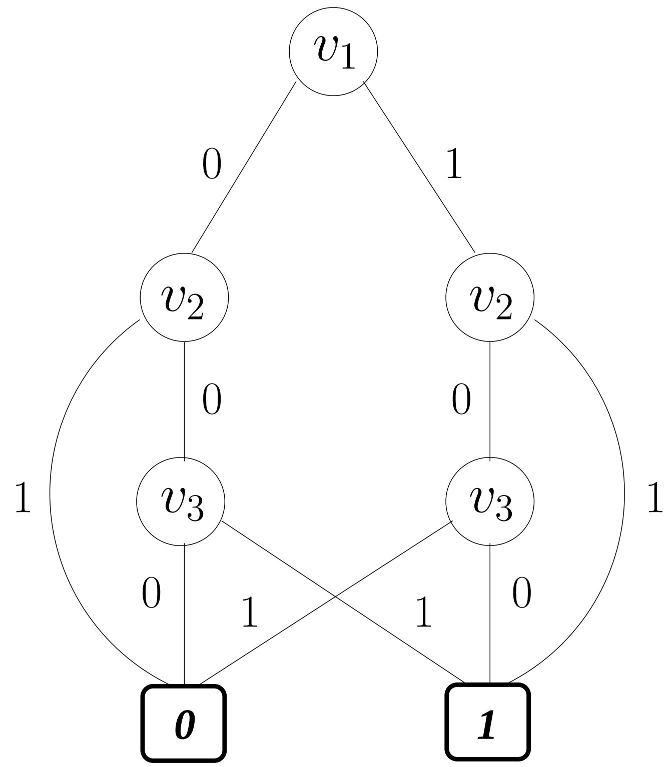
Conjoin the two functions then find $v_4 = 0$ is safe

Safe Assignments: Some Are Missed



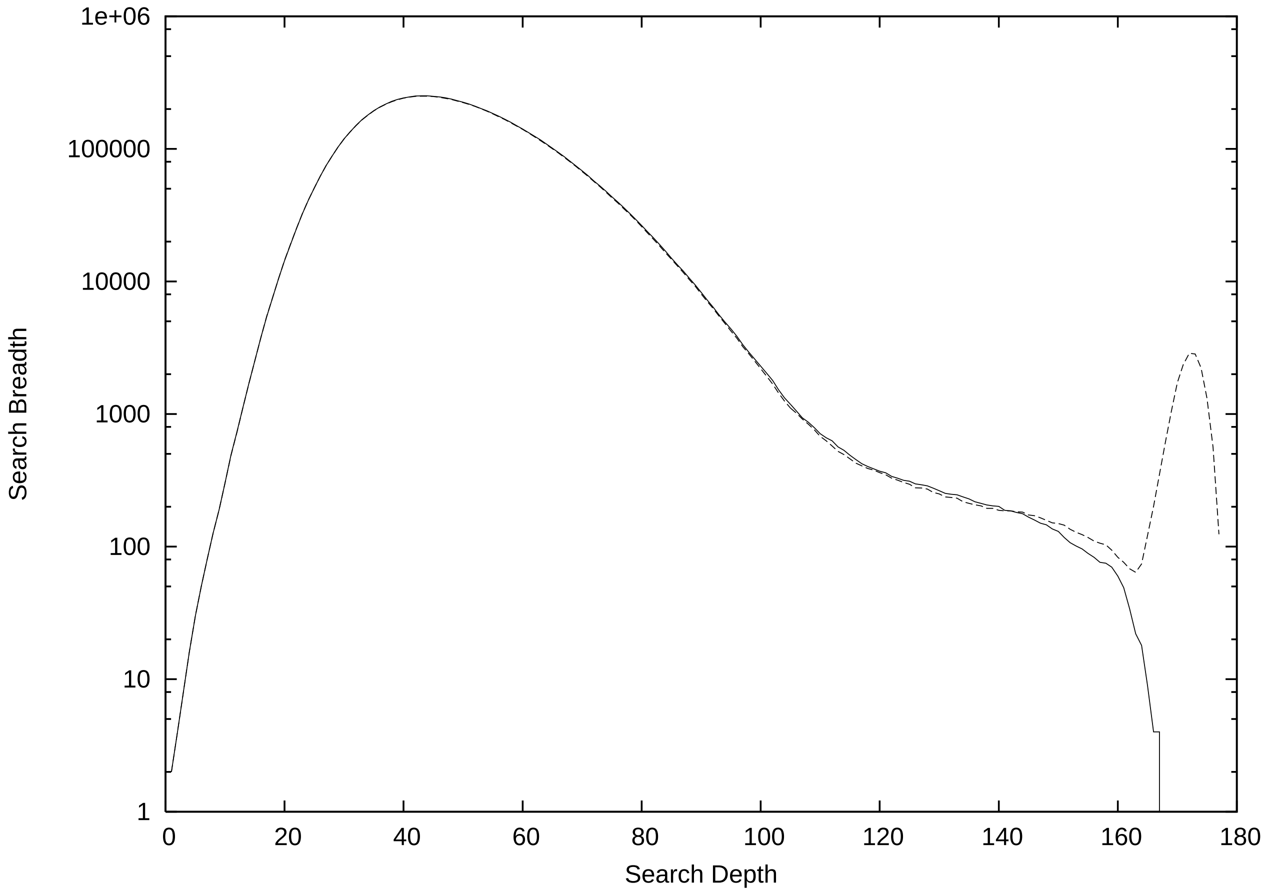
Conjoin the two functions then find $v_4 = 0$ is safe, also $v_2 = 0$ is safe

Safe Assignments: But Multiple Assignments Can Pay Off



Safe assignments $v_1 = v_2 = 1$ and $v_1 = 1, v_3 = 0$ are found as pairs only

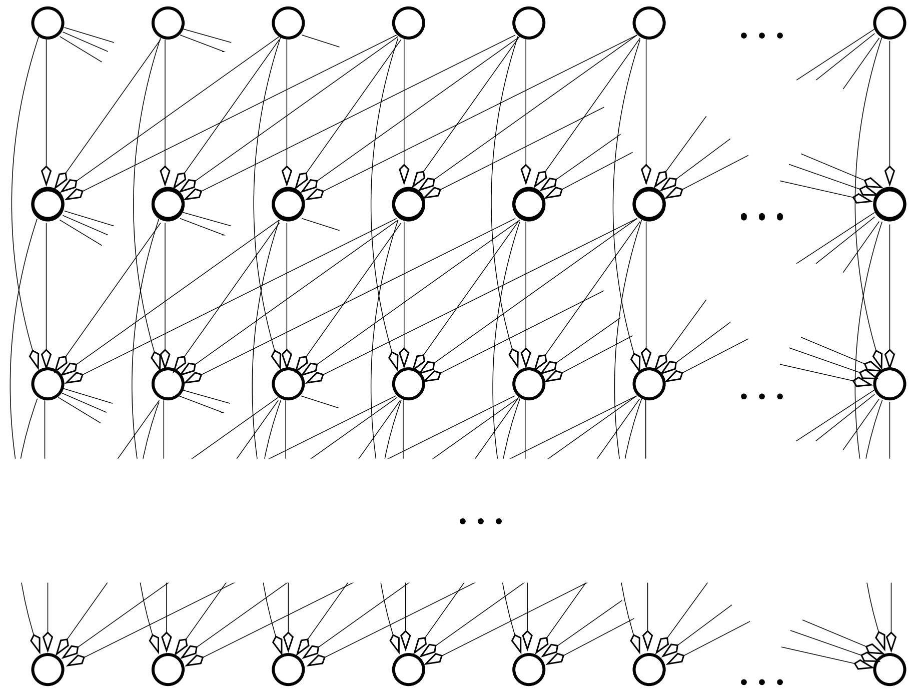
Search Space Profile for Hard Problems



Unsafe Assignments

- Guess some (uninferred) constraints based on *solution* structure in the same family
- Add those constraints initially to reduce the "hump"
- Run the search breadth-first
- When search breadth begins to decline, remove the constraints
- Solve to completion
- Possibly no solution found for a satisfiable input

Example: Van der Waerden Numbers



An ordering of the input variables is natural

Example: Van der Waerden Numbers

```
1 1     1 1           1 1     1 1
 0 0   1 0 0 1     1 1 0 0   1 0 0
    0 1     0 0   1 0     0 1
     0         0 1         0
                0

1010001110100100011101101000111010
```

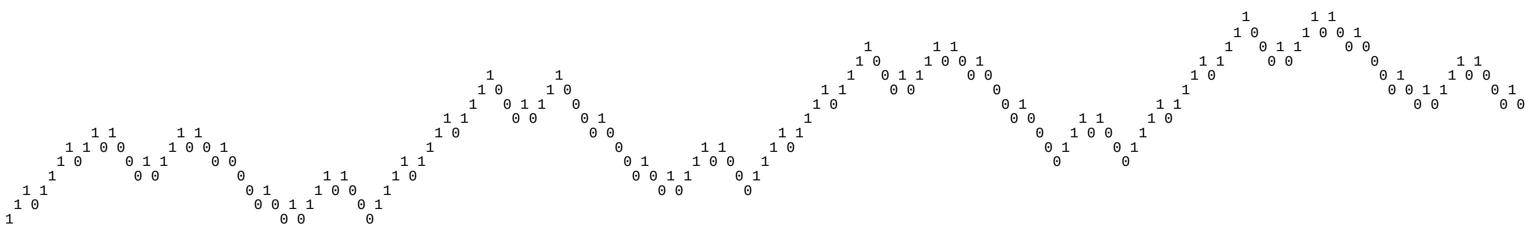
2 categories, progression length 4 ($W_{max}(2, 4)$ formula)

Example: Van der Waerden Numbers

```
1 1     1 1           1 1     1 1
 0 0   1 0 0 1     1 1 0 0   1 0 0
    0 1     0 0   1 0     0 1
     0         0 1         0
                0
```

```
1010001110100100011101101000111010
```

2 categories, progression length 4 ($W_{max}(2,4)$ formula)



```
111011110110100010111010010000100101110100011110111101111000101110000100001001011101000111101111011110001011101001000010000111010001111011110111100010111010010000100101110100010
```

2 categories, progression length 5 ($W_{max}(2,5)$ formula)

Analysis of Solutions Suggests...

Conjecture:

For every $W_{max}(2, l)$ formula there exists a solution that contains at least one reflected pattern of length $W(2, l)/(2 * (l - 1))$ with the middle positioned somewhere between $W(2, l)/(l - 1)$ and $W(2, l) * (l - 2)/(l - 1)$.

Analysis of Solutions Suggests...

Conjecture:

For every $W_{max}(2, l)$ formula there exists a solution that contains at least one reflected pattern of length $W(2, l)/(2 * (l - 1))$ with the middle positioned somewhere between $W(2, l)/(l - 1)$ and $W(2, l) * (l - 2)/(l - 1)$.

From search profile:

The maximum breadth occurs near depth $W(2, l)/(2(l - 1))$.

$W(2, l) \approx l * W(2, l - 1)$, for small l anyway.

Continuing...

Add the unsafe constraints:

Function Seg	Range	Meaning
$(v_{-i} \vee v_{i+1}) \wedge (\bar{v}_{-i} \vee \bar{v}_{i+1})$	$0 \leq i < s/2$	force $v_{-i} \equiv \bar{v}_{i+1}$.

Continuing...

Add the unsafe constraints:

Function Seg	**Range**	**Meaning**
$(v_{-i} \vee v_{i+1}) \wedge (\bar{v}_{-i} \vee \bar{v}_{i+1})$	$0 \leq i < s/2$	force $v_{-i} \equiv \bar{v}_{i+1}$.

Retract the constraints at depth

$$(l * W(2, l-1))/(2(l-1))$$

Continuing...

Add the unsafe constraints:

Function Seg	Range	Meaning
$(v_{-i} \vee v_{i+1}) \wedge (\bar{v}_{-i} \vee \bar{v}_{i+1})$	$0 \leq i < s/2$	force $v_{-i} \equiv \bar{v}_{i+1}$.

Retract the constraints at depth

$$(l * W(2, l-1))/(2(l-1))$$

Continue without unsafe constraints until the end

Summary

State Machines

- Support function-complete look-ahead
- Efficiently support complex heuristics
- Efficiently admit special forms, e.g. cardinality constraints
- Efficiently admit backjumping, lemmas, restarts, etc.

Preprocessing

- Restrict, Existential Quantification, Strengthening help

Safe Assignments

UnSafe Assignments