

CSC2512
Advanced Propositional
Reasoning

CSC2512: MaxSAT

- So far we have addressed decision problems.
- Frequently applications involve optimization.
- **MaxSAT**: generalization of SAT for dealing with optimization.

MaxSat

- MaxSat is a formalism for expressing Boolean **optimization** problems expressed in CNF.
 - Hard clauses: must be satisfied
 - Soft clauses each with a weight, falsifying these incurs a cost equal to their weight.
- MaxSat: find a truth assignment that **satisfies all of the hard clauses** while falsifying **a minimum weight** of soft clauses. (Equivalently satisfying a maximum weight of soft clauses).
- MaxSat solvers are effective in a growing range of applications.

MaxSat Applications

probabilistic inference
design debugging

[Park, 2002]

[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
[Chen, Safarpour, Marques-Silva, and Veneris, 2010]

maximum quartet consistency
software package management

[Morgado and Marques-Silva, 2010]

[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
[Ignatiev, Janota, and Marques-Silva, 2014]

Max-Clique
2015]

[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu,

fault localization

[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]

restoring CSP consistency

[Lynce and Marques-Silva, 2011]

reasoning over bionetworks

[Guerra and Lynce, 2012]

MCS enumeration

[Morgado, Liffiton, and Marques-Silva, 2012]

heuristics for cost-optimal planning

[Zhang and Bacchus, 2012]

optimal covering arrays

[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]

correlation clustering

[Berg and Jarvisalo, 2013; Berg and Jarvisalo, 2016]

treewidth computation

[Berg and Jarvisalo, 2014]

Bayesian network structure learning

[Berg, Jarvisalo, and Malone, 2014]

causal discovery

[Hyttinen, Eberhardt, and Jarvisalo, 2014]

visualization

[Bunte, Jarvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]

model-based diagnosis

[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]

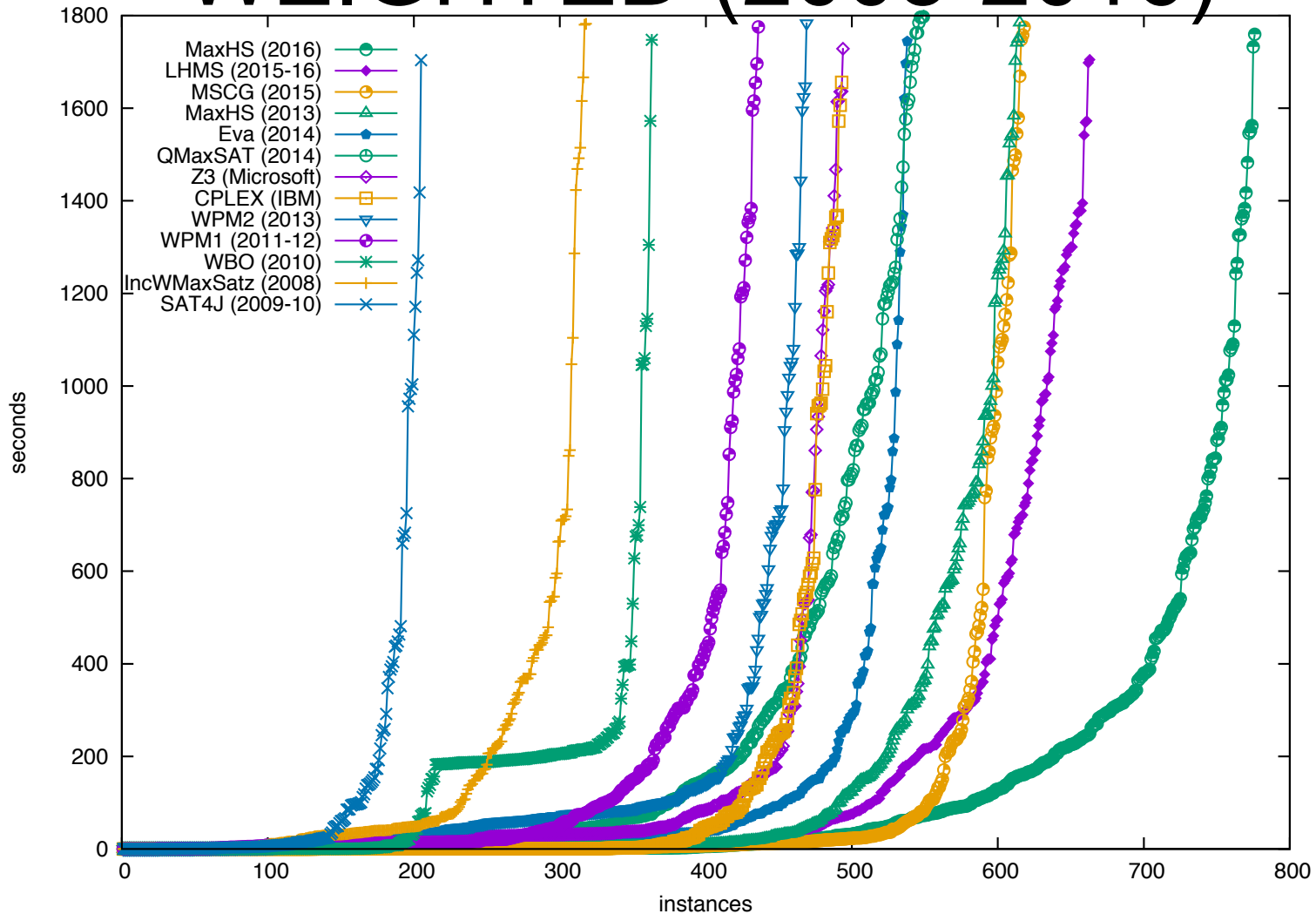
cutting planes for IPs

[Saikko, Malone, and Jarvisalo, 2015]

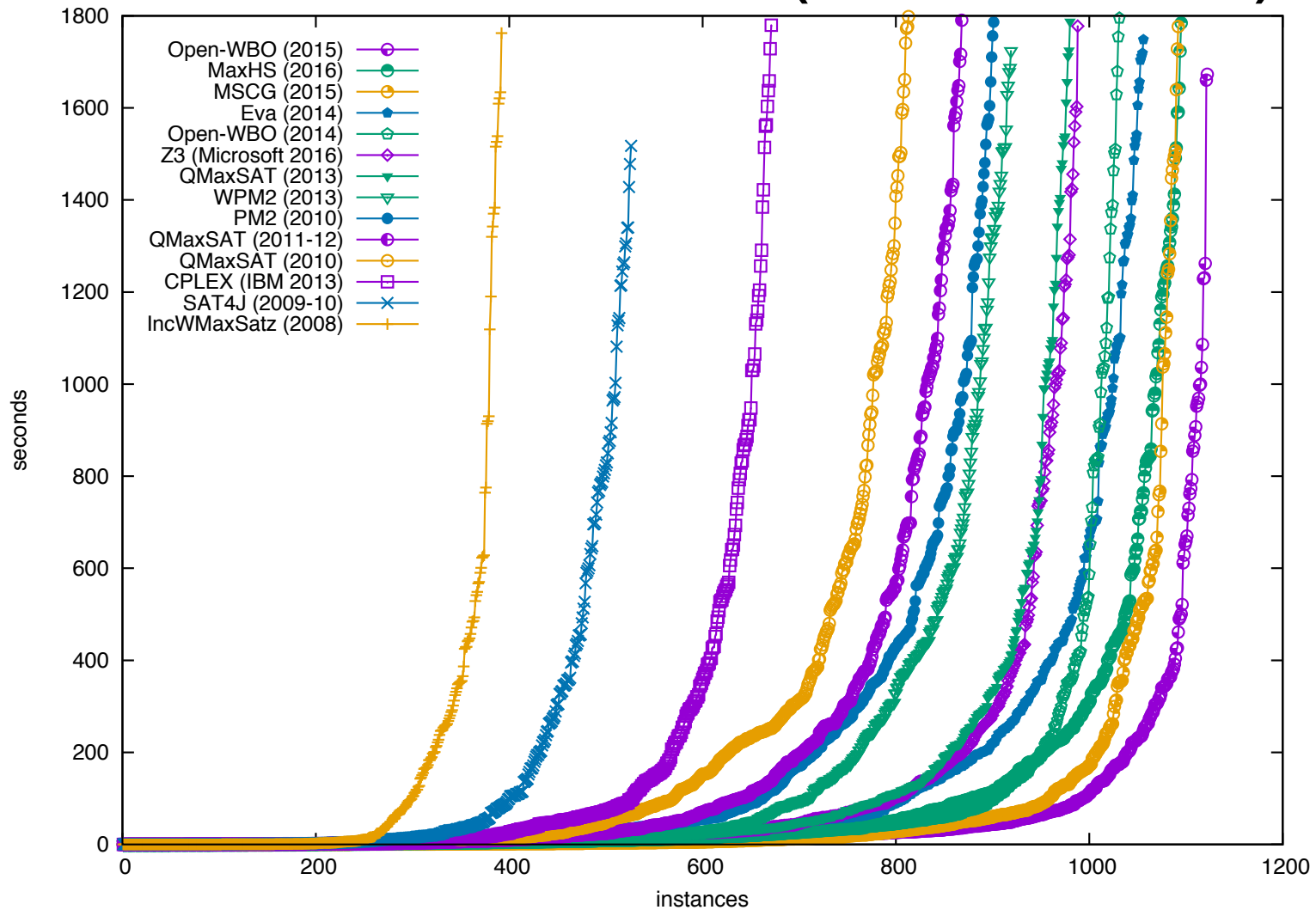
argumentation dynamics

[Wallner, Niskanen, and Jarvisalo, 2016]

Improvements in MaxSat Solving WEIGHTED (2008-2016)



Improvements in MaxSat Solving UNWEIGHTED (2008-2016)



Problem Sizes

- Largest problems solved in 2017 MaxSat Evaluation, >6,000,000 variables and > 13,000,000 clauses (solved by MaxHS in < 800 sec.)
- MaxSat is considerably harder than SAT, SAT solvers can solve bigger problems

CSC2512: MaxSAT

MaxSAT

- A set of clauses each with an associated rational valued or infinite weight greater than zero:

$$\{C_1:w_1, C_2:w_2, \dots, C_m:w_m\} \text{ with } w_i > 0$$

CSC2512: SoftCSPs and MaxSAT

MaxSAT

- A cost of a truth assignment π is the sum of the weights of the clauses it falsifies.
- If any of the weights are ∞ then these clauses must be satisfied. Such clauses are called **hard clauses**.
- A **feasible solution** is a truth assignment satisfying all of the hard clauses (Equivalently it is a truth assignment with finite cost).
- An **Optimal solution** is a feasible solution of minimum cost.

CSC2512: SoftCSPs and MaxSAT

MaxSAT

- If all weights are 1, this is equivalent to finding a truth assignment that satisfies a maximum number of clauses.
- With non-unit weights, equivalent to finding a truth assignment satisfying a maximum weight of clauses.
- If w_i was zero we could discard the clause. If w_i was negative then we would be trying to solve a different type of optimization (we would want to falsify the clause)...we could however reformulate the problem to make it into a maxsat problem.

CSC2512: SoftCSPs and MaxSAT

MaxSAT

- In the literature the following classes are sometimes distinguished:
 1. Maxsat all weights are 1 (theoretical studies often restricted to this case)
 2. Partial Maxsat has some weights of ∞ (i.e., some hard clauses)
- When weights are 1, we can see that MaxSat is solvable by a sequence of SAT calls: Can the formula be satisfied by falsifying k clauses for $k=1, k=2, k=3 \dots$
- This illustrates that MaxSat with unit weights is in the class FP^{NP} which are the functions that can be computed by a polytime function that has access to a NP oracle (each oracle call is charged as a single operation)
- MaxSat is also hard to approximate being in the class APX-Complete.

CSC2512: SoftCSPs and MaxSAT

MaxSAT

- When the weights are finite precision rationals MaxSat remains in the class FP^{NP} as long as there is a fixed bound on the precision.

CSC2512: MAXSAT

Resolution.

- Resolution in SAT preserves the set of models of the formula:
 - If Φ is a set of clauses and $\{(A,x), (B,-x)\} \subset \Phi$
 - Then for any truth assignment π such that $\pi \models \Phi$, we have that $\pi \models \Phi \cup \{(A,B)\}$

MaxSAT requires that models retain the same cost.

- An inference rule that transforms Φ to Φ' (e.g., by adding new clauses to Φ) is sound for MaxSAT iff for all truth assignments π , the sum of the costs of the clauses of Φ falsified by π is equal to the sum of the costs of the clauses of Φ' falsified by π .

CSC2512: MAXSAT

Resolution is not sound for MaxSAT.

- Consider simple case where the weight of each clauses is 1.
- $\{(y,x), (z,-x)\}$
- $\pi(x) = T, \pi(y) = F, \pi(z) = F$: $\text{Cost}(\pi) = 1$
- $\{(y,x), (z,-x), (y,z)\}$ (adding the resolvent)
- $\text{Cost}(\pi) = 2$

MaxSat Resolution is a version of resolution that is sound for MaxSAT. Ordinary resolution adds a new clause to the formula. MaxSat resolution makes a more complex transformation of the formula in order to preserve costs.

CSC2512: MAXSAT

MaxSat Resolution Rule: Simple case each clause has weight 1

$\text{MaxR}((x, a_1, \dots, a_s), (-x, b_1, \dots, b_t)) =$

- $(a_1, \dots, a_s, b_1, \dots, b_t)$
- $(x, a_1, \dots, a_s, -b_1)$
- $(x, a_1, \dots, a_s, b_1, -b_2)$
- ...
- $(x, a_1, \dots, a_s, b_1, \dots, b_{t-1}, -b_t)$
- $(-x, b_1, \dots, b_t, -a_1)$
- $(-x, b_1, \dots, b_t, a_1, -a_2)$
- ...
- $(-x, b_1, \dots, b_t, a_1, \dots, a_{s-1}, -a_s)$

These are called
Compensation
Clauses

CSC2512: MAXSAT

1. We **remove** the input clauses and replace them with the conclusion clauses.
2. Any tautologies are discarded
3. Any repeated literals are collapsed into one.

CSC2512: MAXSAT

Soundness. π falsifies 1 clause of $\{(x, a_1, \dots, a_s), (-x, b_1, \dots, b_t)\}$

(a) π falsifies $(-x, b_1, \dots, b_t)$ and (a_1, \dots, a_s)

- $(a_1, \dots, a_s, b_1, \dots, b_t)$ X
- $(x, a_1, \dots, a_s, -b_1)$ ✓
- $(x, a_1, \dots, a_s, b_1, -b_2)$ ✓
- ... ✓
- $(x, a_1, \dots, a_s, b_1, \dots, b_{t-1}, -b_t)$ ✓
- $(-x, b_1, \dots, b_t, -a_1)$ ✓
- $(-x, b_1, \dots, b_t, a_1, -a_2)$ ✓
- ... ✓
- $(-x, b_1, \dots, b_t, a_1, \dots, a_{s-1}, -a_s)$ ✓

CSC2512: MAXSAT

Soundness. $\{(x, a_1, \dots, a_{i-1}, a_i, \dots, a_s), (-x, b_1, \dots, b_t)\}$ π falsifies 1 clause

(b) π falsifies $(-x, b_1, \dots, b_t)$ but satisfies $(a_1, \dots, a_{i-1}, a_i, \dots, a_s)$. Let a_i be the **first** literal s.t. $\pi(a_i) = T$

- $(a_1, \dots, a_{i-1}, a_i, \dots, a_s, b_1, \dots, b_t)$ ✓
- $(x, a_1, \dots, a_s, -b_1)$ ✓
- $(x, a_1, \dots, a_s, b_1, -b_2)$ ✓
- ... ✓
- $(x, a_1, \dots, a_s, b_1, \dots, b_{t-1}, -b_t)$ ✓
- $(-x, b_1, \dots, b_t, -a_1)$ ✓
- $(-x, b_1, \dots, b_t, a_1, -a_2) \dots$ ✓
- $(-x, b_1, \dots, b_t, a_1, \dots, a_{i-1}, -a_i)$ ✗
- $(-x, b_1, \dots, b_t, a_1, \dots, a_{i-1}, a_i, -a_{i+1})$ ✓
- $\dots (-x, b_1, \dots, b_t, a_1, \dots, a_{i-1}, a_i, \dots, -a_s)$ ✓

CSC2512: MAXSAT

Soundness. (c) If π falsifies (x, a_1, \dots, a_s) a symmetric argument holds.

CSC2512: MAXSAT

Soundness. (d) If π satisfies both $(-x, b_1, \dots, b_t)$ and (x, a_1, \dots, a_s) , say $\pi(x) = \text{True}$, then some b_j is true.

$\{(x, a_1, \dots, a_s), (-x, b_1, \dots, b_j, \dots, b_t)\}$ π satisfies all clauses

- $(a_1, \dots, a_s, b_1, \dots, b_j, \dots, b_t)$ ✓
- $(x, a_1, \dots, a_s, -b_1)$ ✓
- $(x, a_1, \dots, a_s, b_1, -b_2)$ ✓
- ... ✓
- $(x, a_1, \dots, a_s, b_1, \dots, b_{t-1}, -b_t)$ ✓
- $(-x, b_1, \dots, b_j, \dots, b_t, -a_1)$ ✓
- $(-x, b_1, \dots, b_j, \dots, b_t, a_1, -a_2)$ ✓
- ... ✓
- $(-x, b_1, \dots, b_j, \dots, b_t, a_1, \dots, -a_s)$ ✓

CSC2512: MAXSAT

Completeness.

- It is more difficult to prove that this rule is complete. But this was done in “Resolution for Max-Sat” by Maria L. Bonet, Jordi Levi, and Felip Manya (Artificial Intelligence 171 (2007) pp. 606-618

Saturation--DP style Algorithm

A set of clauses C is said to be **saturated** w.r.t. a variable x if for every pair of clauses

$C_1 = (x, A)$ and $C_2 = (-x, B)$

there is a literal ℓ such that $\ell \in A$ and $-\ell \in B$

A set of clauses C' is a **saturation** of C w.r.t. x if C' is obtained from C by applying MaxR a finite number of times resolving on x and C' is saturated w.r.t. x .

CSC2512: MAXSAT

Saturation: A set of clauses C is saturated w.r.t. x if and only if every possible application of MaxR resolving on x only introduces compensation clauses (i.e., the resolvent (A,B) is a tautology).

It can be shown that from C we can obtain a saturation of C w.r.t. x by applying MaxR resolving on x until we obtain saturation. That is, this process must terminate.

- Saturation is not unique—different saturations can be obtained depending on the sequencing of our MaxR resolutions
- If we saturate w.r.t. x and then saturate w.r.t. y , the result might no longer be saturated w.r.t. x . In general, we might not be able to saturate w.r.t. two variables at once.

CSC2512: MAXSAT

- If C is saturated w.r.t. x , and C' is the subset of C consisting of those clauses not containing x . Then any assignment π satisfying C' and not assigning x can be extended to an assignment satisfying C .
 - This is a DP like property—once we saturate w.r.t x we can extend any solution to be over x .

CSC2512: MAXSAT

Complete procedure for solving MaxSat with MaxR:

1. Order the variables x_1, \dots, x_n
2. Construct two sequences of sets of clauses C_0, \dots, C_n and D_1, \dots, D_n .
 1. C_0 is the original set of clauses of the MaxSat theory
 2. for $i = 1, \dots, n$, $C_i \cup D_i$ is a saturation of C_{i-1} w.r.t. x_i where C_i is a set of clauses not containing x_1, \dots, x_i , and D_i is a set of clauses containing x_i .

In other words. Saturate the original formula w.r.t. x_1 , then put aside the clauses with x_1 in D_1 , saturate the other clauses C_1 w.r.t. x_2 , put aside the clauses with x_2 in D_2 , saturate the other clauses C_2 w.r.t., $x_3 \dots$

CSC2512: MAXSAT

Complete procedure for solving MaxSat with MaxR:

1. Order the variables x_1, \dots, x_n
2. Construct two sequences of sets of clauses C_0, \dots, C_n and D_1, \dots, D_n .
 1. C_0 is the original set of clauses of the MaxSat theory
 2. for $i = 1, \dots, n$, $C_i \cup D_i$ is a saturation of C_{i-1} w.r.t. x_i where C_i is a set of clauses not containing x_1, \dots, x_i , and D_i is a set of clauses containing x_i .

Notice that C_n does not contain any variables. So it is either empty (indicating that the original formula is SAT or it contains multiple copies of the empty clause. The number of copies is equal to the **minimum number** of clauses of the original theory that must be falsified:

$$C \vdash \emptyset_1, \dots, \emptyset_m, D_1, \dots, D_n$$

CSC2512: MAXSAT

The number of copies is equal to the minimum number of clauses of the original theory that must be falsified:

$$C \vdash \emptyset_1, \dots, \emptyset_m, D_1, \dots, D_n$$

So after these rounds of saturation the answer is given by counting the number of empty clauses remaining (the D_i are satisfiable).

CSC2512: MAXSAT

Used in this way MaxR is mostly of theoretical interest, and there are many open theoretical questions:

1. It is not simple to convert an ordinary resolution proof (in which a clause might be used multiple times) into a MaxR proof (where every inference step removes the clauses being resolved on).
2. Not much is known about the minimum sizes of MaxR proofs: for the saturation procedure we have that $n*m*2^n$ (where m is the number of clauses) is an upper bound on the number of MaxR steps required.
3. Are there other non-ordered MaxR proof procedures? Can these be smaller?

CSC2512: MAXSAT

Weighted MaxR: MaxR as defined only works with unweighted (weight 1) clauses but it can be extended to the weighted case

CSC2512: MAXSAT

$\text{wtMaxR}((x, a_1, \dots, a_s : w_1), (-x, b_1, \dots, b_t : w_2)) =$

- $(a_1, \dots, a_s, b_1, \dots, b_t : \min(w_1, w_2))$
- $(x, a_1, \dots, a_s : w_1 - \min(w_1, w_2))$
- $(-x, b_1, \dots, b_t : w_2 - \min(w_1, w_2))$
- $(x, a_1, \dots, a_s, -b_1 : \min(w_1, w_2))$
- $(x, a_1, \dots, a_s, b_1, -b_2 : \min(w_1, w_2))$
- ...
- $(x, a_1, \dots, a_s, b_1, \dots, b_{t-1}, -b_t : \min(w_1, w_2))$
- $(-x, b_1, \dots, b_t, -a_1 : \min(w_1, w_2))$
- $(-x, b_1, \dots, b_t, a_1, -a_2 : \min(w_1, w_2))$
- ...
- $(-x, b_1, \dots, b_t, a_1, \dots, a_{s-1}, -a_s : \min(w_1, w_2))$

CSC2512: MAXSAT

1. The input clauses are retained, but their weights reduced.
2. Any tautologies are removed
3. Any repeated literals are collapsed into one.
4. Clauses of zero weight are removed—one of the input clauses is always removed.

Contraction: replace (A, w_1) and (A, w_2) with (A, w_1+w_2)

CSC2512: MAXSAT

1. The weighted max sat resolution rule can be shown to be sound (easy) and complete. That is, from a weighted MaxSat instance C we can derive via wtMaxR:

$$C \vdash (\emptyset : w), D$$

where D is satisfiable, and w is the weight of the optimal solution to C .

CSC2512: MAXSAT

Using MaxR in practice:

Certain special cases of (A, x) , $(B, -x)$ yield simple sets of clauses under MaxR.

MaxR is used in Branch and Bound search exploiting various special cases:

- In the reduced theory at the current node of the search space look for special cases (often detectable via unit propagation), apply MaxR to obtain an increase in the lower bound (i.e., derive an empty clause with certain weight).
 - That is MaxR is used as a lower bounding technique.

CSC2512: Branch and Bound

- B&B relies on two bounds
- UB: $\text{mincost}(P) \leq \text{UB}$.
- LB: A lower bound **function**. This function must be able to supply a lower bound on the cost that must be incurred below each node that will be encountered during search

CSC2512: Branch and Bound

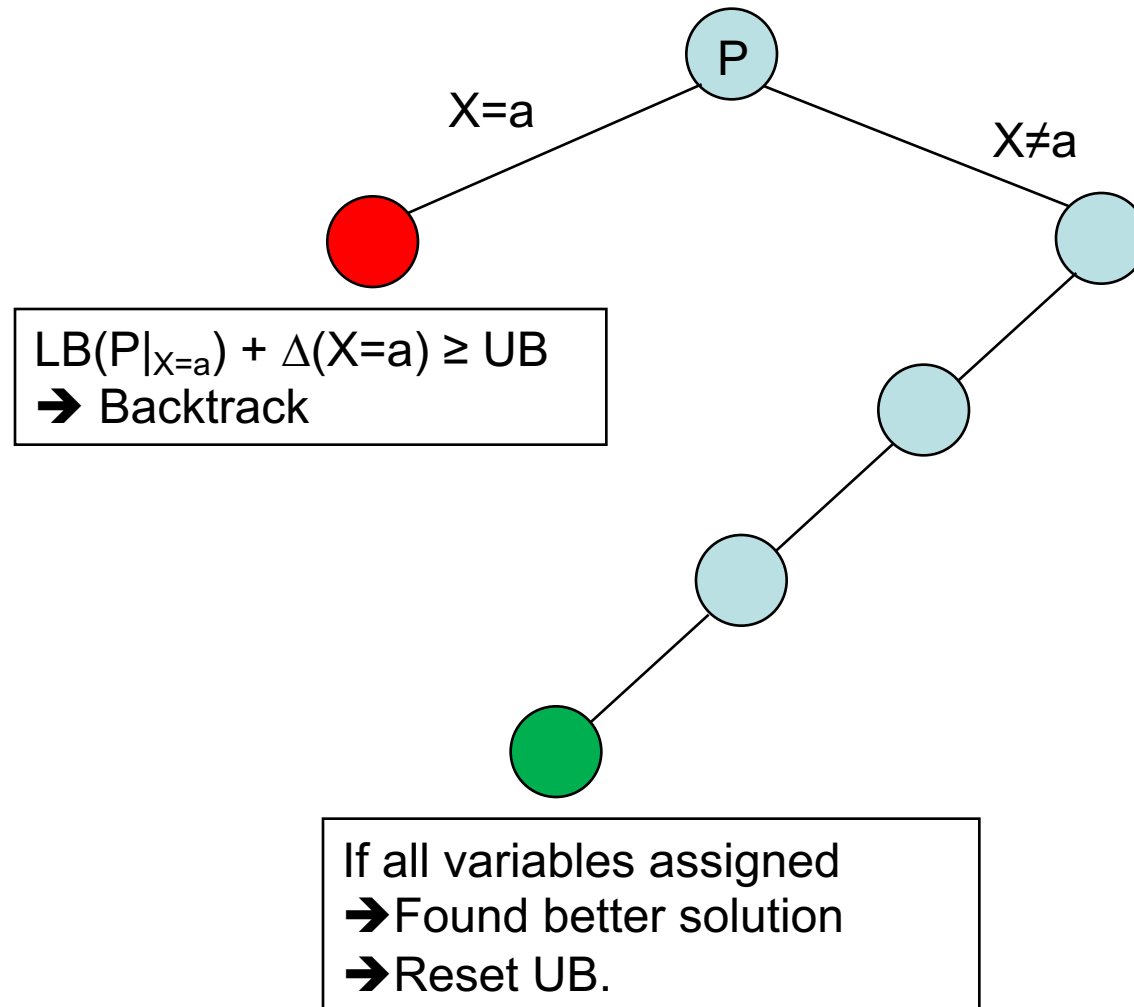
UB

- Initially we can set UB by finding any low cost solution, e.g., via local search or a greedy solution.
- If there are hard constraints we need to find a feasible solution, any feasible solution will provide an upper bound. If feasible solution to the hard constraints then all complete assignments have infinite cost.

LB

- Various techniques are used for computing LB including
- Linear programming relaxations.
- **MaxR resolutions**

CSC2512: Branch and Bound



CSC2512: MAXSAT

MiniMaxSat

- A Branch and Bound MaxSat Solver. At each node it selects a variable to branch on, reduces the input CNF by that assignment, then tries to compute a lower bound for the cost of that reduced CNF.
- If this lower bound exceeds the current upper bound we backtrack.

CSC2512: MAXSAT

MiniMaxSat

- Computes the lower bound using MaxR and unit propagation.
- In the reduced theory we perform Unit Propagation treating all soft clauses as hard allowing the solver to detect special cases where MaxR can yield a new weighted empty clause. That weight can be added to the LB (**at this node**)
- All inferences must be undone on backtrack, as the MaxR steps are wrt to the node's formula not wrt to the global formula.
- B&B effective on small combinatorially hard problems, e.g., hard instances of maxcut and maxclique. But it does not scale well to larger instances (the B&B tree grows too large).

CSC2512: MAXSAT

Readings:

1. **MiniMaxSat: a New Weighted Max-SAT Solver** Federico Heras, Javier Larrosa, and Albert Oliveras. SAT 2007 (a branch and bound approach using MaxR to compute lower bounds)

CSC2512: MAXSAT

Solving MaxSat as a sequence of Decision Problems.

- Branch and Bound solvers do not work well when we have larger MaxSat theories. However, we can still run SAT solvers on such theories.
- This leads to a method of solving MaxSat by solving a sequence of SAT decision instances.
- Various techniques have been developed to make this process effective.

CSC2512: MAXSAT

Simplest version: (Een & Sorensson, 2006, MiniSat+) UNIT WEIGHTS

1. Input MaxSat CNF Φ
2. Add a **blocking variable** b_i to every soft clause C_i of Φ . That is we replace the clause C_i with the clause $C_i \vee b_i$
3. $k = 0$
4. If $\text{Sat}(\Phi \cup \text{cnf}(\sum b_i \leq k))$ return k .
5. Else $k = k+1$ GOTO 4.

The blocking variables do not appear anywhere else in the formula. If we turn them on---the soft clause is “**blocked**” (i.e., automatically satisfied).

By making b_i true, we satisfy the clause $C_i \vee b_i$ and now the Sat solver can find a solution that does not have to satisfy the original soft clause C_i . The cardinality constraint restricts the Sat solver from falsifying more than k soft clauses

CSC2512: MAXSAT

Simplest version: (Een & Sorensson, 2006, MiniSat+) UNIT WEIGHTS

$$\Phi = \{(x), (y), (-x, -y), (p), (q)\}$$

1. $\Phi = \{(x, b_1), (y, b_2), (-x, -y, b_3), (p, b_4), (q, b_5)\}$
 $b_1 + b_2 + b_3 + b_4 + b_5 = 0$
2. $\Phi = \{(x, b_1), (y, b_2), (-x, -y, b_3), (p, b_4), (q, b_5)\}$
 $b_1 + b_2 + b_3 + b_4 + b_5 = 1$
→ SAT, e.g., $b_1 = 1, x = 0, y = 1, p = 1, q = 1.$

The theory becomes hard to solve as the sum of the b variables becomes larger. $\sum b_i = k$ has n choose k different solutions which grows exponentially with k .

CSC2512: MAXSAT

More refined versions exploit CORES.

First, it is useful to divide the MaxSat formula F into $H \cup S$ where H is the set of hard clauses (those with infinite weight) and S is the set of soft clauses.

Cores

- A set of soft clauses $\kappa \subseteq \mathbf{S}$ is a **core** of \mathbf{F} if

$\kappa \cup \text{hard}(\mathbf{F})$ is **UNSAT**

- Note that in SAT a core is a subset of clauses that are UNSAT. In MaxSat we always have to satisfy the hard clauses, so more useful to define cores relative to the hard clauses.

Cores via Assumptions

- The current best performing algorithms for MaxSat need to extract cores of the formula
- Currently most accessible way to do this is to use SAT solving with assumptions...built into most SAT solvers.
- Assumptions must be a set of literals.
- **SAT_ASSUME(H, A_{sm}p)**
 - Sat solve the CNF H under the assumption that every literal in A_{sm}p is true.
 - Return SAT and π if $\pi \models H$ and makes every literal in A_{sm}p true
 - Return UNSAT and a conflict clause $(\neg l_1, \neg l_2, \dots, \neg l_k)$ implied by H where each $l_i \in A_{sm}p$.
 - At least one of the subset of assumptions $\{l_1, l_2, \dots, l_k\}$ must be falsified in every model of H.

Cores via Assumptions

- To extract MaxSat cores we use as assumptions the negation of the blocking variables—if these are assumed to be true then their corresponding soft clause must be satisfied.
- So if we have the clauses $C_1 \vee b_1, C_2 \vee b_2, \dots, C_m \vee b_m$ with the blocking variables added to the soft clauses C_i
- Then if we SAT solve $\text{SAT_ASSUME}(H, \{\neg b_1, \neg b_2, \dots, \neg b_m\})$
 - If the sat solver returns SAT we obtain a truth assignment π that satisfies all hard clauses H , and makes every literal b_i in the assumptions true. Thus since $H \models \neg b_i \rightarrow c_i$ we have that $\pi \models c_i$. That is, π satisfies all soft clauses.
 - If the sat solver returns UNSAT, we obtain a conflict clause $(b_{k1}, b_{k2}, \dots, b_{kn})$ entailed by H . That is any model of H must make at least one of these b_i false \Leftrightarrow any model must falsify at least one of the corresponding soft clauses. **Thus, the conflict clause specifies a core of F .**

Cores via Assumptions

- Things are easier if we first transform F to an equivalent MaxSat formula with only unit soft clauses $(\neg b_i)$. Now the $\neg b_i$ can directly be used as assumptions. This can be done by taking every soft clause C_i and replacing it with
 - The soft clause $(\neg b_i)$
 - The hard clause $(C_i \vee b_i)$
- This transformation preserves the cost of all models.

Conversion to Unit Softs

The F^b conversion.

$$F = H \cup S \rightarrow F^b = H^b \cup S^b$$

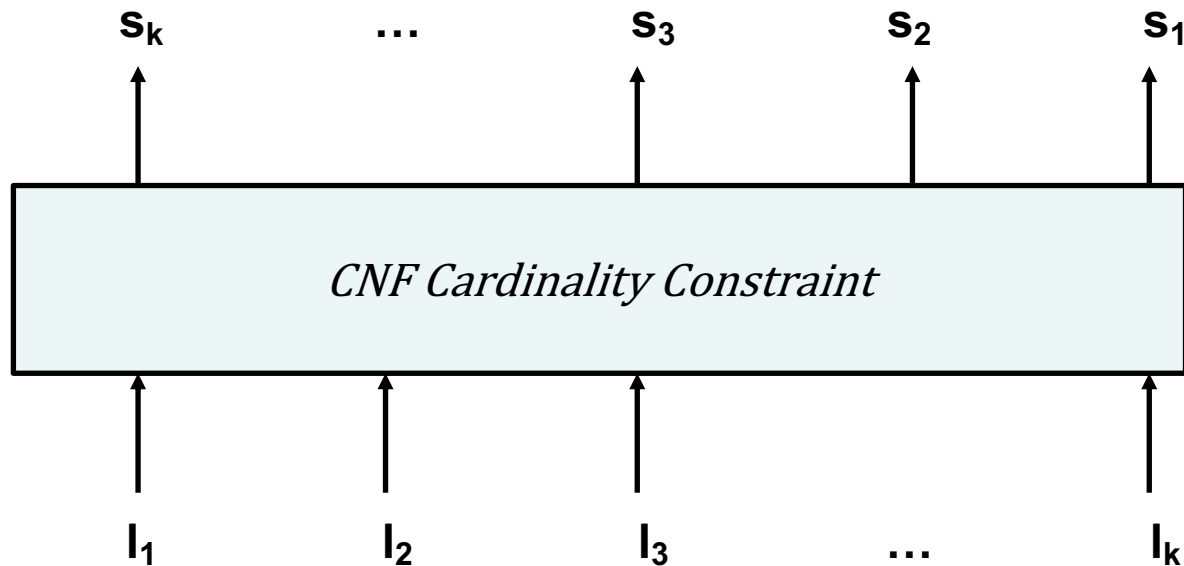
- $H^b = H \cup \{ (C_i \vee b_i) \mid C_i \in S \}$ with new variable b_i
 - Make softs into hard clause with new “blocking variables”
- $S^b = \{ (\neg b_i) \mid C_i \in S \}$
 - Add new literals as unit soft clauses.

Conversion to Unit Softs

- This new "only unit softs" instance is equivalent:
- Every feasible model of F , π , can be extended to a feasible model π^b of F^b with the same cost.
 - For each new variable b^i in F^b set $\pi^b \models b^i$ to FALSE iff $\pi \models c^i$
 - Hence π^b satisfies the soft unit clause $(\neg b^i)$ iff π satisfies the corresponding soft clause c^i .
- Every feasible model of F^b , π^b restricted to the variables of F is a feasible model π of F . $\text{cost}(\pi) \leq \text{cost}(\pi^b)$. $\pi^b \models (c_i \vee b_i)$ so if π^b satisfies $(\neg b^i)$ its restriction π must satisfy c_i .
 - Note $\text{cost}(\pi) \leq \text{cost}(\pi^b)$ since we can have feasible models π^b that have more cost than they need to in F^b . E.g., we could have that $\pi^b \models c_i$ and yet $\pi^b \models b_i$. π^b incurs the cost of falsifying $(\neg b_i)$ when it doesn't need to as it already satisfies c_i .
- **However, this is sufficient to show that an optimal model of F^b restricted to the variables of F is an optimal model of F .**

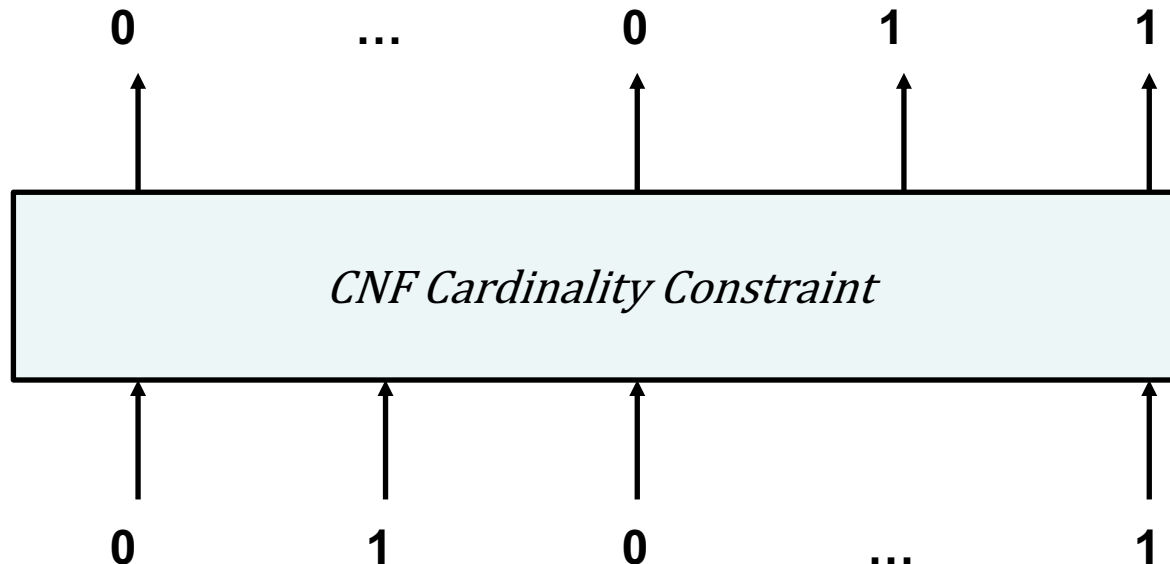
Summation Circuits

- MaxSat solvers using sequences of relaxations exploit cardinality constraints.
- Often these are CNF encodings of summation circuits that output a unary representation of the sum of their inputs.



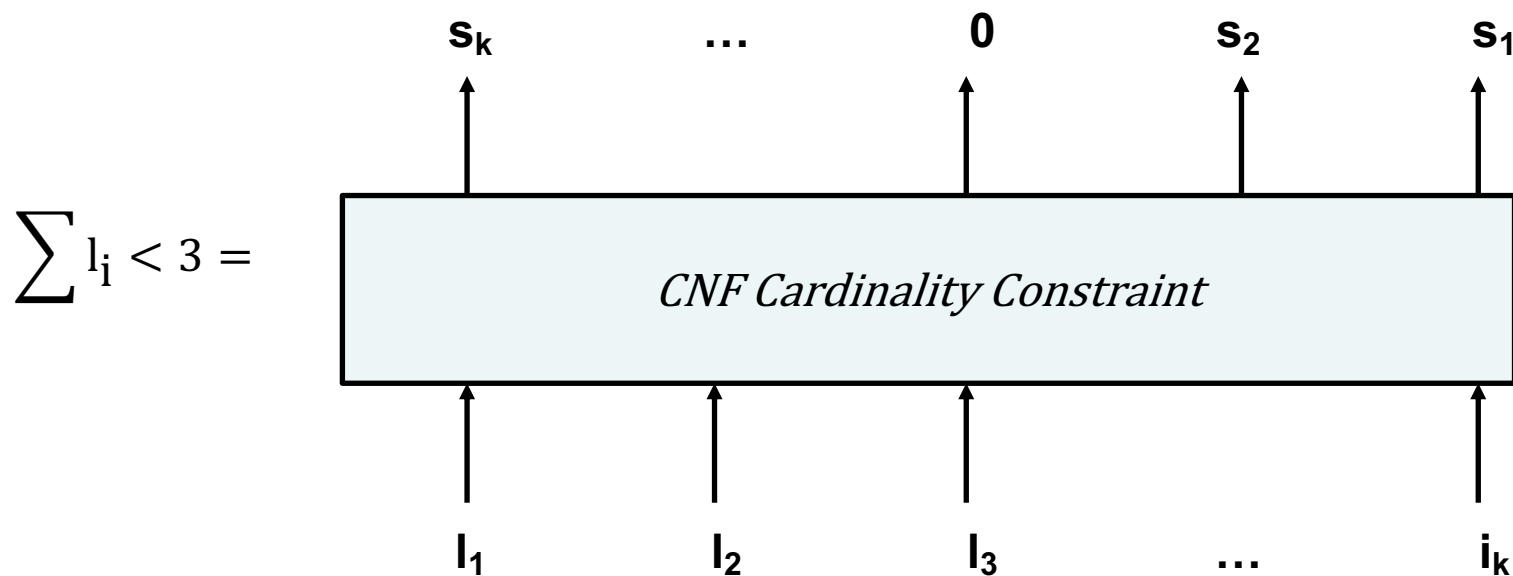
Summation Circuits

- The output is a string of 0's followed by 1's with (as many 1's as there are true inputs).
- The clauses of the summation circuit ensure that
 - $s_i \Leftrightarrow l_1 + l_2 \dots + l_k \geq i$
 - $\neg s_i \Leftrightarrow l_1 + l_2 \dots + l_k < i$



Cardinality Constraints

- So we can impose the constraint $\sum l_i < k$ by adding the clauses of the summation circuit along with the assumption $\neg s_k$



Sat Solving a sequence of relaxations

- A sequence of SAT instances are created (usually incrementally) where each instance is more relaxed so that it allows a larger weight of soft clauses to be falsified.
- The amount of additional weight that can be falsified is restricted so that the first time the formula becomes SAT the resulting satisfying models are optimal solutions to the original MaxSat formula.
 - i.e., there is no lesser relaxation that is satisfiable.
- We can also go the other direction starting with most relaxed and working down until we reach a relaxation that is UNSAT.
 - This works ok, better on some problems but not as well on most problems.

CARDINALITY LAYER

- The needed relaxations are achieved by adding a **CARDINALITY LAYER** to the **hard clauses of F**.
- The **inputs** of the **CARDINALITY LAYER** are the **blocking variables** whose truth counts the number of soft clauses that are relaxed (and thus can be falsified)—limiting how many of these that can be true limits the number of soft clauses that can be falsified.
- The **outputs** of the **CARDINALITY LAYER** can be set by assumptions to restrict how many and which groups of softs that can be falsified.

UNWEIGHTED INSTANCES

- First we assume that all soft clauses have the same weight—so solving MaxSat is the same as minimizing the number of falsified softs.
- Later we will show the technique for lifting to the weighted case (i.e., differing weights).
 - This technique is simple but it seems to lead to some inefficiencies.

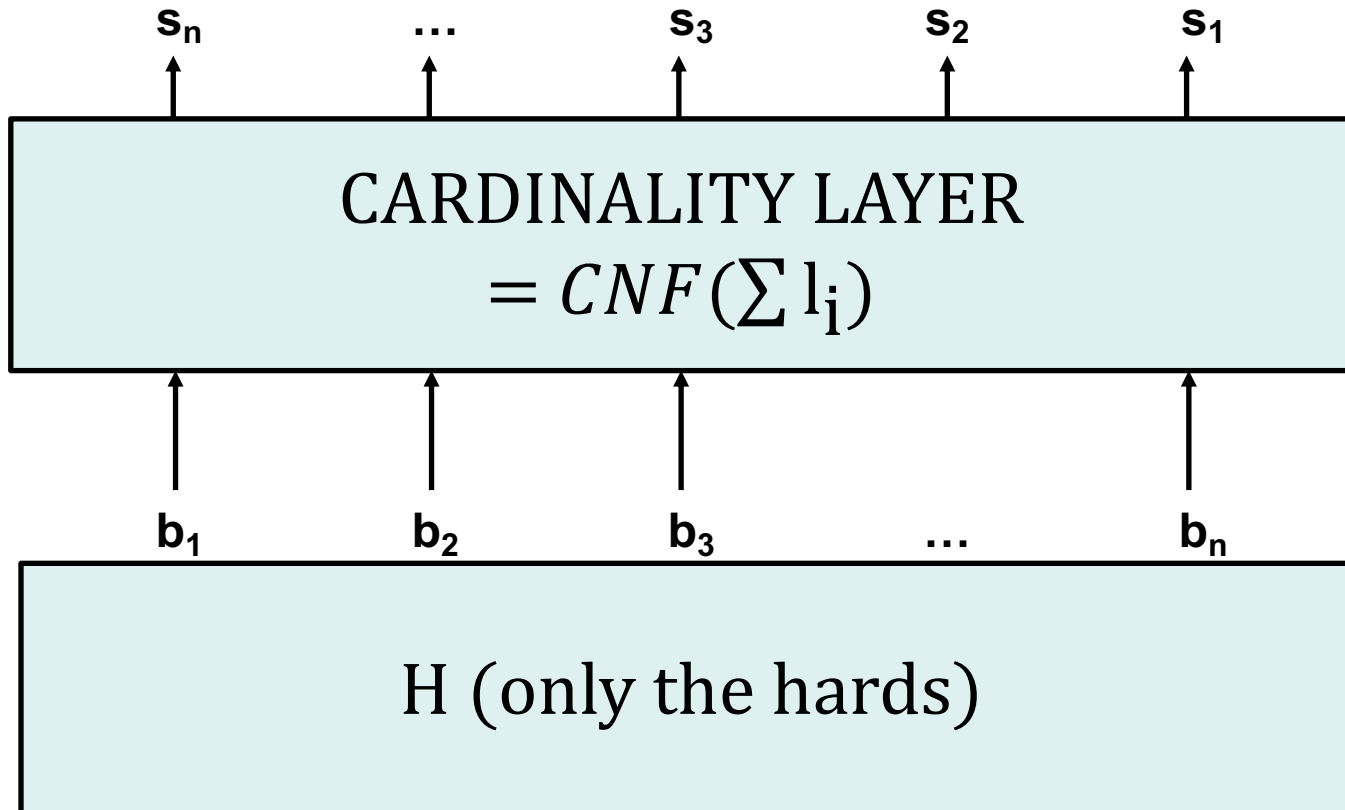
Example

- Standard reduction of MaxSat to a sequence of SAT problems
 - Is the formula satisfiable falsifying 0 softs?
 - Is the formula satisfiable falsifying 1 soft?
 - ...
 - Is the formula satisfiable falsifying k softs?
- Stop as soon as the answer is yes...the truth assignment (excluding the new variables in the cardinality layer) is an optimal solution.

Cardinality Layer Simple Example

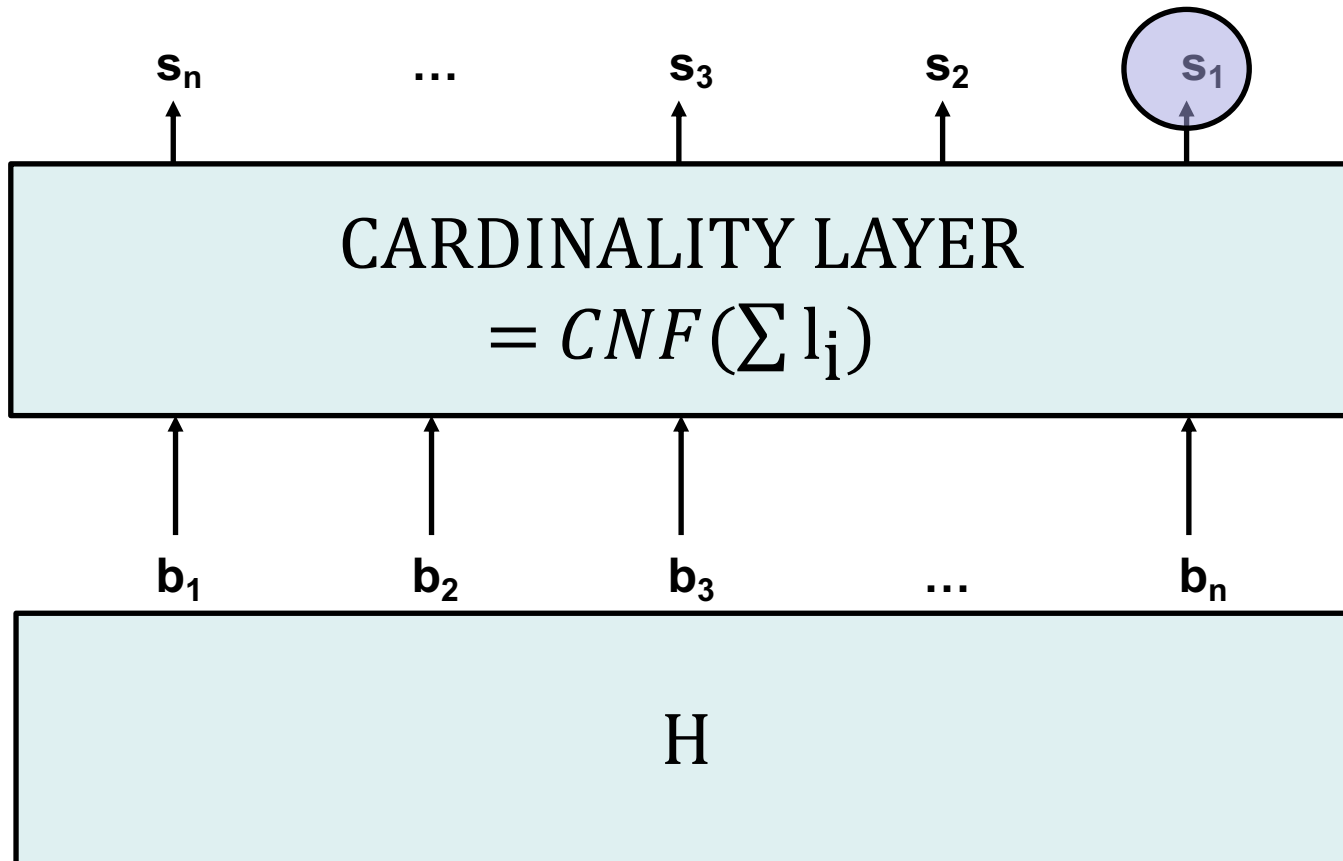
$F = H \cup S$

$S = \{ (\neg b_1), (\neg b_2), \dots, (\neg b_n) \}$



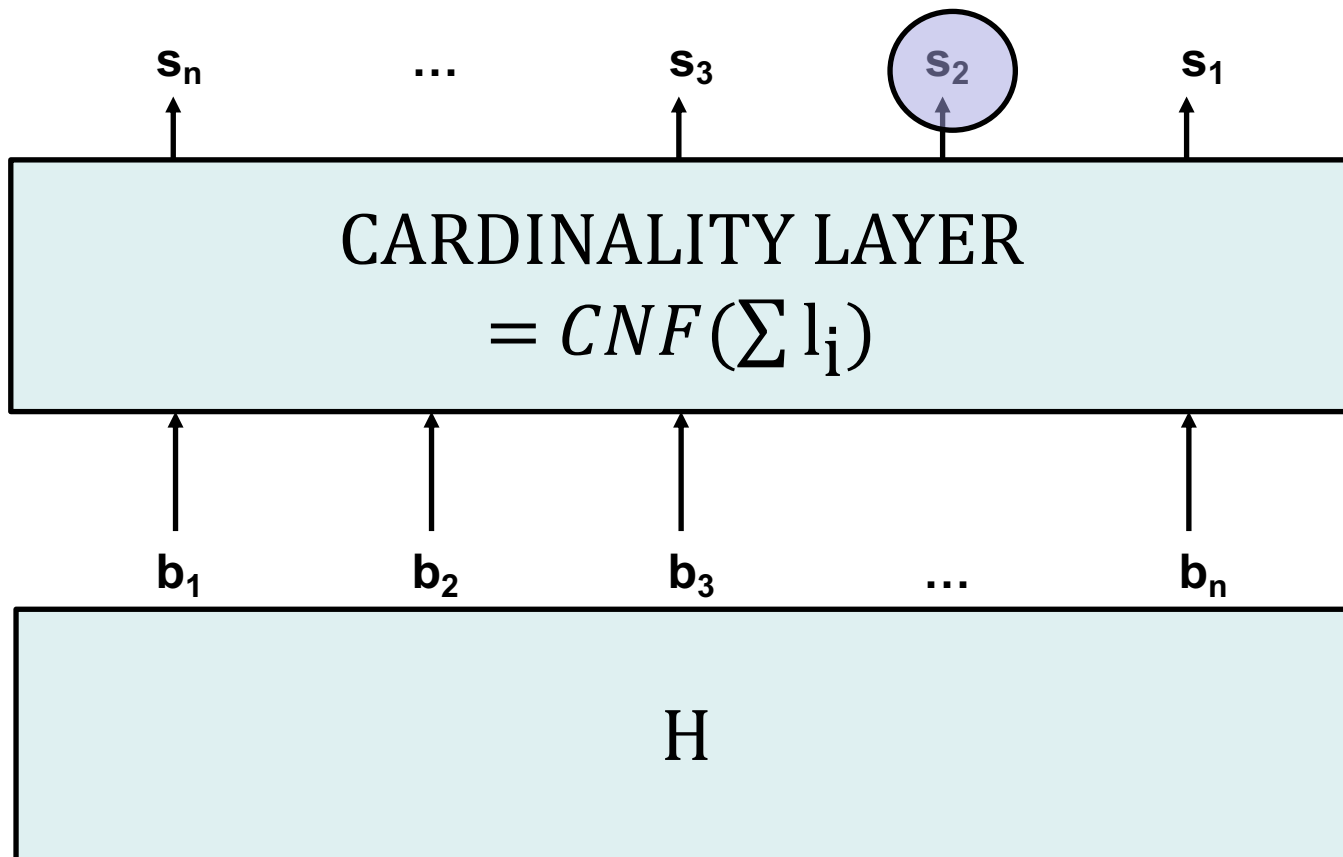
Cardinality Layer Simple Example

On the i -th iteration solve $\text{SAT_ASSUME}(H \cup \text{Card}, \neg s_i)$



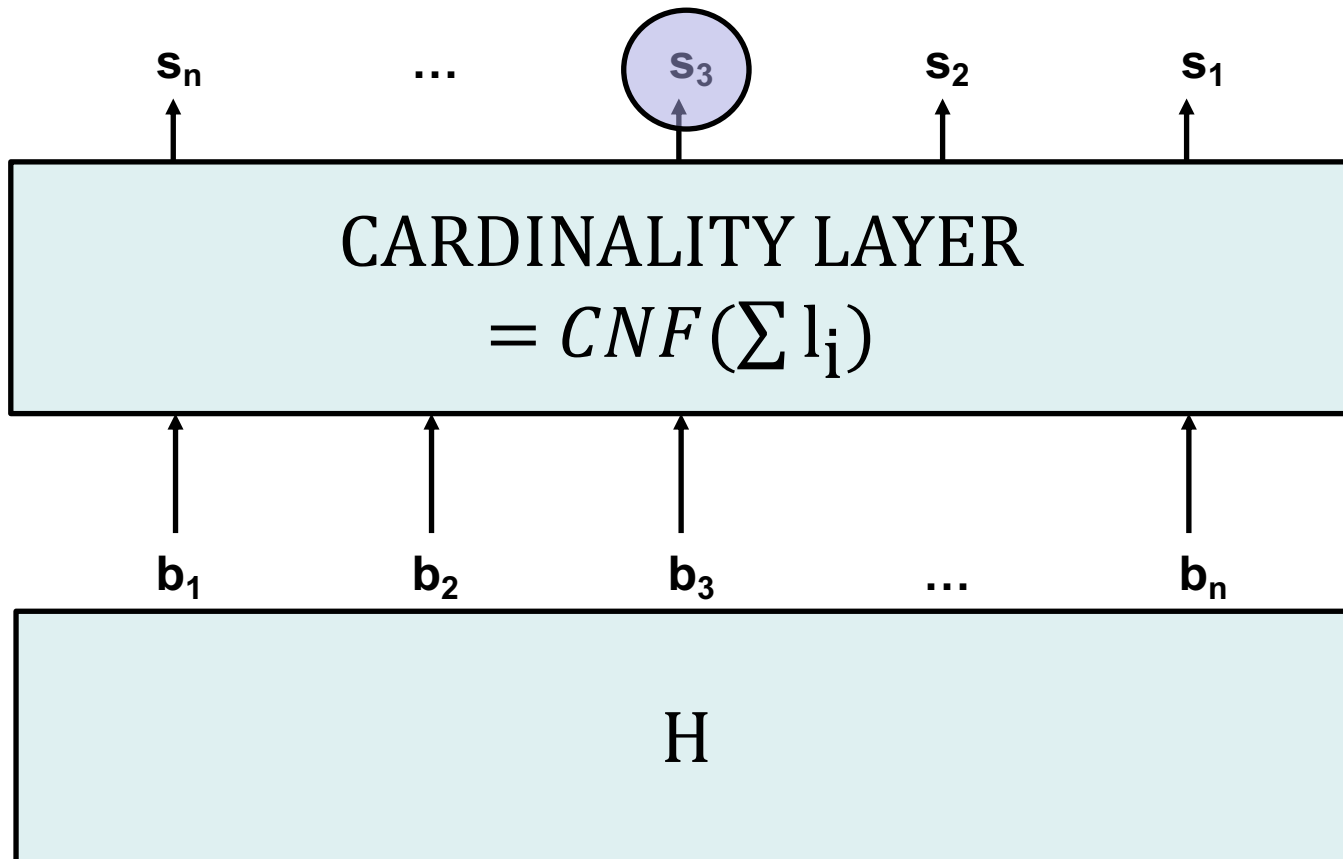
Cardinality Layer Simple Example

On the i -th iteration solve $\text{SAT_ASSUME}(H \cup \text{Card}, \neg s_i)$



Cardinality Layer Simple Example

On the i -th iteration solve $\text{SAT_ASSUME}(H \cup \text{Card}, \neg s_i)$



Cardinality Layer

- In these algorithms the unit soft clauses are not part of the SAT solver's CNF.
 - Their literals serve as inputs to the cardinality layer.
- When we set various various literals in the cardinality layer as assumptions these assumptions restrict the allowed T/F settings of unit soft clauses.

Cardinality Layer

- Clause learning can allow the SAT solver to refute entire subsets of $\text{CARD}(A)$ rather than having to refute each $\rho \in \text{CARD}(A)$ individually.
- The learnt clauses can contain the new variables added to construct the Cardinality Layer. This can support learning more powerful clauses that speed up the refutation.
- Different proposed algorithms construct differently structured Cardinality Layers.

MSU3 using Incremental Cardinality Constraints

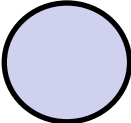
- Used in OpenWBO.

MSU3 Incremental

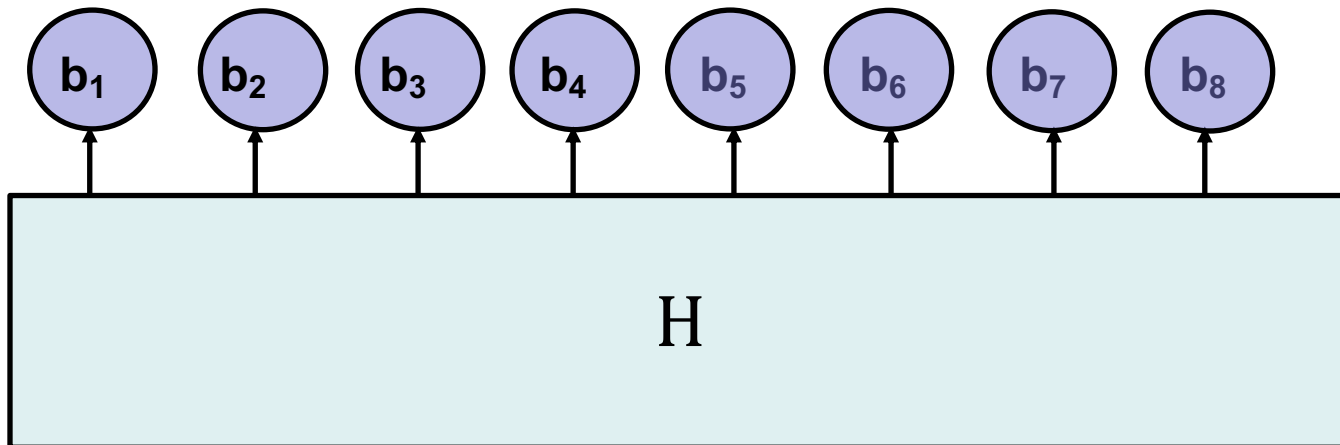
- Consider $F = H \cup S$ where
- $S = \{(\neg b_1), (\neg b_2), \dots, (\neg b_8)\}$
- $H = \sum_{i=1}^8 b_i \geq 4$
- We want to falsify all 8 literals $\neg b_i$ but at least 4 of these literals must be made true—at least 4 soft clauses must be falsified
- Every set of 5 or more soft clauses is a core
- What will MSU3 do on this input formula?

MUS3 Incremental

Start with empty Cardinality Constraint Layer inputs only

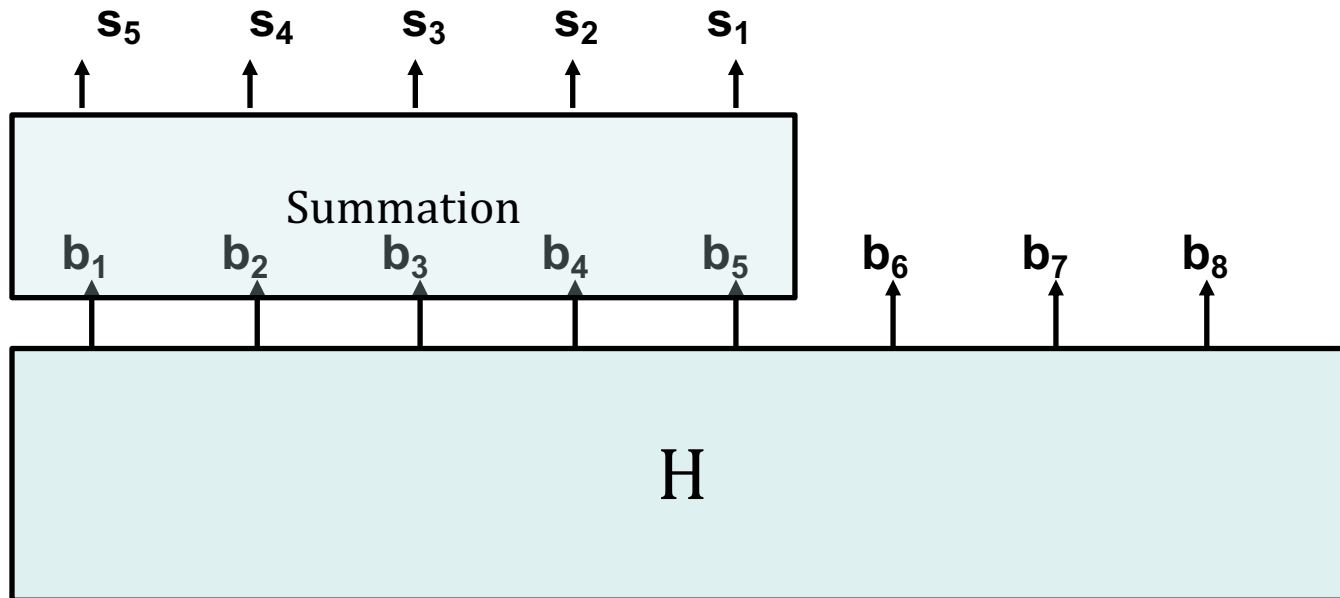
 Literals assumed FALSE

First SAT solve is $\text{SAT_ASSUME}(H, \{\neg b_1, \neg b_2, \dots, \neg b_8\})$,
I.e., try to falsify zero softs.



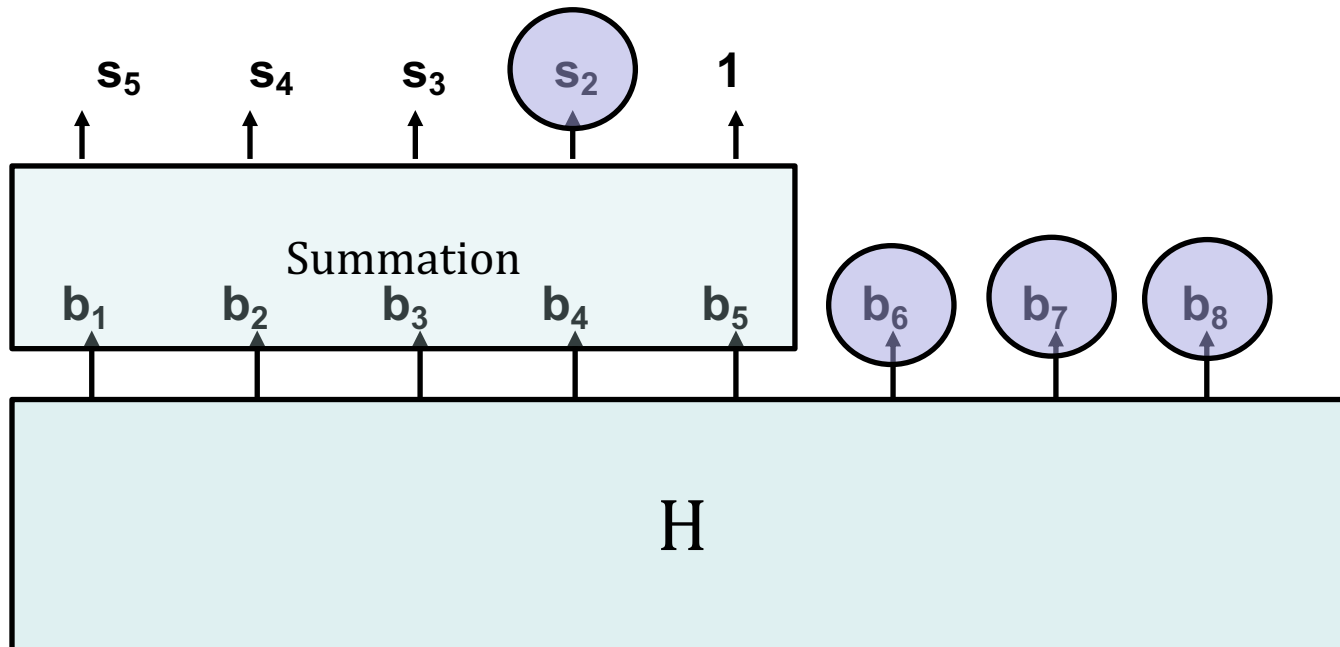
MUS3 Incremental

- UNSAT; say we get the core $(b_1, b_2, b_3, b_4, b_5)$
- Add one to overall cost
- Build summation network over softs in core



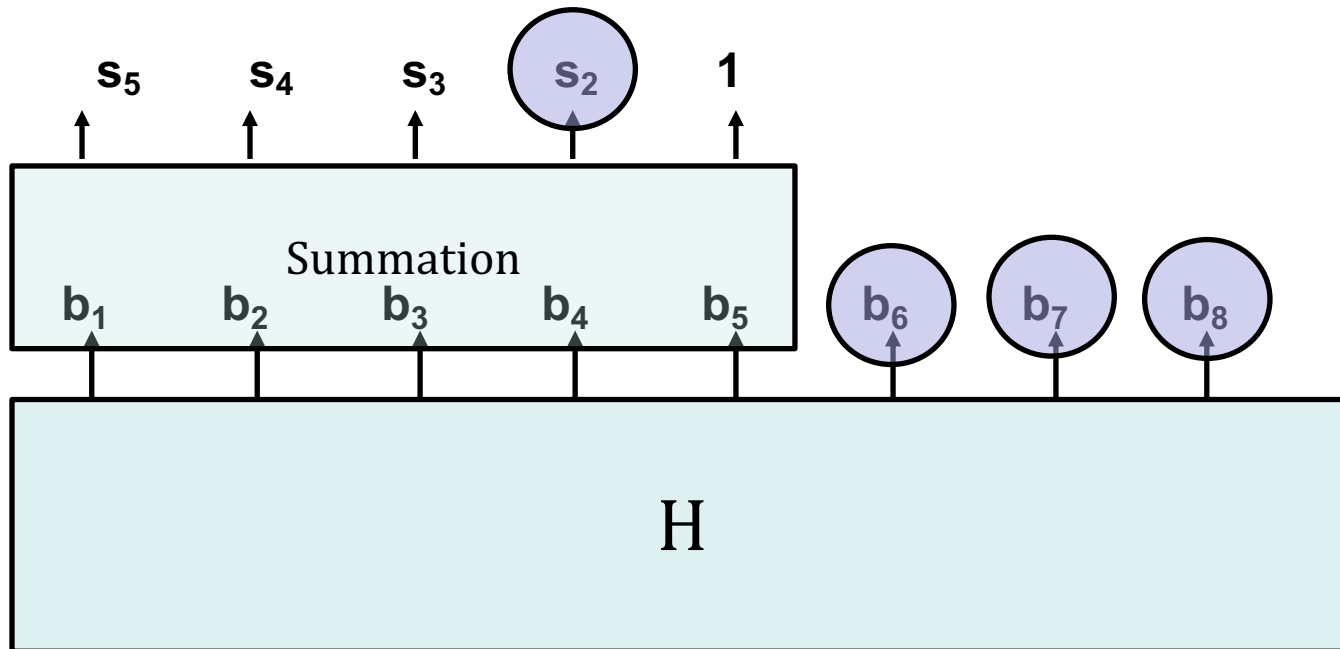
MUS3 Incremental

- We know that the sum must be at least one—set $s_1 = 1$.
- Now **SAT_ASSUME**($H \cup \text{Card}, \{\neg s_2, \neg b_6, \neg b_7, \neg b_8\}$),
- Allow one soft to be falsified among $\neg b_1$ — $\neg b_5$ but no more!



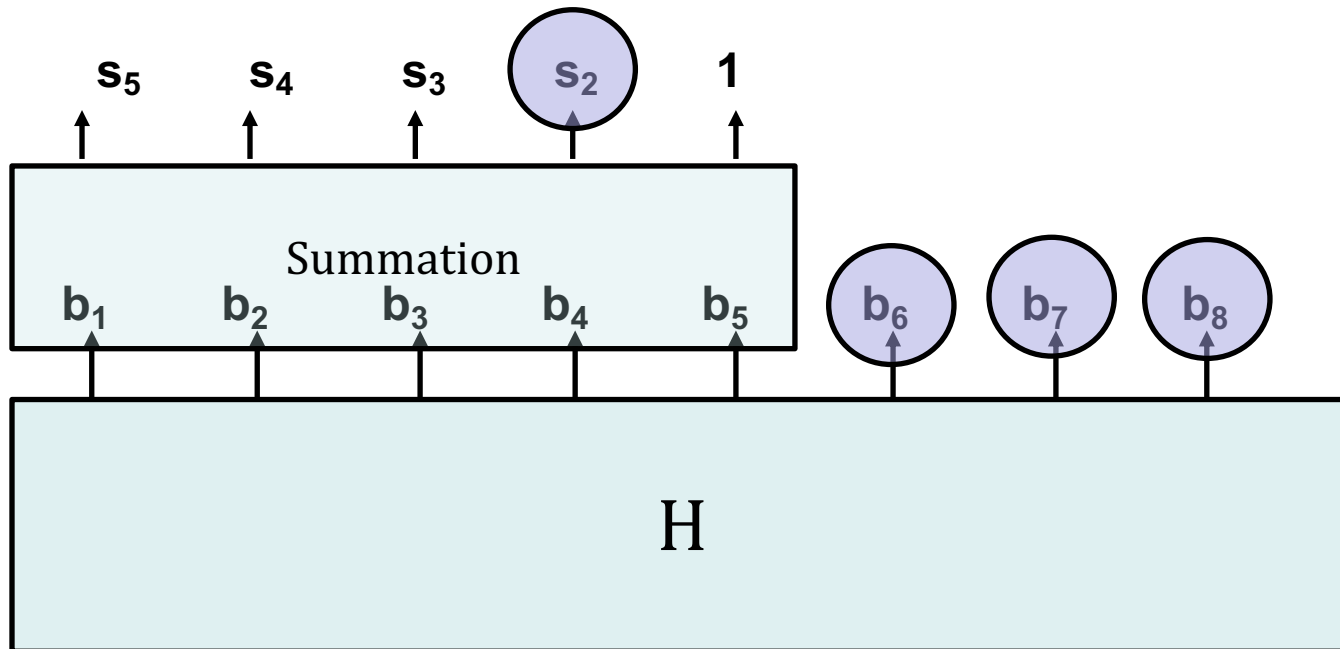
MUS3 Incremental

- Get another core, add 1 to overall cost.
- Get another conflict, (s_2, b_6)
 - ▶ Either we must falsify two of $\neg b_1 \dots \neg b_5$ or one of $\neg b_1 \dots \neg b_5$ and $\neg b_6$



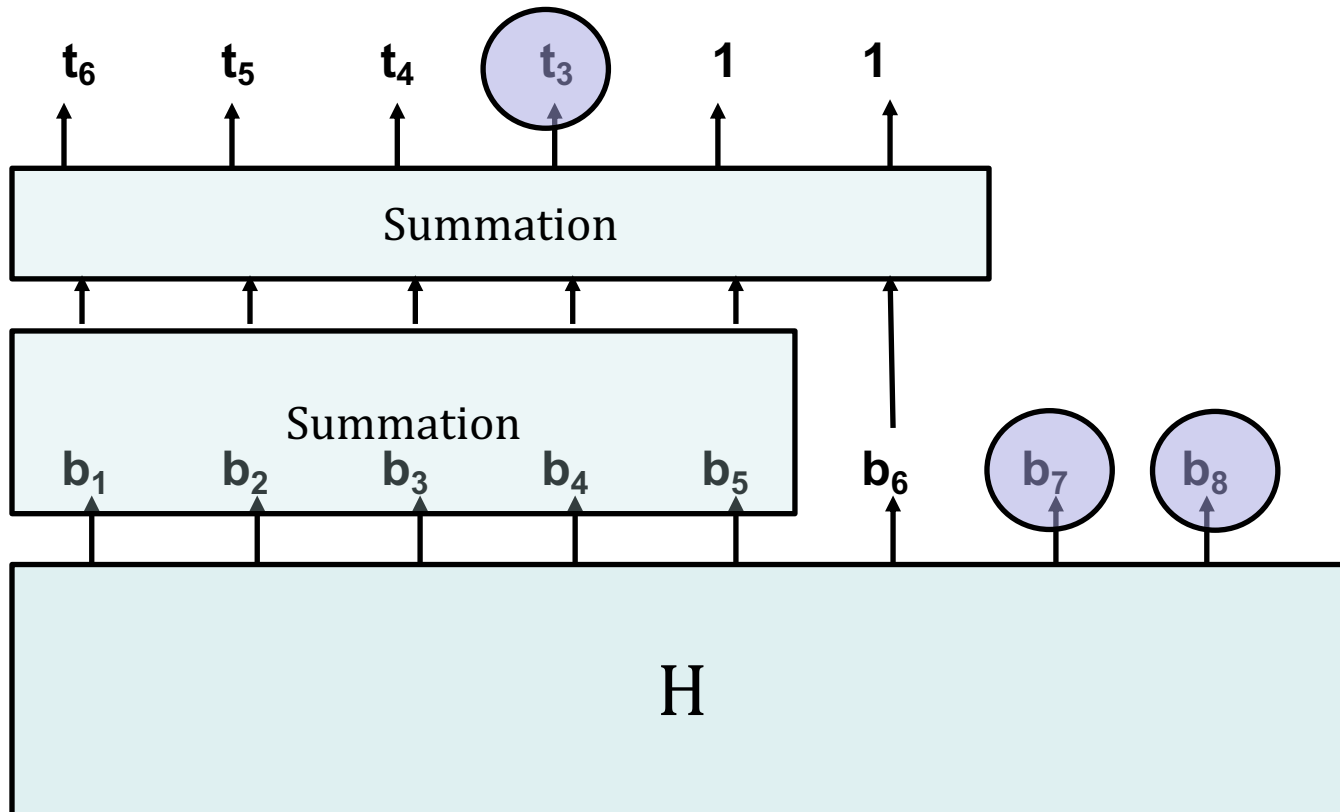
MUS3 Incremental

- Now the incremental part. MUS3-Incremental reuses the previous summation constraint and its variables **subsuming** them into a new summation constraint that accounts for the new core (s_2, b_6)



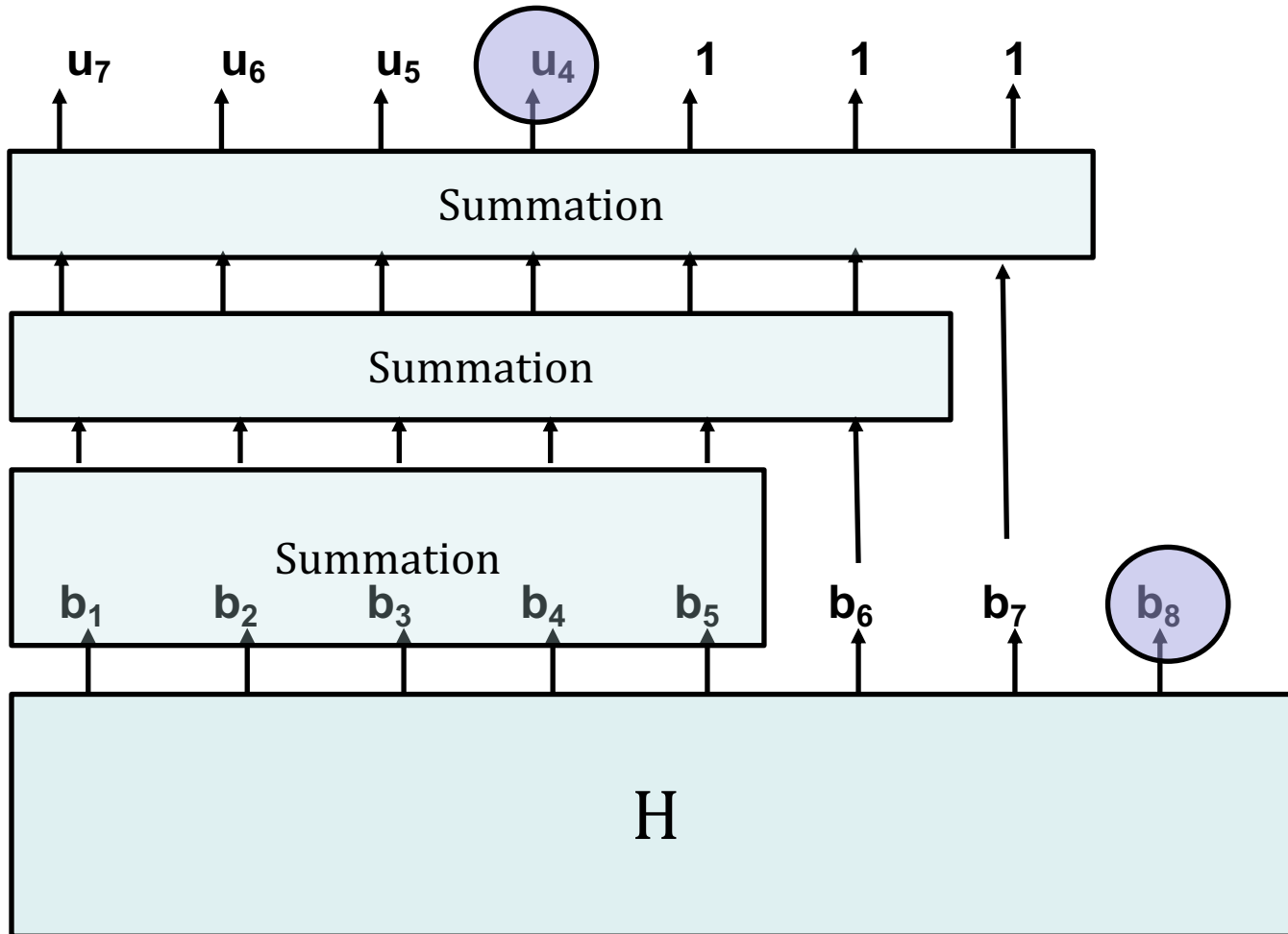
MUS3 Incremental

- We know that the sum must be at least 2—set t_1 and t_2 to 1.
- Now **SAT_ASSUME**($H \cup \text{Card}, \{\neg t_3, \neg b_7, \neg b_8\}$),
- Allow two softs to be falsified among $\neg b_1$ — $\neg b_6$



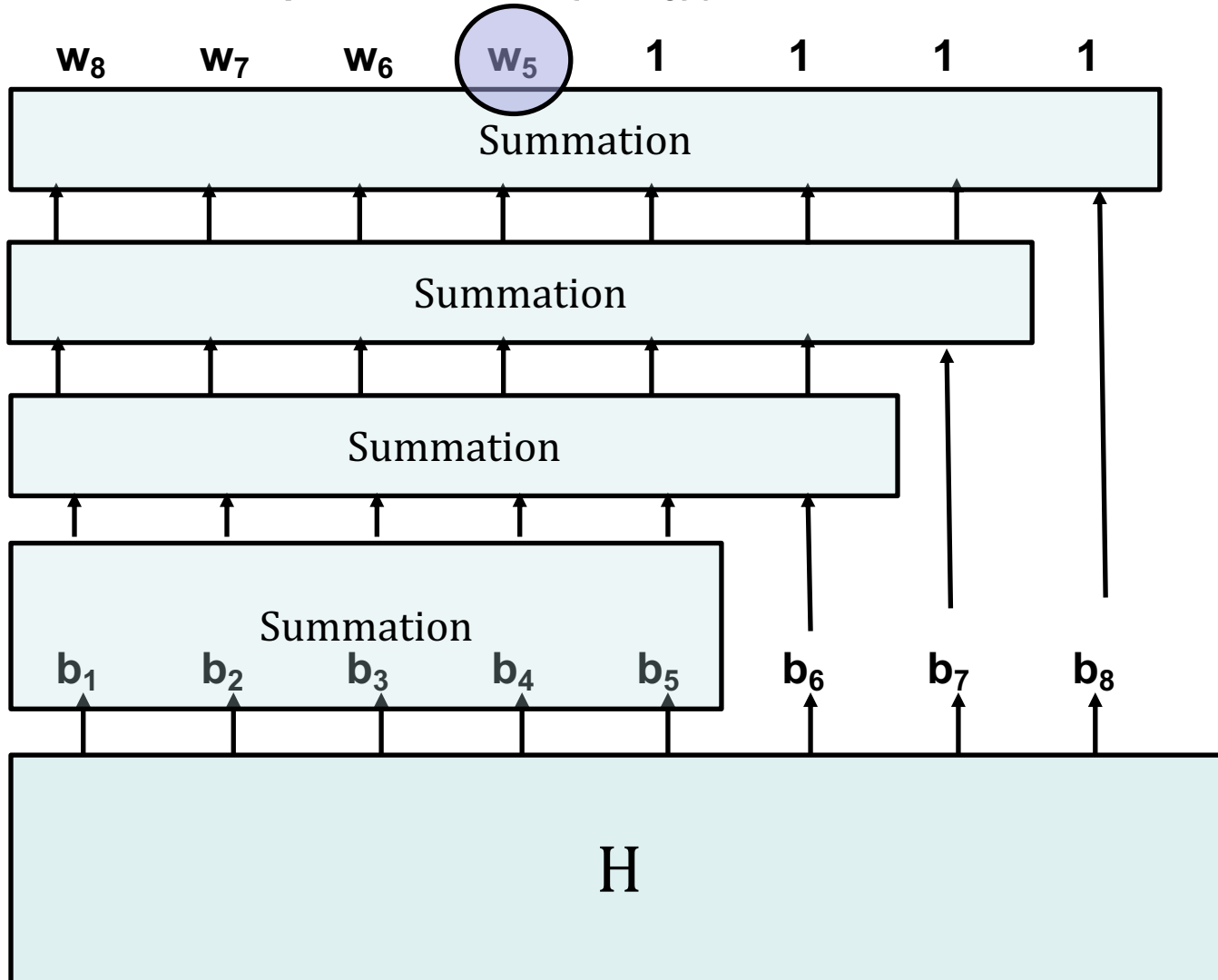
MUS3 Incremental

- Another core (t_3, b_7), add 1 to the overall cost.
- Now **SAT_ASSUME(H U Card, $\{\neg u_4, \neg b_8\}$)**, allow up to 3 falsified softs



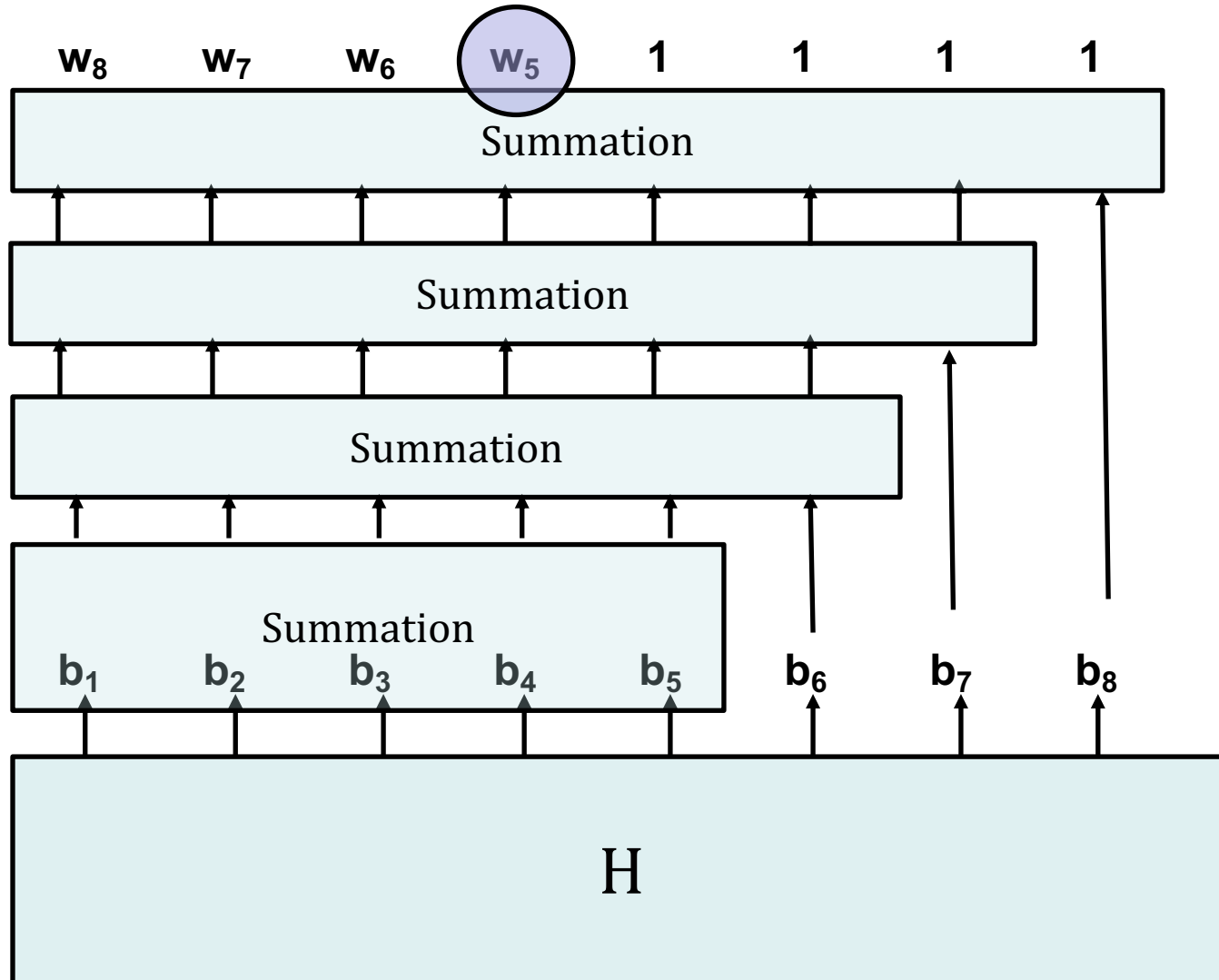
MUS3 Incremental

- Another core (u_4, b_8); Add one to overall cost. Then **SAT_ASSUME(H U Card, $\{\neg w_5\}$)**



MUS3 Incremental

- SAT – optimal solution with overall cost = accumulated overall cost



OLL

- Used in RC2.
- A. Morgado, C. Dodaro, and J. Marques-Silva. **Core-guided MaxSAT with soft cardinality constraints**. CP 2014
- Uses summation circuits like MUS3 but links up the summation circuits in a more flexible way.

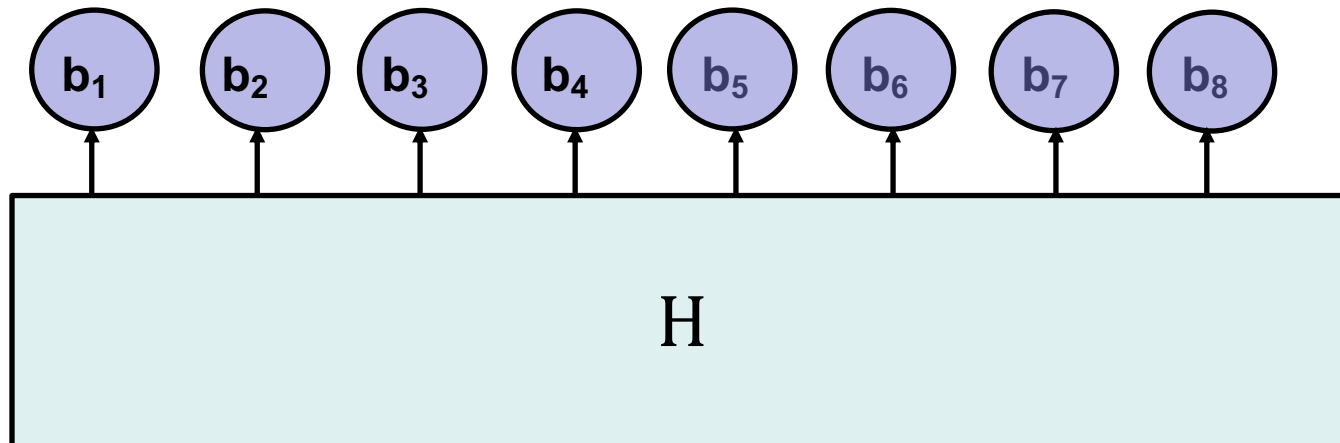
OLL

With same example

Start with Cardinality Constraint Layer with inputs only

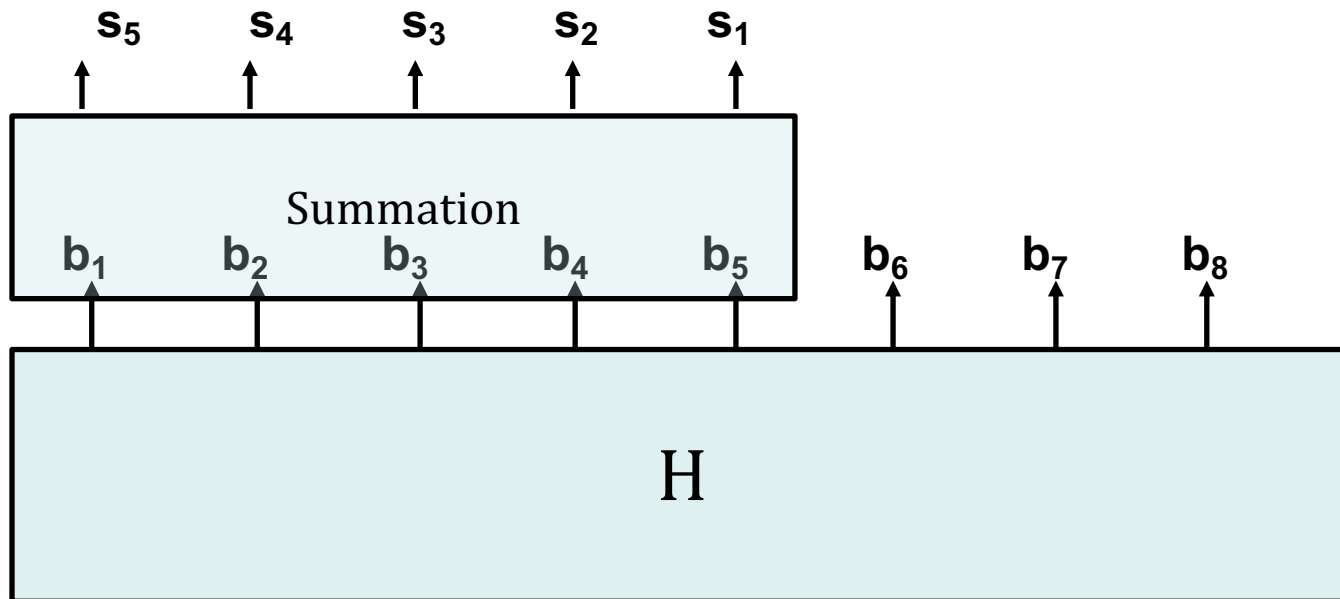
 Literals assumed FALSE

First SAT solve is $\text{SAT_ASSUME}(H, \{\neg b_1, \neg b_2, \dots, \neg b_8\})$,
I.e., try to falsify zero softs.



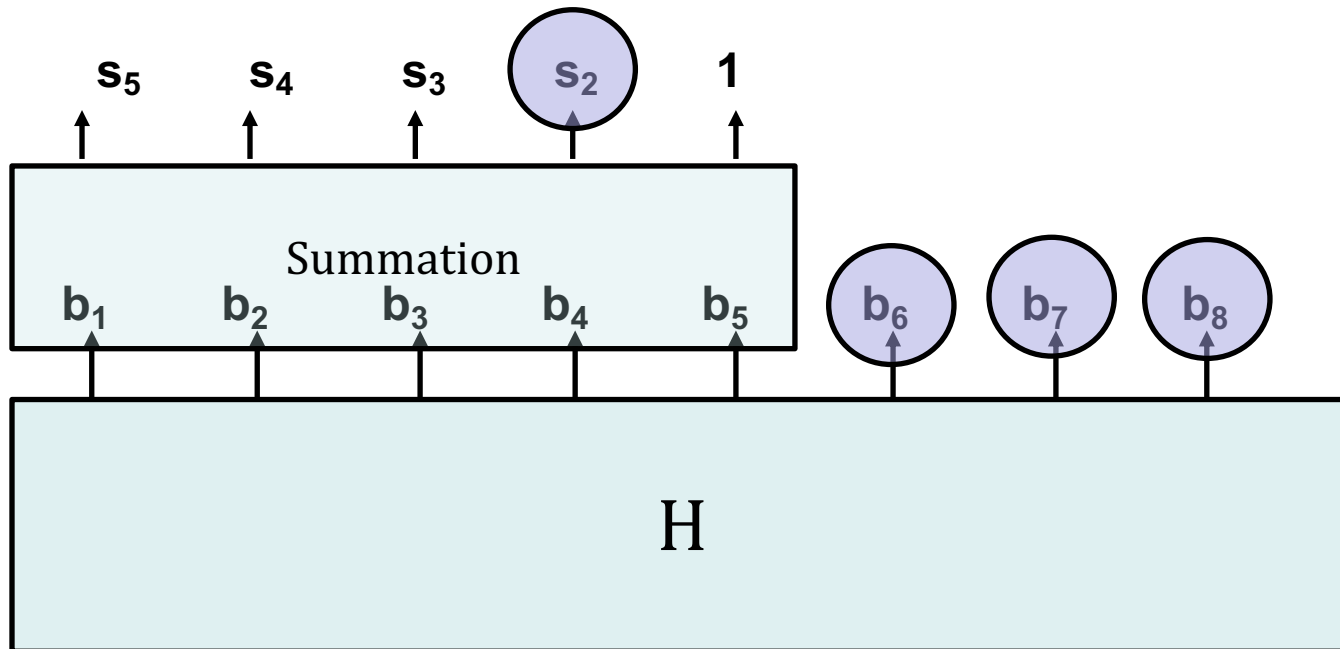
OLL

- UNSAT; say we get the core $(b_1, b_2, b_3, b_4, b_5)$
- Build summation network over softs in core



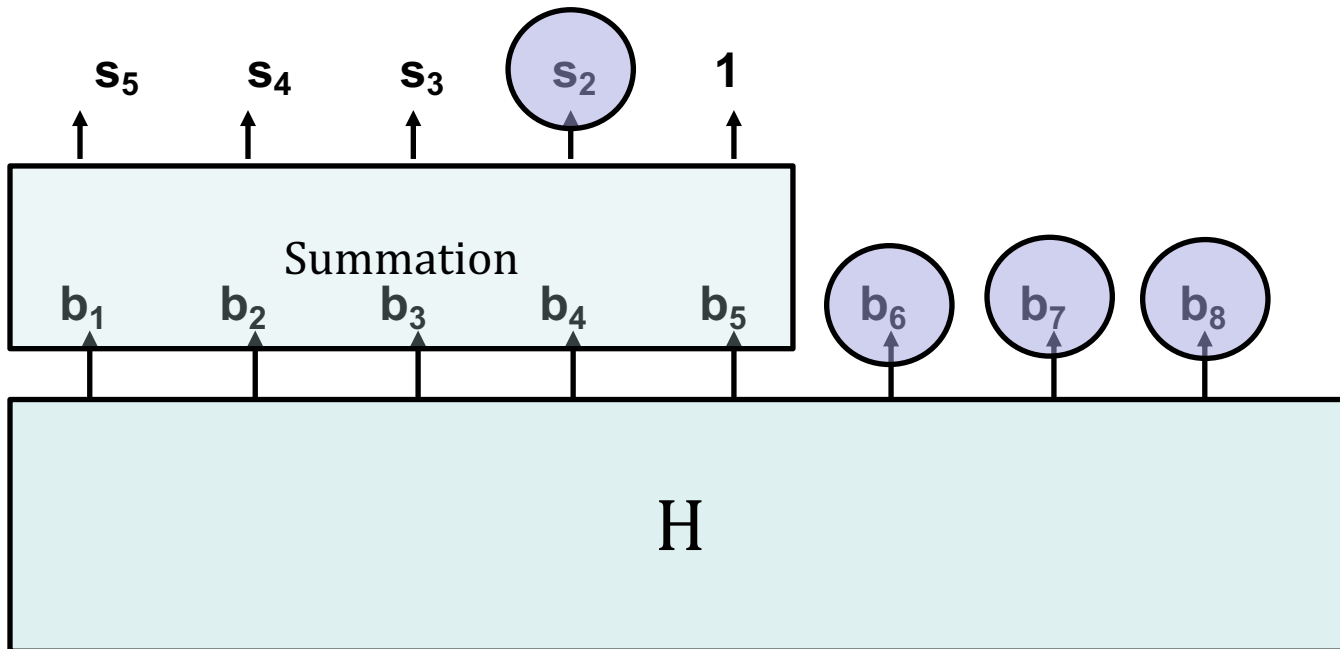
OLL

- We know that the sum must be at least one—set $s_1 = 1$.
- Now **SAT_ASSUME(H U CARD, { $\neg s_2, \neg b_6, \neg b_7, \neg b_8$ })**,
- Allow one soft to be falsified among $\neg b_1$ — $\neg b_5$



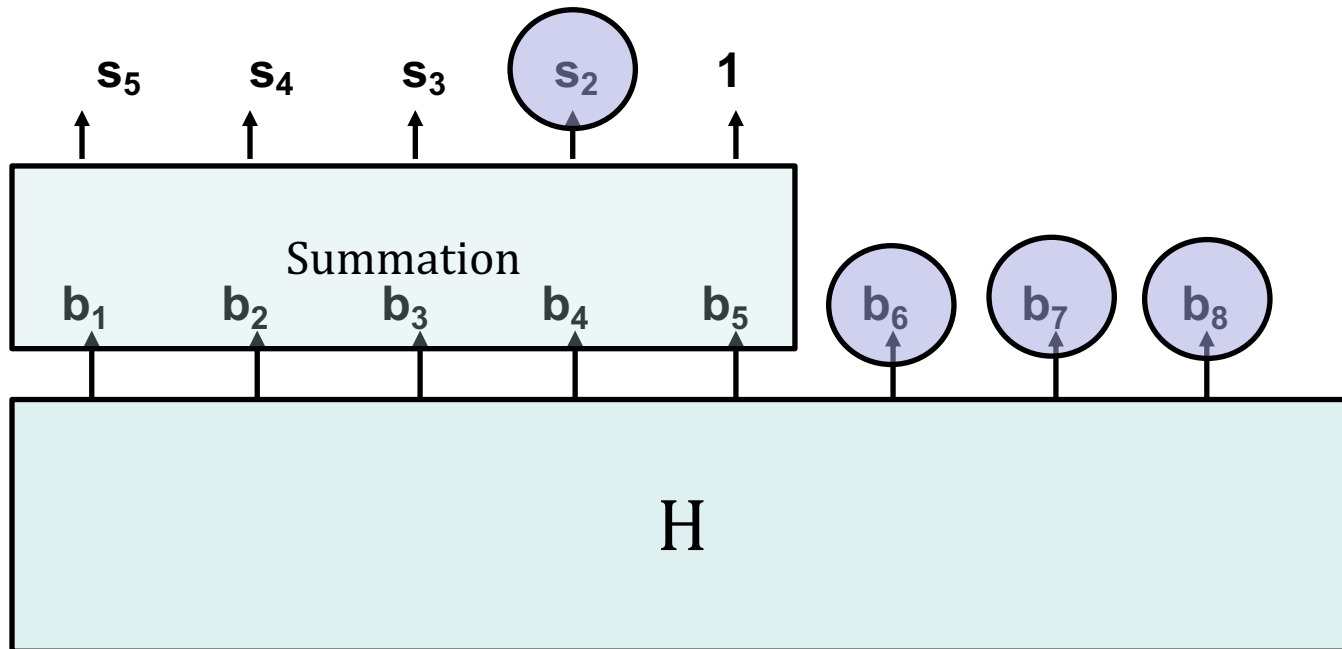
OLL

- Get another core
- Get another conflict, e.g., (s_2, b_6)

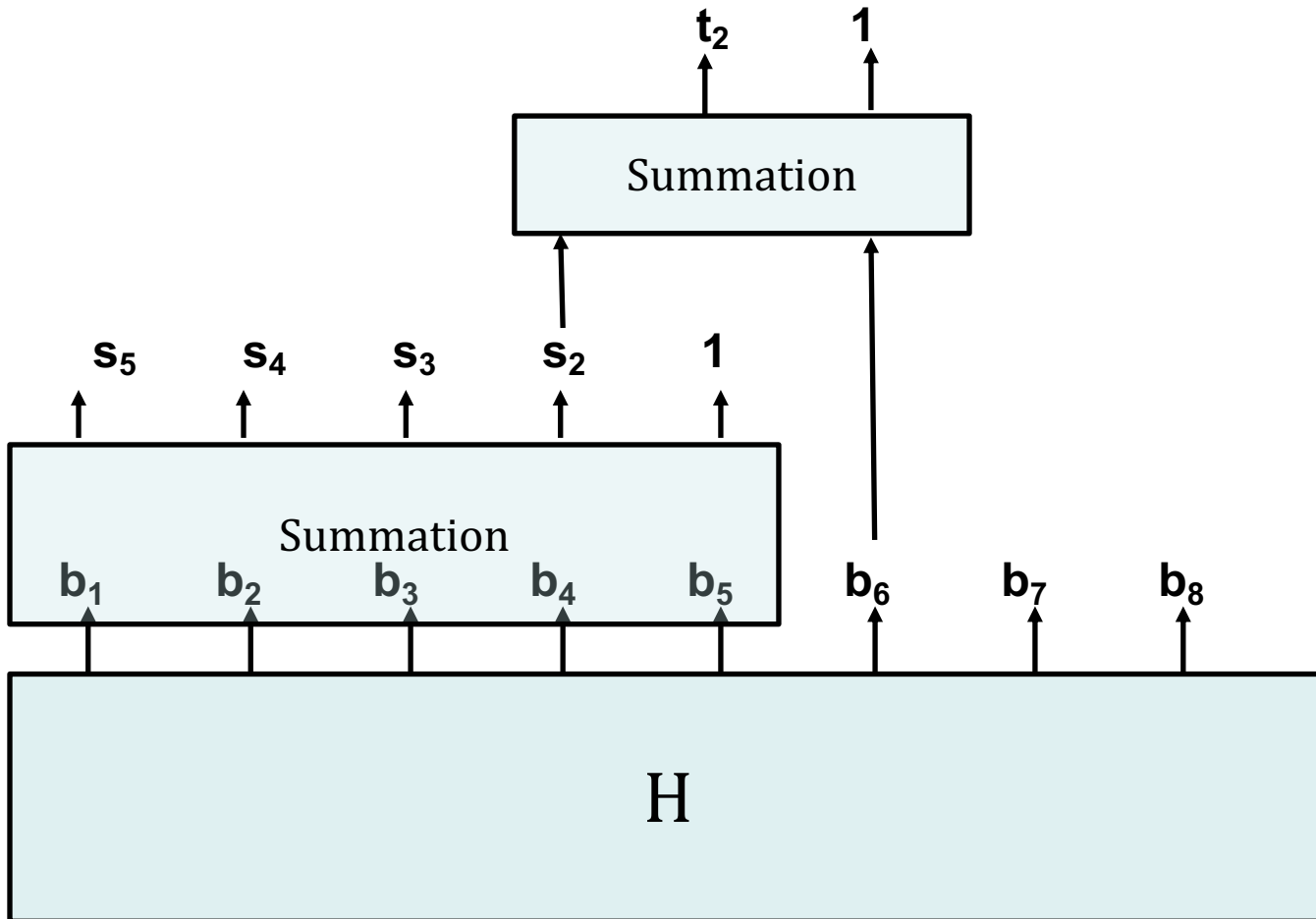


OLL

- Like MSU3 we create a new summation constraint but unlike MSU3 we do not subsume the prior summation constraint into the new one. Only the literals of the conflict.

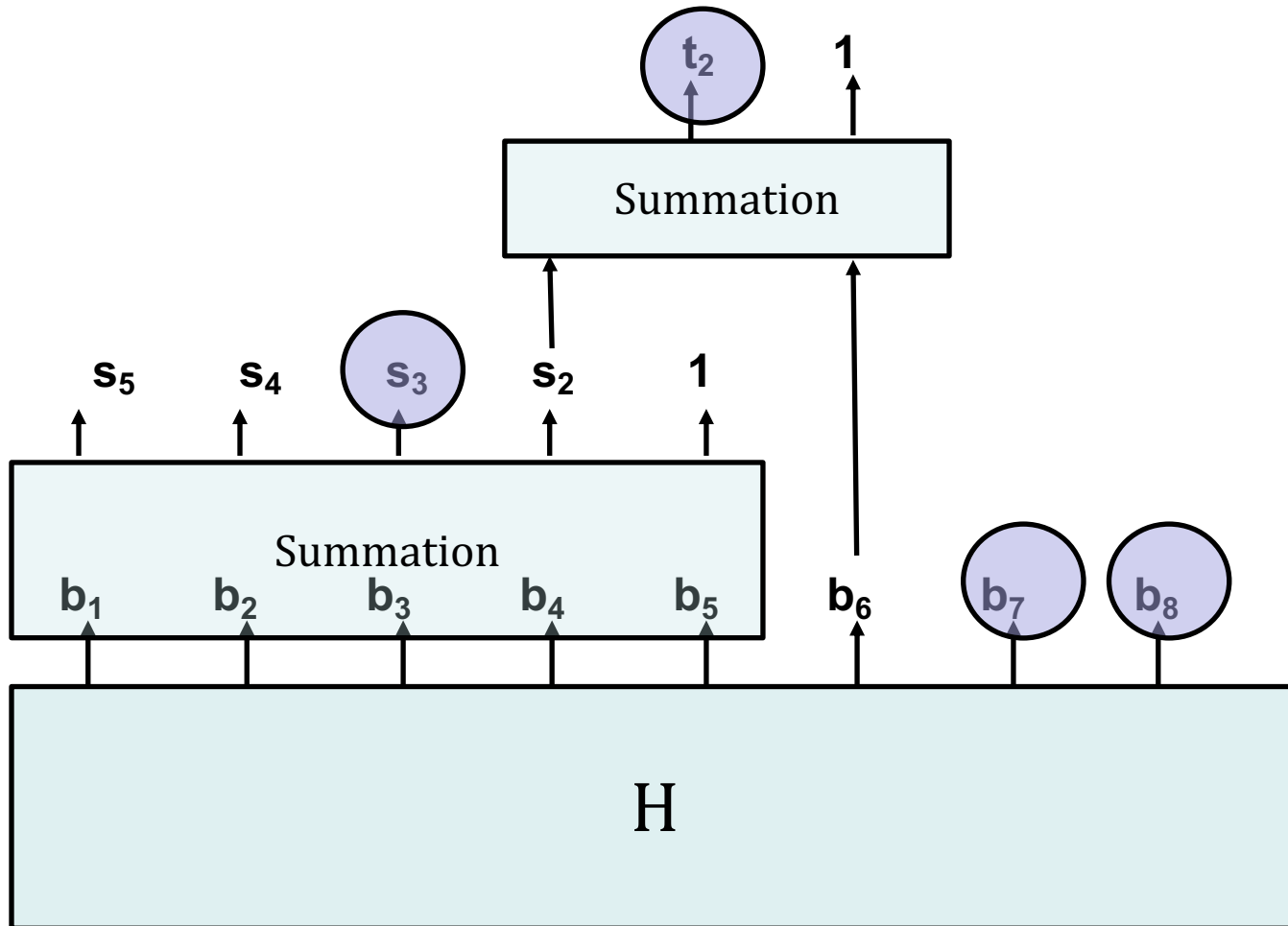


OLL



OLL

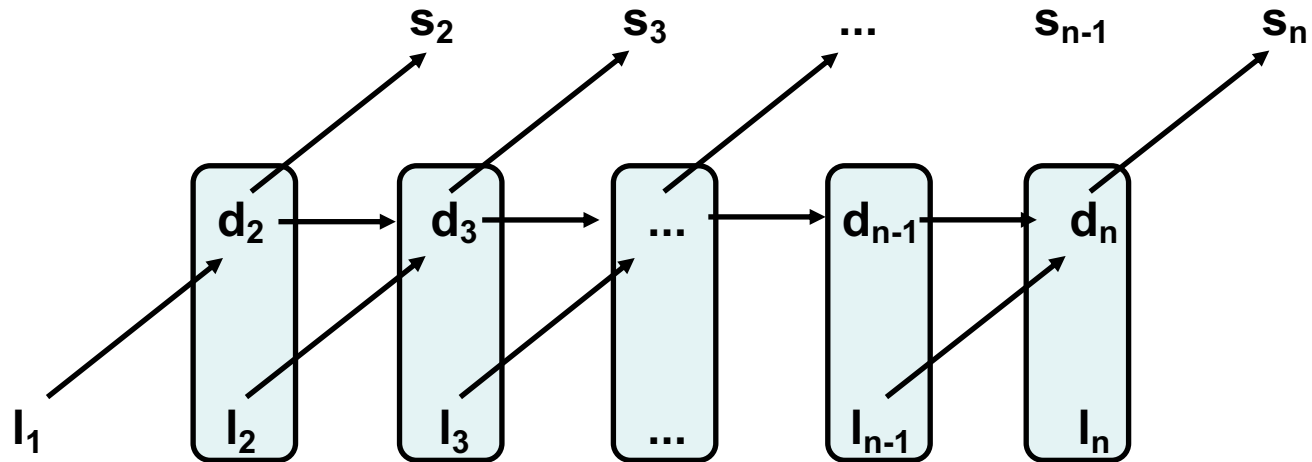
- Then $\text{SAT_ASSUME}(H \cup \text{CARD}, \{\neg s_3, \neg t_2, \neg b_7, \neg b_8\})$...continue in this way



PMRES

- Used in Eva500a.
- Narodytska, N., Bacchus, F. **Maximum satisfiability using core-guided MaxSAT resolution.** AAI 2014
- Does not use summation circuits, rather it uses a circuit that detects if more than one literal is true.

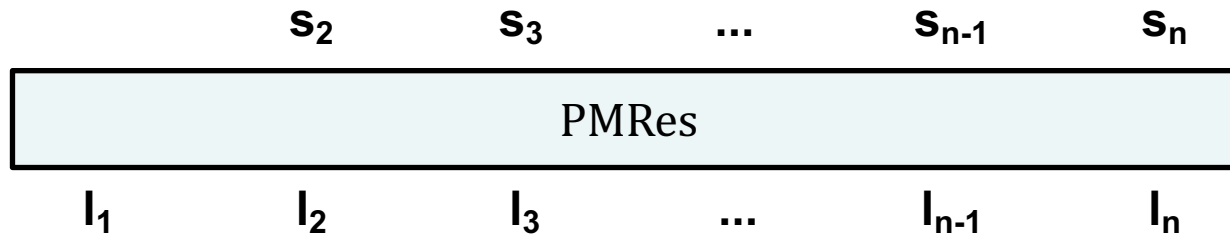
PM-Res



- The d_i and s_i variables are new.
- We make d_i if d_{i-1} or l_{i-1} are true.
 - d_i encodes that at least one of the first $i-1$ inputs is true.
- $d_i \wedge l_i \rightarrow s_i$ one of the first $i-1$ inputs was true and l_i is true (i.e., sum of $l_1 .. l_i$ is ≥ 1 AND l_i is true)

PM-Res

- Draw this as below.



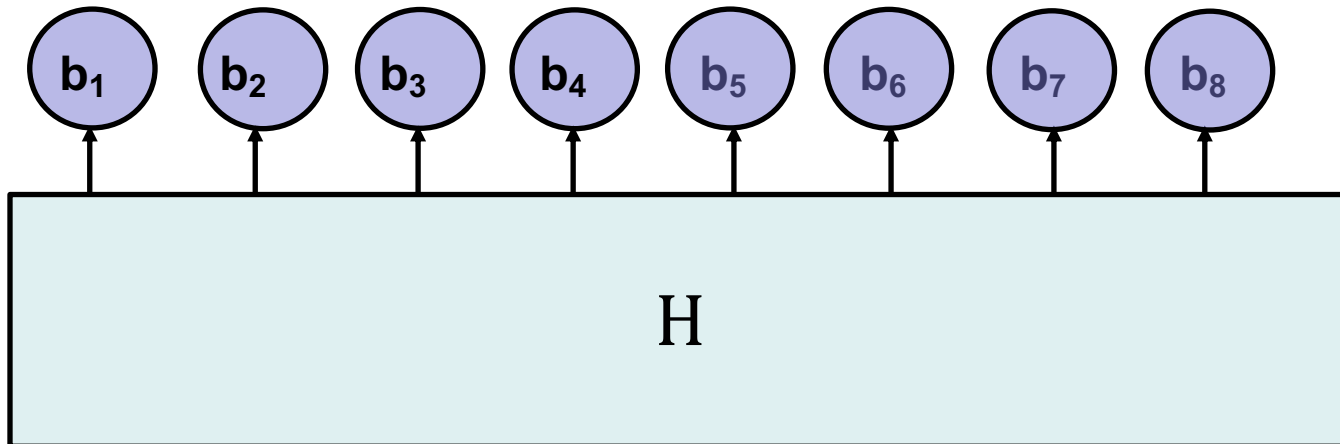
PMRes

Using same example

Start with Cardinality Constraint Layer with inputs only

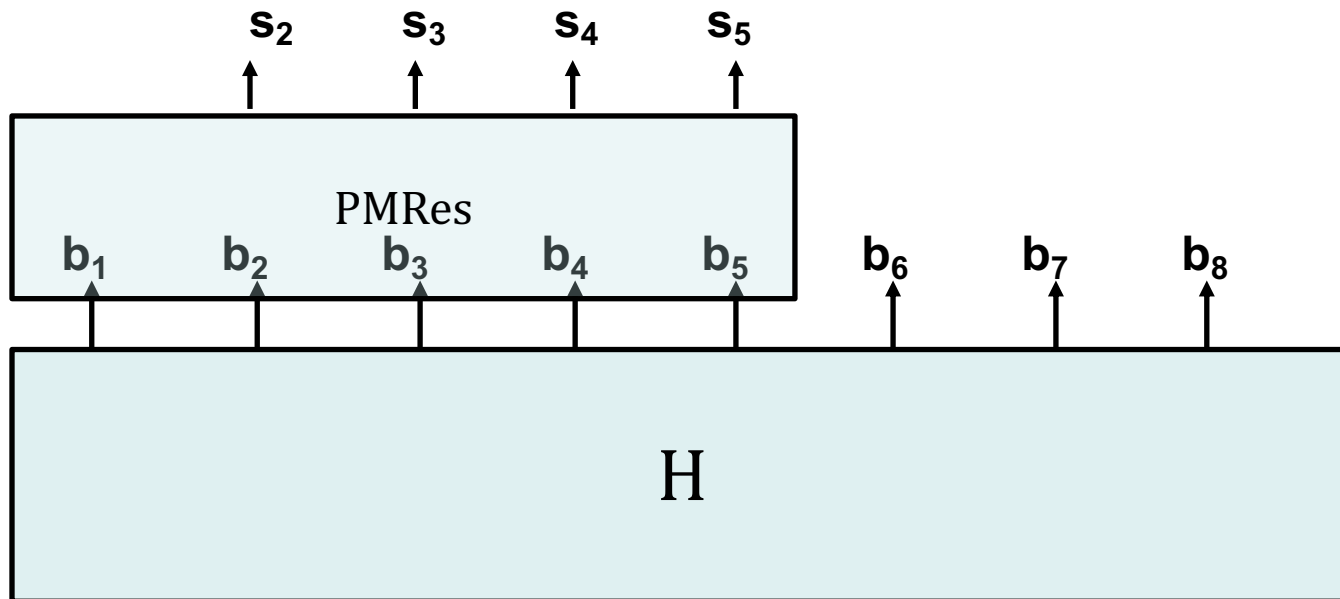
 Literals assumed FALSE

First SAT solve is $\text{SAT_ASSUME}(H, \{\neg b_1, \neg b_2, \dots, \neg b_8\})$,
I.e., try to falsify zero softs.



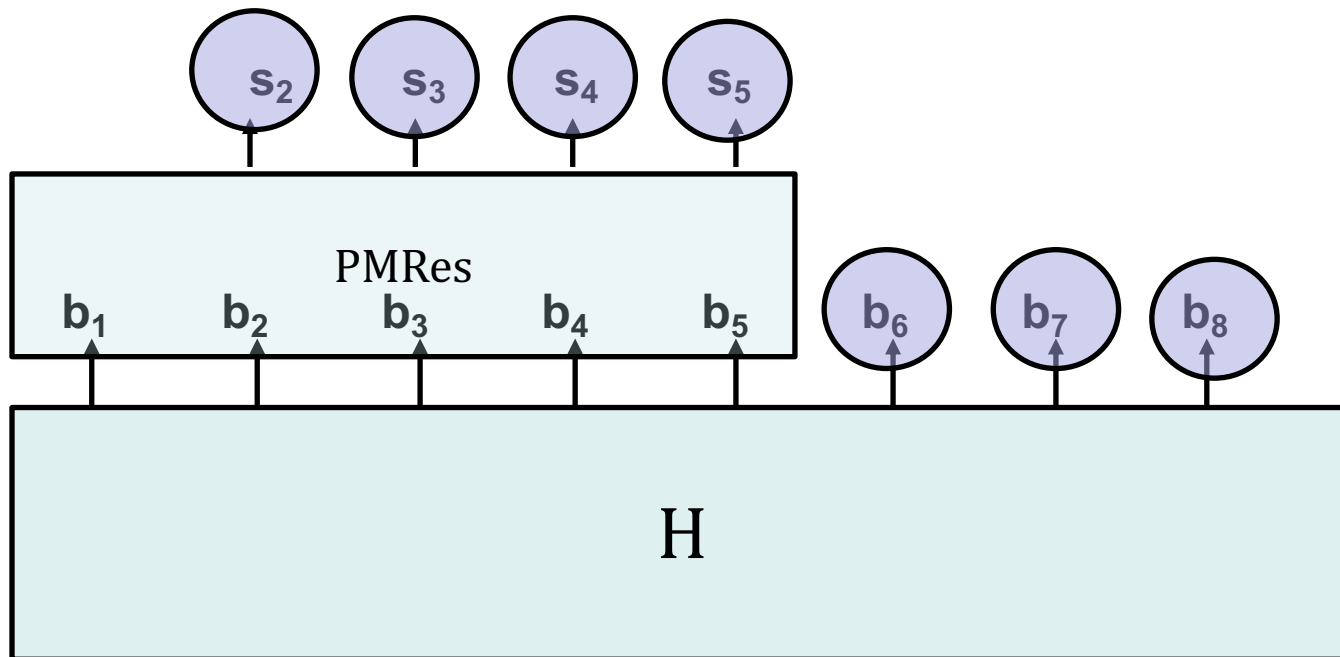
PMRes

- UNSAT; say we get the core $(b_1, b_2, b_3, b_4, b_5)$
- Build PMRes circuit over softs in core.
- In the new formula (with the PMRes circuit) we always assume the negation of all outputs of the cardinality layer



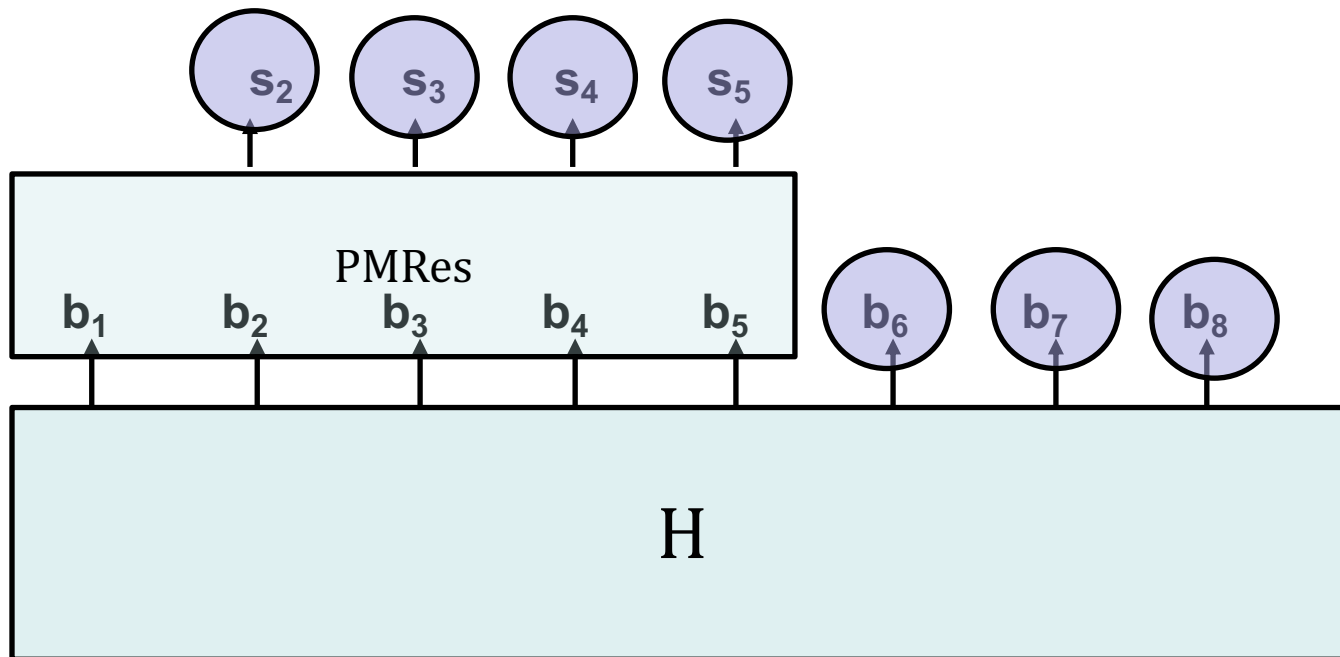
PMRes

- Now **SAT_ASSUME(HUCard, { $\neg s_2, \neg s_3, \neg s_4, \neg s_5, \neg b_6, \neg b_7, \neg b_8$ })**,
- **→** We can falsify at most one of the softs $\neg b_1, \dots, \neg b_5$ but no other softs.
- UNSAT (we must falsify at least 4)



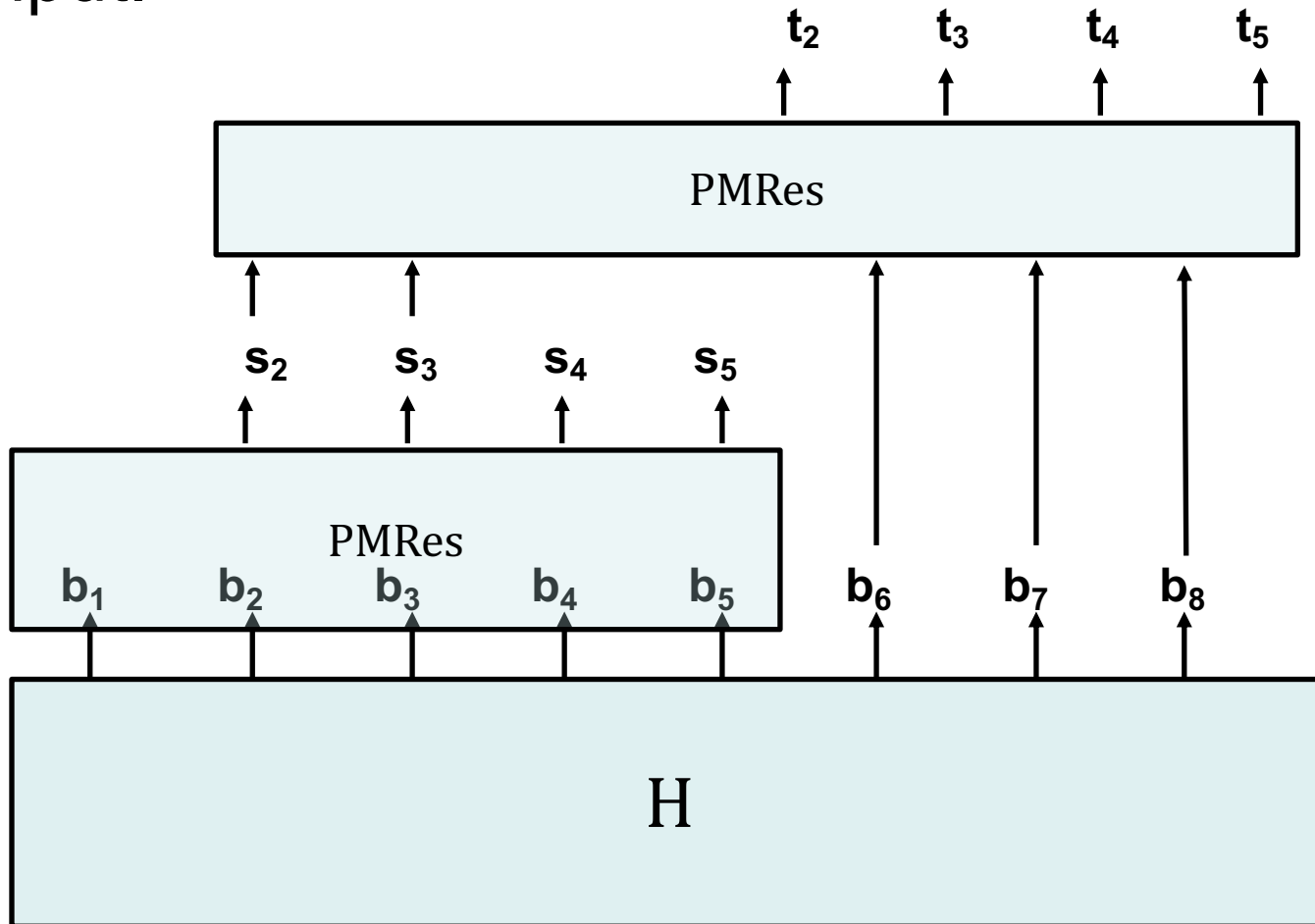
PMRes

- Various conflicts, say **SAT_ASSUME(H, { $\neg s_2, \neg s_3, \neg s_4, \neg s_5, \neg b_6, \neg b_7, \neg b_8$ })** returns the core $(b_6, b_7, b_8, \neg s_2, \neg s_3)$
- PMRes then builds a new PMRes circuit with these literals as input.



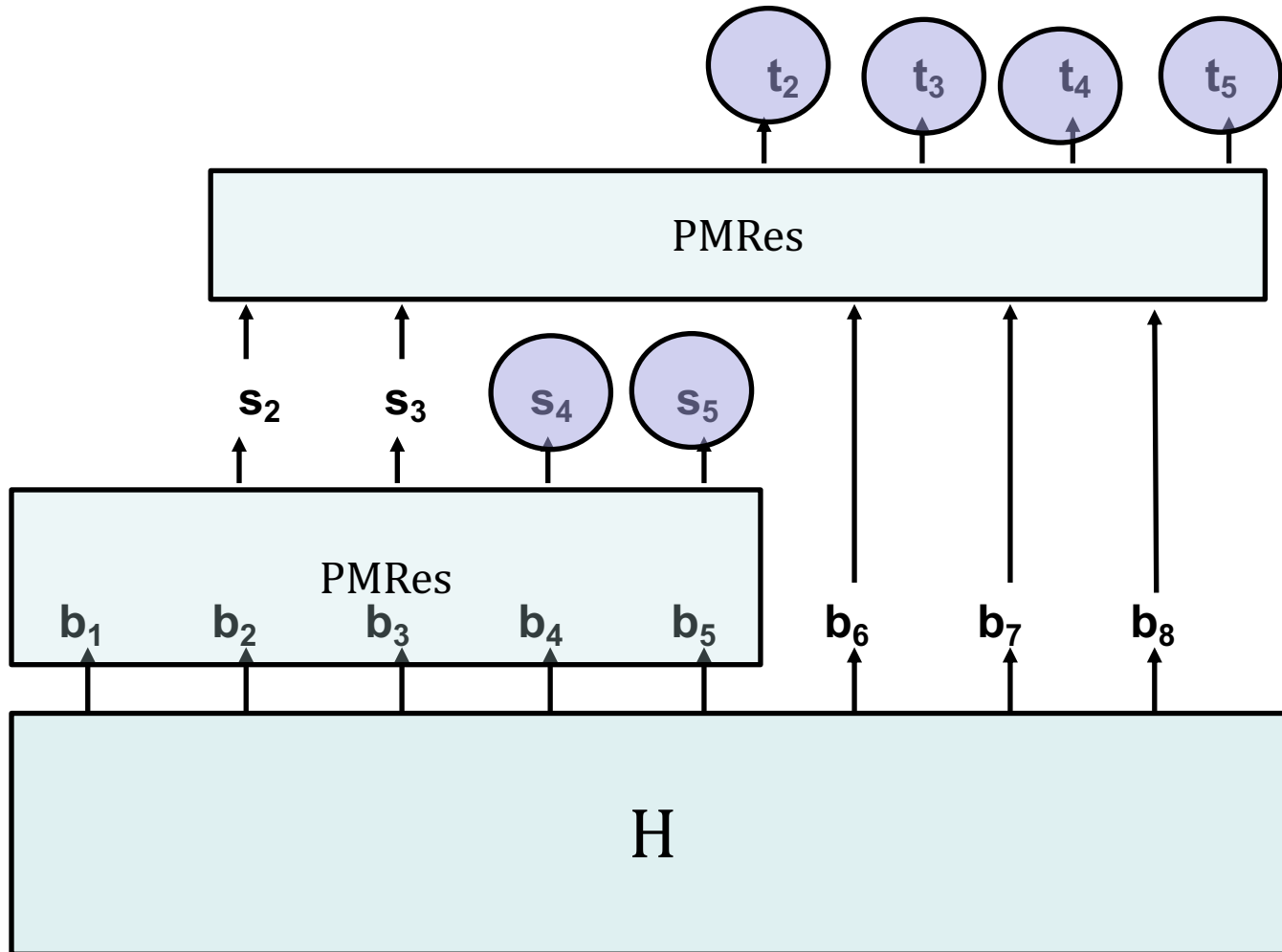
PMRes

- new core ($b_6, b_7, b_8, \neg s_2, \neg s_3$)
- PMRes then builds a new PMRes circuit with these as input.



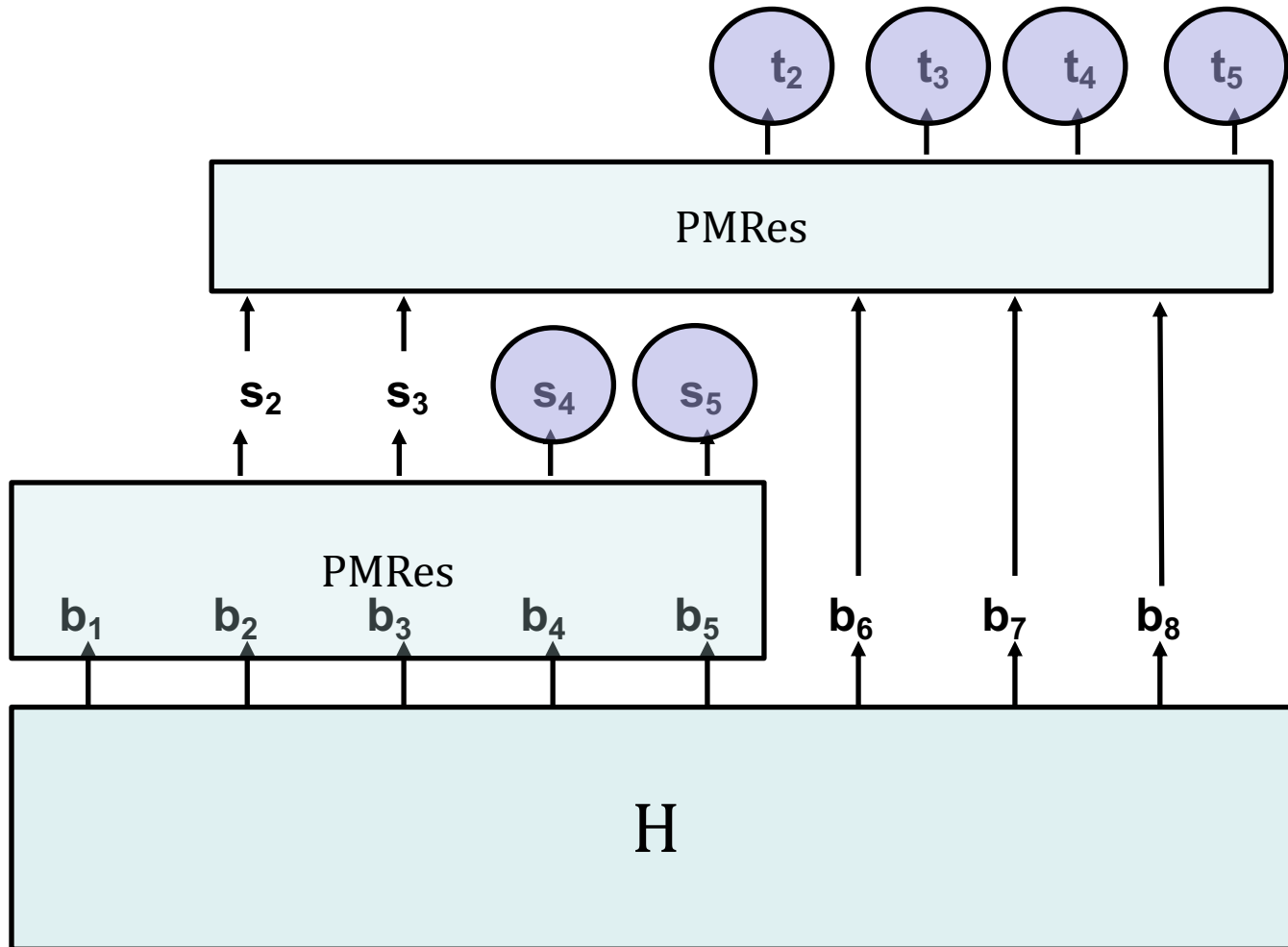
PMRes

- **SAT_ASSUME(H, { $\neg s_4$, $\neg s_5$, $\neg t_2$, $\neg t_3$, $\neg t_4$, $\neg t_5$ })**
- It gets complex...



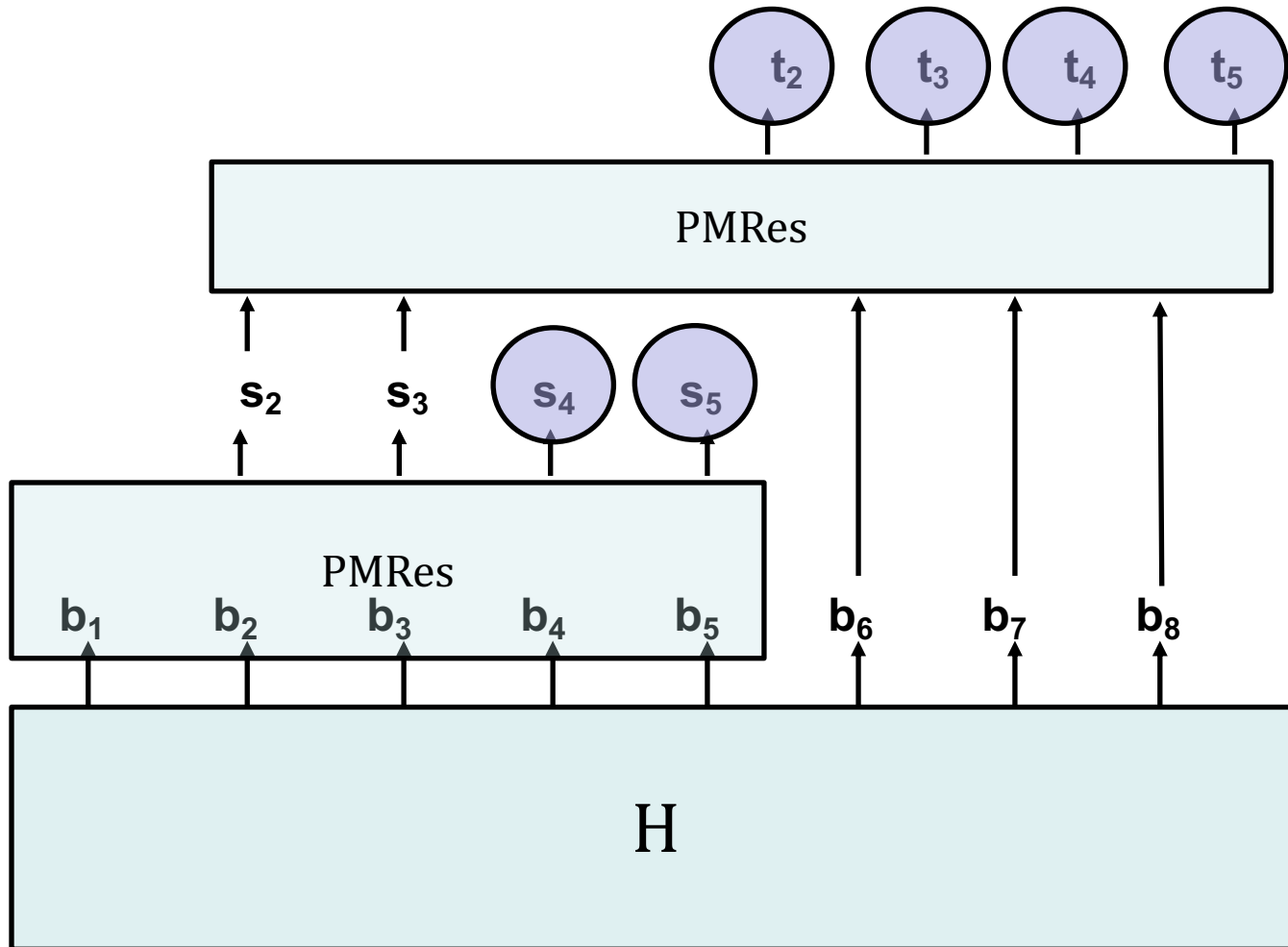
PMRes

- But subject to these assumptions at most two original softs can be falsified.



PMRes

- PMRes continues in this way until it achieves SAT.



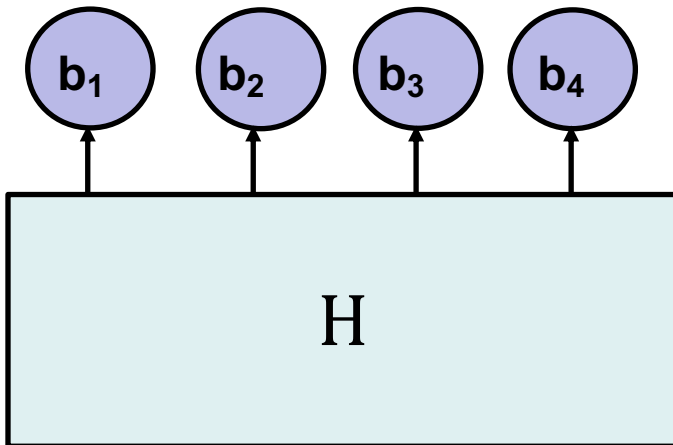
Dealing with Weights.

- These algorithms employ the technique of **clause cloning** to deal with weighted softs.
- When a core is found e.g., $\{b_1, b_2, b_3, b_4\}$ where the soft clauses $(\neg b_i)$ have different weights.
 - Let minWt be the minimum weight among the softs in the core.
 - Add minWt to the overall cost (instead of 1)
 - Make a copy of those literals that have cost greater than minWt . Give these copies weight = original weight – minWt

Weighted instances

Say $S = \{(\neg b_1), (\neg b_2), (\neg b_3), (\neg b_4)\}$ with $\text{wt}(\neg b_i) = i$

First SAT solve is $\text{SAT_ASSUME}(H, \{\neg b_1, \neg b_2, \neg b_3, \neg b_4\})$,

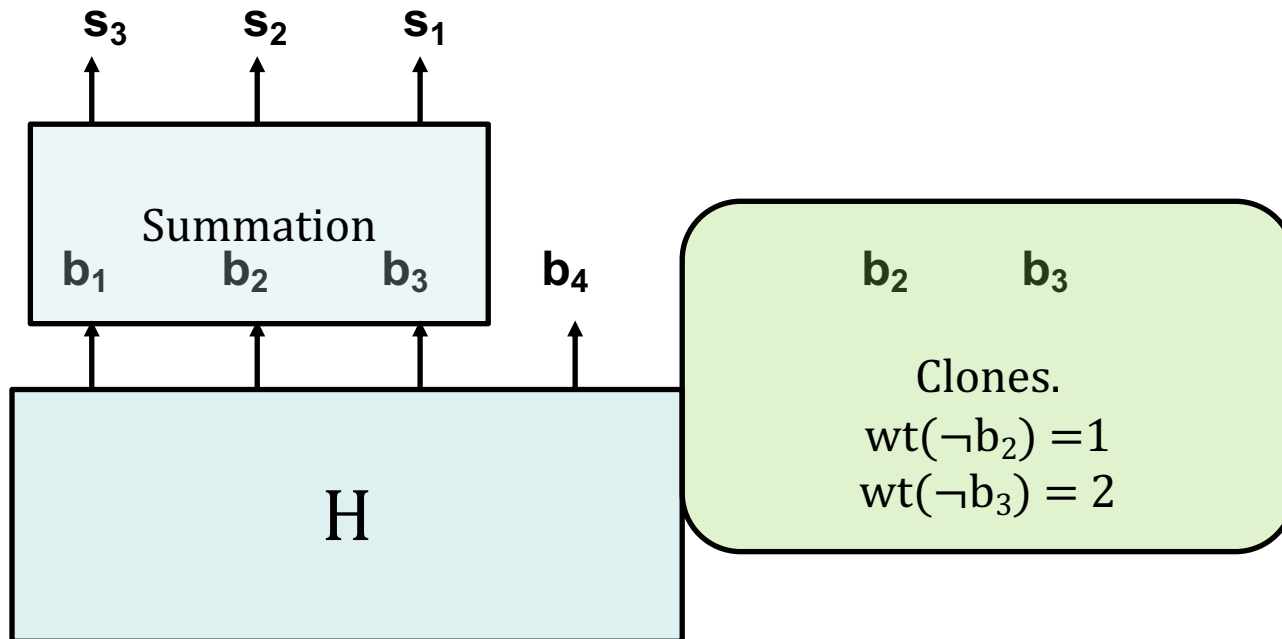


Weighted instances

UNSAT; say we get the core (b_1, b_2, b_3)

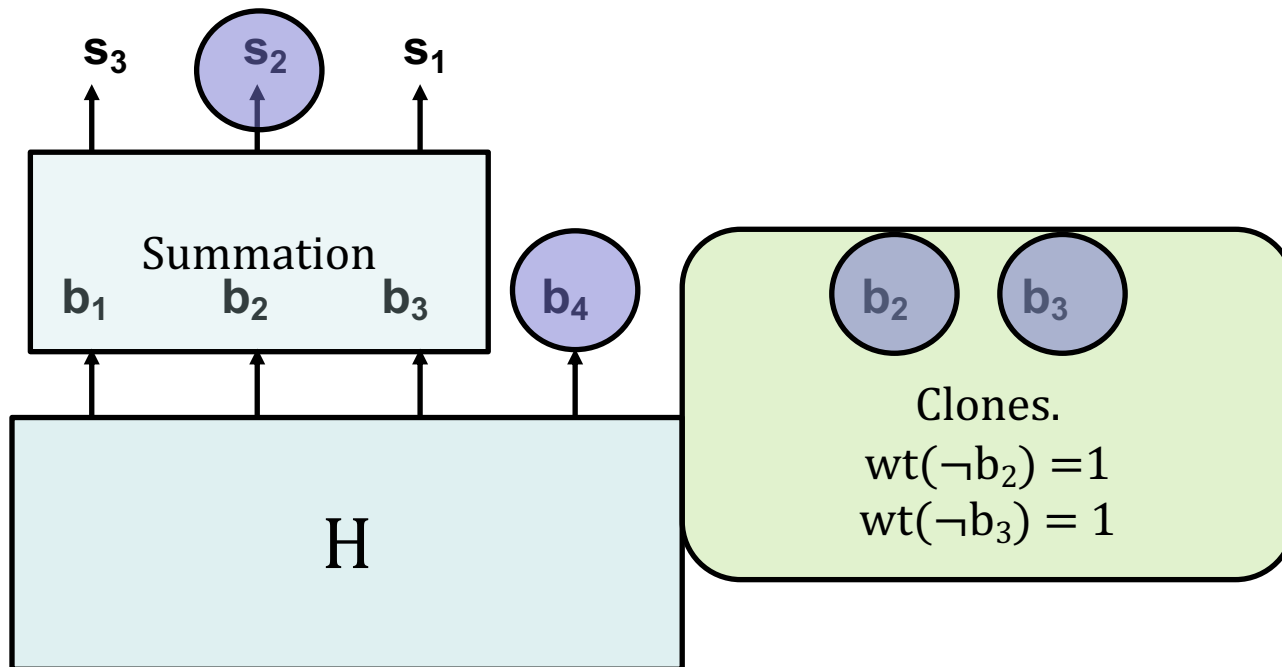
As before build summation network over softs in core.

$\min Wt=1$, so we must clone $\neg b_2$ and $\neg b_2$.



Weighted instances

Now $\text{SAT_ASSUME}(H, \{\neg s_2, \neg b_4, \neg b_2, \neg b_3\})$,



Solving a sequence of relaxations

- The approach can be very effective. Much more effective than the naive approach of restricting the sum over all falsified soft clauses.
- The discovered cores are exploited in a non-trivial manner to constraint the search for the set of soft clauses that can be falsified.

Solving a sequence of relaxations

- The structure of the Cardinality layer becomes quite complex, and although clear empirical differences can be observed among the different ways of constructing the Cardinality layer, there is no real understanding of this.

Solving a sequence of relaxations

- As the cardinality layer grows the SAT solver has a harder and harder time solving it.
- The approach of clause cloning for dealing with weighted instances is limited.
- On unweighted instances (all softs have the same weight) sequence of sat approaches are often the best approaches.

Implicit Hitting Set (IHS) Approach to MaxSat

- First developed by [Davies PhD]
- IHS solvers utilize both a SAT solver and an Integer Program solver (IP)

Implicit Hitting Set (IHS) Approach to MaxSat

- Unlike the previous approaches, IHS solvers never modify the input MaxSat \mathbf{F} .
 - The SAT solver is always run on instances that are no more complex than the input \mathbf{F} .
- The cores exploited by IHS solvers are cores of the input formula \mathbf{F} (not cores of \mathbf{F} augmented by cardinality constraints).
- All numeric reasoning about weights is delegated to an Integer Programming solver (e.g., CPLEX)
 - designed for optimization
 - weights can be floating point numbers
 - the underlying LP + Cuts approach is very powerful

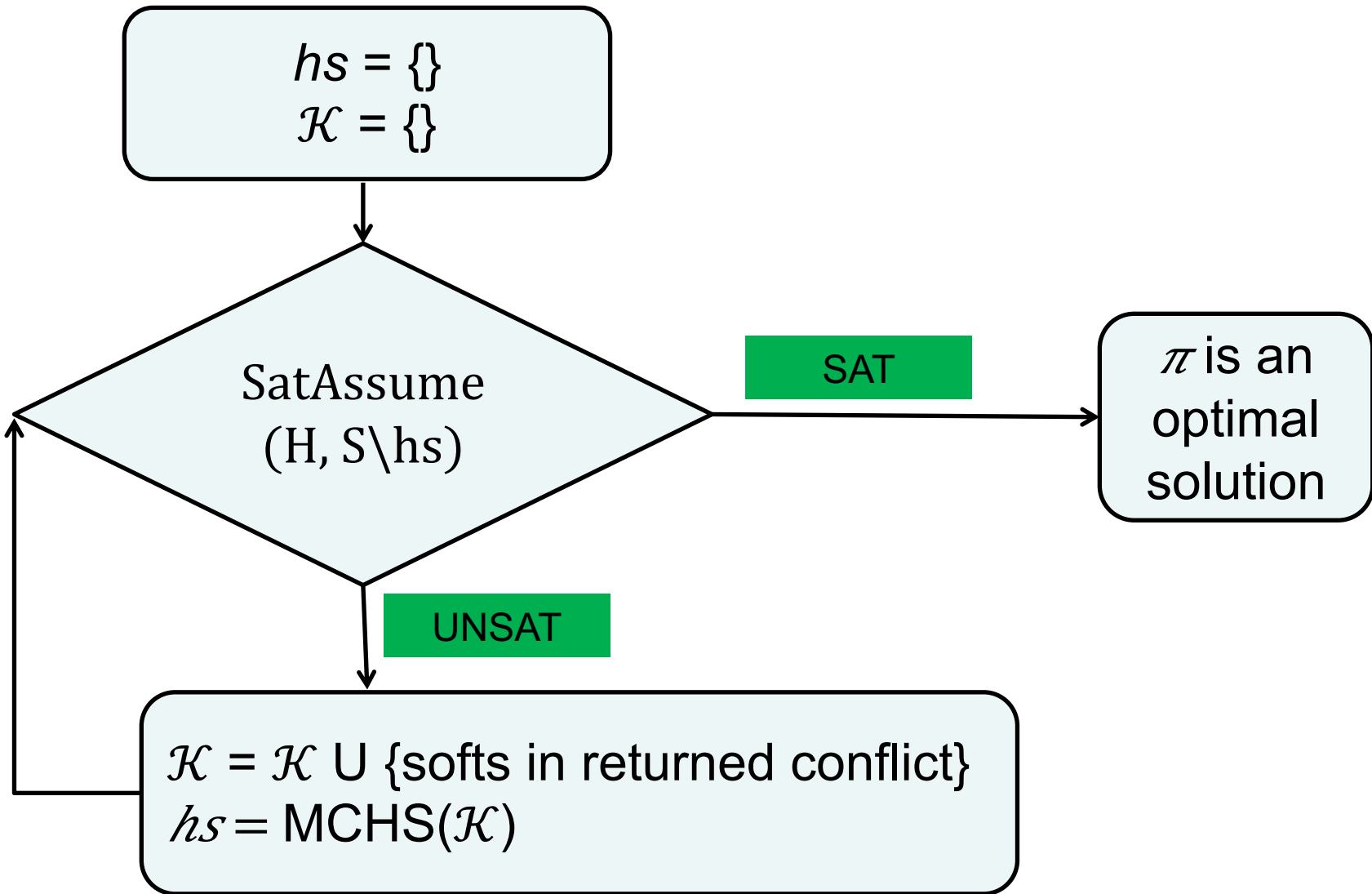
Cores and hitting sets

- Remember
 - A set of soft clauses $\kappa \subseteq \mathbf{S}$ is a **core** of \mathbf{F} if $\kappa \cup \mathbf{H}$ is **UNSAT**
 - Feasible solutions satisfy the hard clauses \mathbf{H}
- Let \mathbf{K} be any set of cores of \mathbf{F} and π any feasible solution. π **must falsify** at least one soft clause of every core in \mathbf{K} .
- Let $A = \{c \mid \pi \not\models c\}$ be the set of clauses falsified by π
- Then A is a hitting set of \mathbf{K} (non-empty intersection with every member of \mathbf{K}).

Cores and hitting sets

- Let $\text{MCHS}(\mathbf{K})$ be a minimum cost hitting set of \mathbf{K} —this is a set of soft clauses.
- For every feasible solution π
$$\text{cost}(\pi) = \text{wt}(A) \geq \text{wt}(\text{MCHS}(\mathbf{K}))$$
- The weight of a minimum cost hitting set of **any** set of cores is a **lower bound** on the cost of an optimal solution.
- Therefore, for **any** set of cores \mathbf{K} and **any** feasible solution π if $\text{cost}(\pi) = \text{wt}(\text{MCHS}(\mathbf{K}))$, π must be an **optimal solution**.
- This leads to a simple algorithm for finding an optimal solution.

The IHS Algorithm



The IHS Algorithm

$hs = \{\}$
 $\mathcal{K} = \{\}$

π satisfies H and all soft clauses except possibly the softs in hs . So $cost(\pi) \leq wt(MCHS(\mathcal{K}))$

SatAssume
($H, S \setminus hs$)

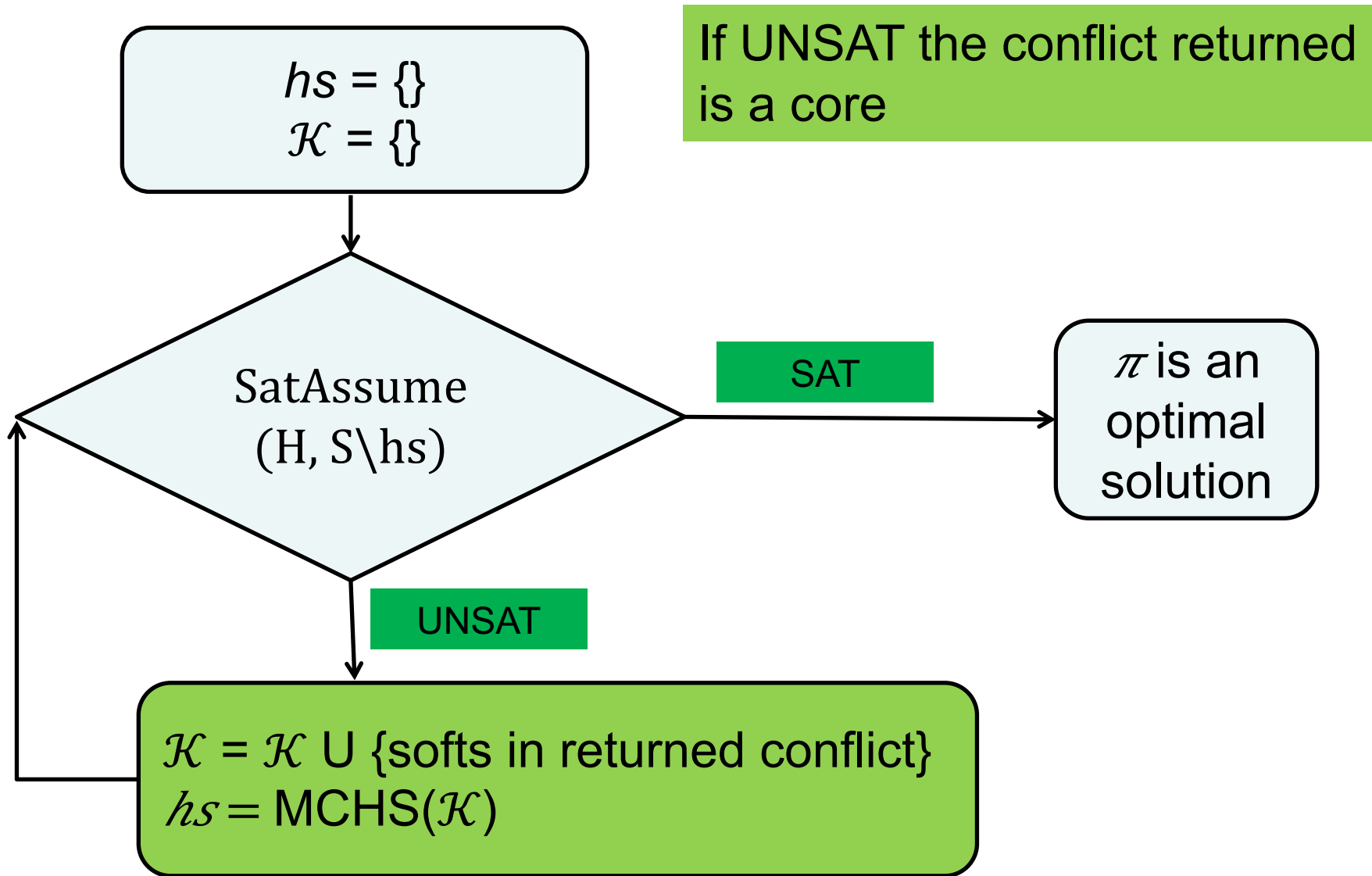
SAT

π is an
optimal
solution

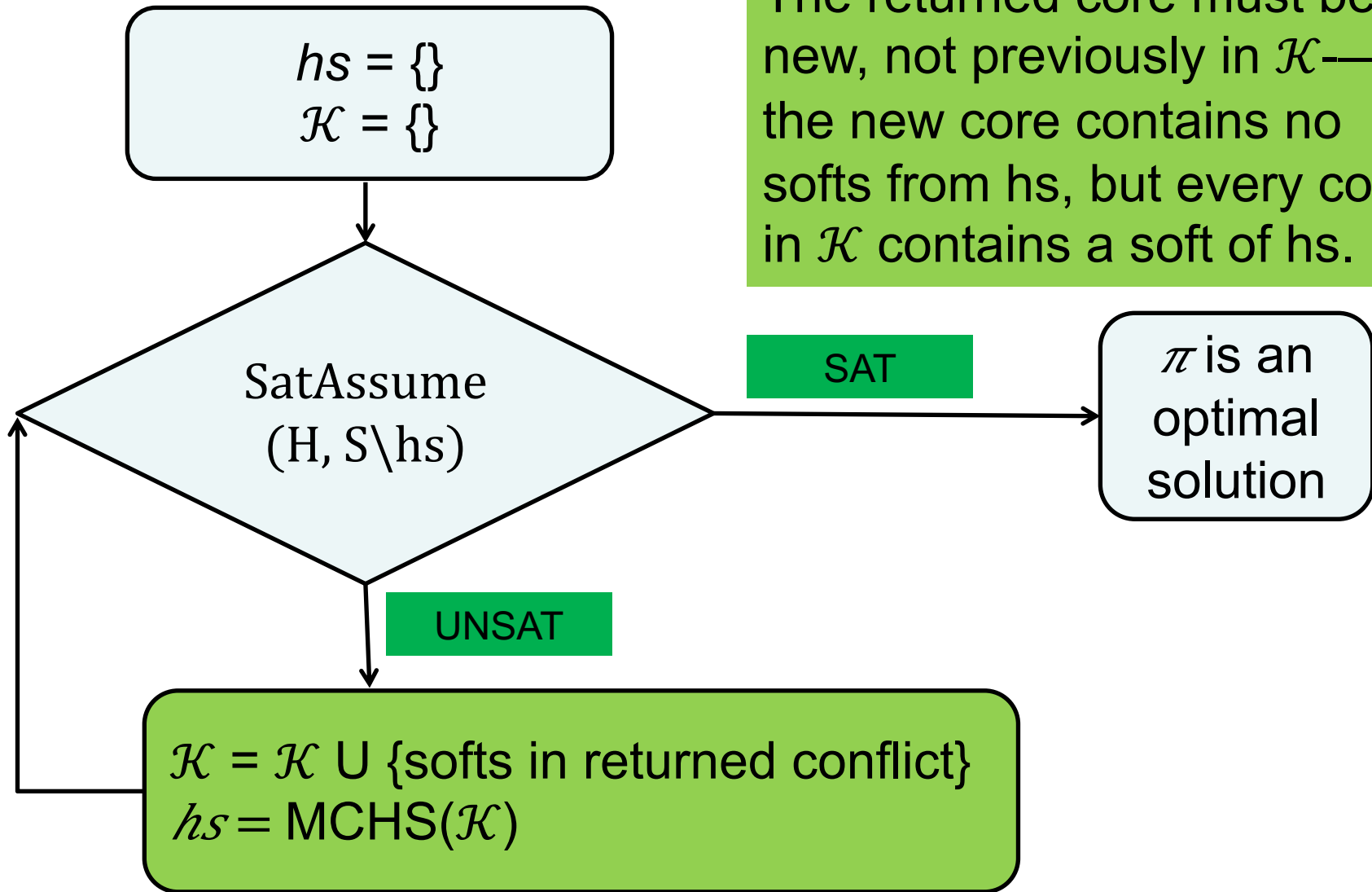
UNSAT

$\mathcal{K} = \mathcal{K} \cup \{\text{softs in returned conflict}\}$
 $hs = MCHS(\mathcal{K})$

The IHS Algorithm



The IHS Algorithm



The returned core must be new, not previously in \mathcal{K} —the new core contains no softs from hs , but every core in \mathcal{K} contains a soft of hs .

The IHS Algorithm

$hs = \{\}$
 $\mathcal{K} = \{\}$

SatAssume
($H, S \setminus hs$)

SAT

π is an
optimal
solution

UNSAT

$\mathcal{K} = \mathcal{K} \cup \{\text{softs in returned conflict}\}$
 $hs = \text{MCHS}(\mathcal{K})$

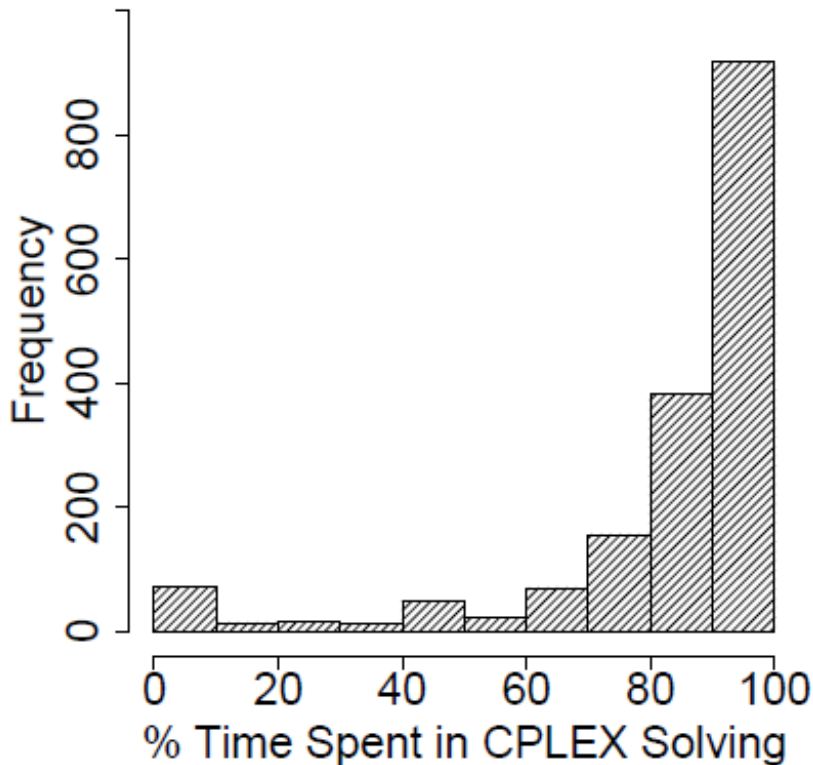
This process must terminate
as there are only a finite
number of cores.

IP solver used to compute MCHS
(when an MCHS is needed)

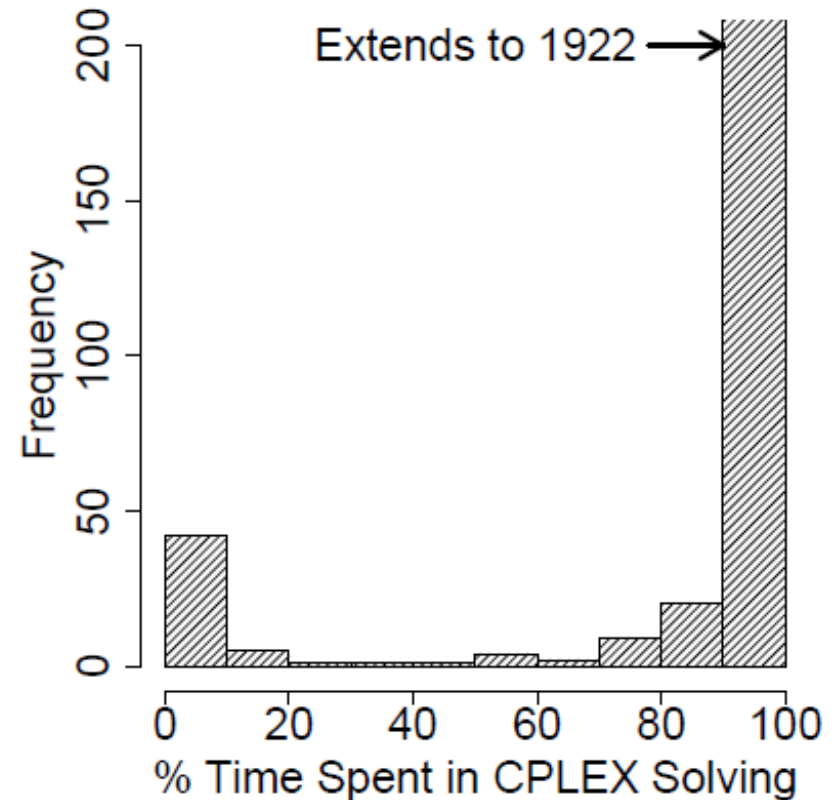
- The MCHS (aka, set-cover) problem is an NP-Hard optimization problem. But in practice it can often be solved efficiently by an integer programming solver.
 - Typically IBM's CPLEX is used
 - Seems to be the most effective way of finding an MCHS.

The IHS Algorithm

- This basic IHS algorithm is not that effective.



Solved



Unsolved

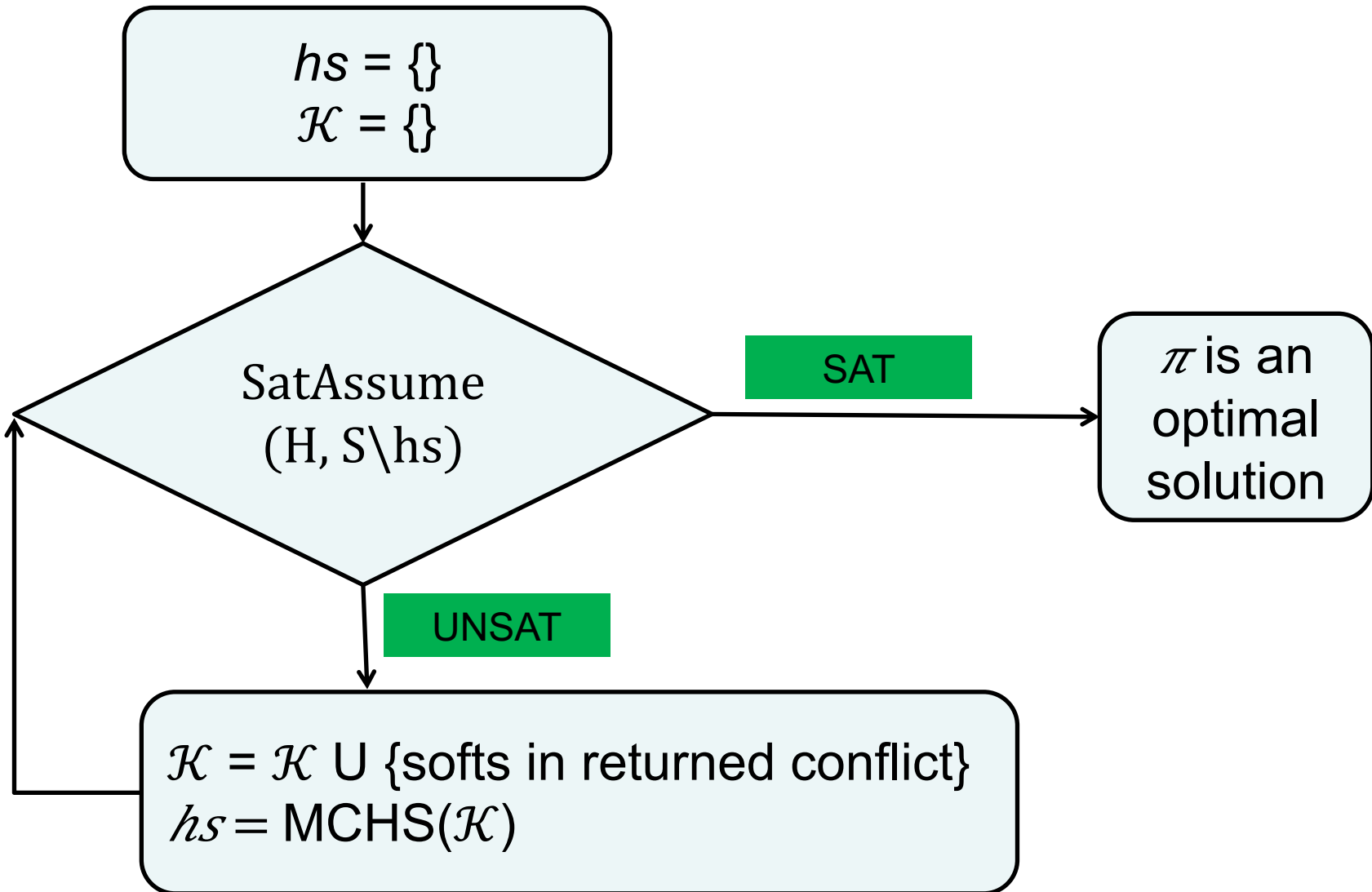
The IHS Algorithm

- The IP hitting set model is being incrementally improved by adding new cores.
- Generally many cores have to be accumulated before the IP model is strong enough to yield hitting sets whose removal yields SAT.
- Always computing an MCHS on these “too weak” IP models becomes very expensive.

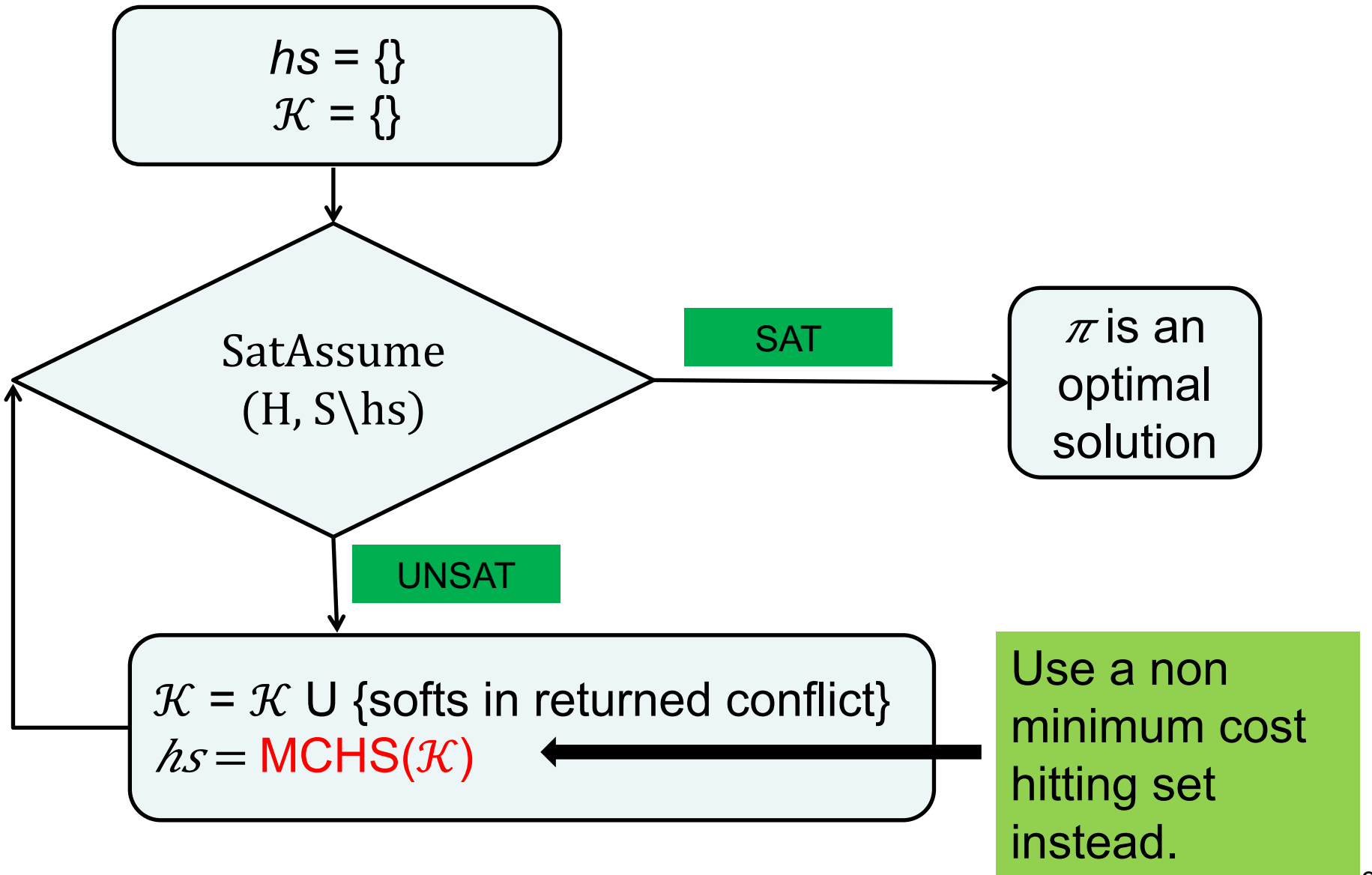
The IHS Algorithm

- Various are employed techniques to improve the IP model more quickly
- Most importantly computing an MCHS can be delayed and performed only occasionally. Much cheaper to compute non-minimum hitting sets can be used instead.
- This leads to a different formulation of IHS algorithms (this formulation is what is used in current IHS solvers).

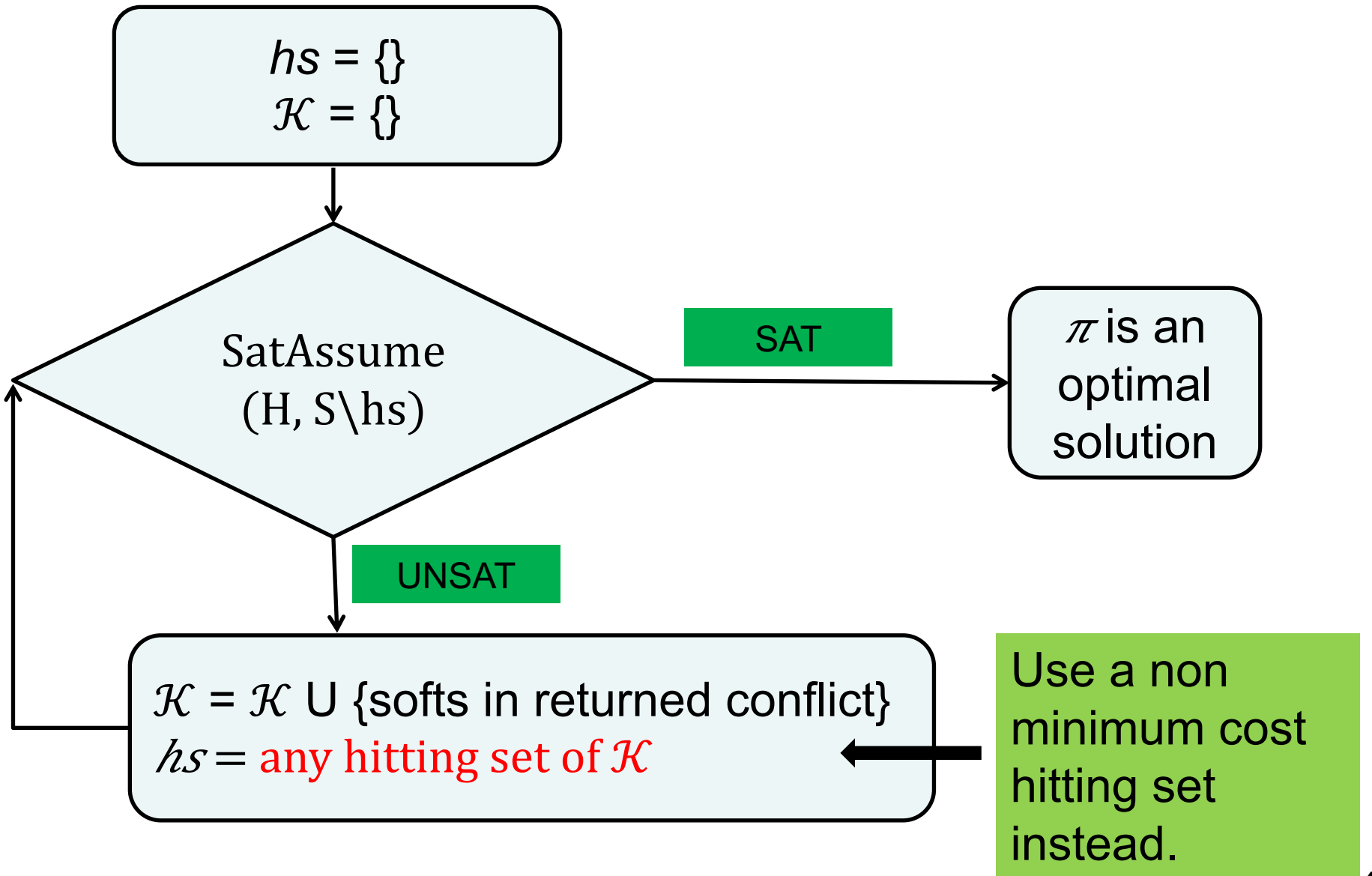
The IHS Algorithm



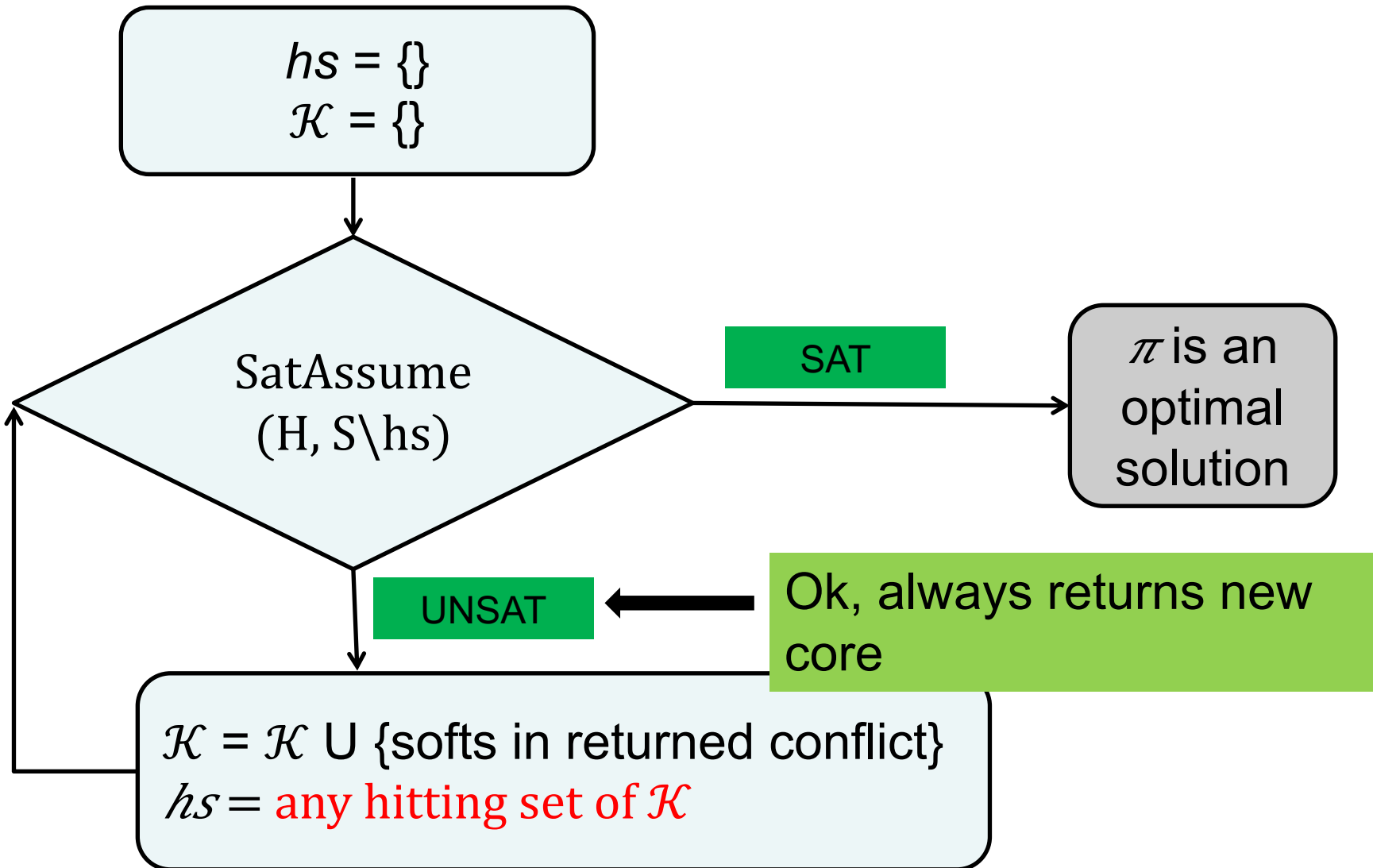
The IHS Algorithm



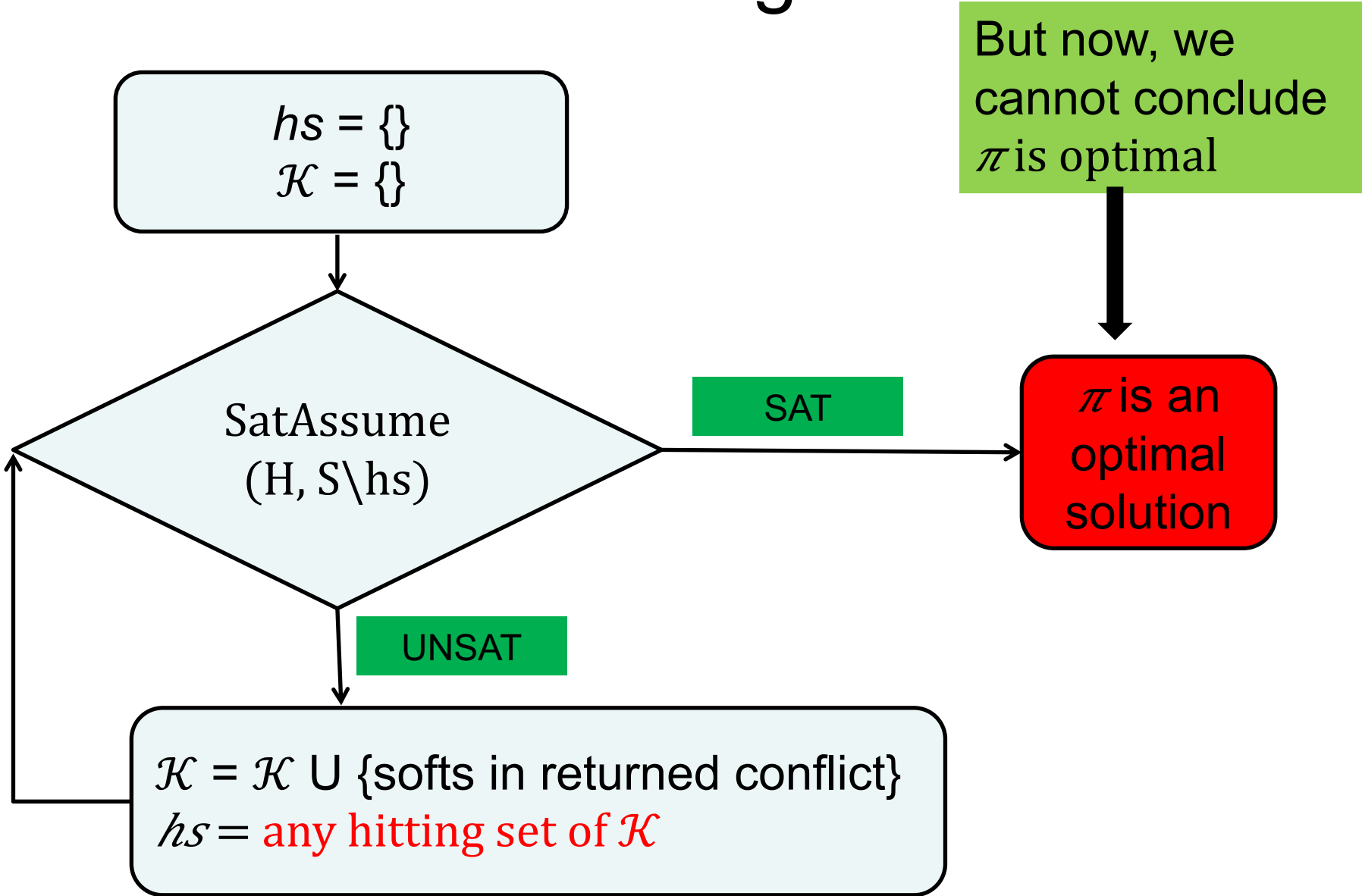
The IHS Algorithm



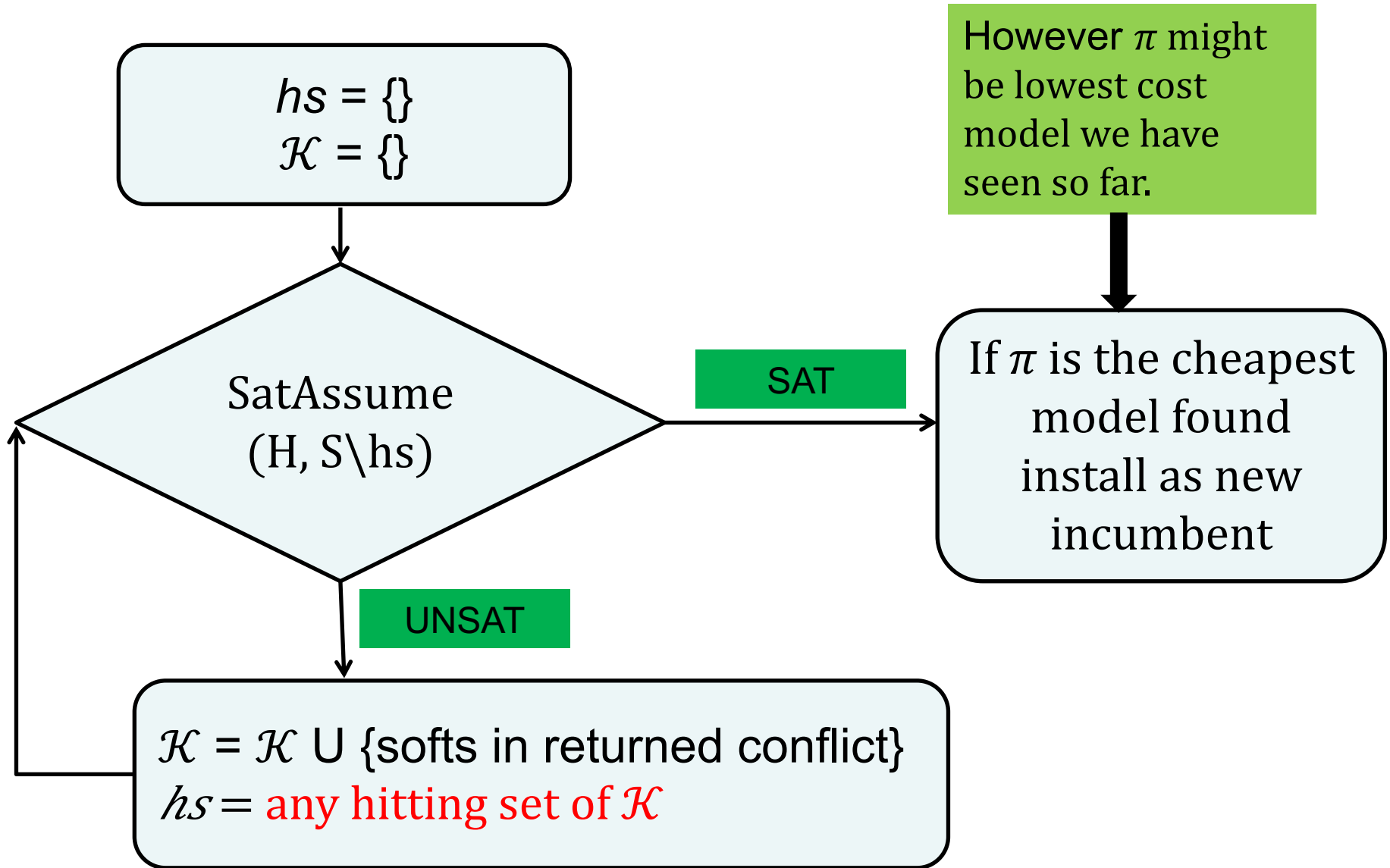
The IHS Algorithm



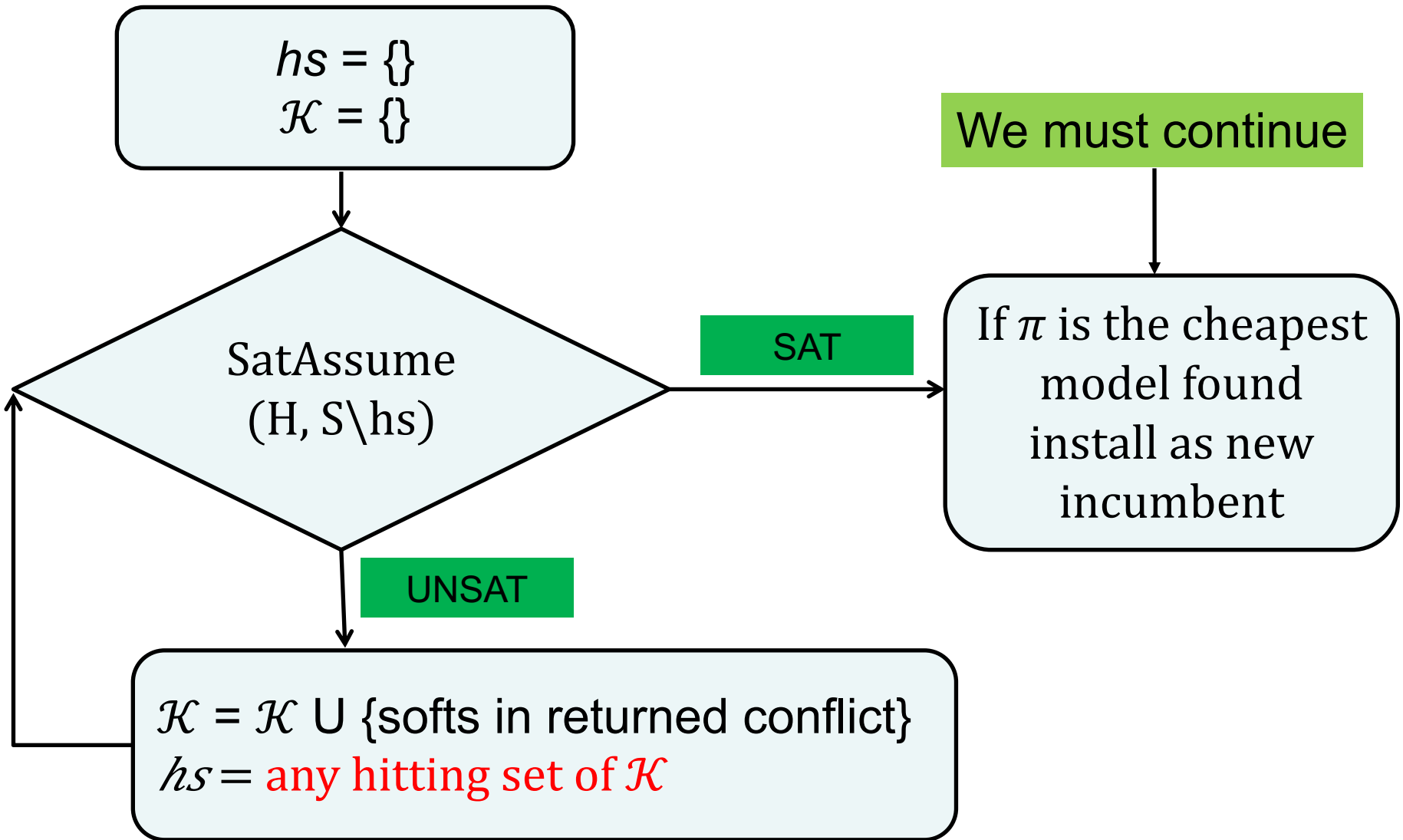
The IHS Algorithm



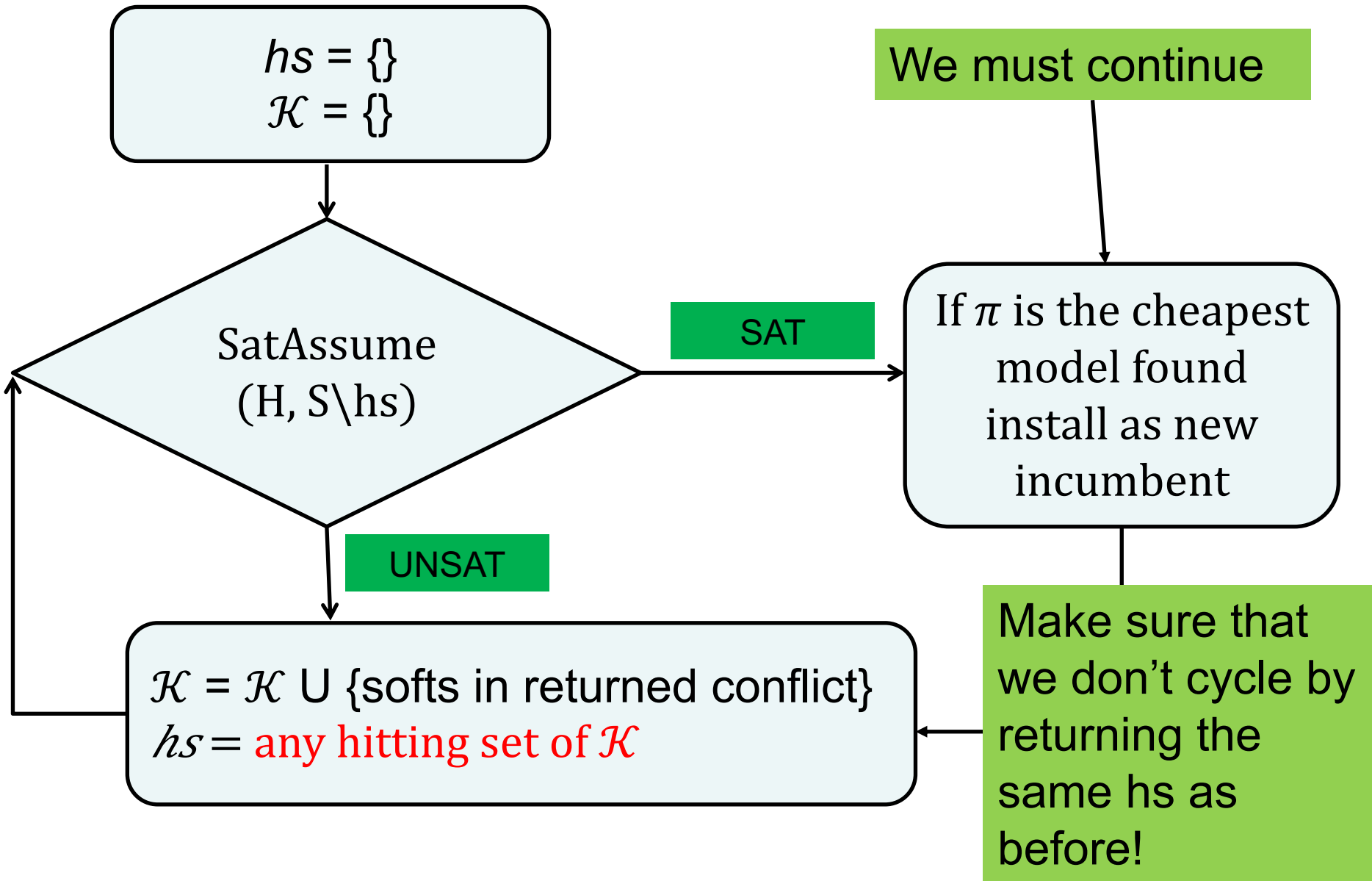
The IHS Algorithm



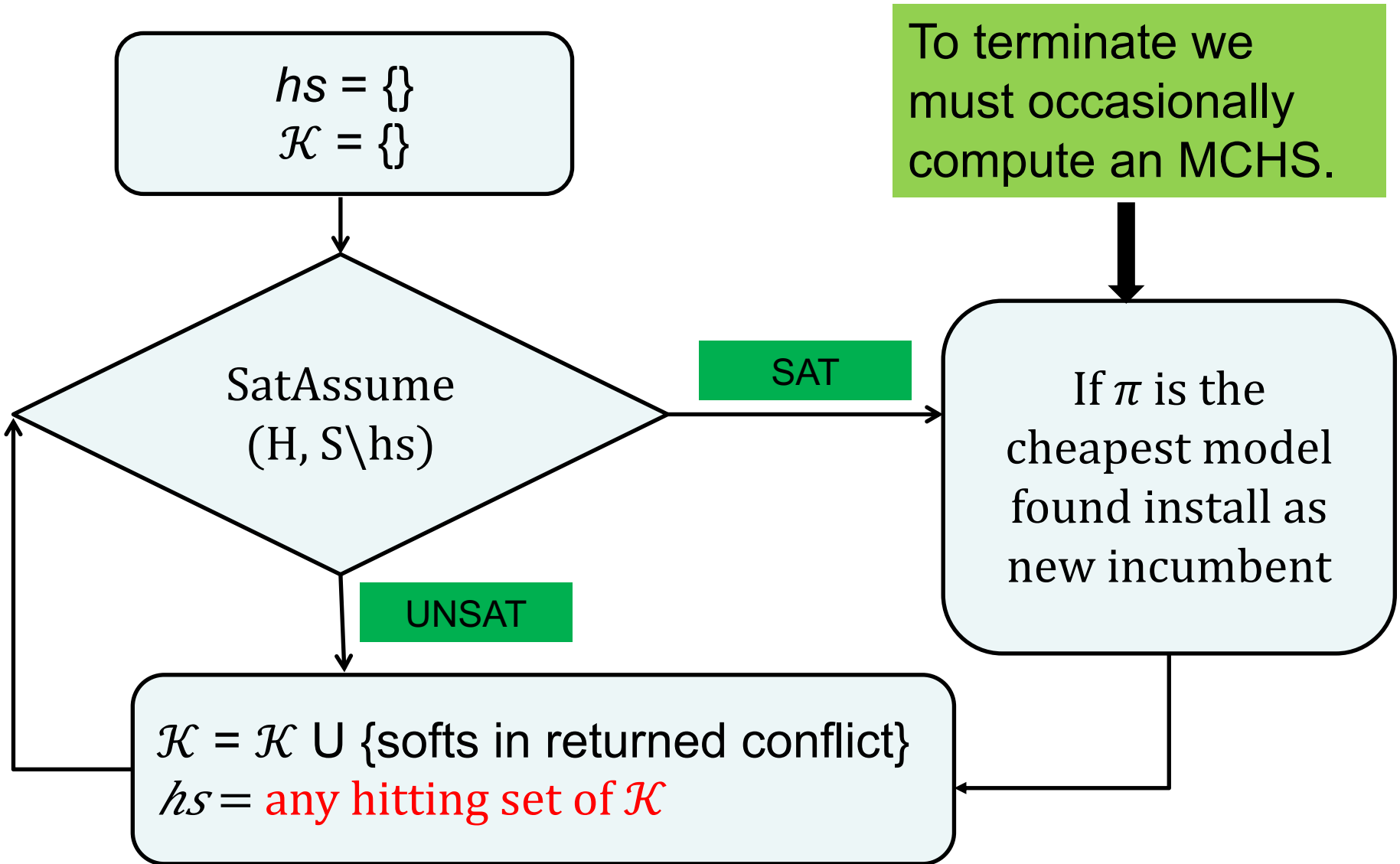
The IHS Algorithm



The IHS Algorithm



The IHS Algorithm



$hs = \{\}$
 $\mathcal{K} = \{\}$

To terminate we must occasionally compute an MCHS.

SatAssume
($H, S \setminus hs$)

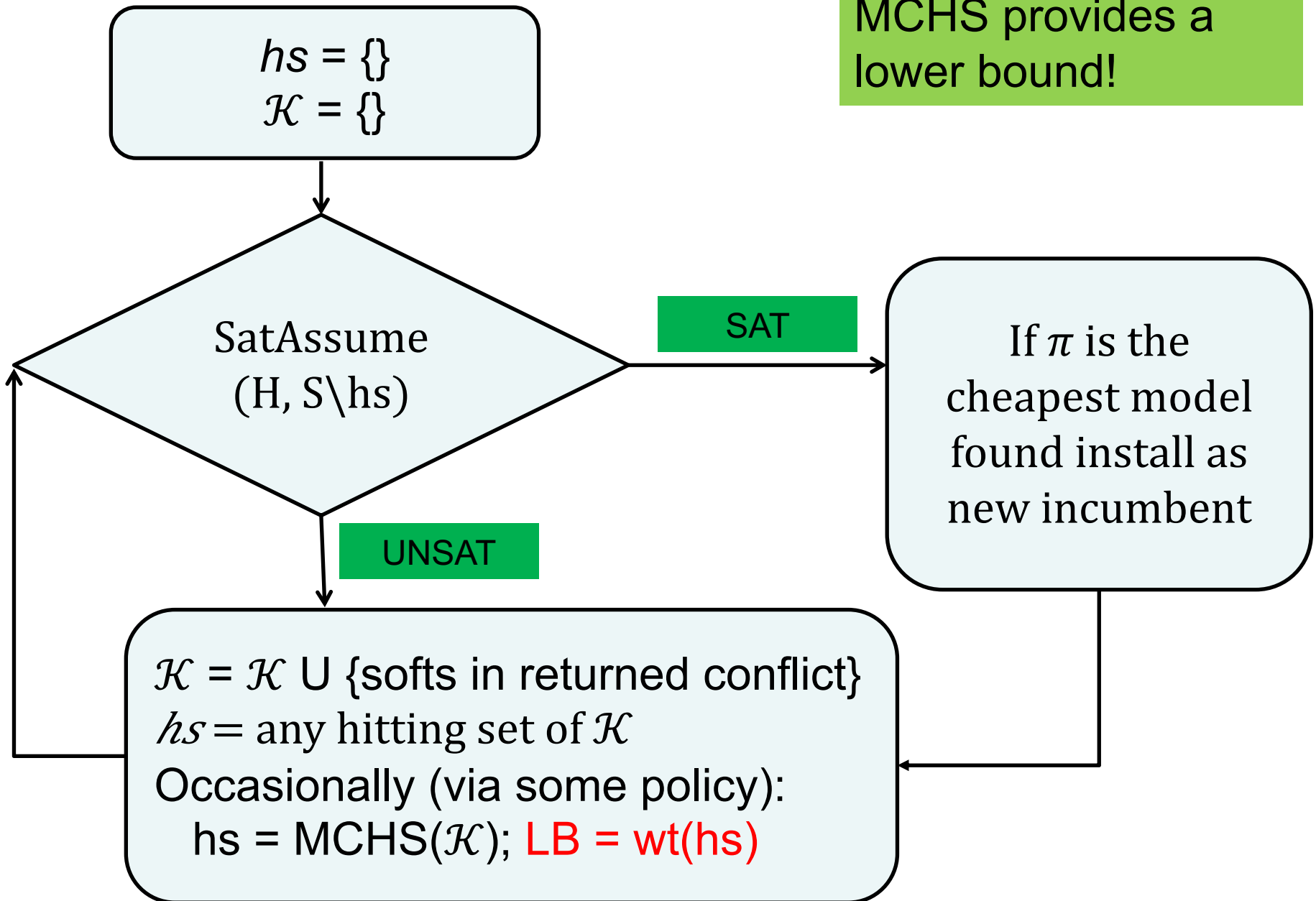
SAT

If π is the
cheapest model
found install as
new incumbent

UNSAT

$\mathcal{K} = \mathcal{K} \cup \{\text{softs in returned conflict}\}$
 $hs = \text{any hitting set of } \mathcal{K}$
Occasionally (via some policy):
 $hs = \text{MCHS}(\mathcal{K})$

MCHS provides a lower bound!



$hs = \{\}$
 $\mathcal{K} = \{\}$

Lower bound meets
upper bound becomes
new termination
condition.

SatAssume
($H, S \setminus hs$)

SAT

If π is the cheapest
model found install as
new incumbent.

If $LB \geq$
 $cost(incumbent)$ return
incumbent

UNSAT

$\mathcal{K} = \mathcal{K} \cup \{\text{softs in returned conflict}\}$

$hs = \text{any hitting set of } \mathcal{K}$

Occasionally (via some policy):

$hs = \text{MCHS}(\mathcal{K}); LB = wt(hs)$

If $LB \geq cost(incumbent)$ return incumbent

IHS Algorithm

- As long as computing an MCHS is never “starved” (i.e., always eventually we compute the MCHS) the algorithm must terminate.
- Maintaining an UB model also allows the IP technique of reduced cost fixing to be exploited
 - Fahiem Bacchus, Antti Hyttinen, Matti Jarvisalo, and Paul Saikko; **Reduced Cost Fixing in MaxSAT, CP 2018**

Implicit Hitting Set Solvers

- The SAT solving episodes are much simpler—they involve restrictions of the original MaxSat formula rather than augmentations of that formula.
- In practice, like other CEGAR approaches, only a few thousand cores need to be generated before the MCHS lower bound meets the optimal cost.
- But there are other cases where the number of cores required is too large, making both finding them and solving the MCHS too expensive.
- Currently it is usually a more effective way of dealing with a diverse collection of weights.