MaxSAT
Current Solution Techniques

Fahiem Bacchus
University of Toronto

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

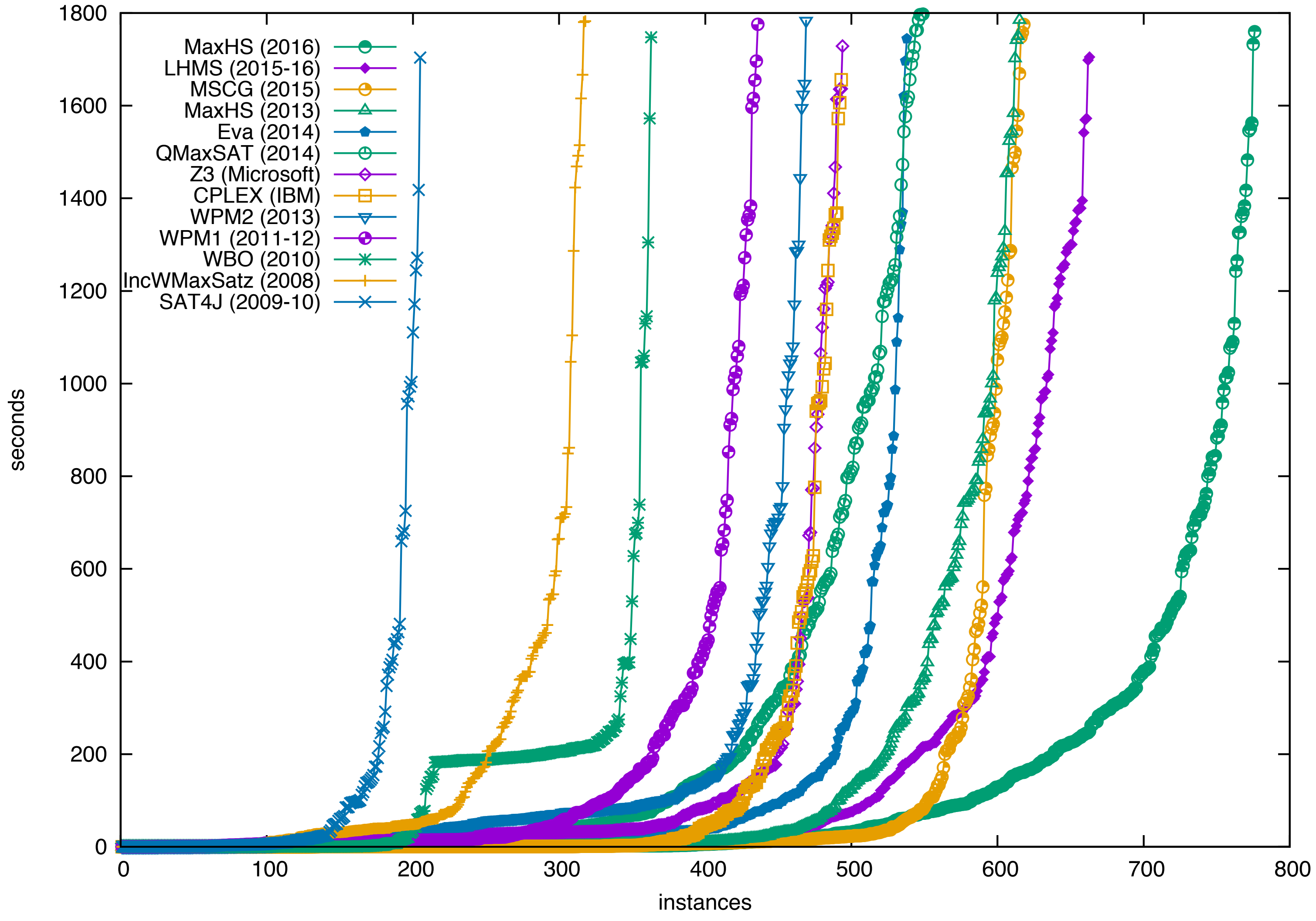
probabilistic inference	[Park, 2002]
design debugging	[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
	[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
maximum quartet consistency	[Morgado and Marques-Silva, 2010]
software package management	[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
	[Ignatiev, Janota, and Marques-Silva, 2014]
Max-Clique	[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
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,andTorres-Jiménez,2013b]
correlation clustering	[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016]
treewidth computation	[Berg and Järvisalo, 2014]
Bayesian network structure learning	[Berg, Järvisalo, and Malone, 2014]
causal discovery	[Hyttinen, Eberhardt, and Järvisalo, 2014]
visualization	[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
model-based diagnosis	[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
cutting planes for IPs	[Saikko, Malone, and Järvisalo, 2015]
argumentation dynamics	[Wallner, Niskanen, and Järvisalo, 2016]

Annual MaxSat Evaluations

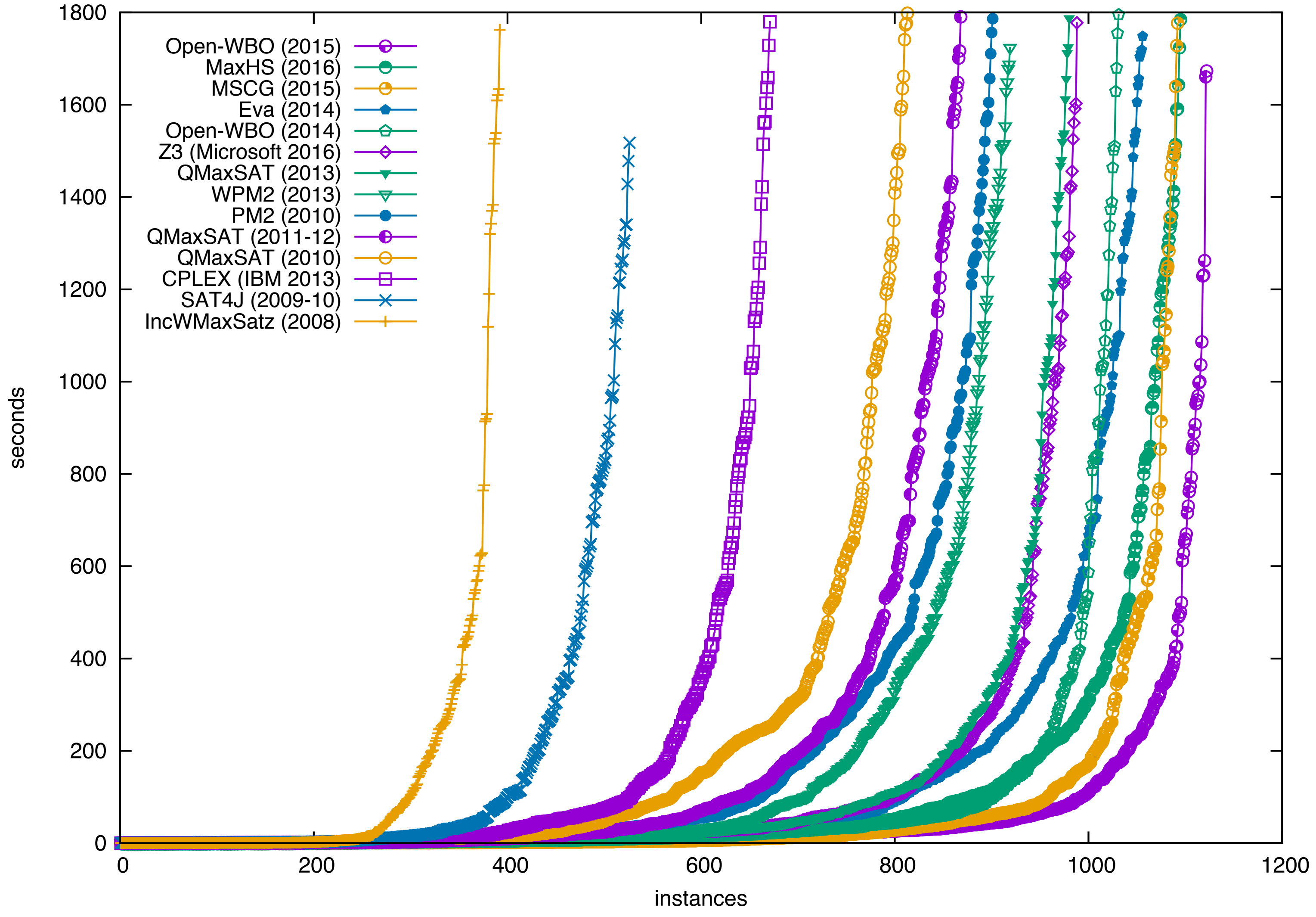
- Each year we receive new maxsat benchmarks in which maxsat is being used to solve new problems.
  - Not surprising as many problems involve optimization.

- The improvement in performance in MaxSat solvers has been impressive.

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

Outline

- Formalizing the MaxSat problem and some necessary notation.
- Solvers that use SAT to solve a sequence of relaxations.
- Implicit hitting set solvers that exploit both SAT and Integer Programming technology.

Necessary Formal Notions

MaxSat

- A MaxSat instance **F** is a CNF where the clauses are partitioned into HARD clauses and SOFT clauses:
  F = **H** ∪ **S**
- Associated with each soft clause c ∈ **S** is a integer weight **wt(c) > 0.**
  - Some solvers can handle floating point weights.
- For any set of soft clauses A ⊆ **S**, wt(A) is the sum of the weights of the clauses in A.

$$\mathrm{wt(A)} = \sum_{c\, \in A} wt(c)$$

MaxSat

- A truth assignment π that satisfies **H** is called a **feasible solution**: $\boldsymbol{\pi} \vDash \mathbf{H}$
- **Cost** of a feasible solution π is the weight of the set of soft clauses it falsifies.
$$cost(\pi) = \sum_{\pi \nvDash c} wt(c)$$
- A feasible solution is an **optimal solution** if it has minimum cost amongst all feasible solutions
$$\forall \boldsymbol{\sigma}.\, \boldsymbol{\sigma} \vDash \mathbf{H} \longrightarrow cost(\boldsymbol{\sigma}) \geq \mathrm{cost}(\boldsymbol{\pi})$$

- Solving a MaxSat instance F means finding an optimal solution.

Cores

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

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

- 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.
- Currently most accessible way to do this is to use SAT with assumptions…built into most SAT solvers.
- Assumptions must be a set of literals.

- **SAT_ASSUME(H, Asmp)**
  - Sat solve the CNF H under the assumption that every literal in Asmp is true.
  - Return SAT and π if $\pi \models$ H and makes every literal in Asmp true
  - Return UNSAT and a conflict clause $(\neg l_1, \neg l_2, \ldots, \neg l_k)$ implied by H where each $l_i \in$ Asmp.
    - At least one of the subset of assumptions $\{l_1, l_2, \ldots, l_k\}$ must be falsified in every model of H.

Cores via Assumptions

- Since assumptions are limited to literals, to extract MaxSat cores we must have literals that imply the satisfaction of soft clauses
- Unit soft clauses, we can use its literal as an assumption
  - (z) we can use the literal z as an assumption.
- Non unit soft clauses **C**, we need to create a new literal b, add to the hard clauses the condition $b \rightarrow C$, and then use b as an assumption.
  - $(x, y, z) \rightarrow (\neg b, x, y, z)$
  - $H = H \cup \{(\neg b, x, y, z)\}$
  - Use b as an assumption

Conversion to Unit Softs

- In fact this “getting the formula ready for using assumptions” transformation is the same as converting it to a new MaxSat instance which has **only unit soft clauses**.

- $F = H \cup S \Rightarrow F^b = H^b \cup S^b$
- Let $S^{>1} = \{c_i \mid c_i \in S \text{ and is non-unit}\}$
- $H^b = H \cup \{ (c_i \vee \neg b_i) \mid c_i \in S^{>1}\}$ with new variable b_i
  - Make non-unit softs into assumable hard clause
- $S = (S \setminus S^{>1}) \cup \{(b_i) \mid c_i \in S^{>1}\}$
  - Add assumption literals as softs.

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 of F^b with the same cost.
  - For each new variable b^i in F^b set b^i to TRUE iff $\pi \models c^i$

- Every feasible model of F^b, π^b restricted to the variables of F is a feasible model π of F. cost(π) $\leq$ cost(π^b)

- **Hence, an optimal model of F^b restricted to the variables of F is an optimal model of F.**

Cores via Assumptions with unit softs.

- With F^b we can easily use assumptions to extract cores.

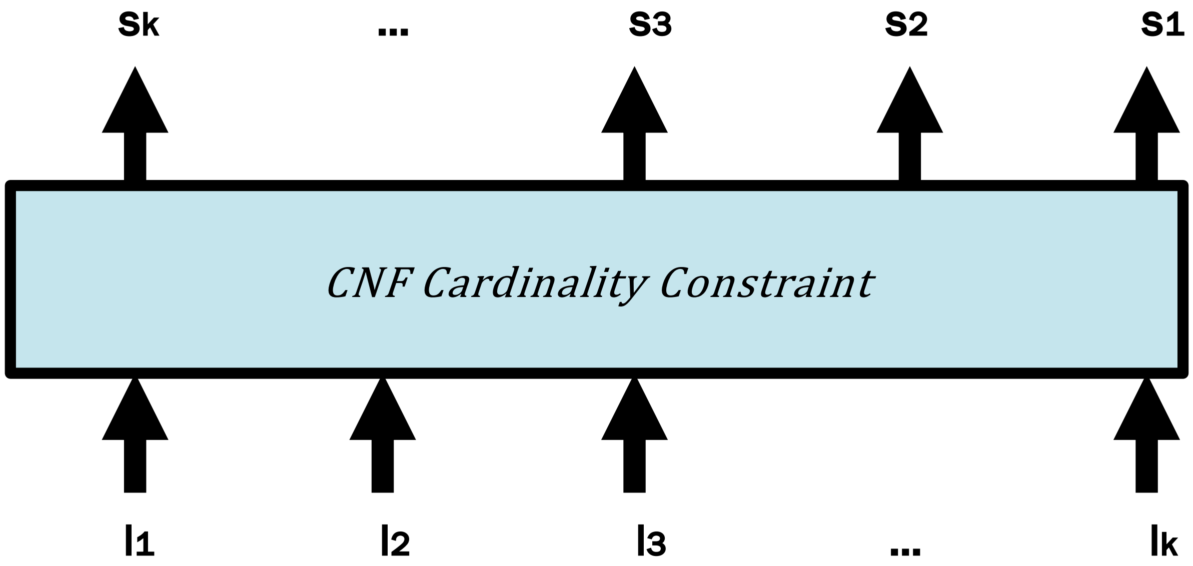
- **SAT_ASSUME(H^b, Asmp)**
  where Asmp are the literals of a set of unit softs.
- If UNSAT, we get a conflict clause $(\neg l_1, \neg l_2, \ldots, \neg l_k)$ where each $\neg l_i$ falsifies the unit soft (l_i)
- The negation of literals in this conflict $\{l_1, l_2, \ldots, l_k\}$ are a **core**.

Unhiding the use of Assumptions

- Note that most MaxSat algorithms in the literature are specified by abstracting away from the assumption mechanisms.
  - An interface with the SAT solvers is assumed that returns cores in a black box manner.
- But I think this can make some of the details these algorithms a bit more difficult to understand.
- So in this talk explain these algorithms in terms of assumptions to the SAT solver.

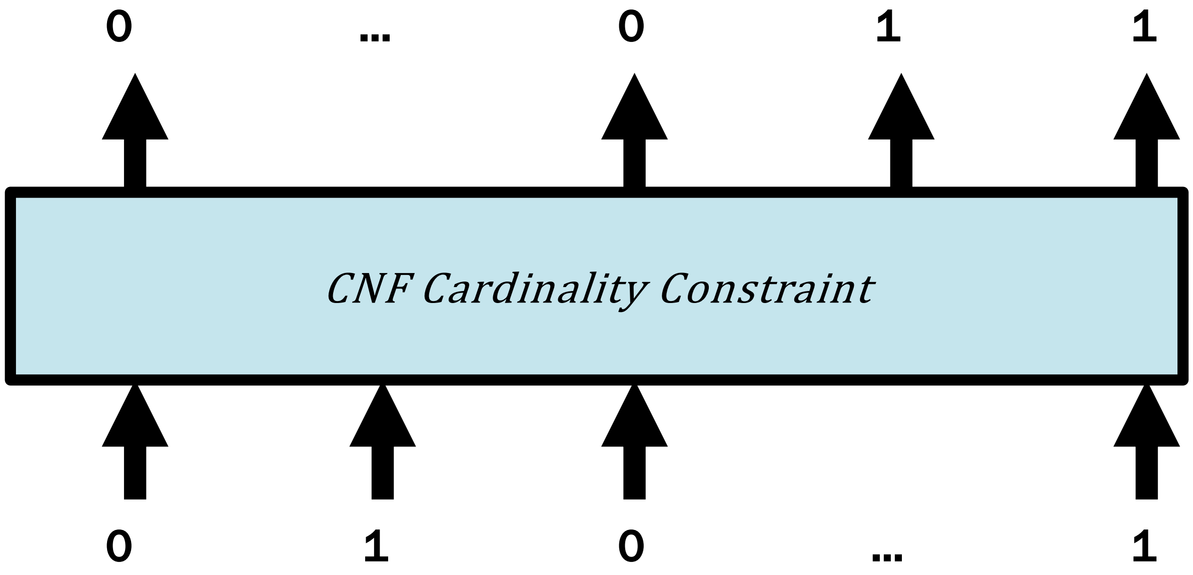
Summation Circuits

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



Summation Circuits

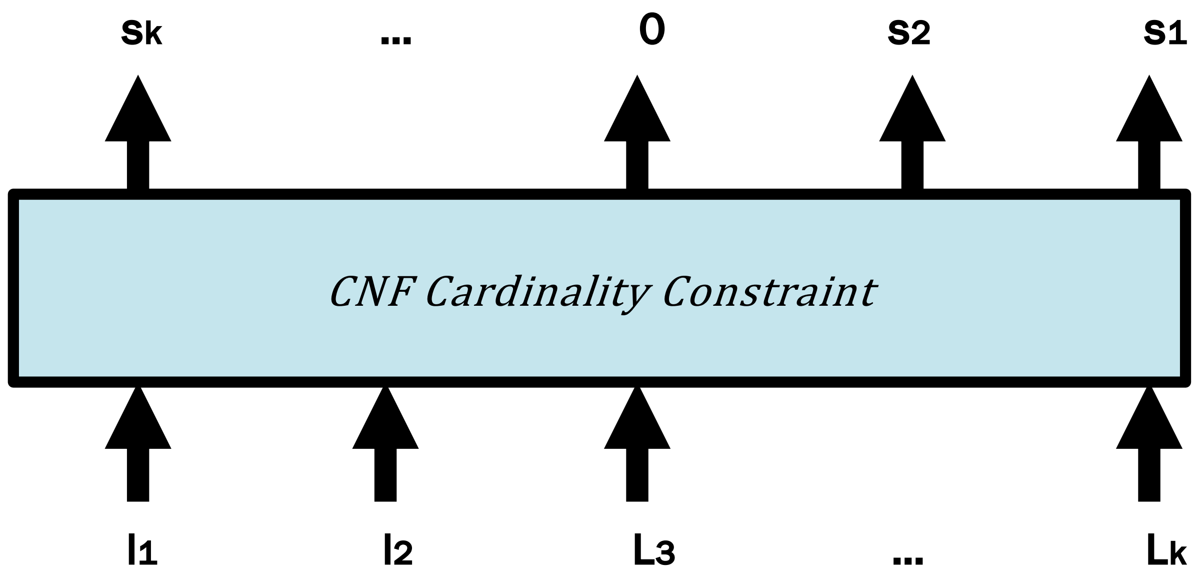
- The output is a string of 0's followed by 1's with (as many 1's as there are true inputs).
- The clauses of the summation circuit ensure that
  - $s_i \leftrightarrow l_1 + l_2 \ldots + l_k \geq i$
  - $\neg s_i \leftrightarrow l_1 + l_2 \ldots + l_k < i$



Cardinality Constraints

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

$\sum l_i < 3 =$



Solving MaxSat by SAT solving a sequence of relaxation

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, but not as well (so far)

CARDINALITY LAYER

- First we start off by ensuring that the input MaxSat **F** has only **unit soft clauses.** (Using the **F^b** conversion discussed before).
- Then the needed relaxations are achieved by adding a **CARDINALITY LAYER** to the **hard clauses of F**.
- The **inputs** of the **CARDINALITY LAYER** are the **negations** of the unit soft clause literals--limiting how many of these that can be true limits how 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

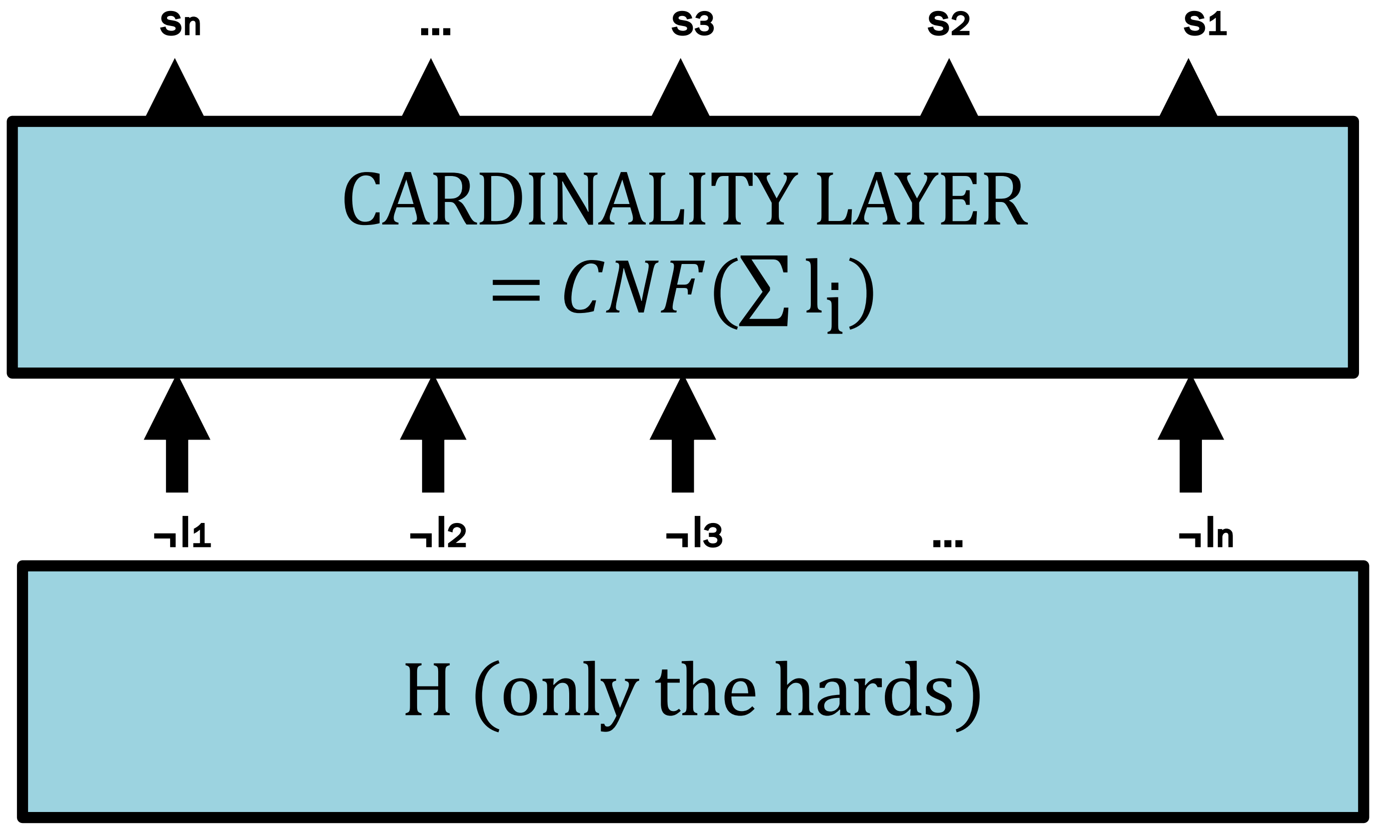
- Standard reduction of MaxSat to a sequence of SAT problems
  - Is the formula satisfiable falsifying 0 softs?
  - Is the formula satisfiable falsifying 1 soft?
  - …
  - Is the formula satisfiable falsifying k softs?

- 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

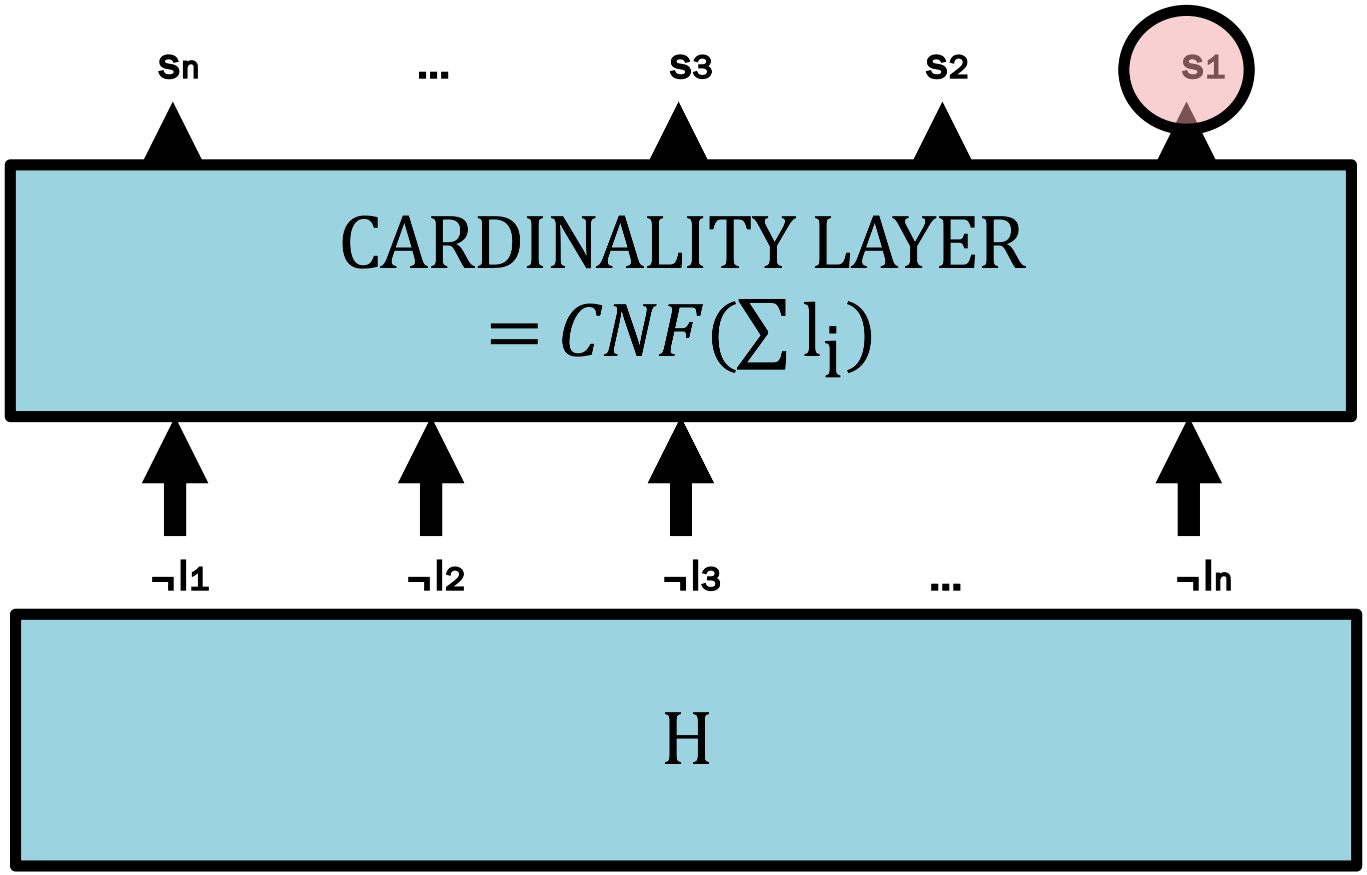
F = H U S

S = { (l_1), (l_2), …, (l_n) }



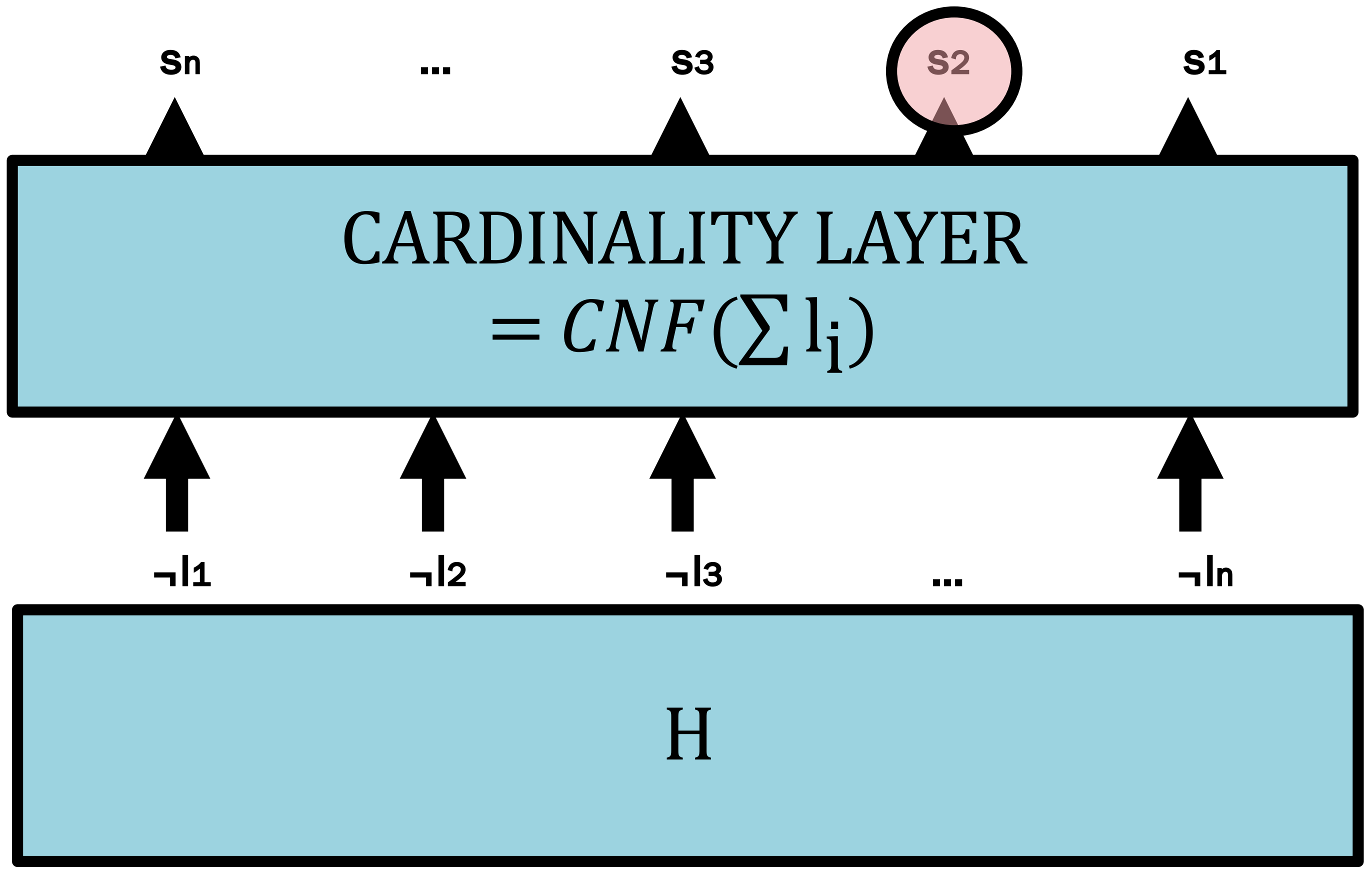
Cardinality Layer Simple Example

On the i-th iteration solve SAT_ASSUME(H U Card,¬s_i)



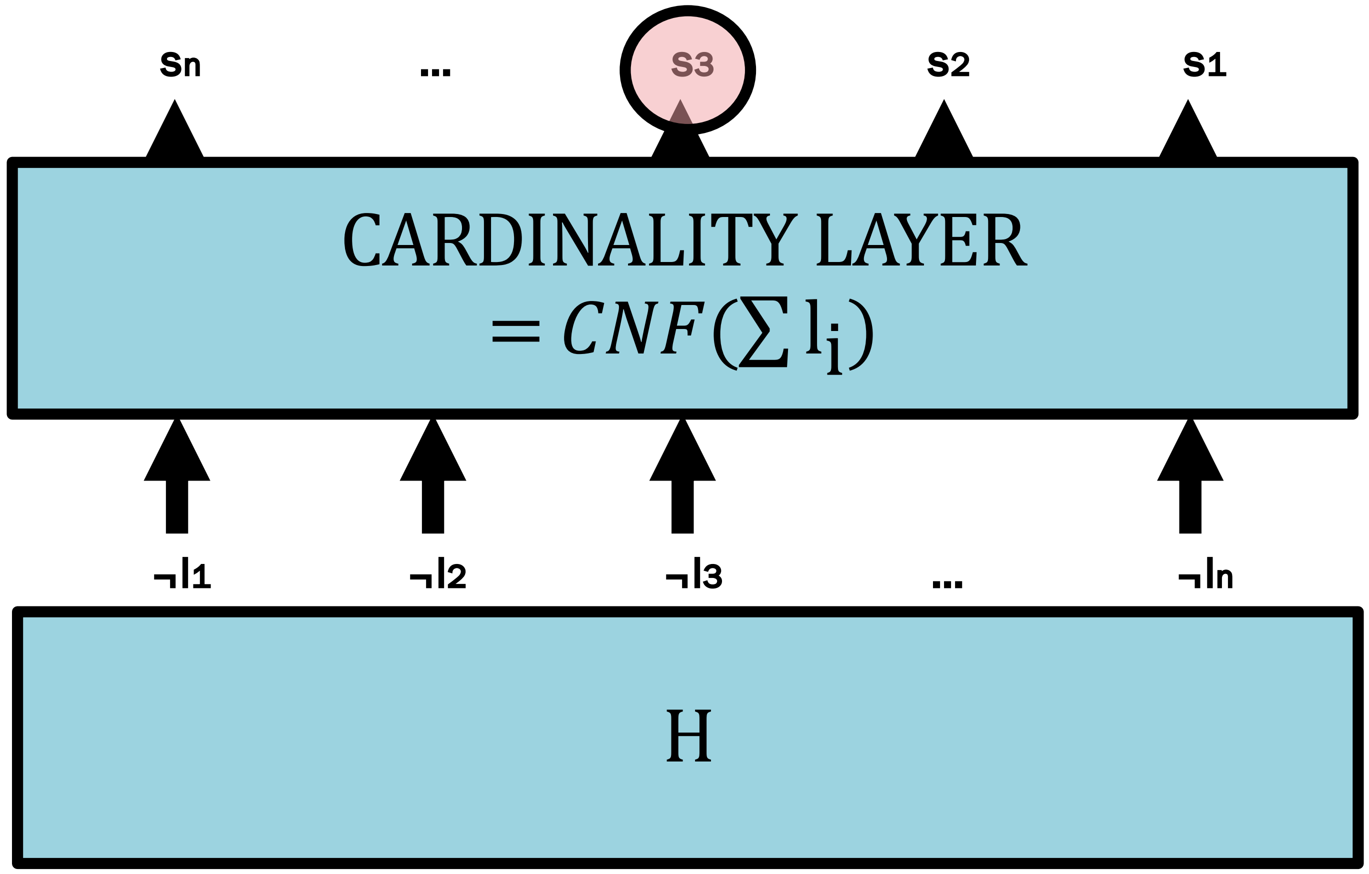
Cardinality Layer Simple Example

On the i-th iteration solve SAT_ASSUME(H U Card,$\neg s_i$)



Cardinality Layer Simple Example

On the i-th iteration solve SAT_ASSUME(H U Card,$\neg s_i$)



Cardinality Layer

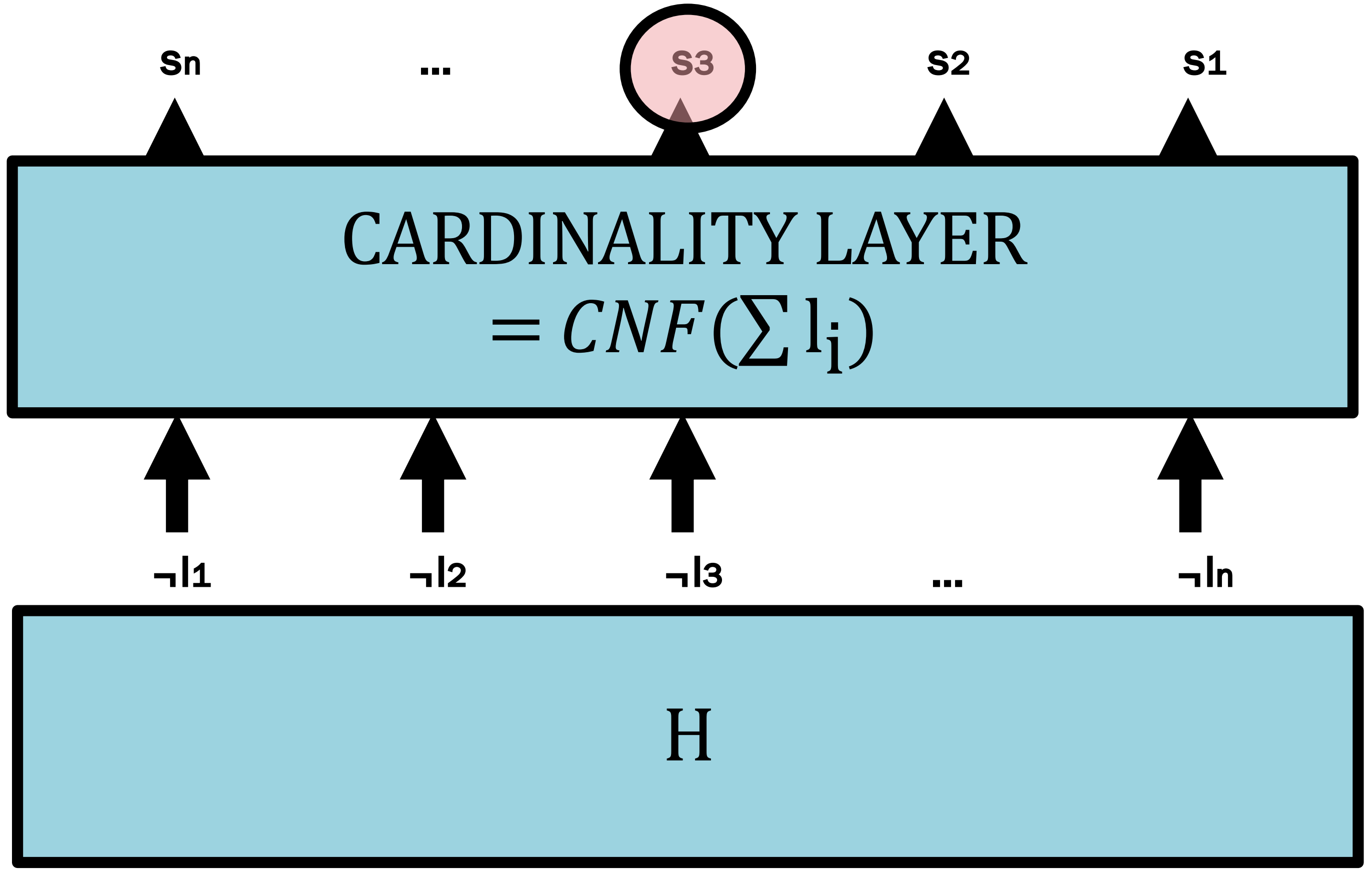
- In these algorithms the unit soft clauses are not part of the SAT solver's CNF.
  - Their literals serve as inputs to the cardinality layer.

- 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 Simple Example

Feasible settings of l_1 to l_n falsify at most 2 soft clauses

$\binom{n}{2} + \binom{n}{1} + \binom{n}{0}$ feasible settings



Cardinality Layer

- Given some assumptions A, let CARD(A) be all of the set of T/F settings of the soft clauses allowed by the cardinality layer under A.
- Since the softs are units, CARD(A) corresponds to a set of truth assignments to $\{l_1, l_2, \ldots, l_n\}$ (the literals in the SOFTS).
- To refute A, the SAT solver must refute every $\rho \in$ CARD(A). That is, the SAT solver must verify that HARDS U ρ is UNSAT for every ρ.
  - If A is refuted then he softs set to true by every ρ form a core of the input MaxSat Formula F. (Fahiem Bacchus, Nina Narodytska, Cores in Core Based MaxSat Algorithms: An Analysis. SAT 2014).

Cardinality Layer

- Clause learning can allow the SAT solver to refute entire subsets of CARD(A) rather than having to refute each $\rho \in$ 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.
- **Understanding more precisely how the structure of the Cardinality layer affects the efficiency of refuting CARD(A) is an important open question.**

MSU3 using Incremental Cardinality Constraints

- Used in OpenWBO.
- R. Martins, S. Joshi, V. Manquinho & I. Lynce
  “Incremental Cardinality Constraints for MaxSAT”
  CP-2014
- (Original MSU3) J. Marques-Silva & J. Planes. “On Using Unsatisfiablity for Maximum Satisfiablity” CoRR, 2007

MSU3 Incremental

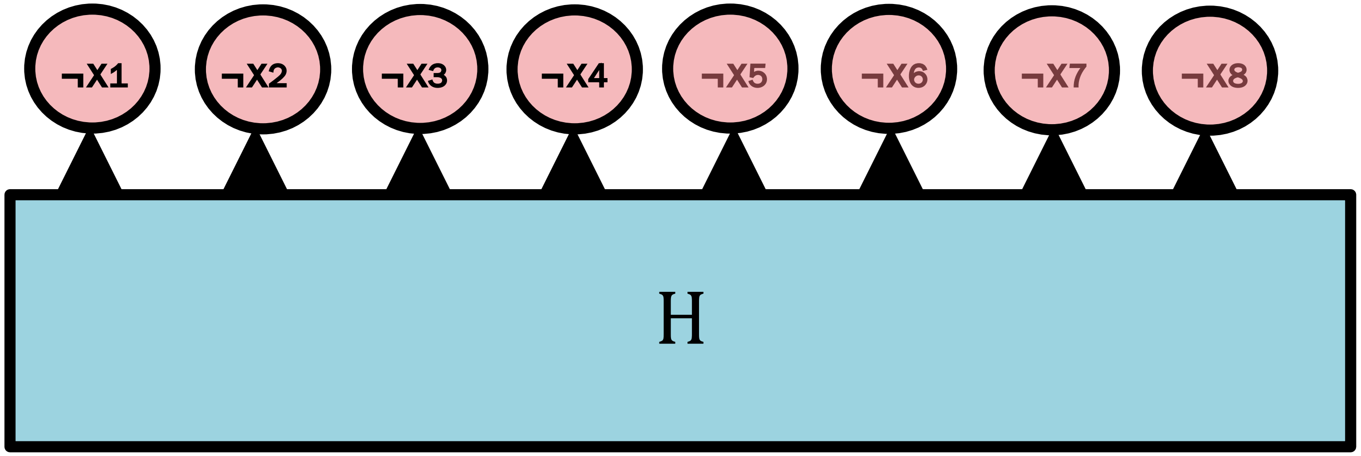
- Consider F = H U S where
- S = {(x_1), (x_2), …, (x_8)}
- H = $\sum_{i=1}^{8} x_i \leq 4$
- We want to satisfy all 8 literals x_i but at most 4 of these literals can be satisfied—at least 4 must be falsified.
- Every set of 5 or more soft clause is a core
- What will MSU3 do on this input formula?

MUS3 Incremental

Start with Cardinality Constraint Layer with inputs only

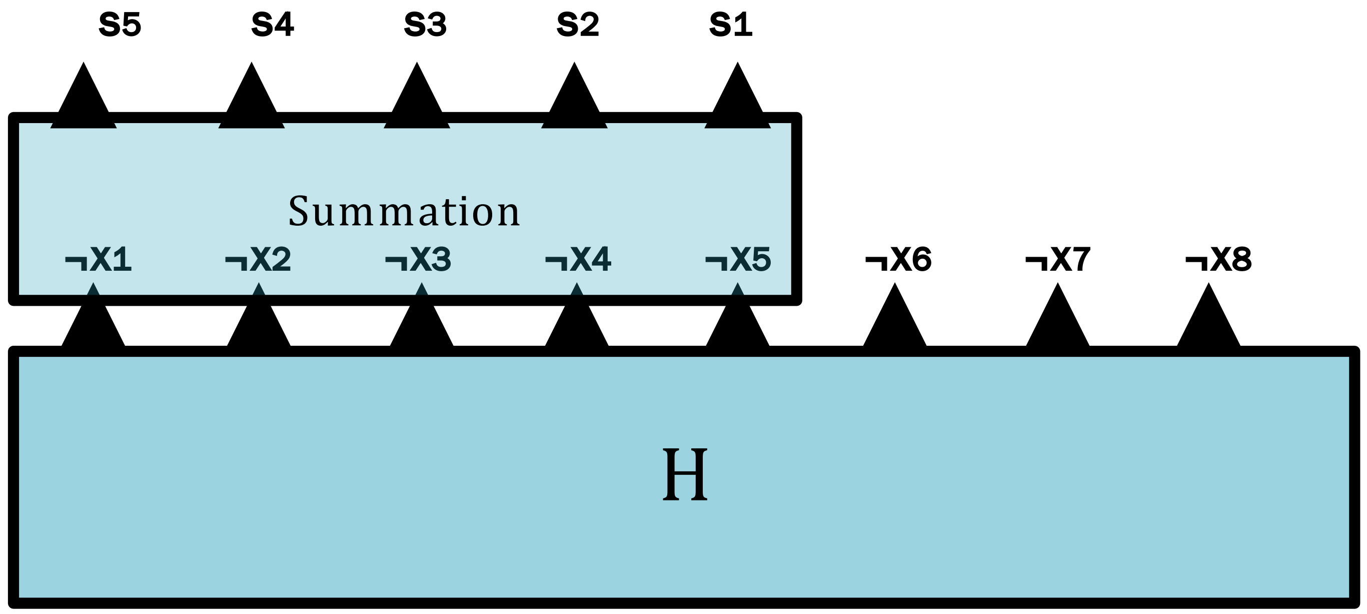
● Literals assumed FALSE

First SAT solve is SAT_ASSUME(H,{x_1,x_2,…,x_8}),
I.e., try to falsify zero softs.



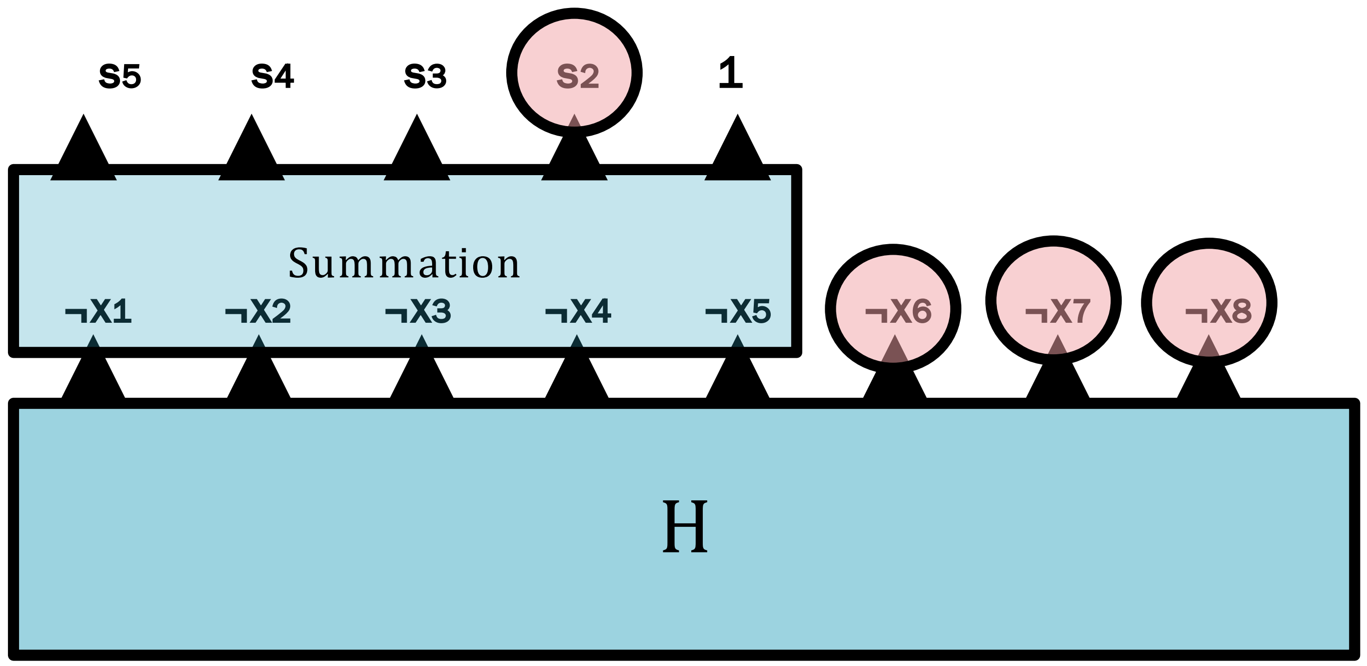
MUS3 Incremental

- UNSAT; say we get the core ($\neg x_1, \neg x_2, \neg x_3, \neg x_4, \neg x_5$)
- Add one to overall cost
- Build summation network over softs in core



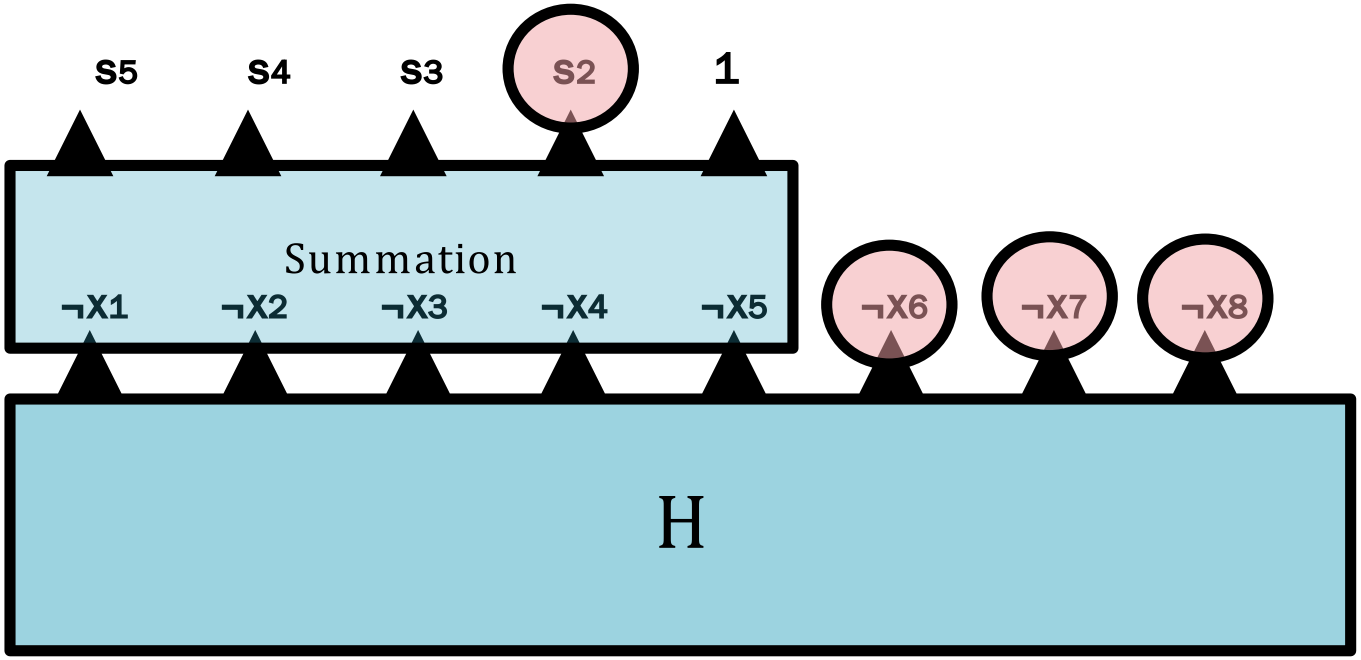
MUS3 Incremental

- We know that the sum must be at least one—set $s_1 = 1$.
- Now **SAT_ASSUME(H U Card,{$\neg s_2, x_6, x_7, x_8$}),**
- Allow one soft to be falsified among **x_1—x_5**



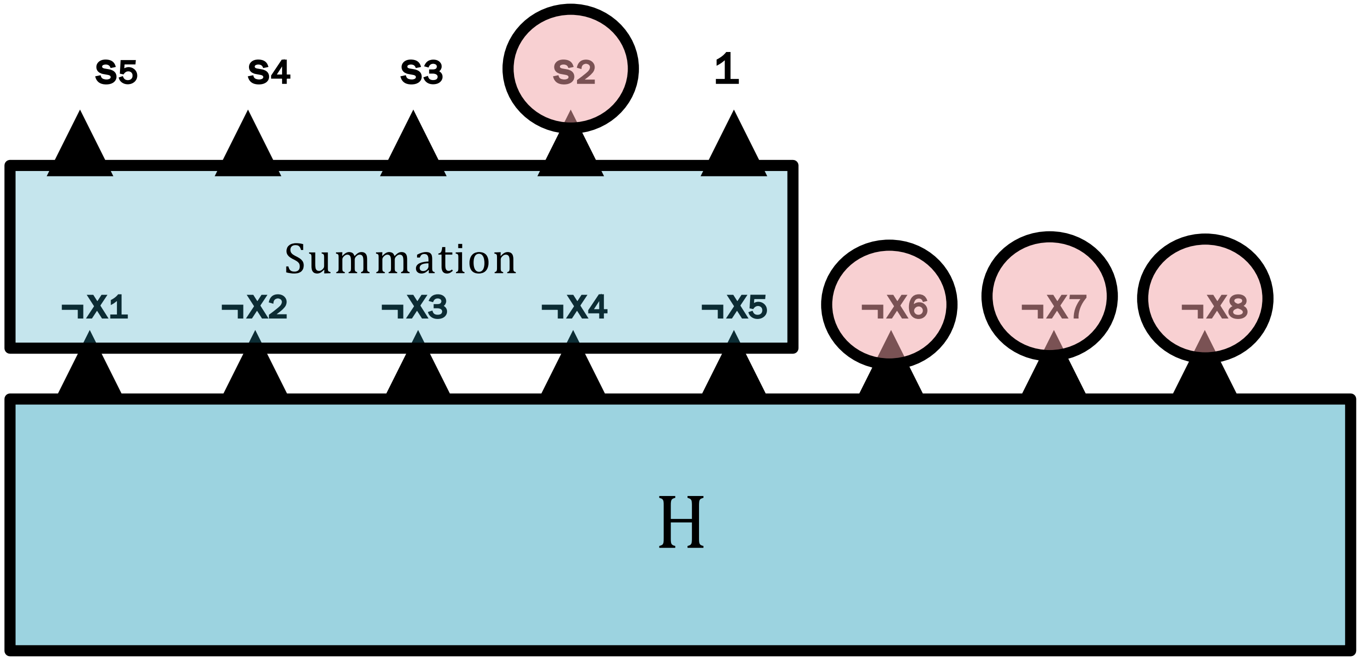
MUS3 Incremental

- Get another core, add 1 to overall cost.
- Get another conflict, $(s_2, \neg x_6)$
  - Either we must falsify two of $x_1 \ldots x_5$ or one of $x_1 \ldots x_5$ and x_6



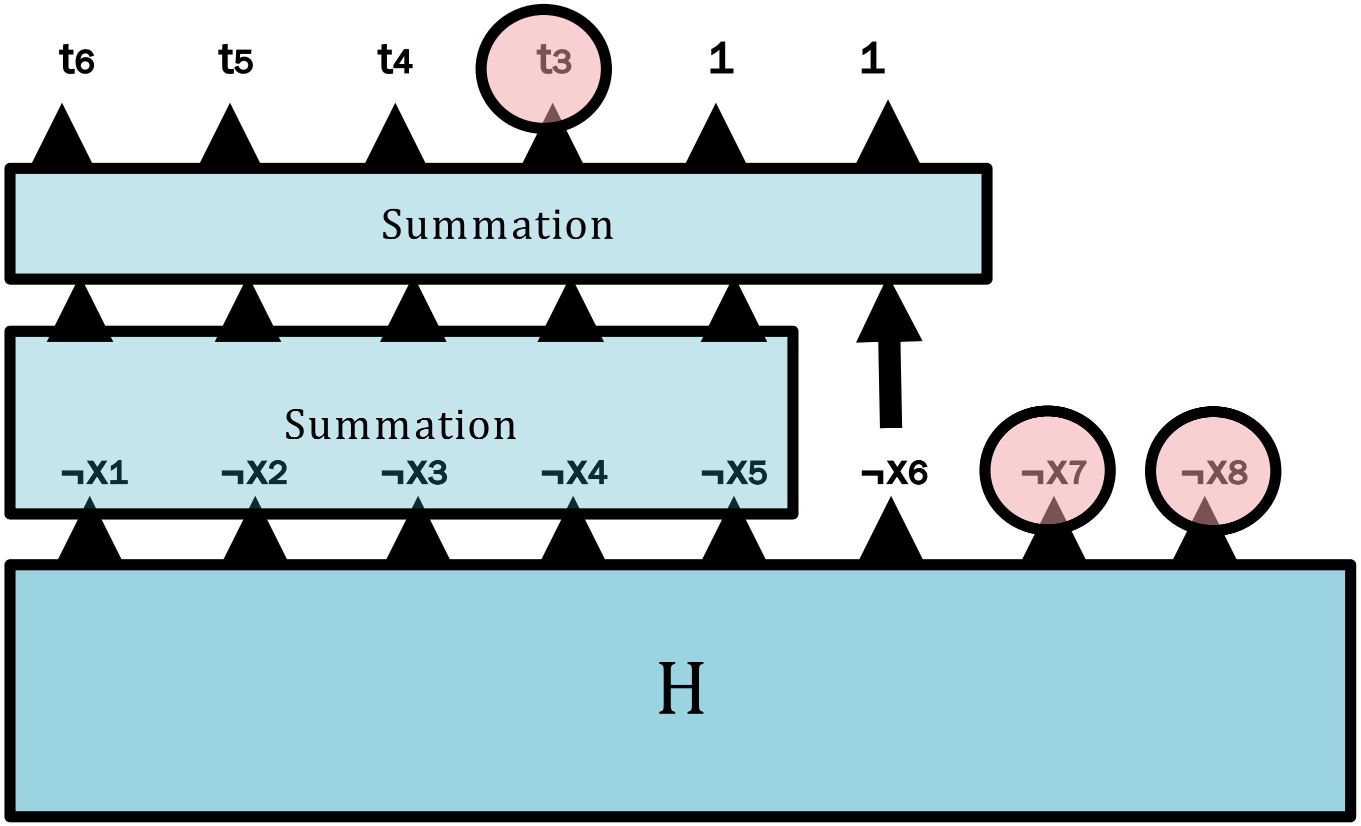
MUS3 Incremental

- 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, $\neg x_6$)



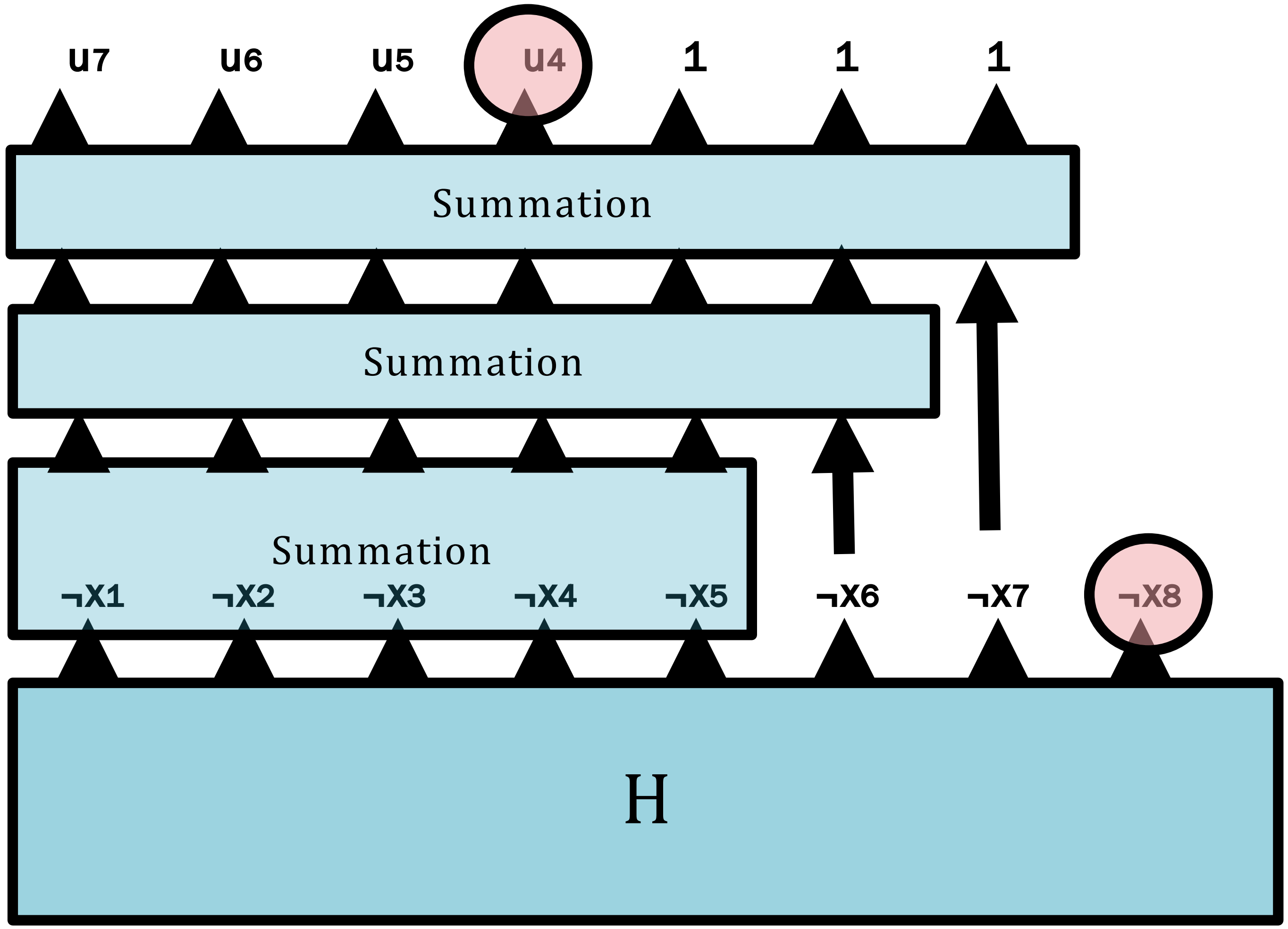
MUS3 Incremental

- We know that the sum must be at least 2—set t_1 and t_2 to 1.
- Now **SAT_ASSUME(H U Card,{¬t_3,x_7,x_8})**,
- Allow two softs to be falsified among **x_1—x_6**



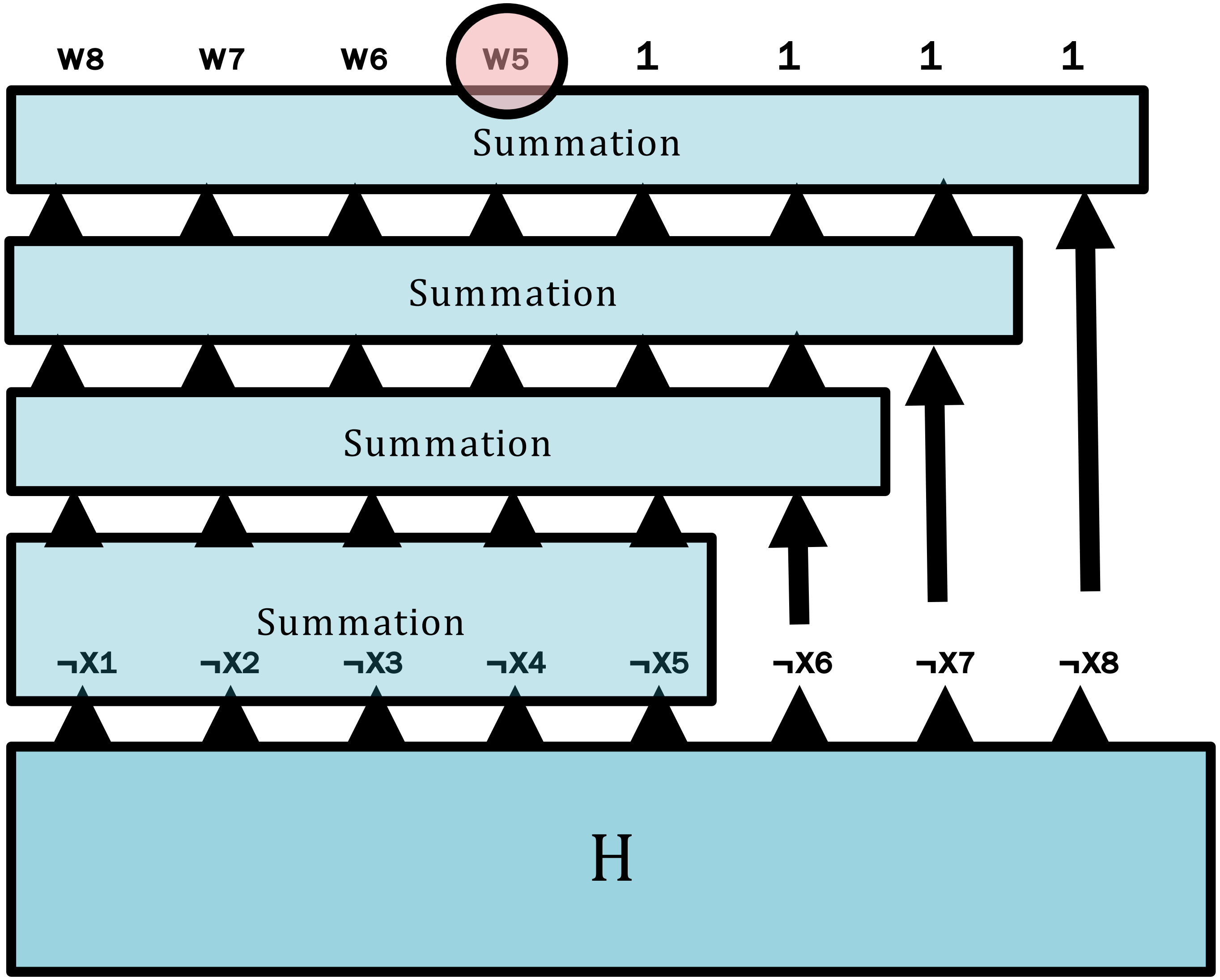
MUS3 Incremental

- Another core (**t_3, ¬x_7**), add 1 to the overall cost.
- Now **SAT_ASSUME(H U Card,{¬u_4,x_8**}), allow up to 3 falsified softs



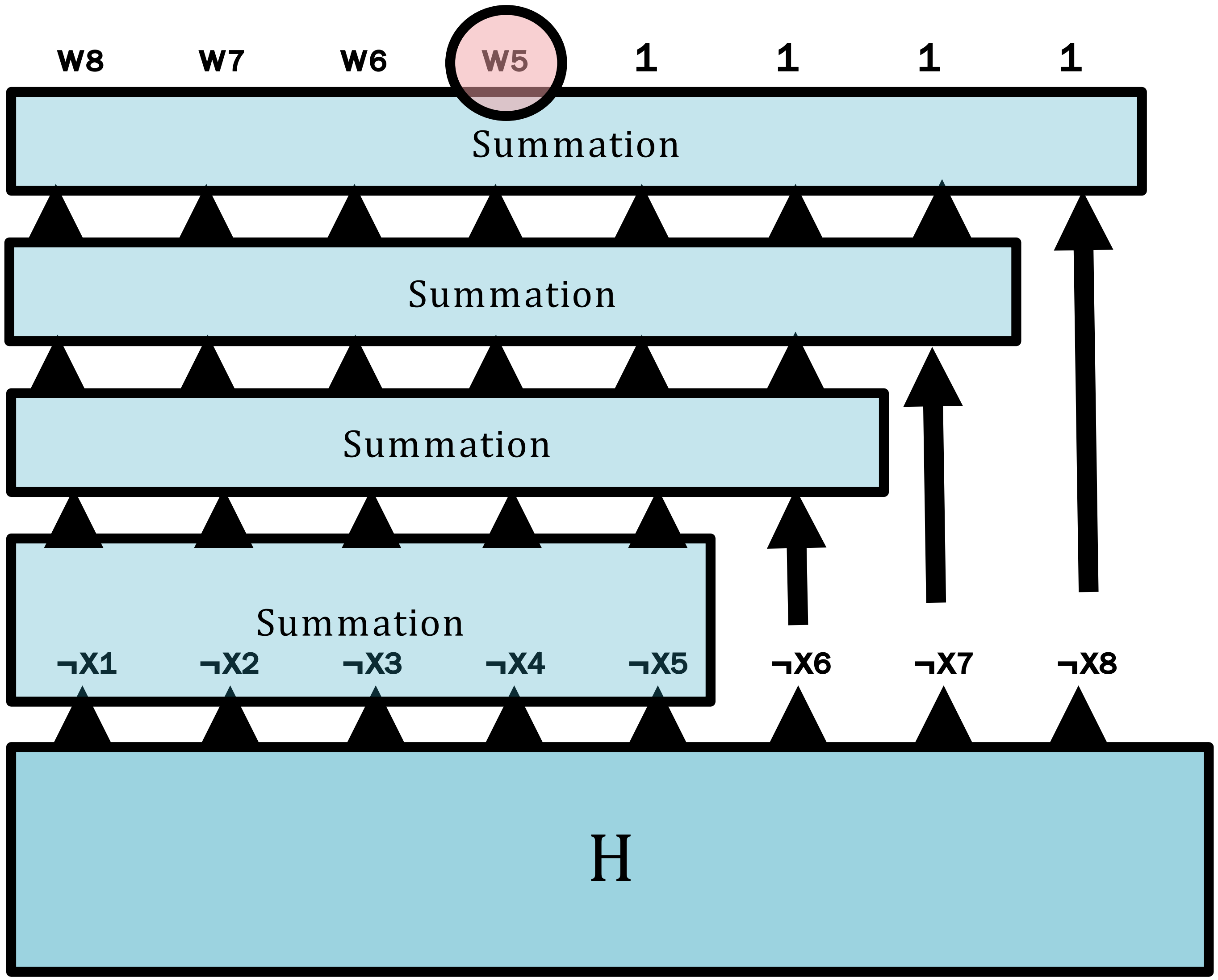
MUS3 Incremental

- Another core (**u_4, ¬x_8); Add** one to overall cost. Then **SAT_ASSUME(H U Card,{¬w_5**})



MUS3 Incremental

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



OLL

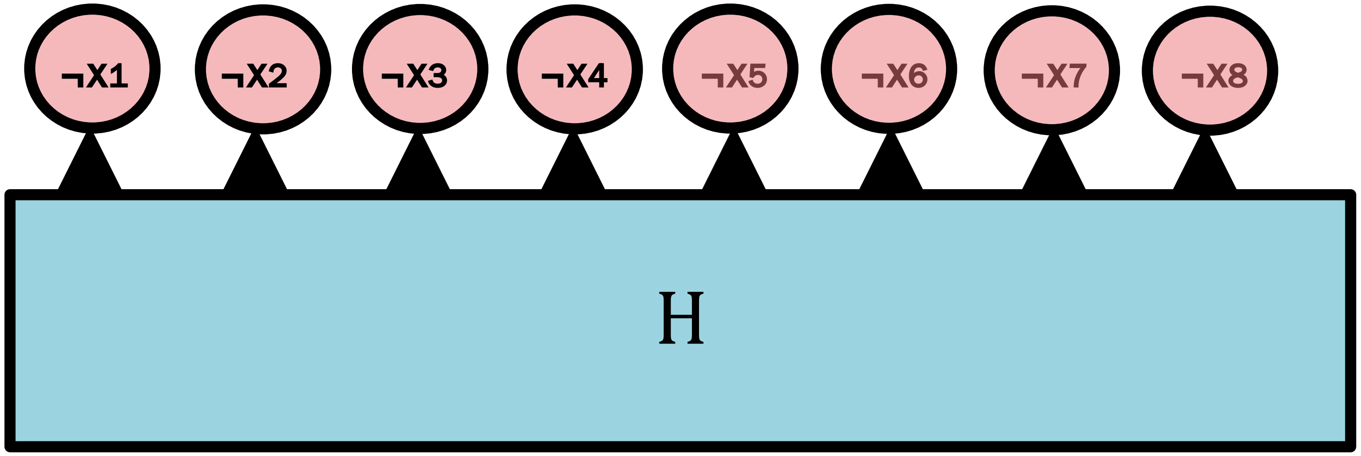
- Used in RC2.
- A. Morgado, C. Dodaro, and J. Marques-Silva. **Core-guided MaxSAT with soft cardinality constraints.** CP 2014
- Uses summation circuits like MUS3 but links up the summation circuits in a more flexible way.

OLL

With same example
Start with Cardinality Constraint Layer with inputs only

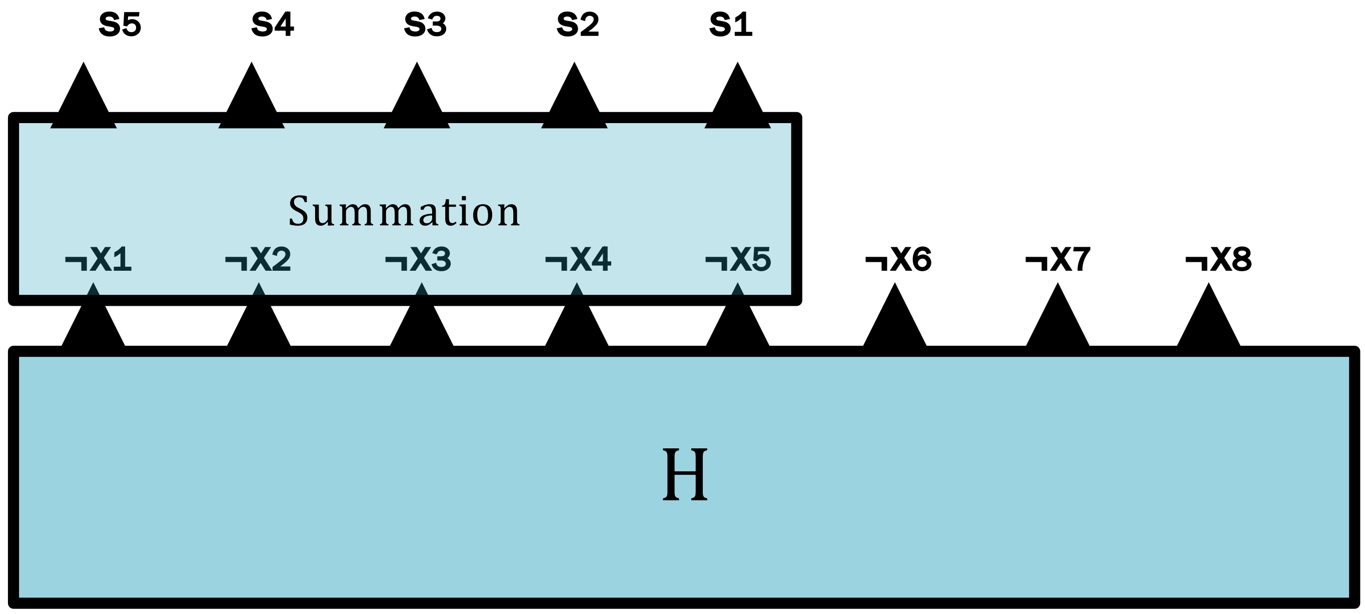
● Literals assumed FALSE

First SAT solve is SAT_ASSUME(H,{x_1,x_2,...,x_8}),
I.e., try to falsify zero softs.



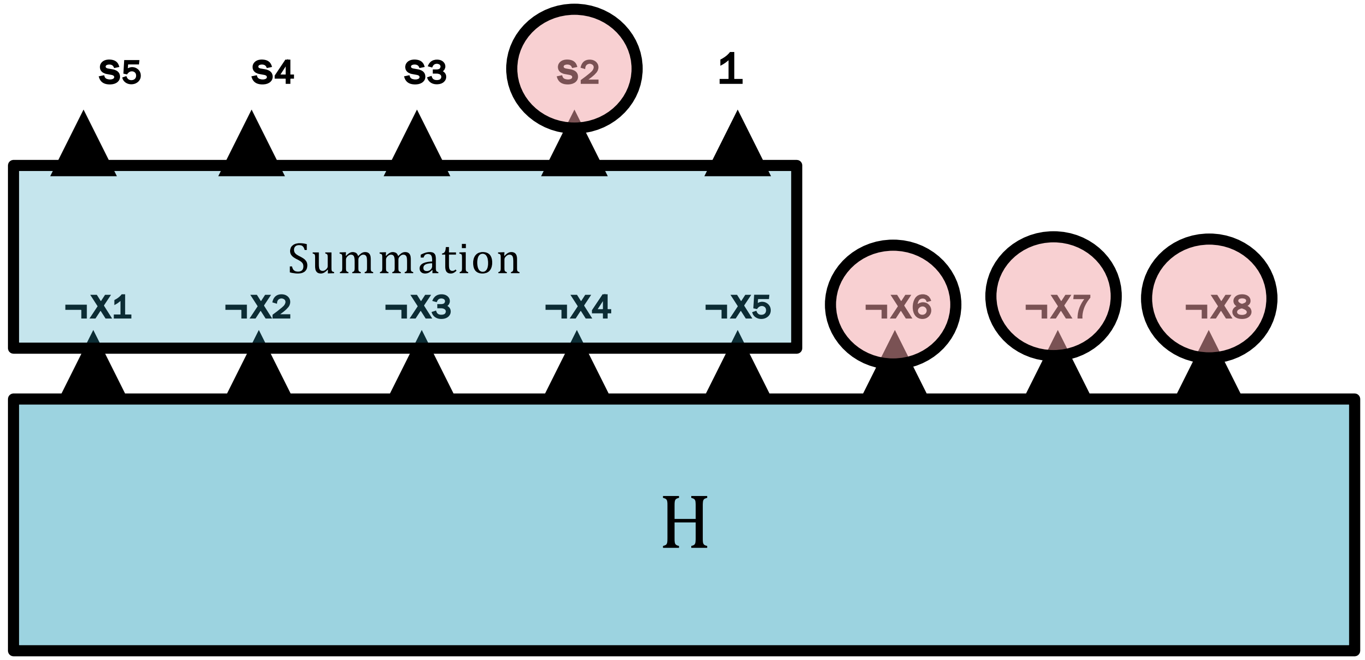
OLL

- UNSAT; say we get the core $(\neg x_1, \neg x_2, \neg x_3, \neg x_4, \neg x_5)$
- Build summation network over softs in core



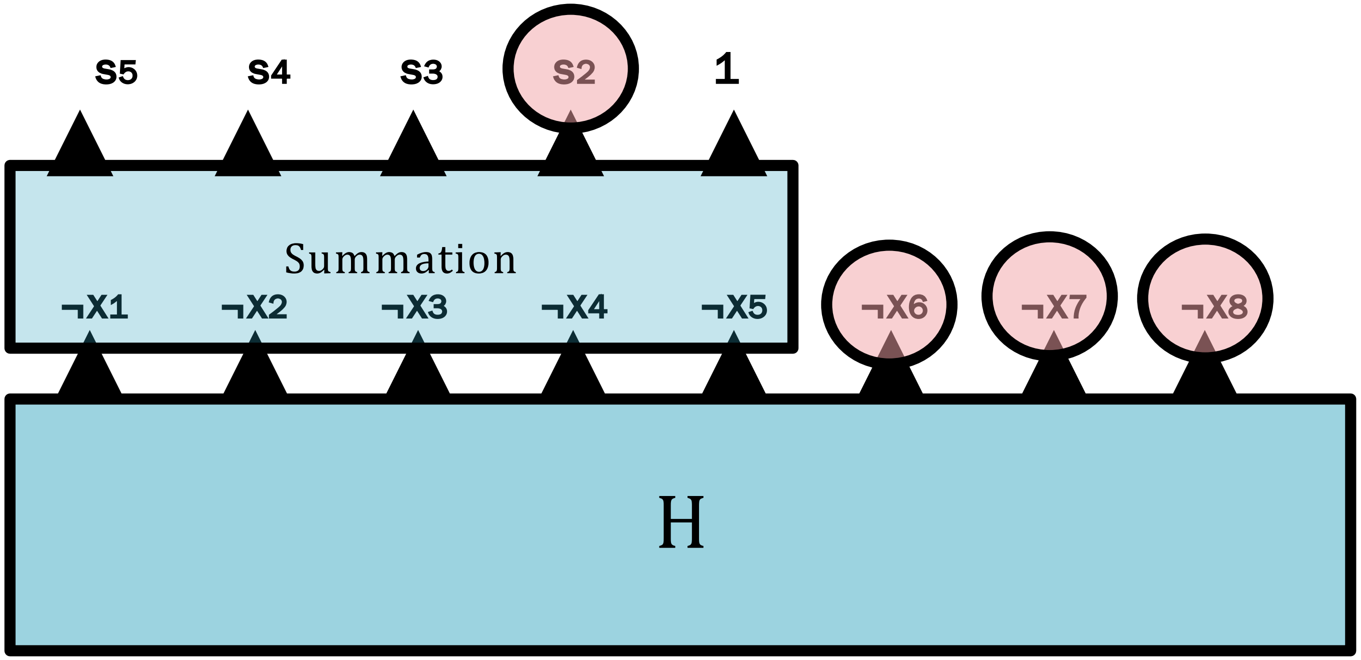
OLL

- We know that the sum must be at least one—set $s_1 = 1$.
- Now **SAT_ASSUME(H U CARD,{¬s_2,x_6,x_7,x_8})**,
- Allow one soft to be falsified among **x_1—x_5**



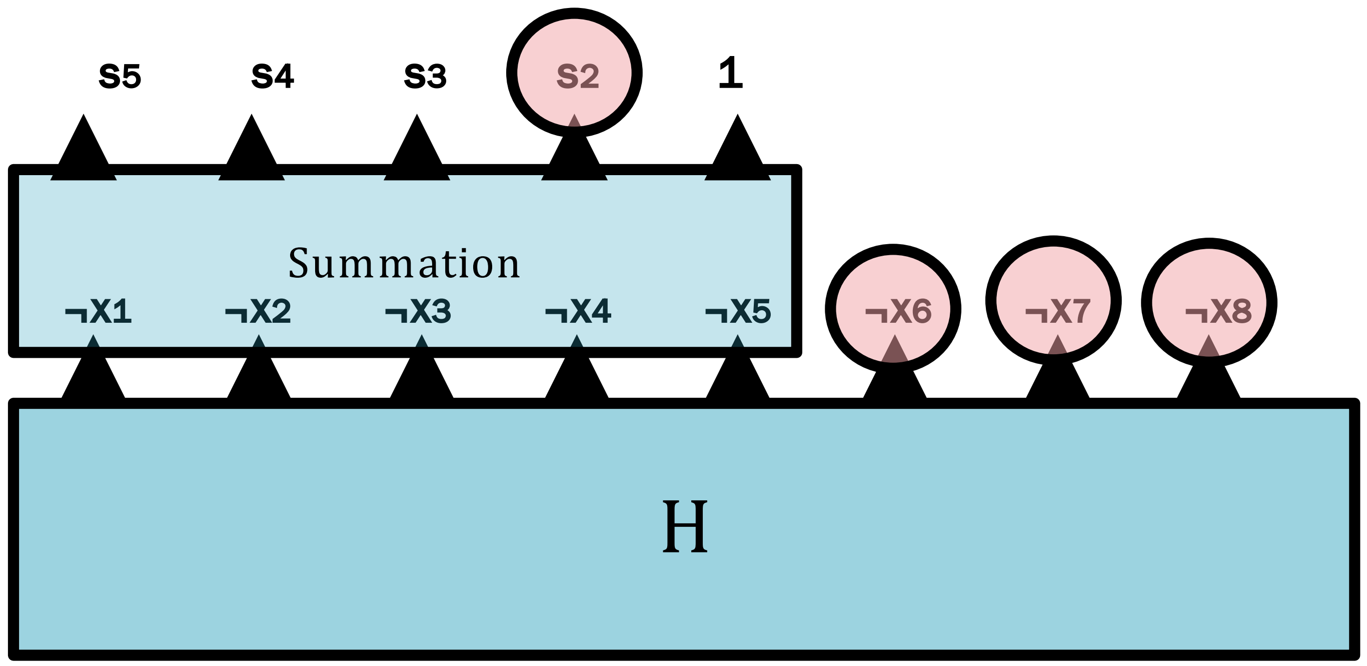
OLL

- Get another core
- Get another conflict, e.g., (s_2, $\neg x_6$)

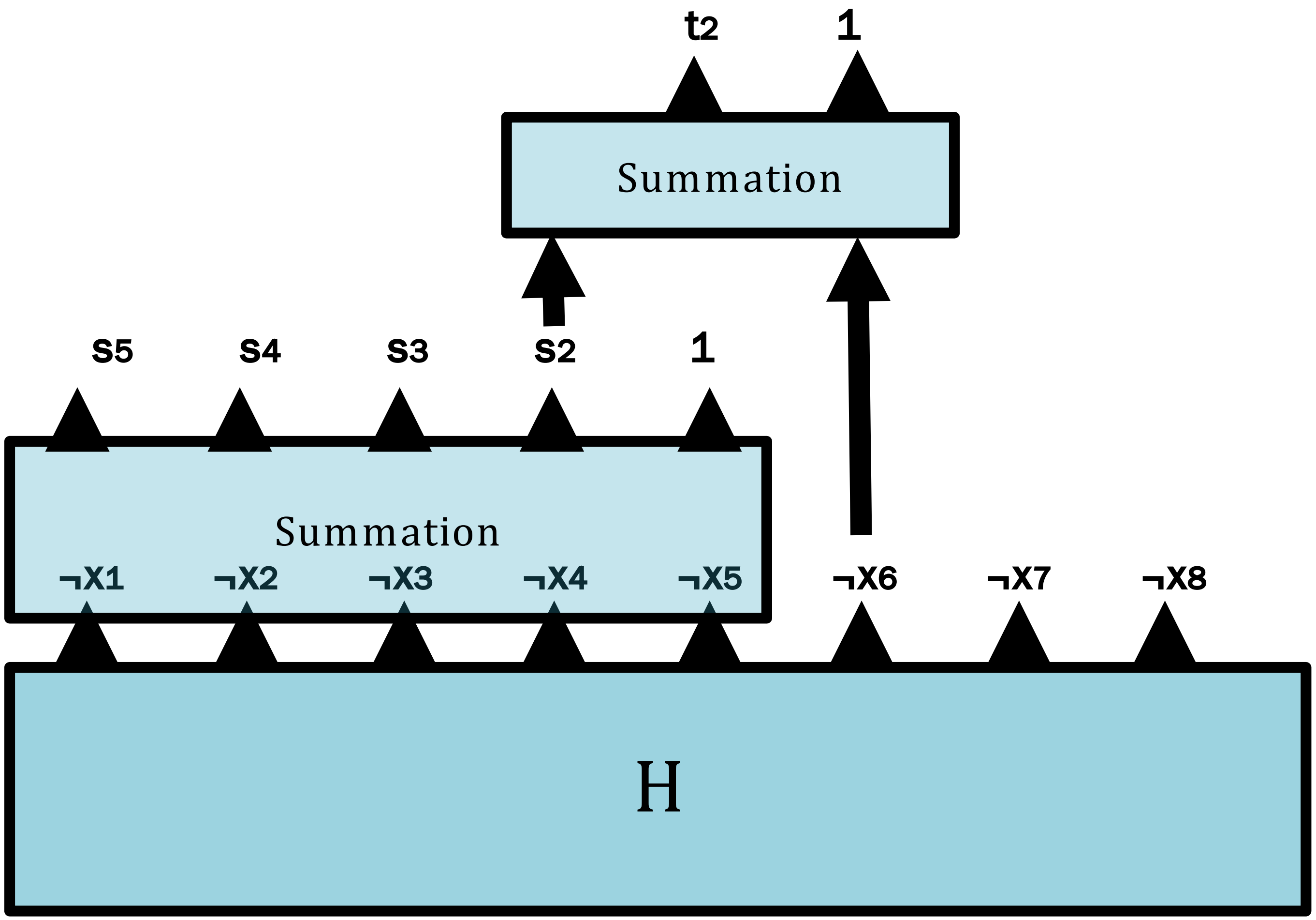


OLL

- 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.

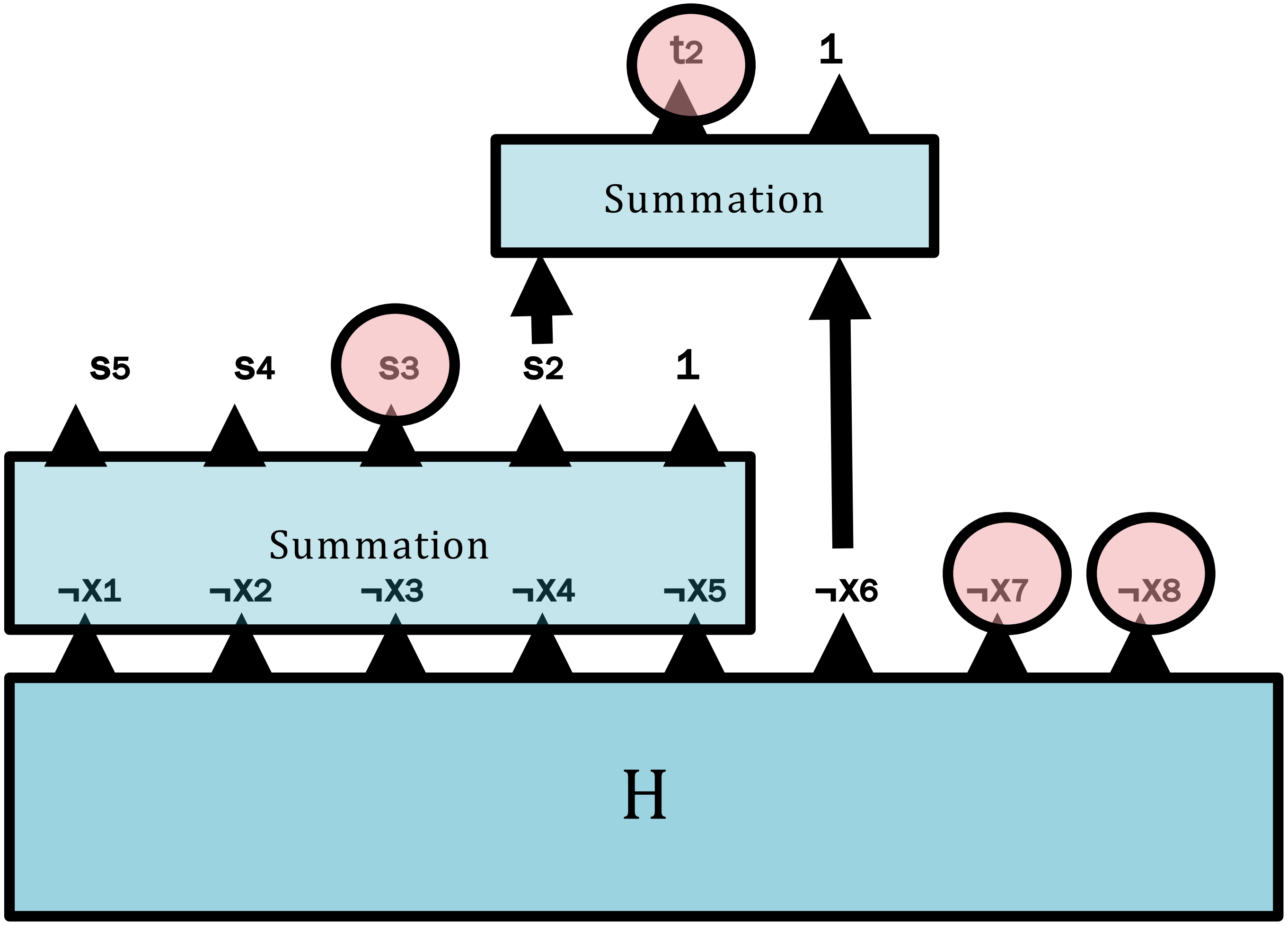


OLL



OLL

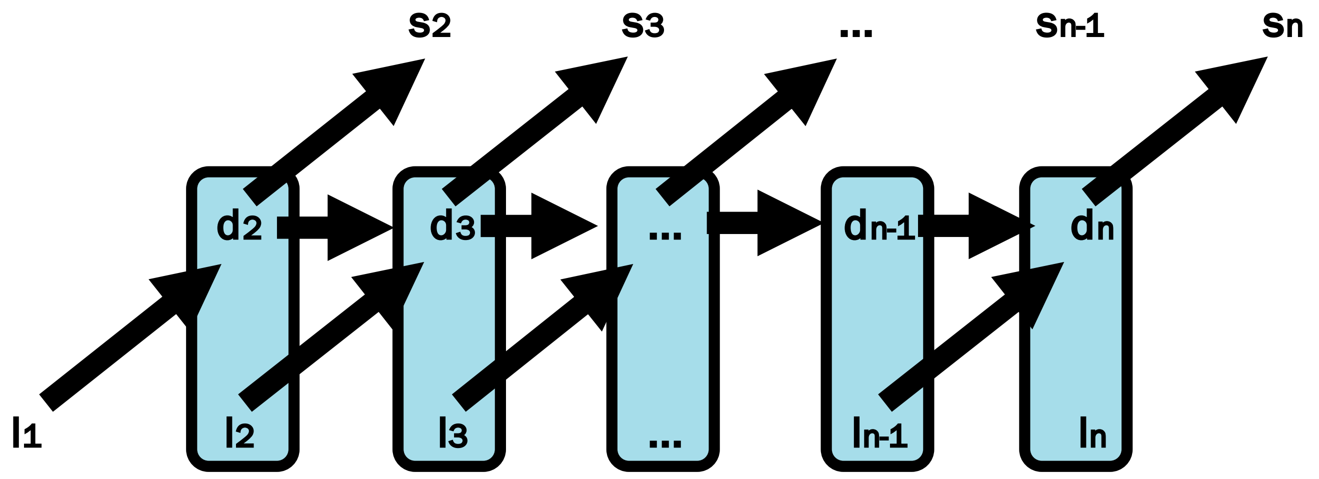
- Then **SAT_ASSUME(H U CARD,{$\neg s_3$,$\neg t_2$,x_7,x_8})**…continue in this way



PMRES

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

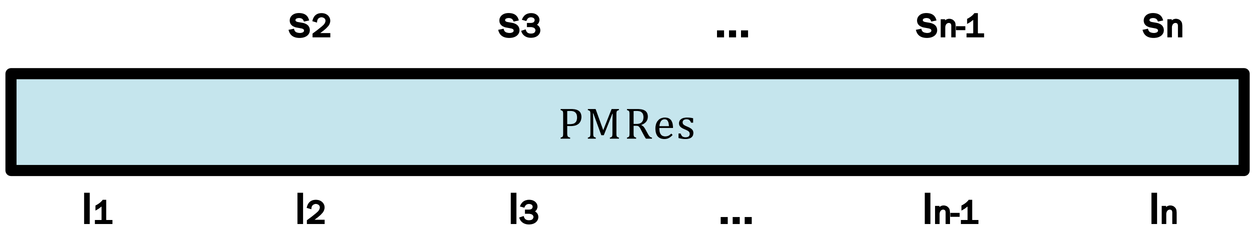
PM-Res



- 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.



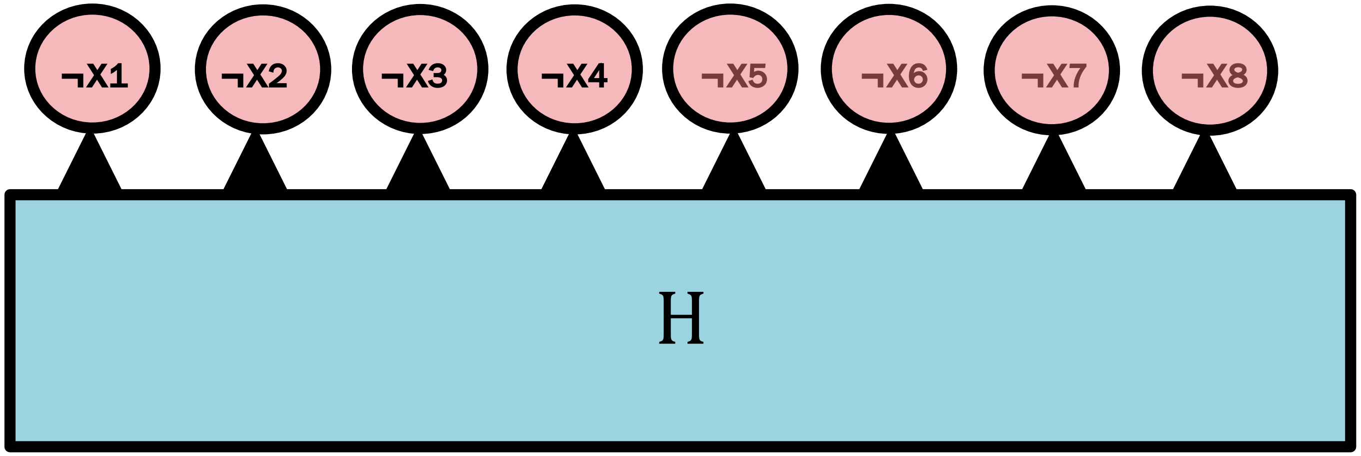
PMRes

Using same example

Start with Cardinality Constraint Layer with inputs only

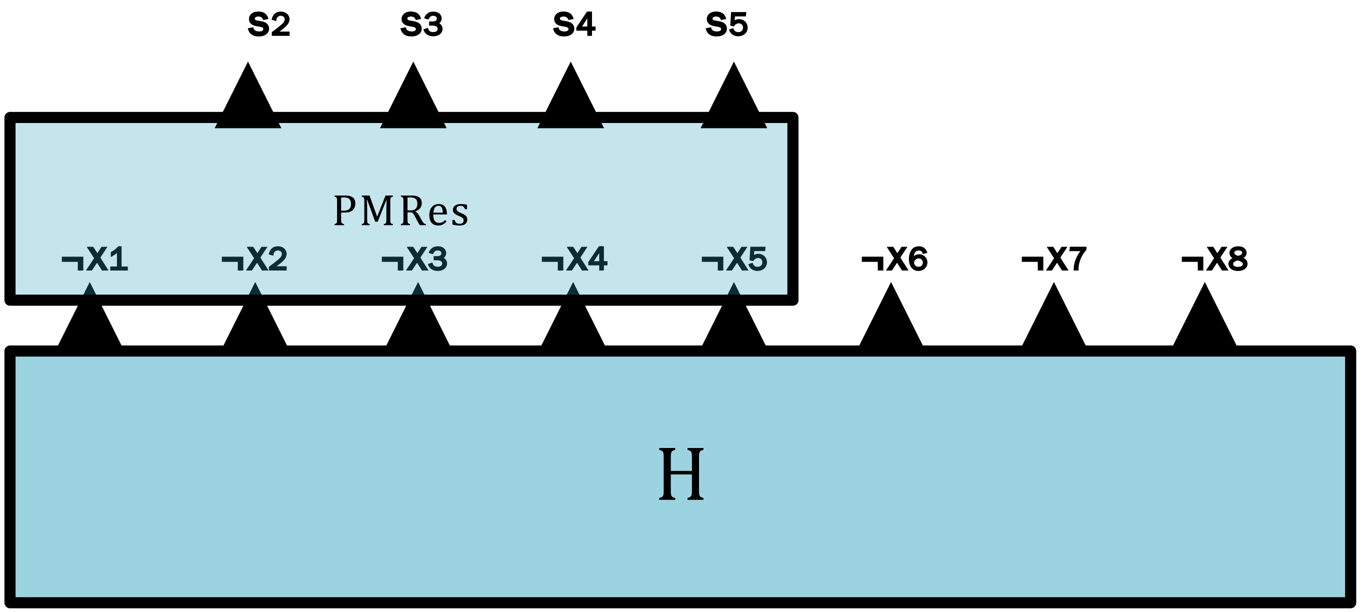
● Literals assumed FALSE

First SAT solve is SAT_ASSUME(H,{x_1,x_2,…,x_8}),
I.e., try to falsify zero softs.



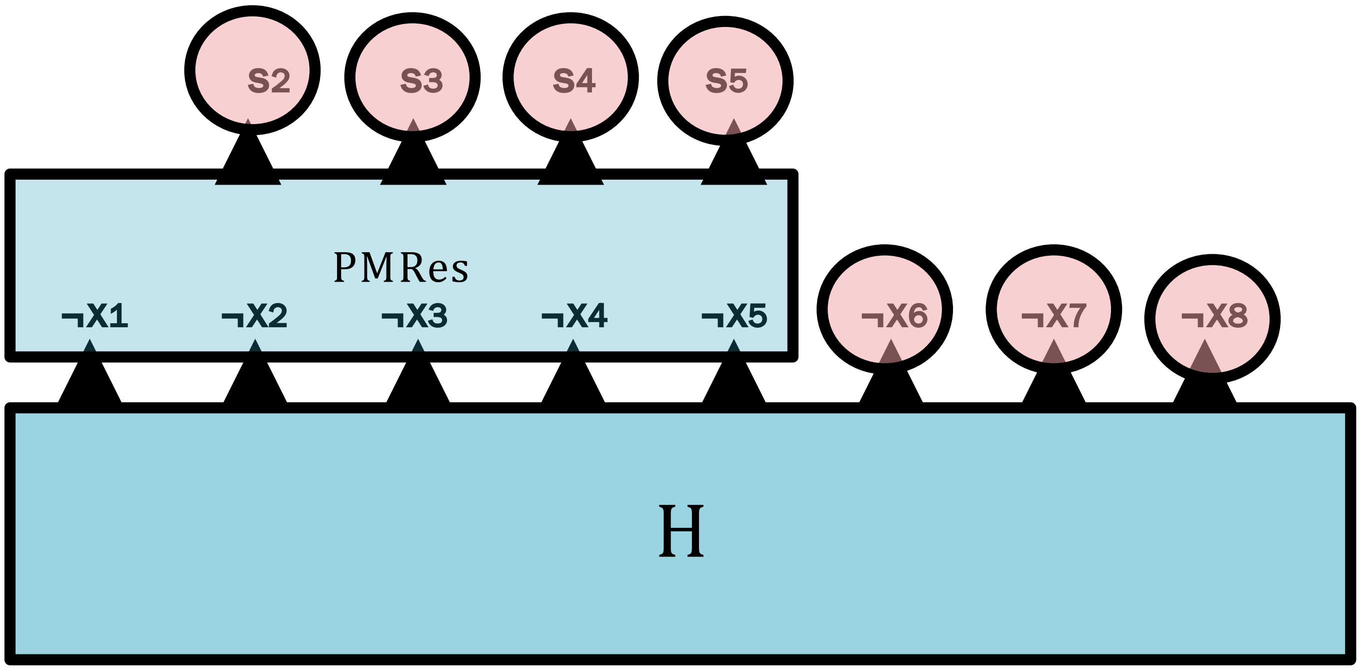
PMRes

- UNSAT; say we get the core $(\neg x_1, \neg x_2, \neg x_3, \neg x_4, \neg x_5)$
- 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



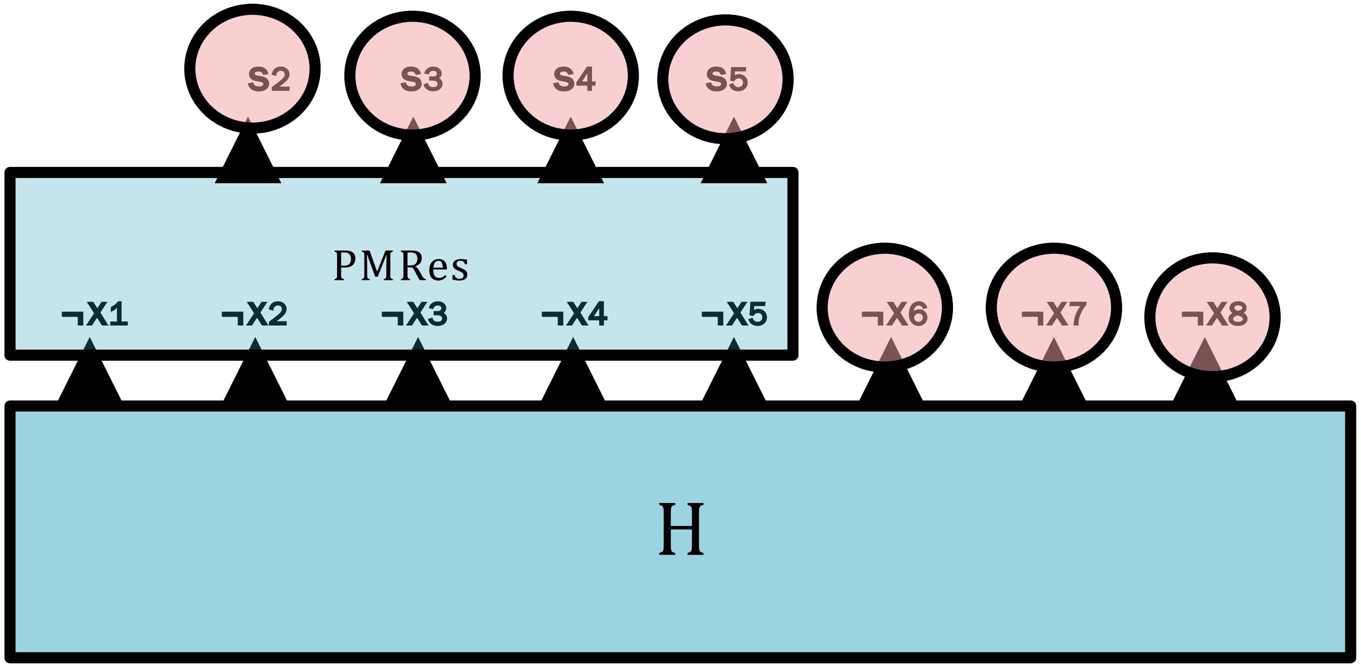
PMRes

- Now **SAT_ASSUME(HUCard,**{¬S2, ¬S3, ¬S4, ¬S5, X6,X7,X8}),
- ➔ We can falsify at most one of the softs **x_1, ..., x_5** but no other softs.
- UNSAT (we must falsify at least 4)



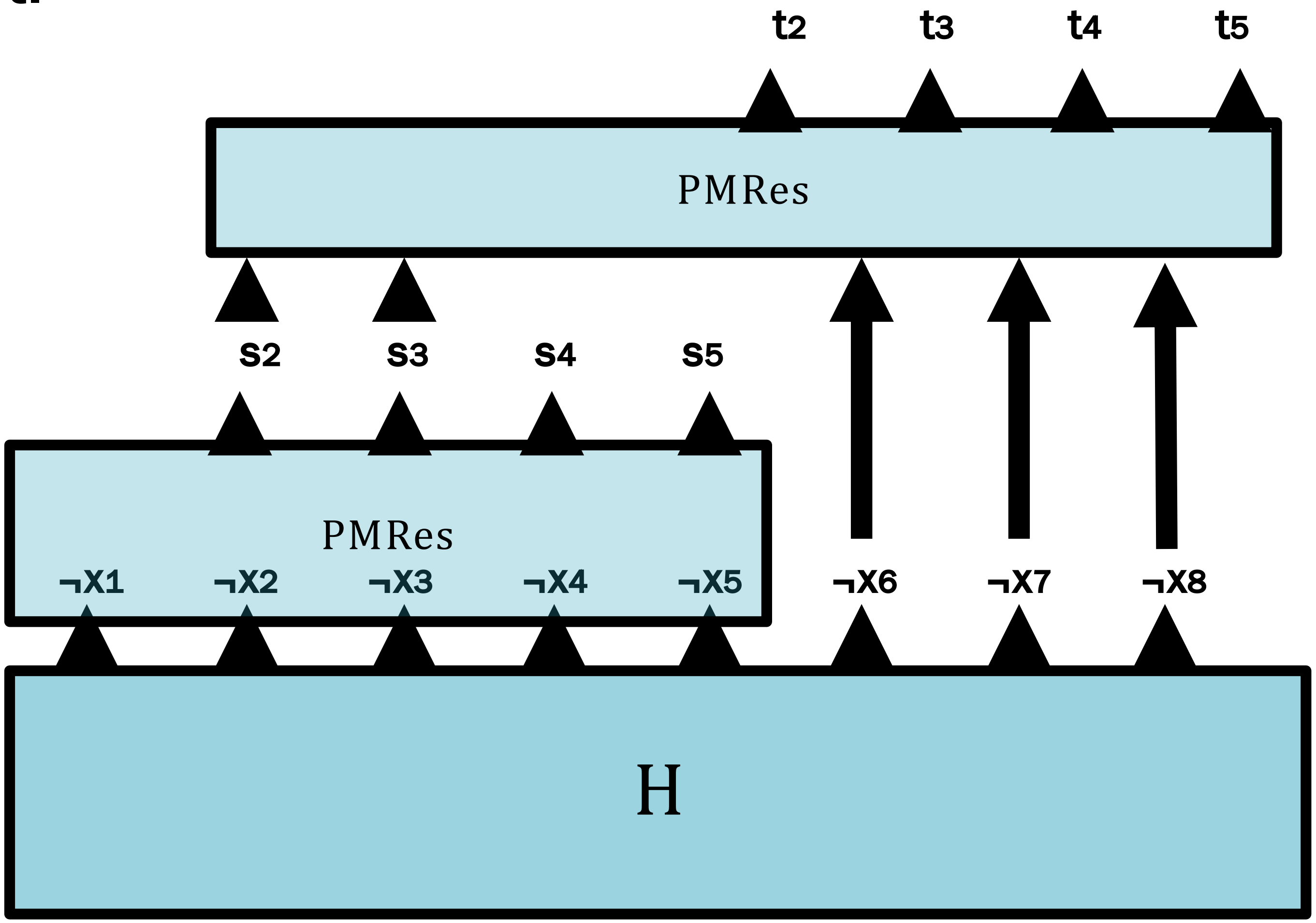
PMRes

- Various conflicts, say **SAT_ASSUME(H,**{¬s_2, ¬s_3, ¬s_4, ¬s_5, x_6,x_7,x_8}) returns the core (x_6,x_7,x_8,¬s_2,¬s_3)
- PMRes then builds a new PMRes circuit with these literals as input.



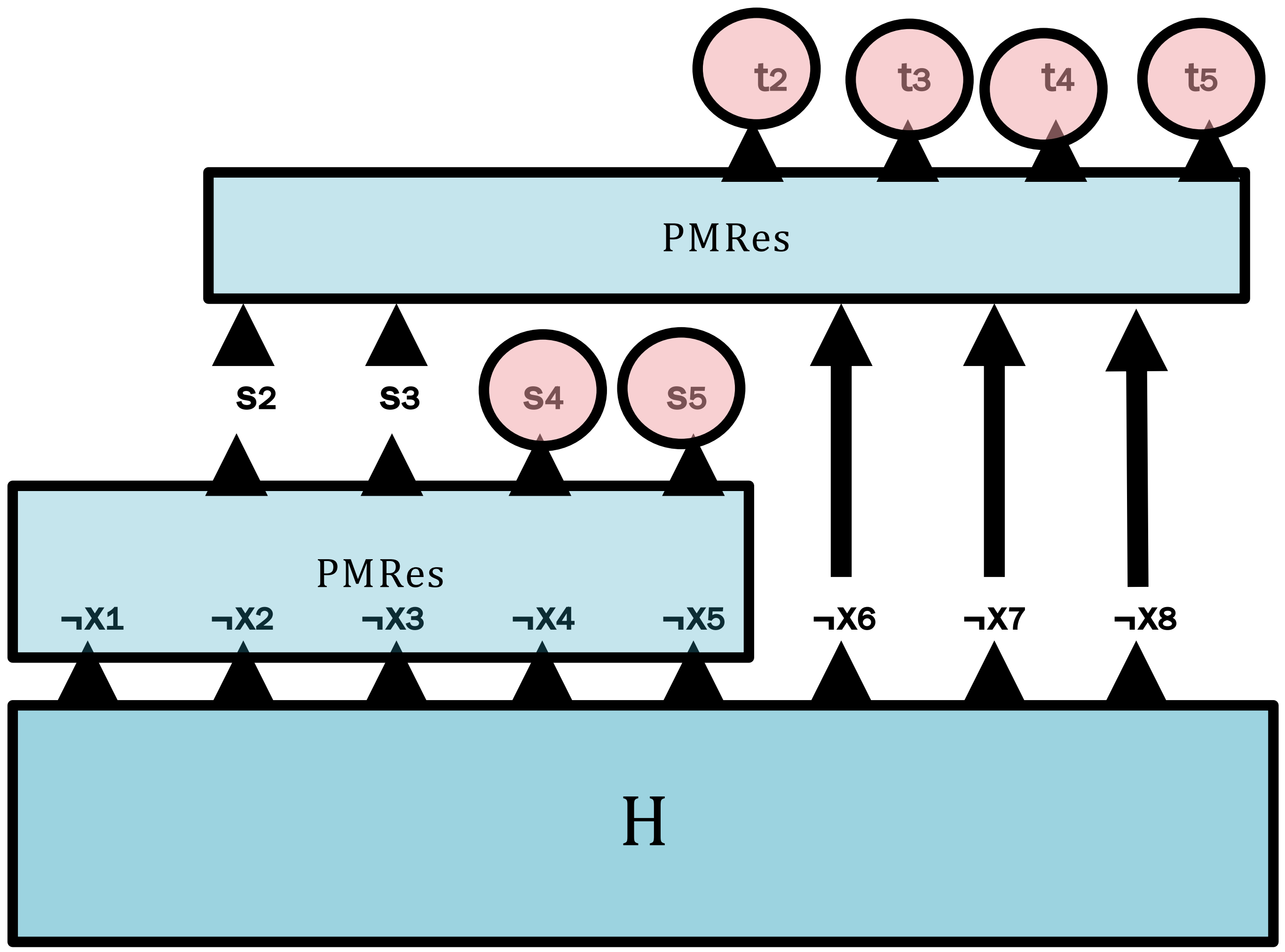
PMRes

- new core ($x_6, x_7, x_8, \neg s_2, \neg s_3$)
- PMRes then builds a new PMRes circuit with these as input.



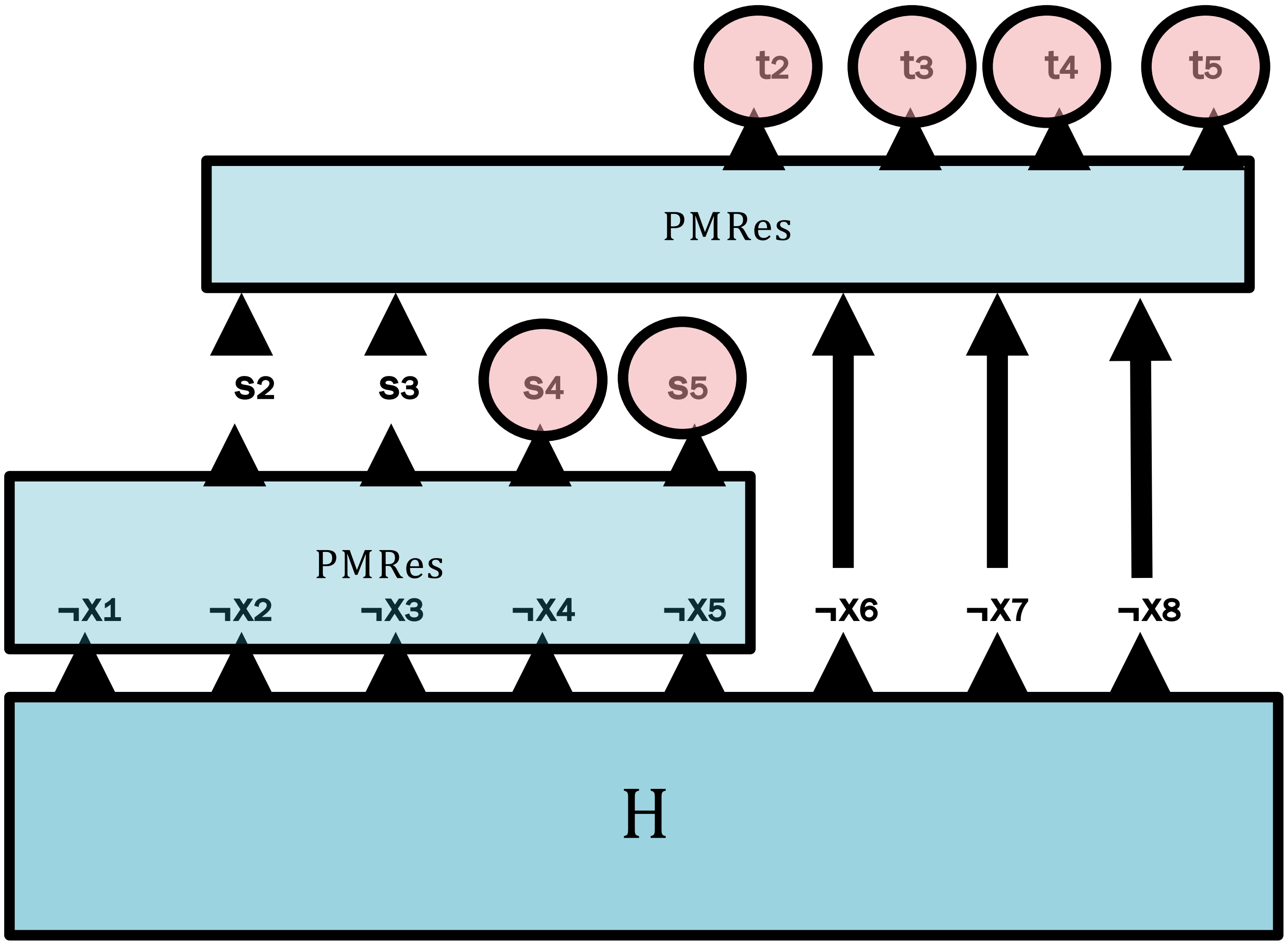
PMRes

- **SAT_ASSUME(H,{¬s_4, ¬s_5, ¬t_2, ¬t_3, ¬t_4, ¬t_5})**
- It gets complex…



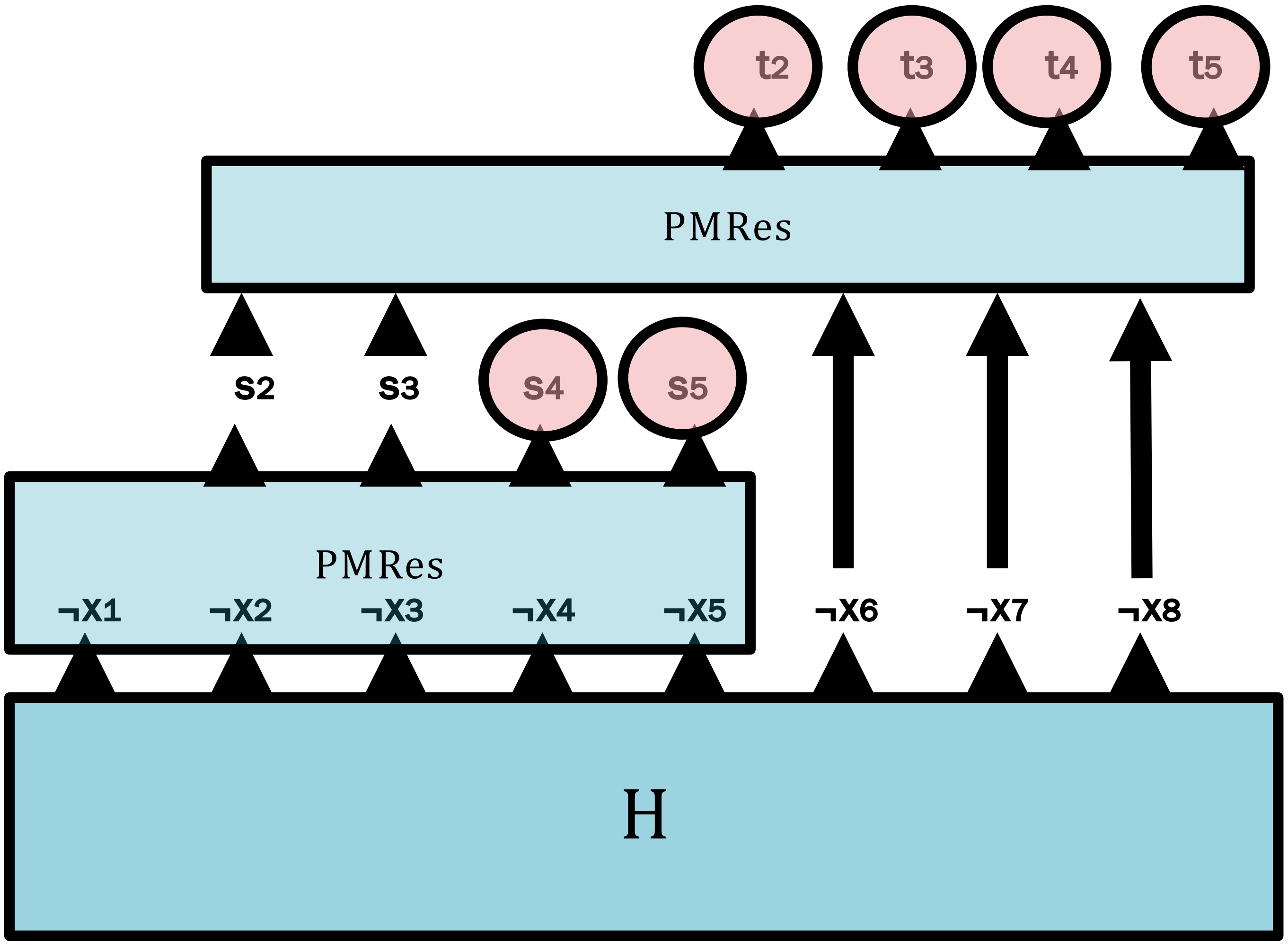
PMRes

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



PMRes

- PMRes continues in this way until it it achieves SAT.



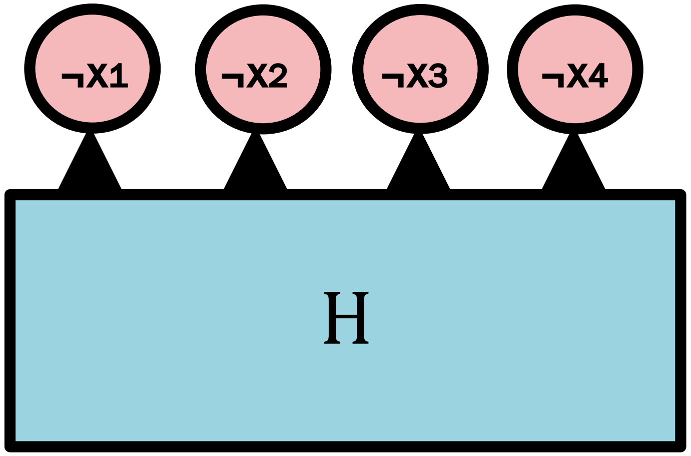
Dealing with Weights.

- These algorithms employ the technique of **clause cloning** to deal with weighted softs.
- When a core is found e.g., $\{l_1, l_2, l_3, l_4\}$ where the soft clauses (l_i) have different weights.
  - Let minWt be the minimum weigh 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

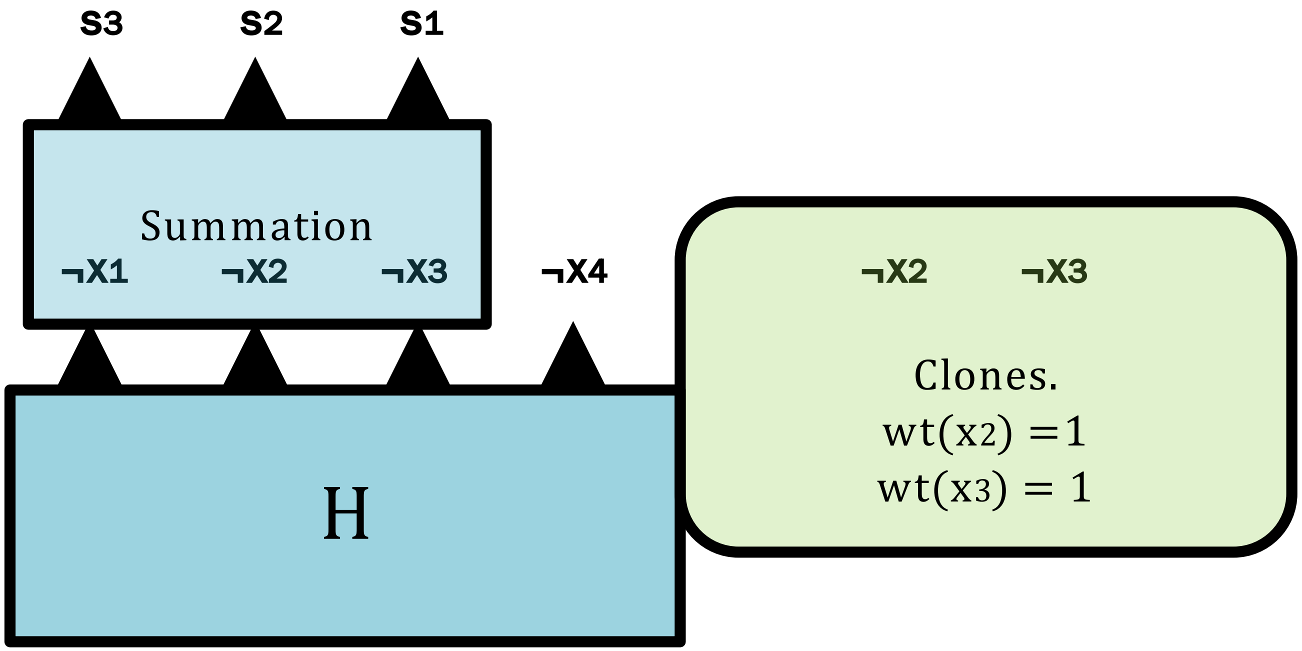
Say S = {(x_1), (x_2), (x_3), (x_4)} with wt(x_i) = i

First SAT solve is SAT_ASSUME(H,{x_1,x_2,x_3,x_4}),



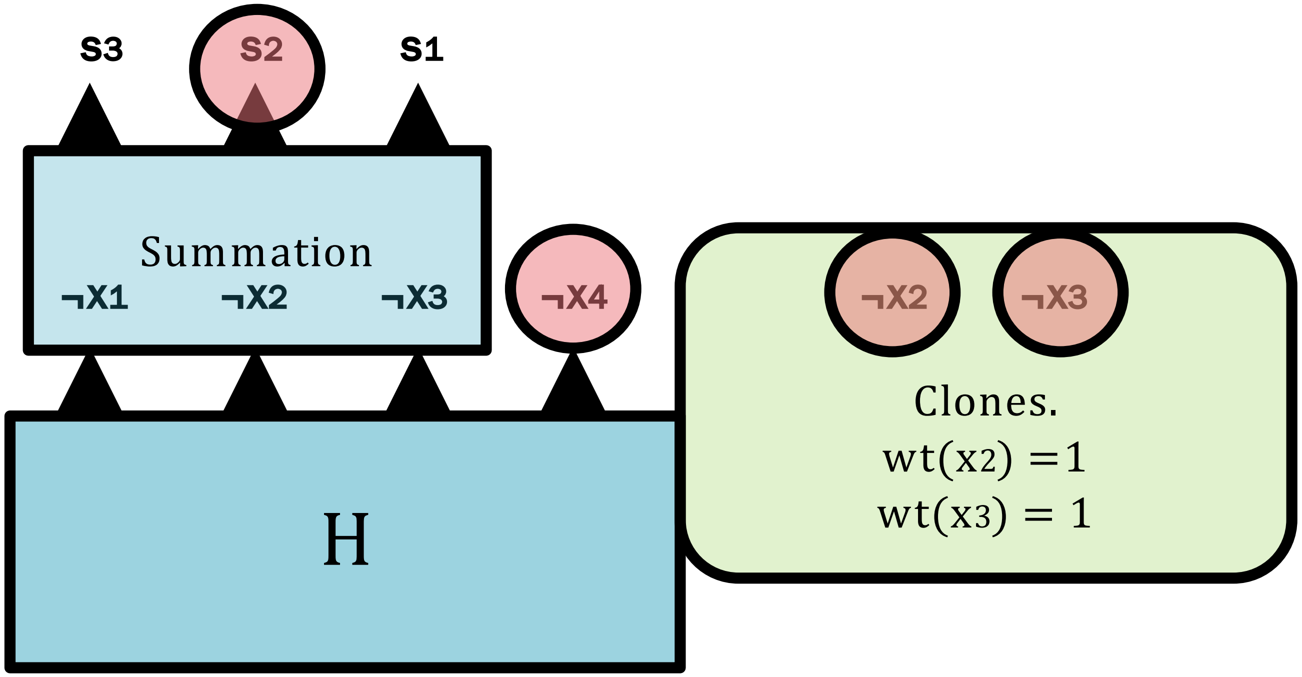
Weighted instances

UNSAT; say we get the core (x_1,x_2,x_3)
As before build summation network over softs in core.
minWt=1, so we must clone x_2 and x_2.



Weighted instances

Now **SAT_ASSUME(H,{¬S2,X4, X2 ,X3}),**



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.
- Instance independent question:
  - From each UNSAT result we extract a conflict clause. Are the current ways of constructing the cardinality layer fully exploiting the information in those conflicts? How can we formalize this question.
- Instance dependent question:
  - Each way of constructing the cardinality layer ends up posing a different type of query to the SAT solver. Do instances with particular structure work better with particular ways of constructing that layer?

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.
  - In the 2018 MaxSat Evaluation a OLL solver RC2 solved more weighted instances among those tested than any other solver.
  - But clearly less effective the implicit hitting set solver MaxHS when run on a larger benchmark suite.
  - On instances where the # of distinct weights > 3, the difference is more profound.
- On unweighted instances (all softs have the same weight) sequence of sat approaches are generally superior to implicit hitting set approaches.

Implicit Hitting Set Solvers

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 **F**.
  - The SAT solver is always run on instances that are no more complex that the input **F**.
- The cores exploited by IHS solvers are cores of the input formula **F** (not cores of **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 $\boldsymbol{\kappa} \subseteq$ **S** is a core of **F** if $\boldsymbol{\kappa} \cup$ **H** is **UNSAT**
  - Feasible solutions satisfy the hard clauses **H**
- Let **K** be any set of cores of F and **π** any feasible solution. **π must falsify** at least one soft clause of every core in **K.**
- Let A = {c | $\pi \nvDash$ c} be the set of clauses falsified by π
- Then A is a hitting set of **K** (non-empty intersection with every member of **K**).

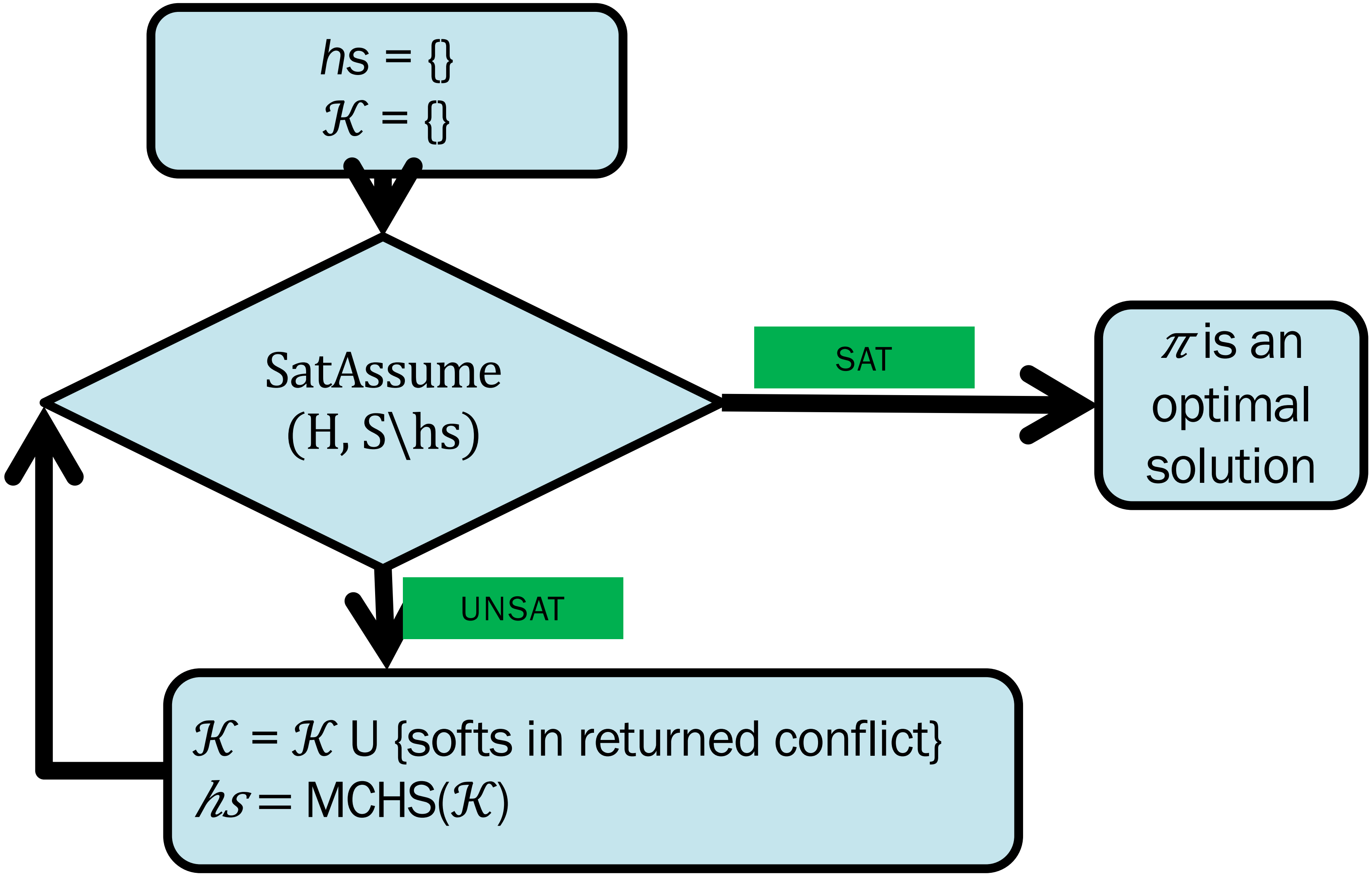
Cores and hitting sets

- Let MCHS(K) be a minimum cost hitting set of **K**–this is a set of soft clauses.
- For every feasible solution π

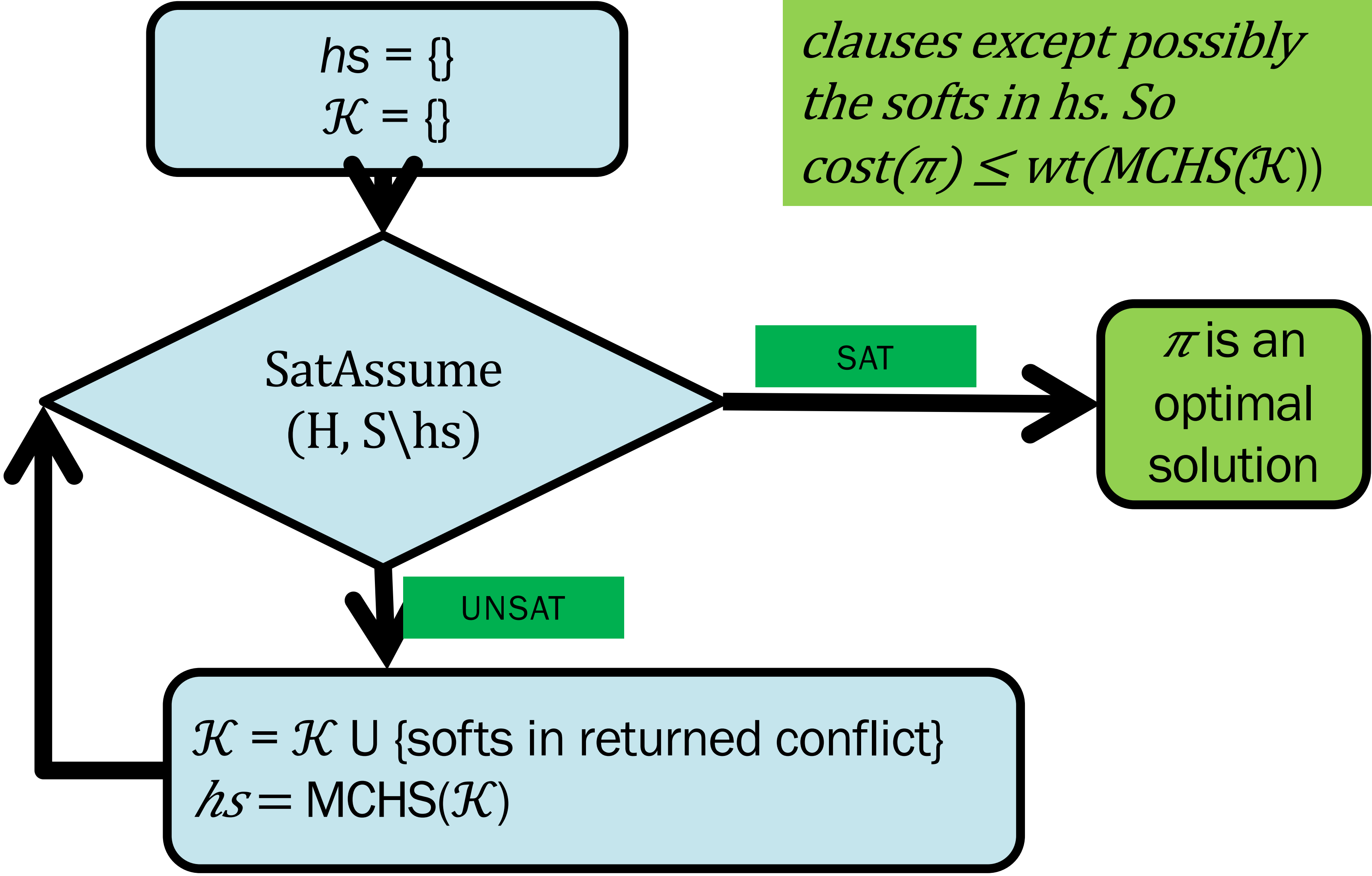
  $$\text{cost}(\pi) = \text{wt}(A) \geq \text{wt}(\text{MCHS}(K))$$
- The weight of a minimum cost hitting set of **any** set of cores is a **lower bound** on the cost of an optimal solution.
- Therefore, for **any** set of cores **K** and **any** feasible solution π if cost(π) = wt(MCHS(K)), π must be an **optimal solution.**

- This leads to a simple algorithm for finding an optimal solution.

The IHS Algorithm


hs = {}
$\mathcal{K}$ = {}
SatAssume
(H, S\hs)
SAT
π is an optimal solution
UNSAT
$\mathcal{K}$ = $\mathcal{K}$ U {softs in returned conflict}
$hs =$ MCHS($\mathcal{K}$)

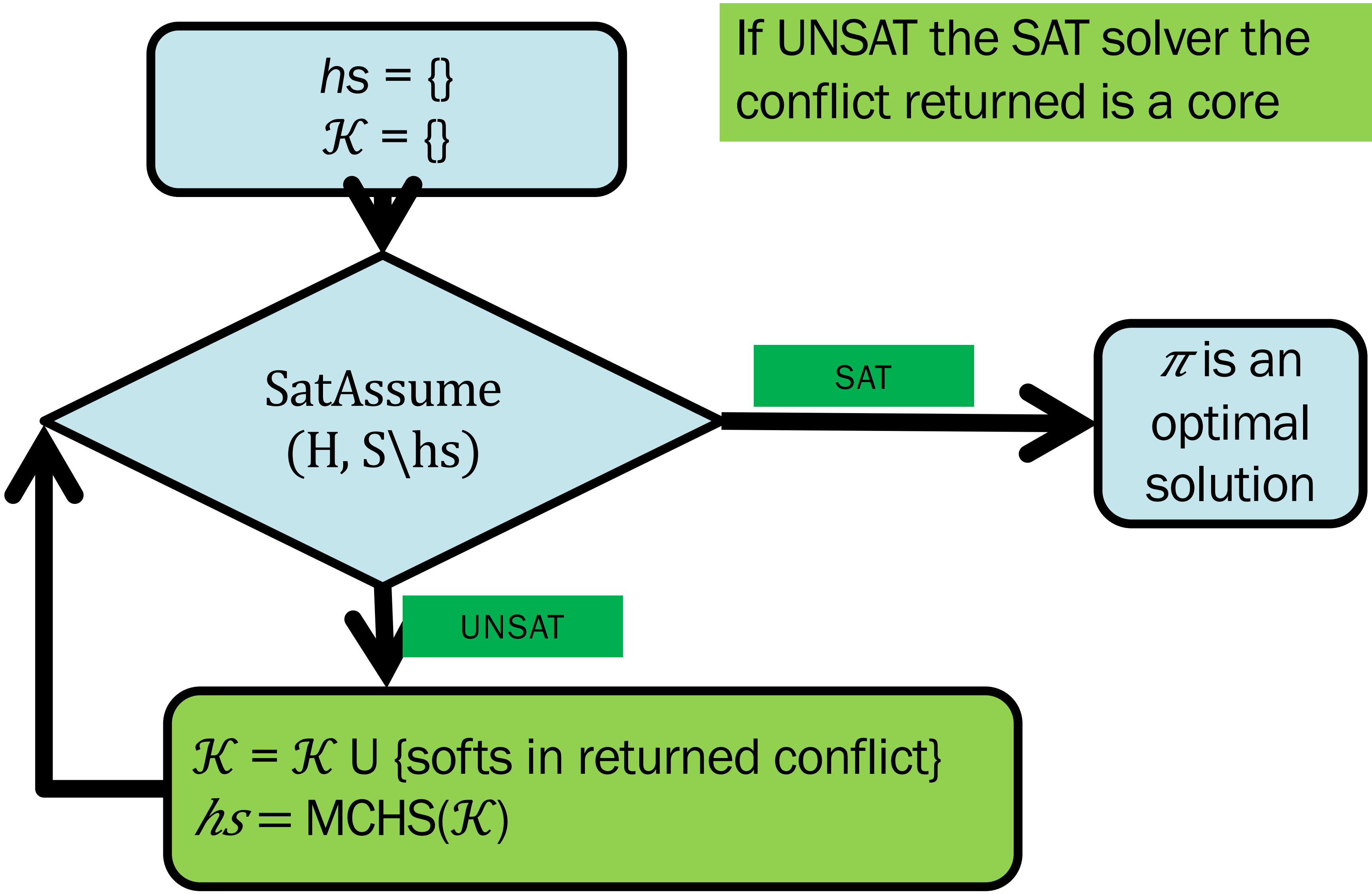
The IHS Algorithm


$hs = \{\}$
$\mathcal{K} = \{\}$
SatAssume (H, S\hs)
SAT
π is an optimal solution
UNSAT
$\mathcal{K} = \mathcal{K}$ U {softs in returned conflict}
$hs = \text{MCHS}(\mathcal{K})$

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

The IHS Algorithm

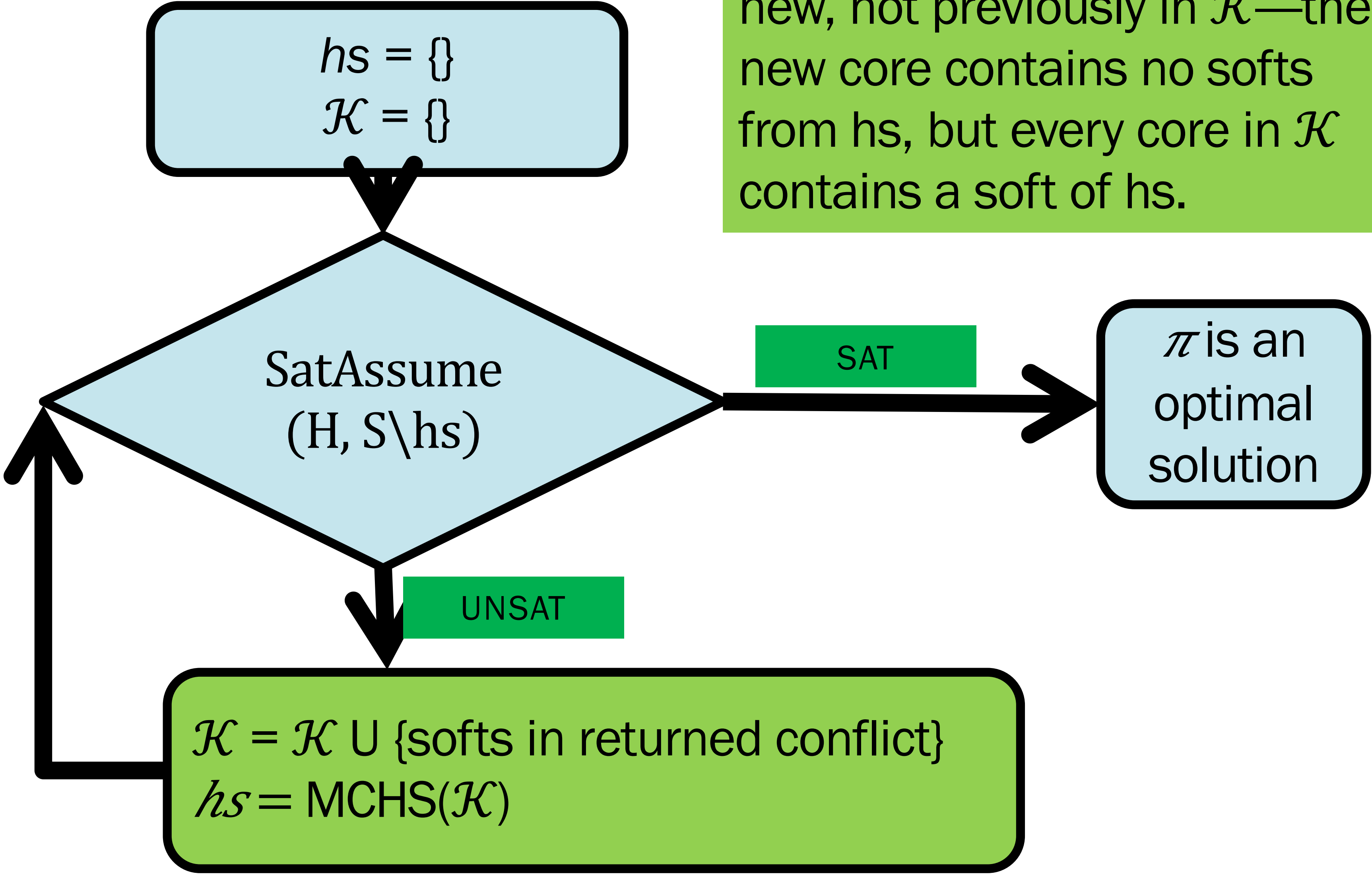
If UNSAT the SAT solver the conflict returned is a core



hs = {}
$\mathcal{K}$ = {}
SatAssume (H, S\hs)
SAT
π is an optimal solution
UNSAT
$\mathcal{K} = \mathcal{K}$ U {softs in returned conflict}
hs = MCHS($\mathcal{K}$)

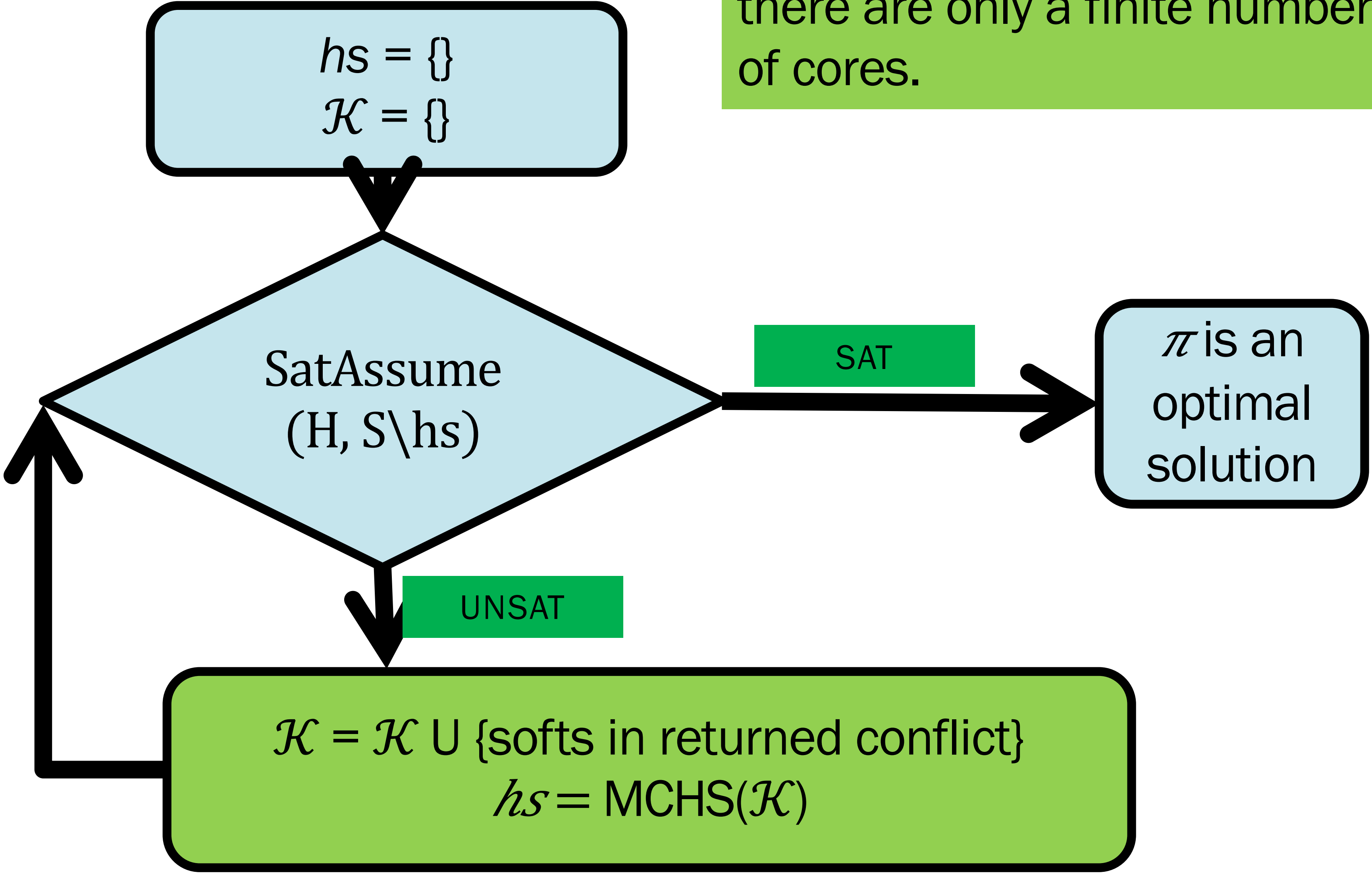
The IHS Algorithm

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.


hs = {}
$\mathcal{K}$ = {}
SatAssume (H, S\hs)
SAT
π is an optimal solution
UNSAT
$\mathcal{K}$ = $\mathcal{K}$ U {softs in returned conflict}
$hs = \text{MCHS}(\mathcal{K})$

The IHS Algorithm

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



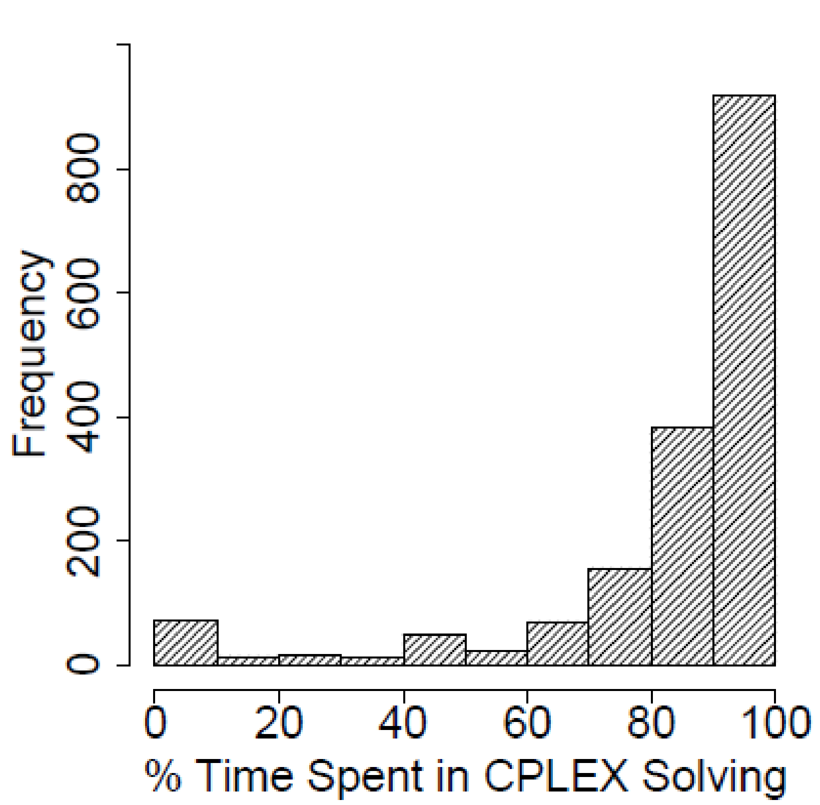
hs = {}
$\mathcal{K}$ = {}
SatAssume (H, S\hs)
SAT
π is an optimal solution
UNSAT
$\mathcal{K} = \mathcal{K}$ U {softs in returned conflict}
$hs = \text{MCHS}(\mathcal{K})$

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

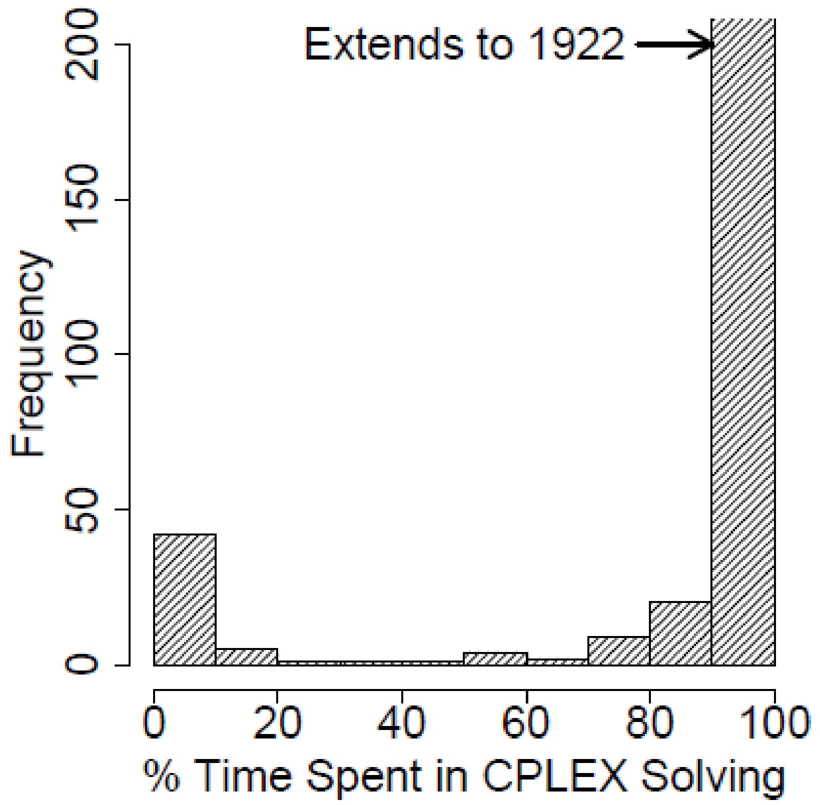
- The MCHS (aka, set-cover) problem is an NP-Hard optimization problem. But in practice it can usually be solved efficiently by an integer programming solver.
  - Typically IBM's CPLEX is used
  - Seems to be by far the most effective way of finding an MCHS.
- The approach is most closely related to the implicit hitting set formalism of Karp. It can also be viewed as being a CEGAR or a logic based Benders approach.
- The direct encoding of MaxSat into an IP is far less effective than this hybrid approach.

The IHS Algorithm

- This basic IHS algorithm is not that effective.



Solved



Unsolved

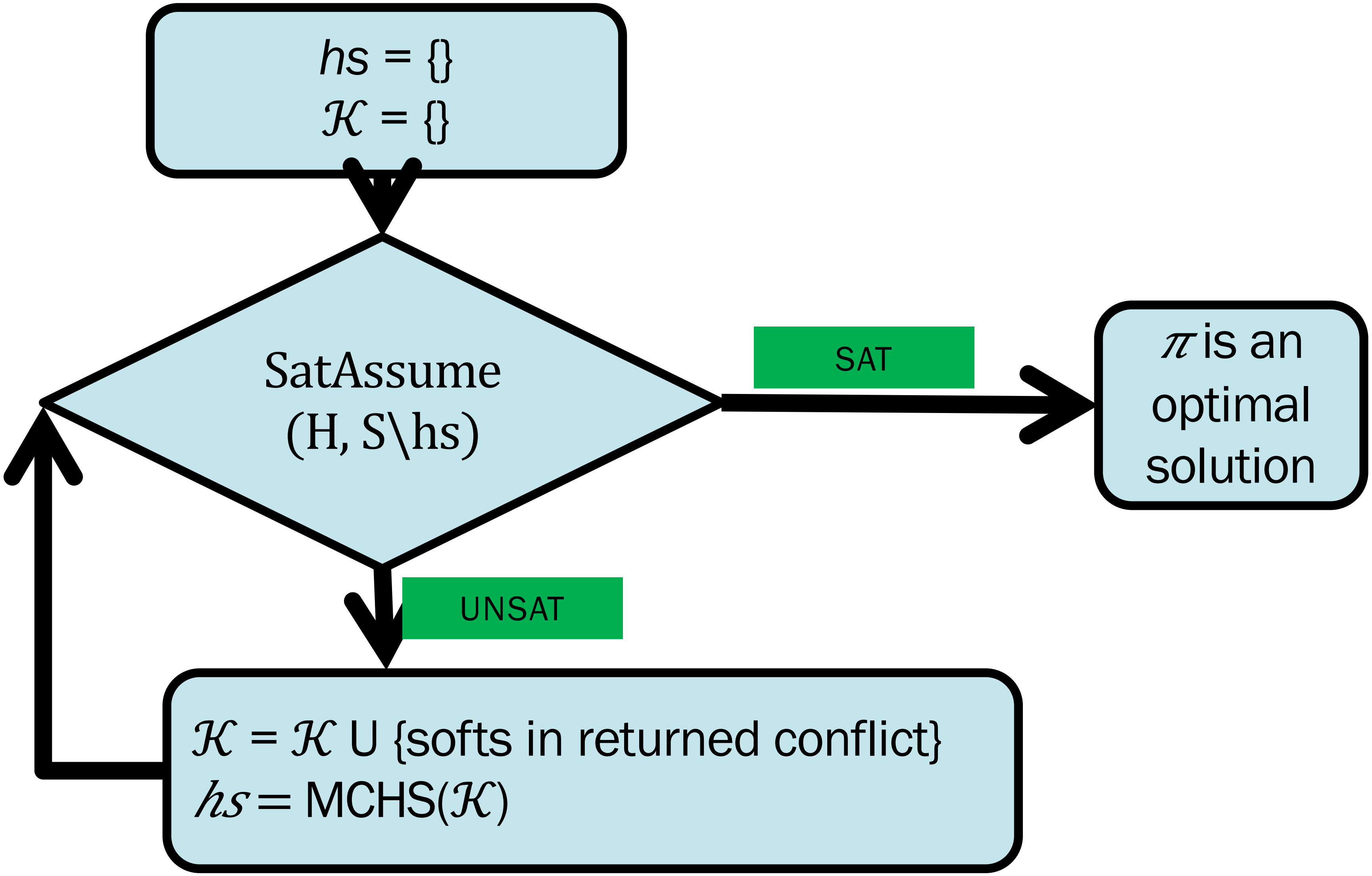
The IHS Algorithm

- 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

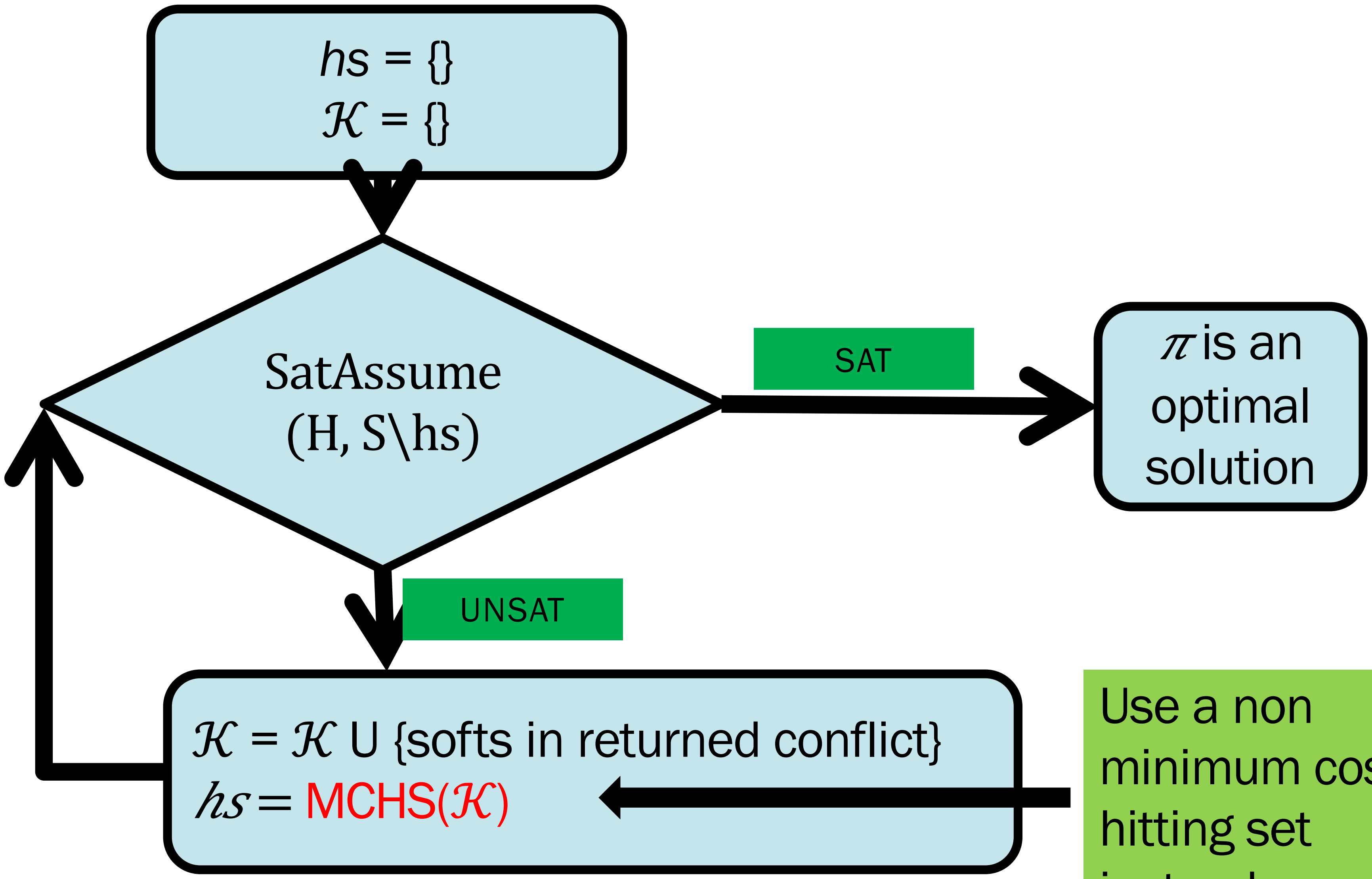
- Various are employed techniques to improve the IP model more quickly
- Most importantly computing an MCHS can be delayed and performed only occasionally. Much cheaper to compute non-minimum hitting sets can be used instead.
- This leads to a different formulation of IHS algorithms (this formulation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

The IHS Algorithm



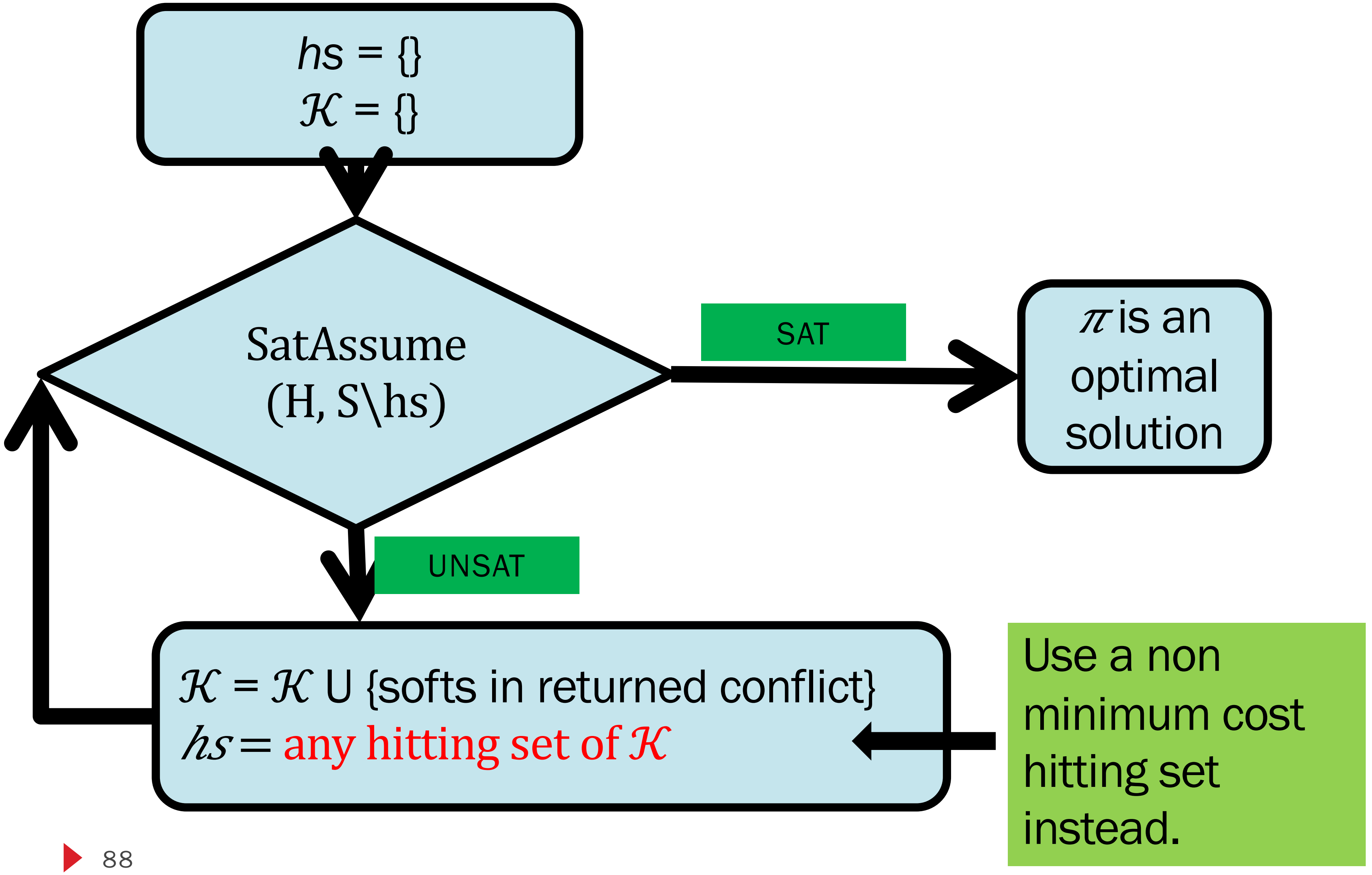
hs = {}
$\mathcal{K}$ = {}
SatAssume (H, S\hs)
SAT
π is an optimal solution
UNSAT
$\mathcal{K}$ = $\mathcal{K}$ U {softs in returned conflict}
$hs = $ MCHS($\mathcal{K}$)

The IHS Algorithm

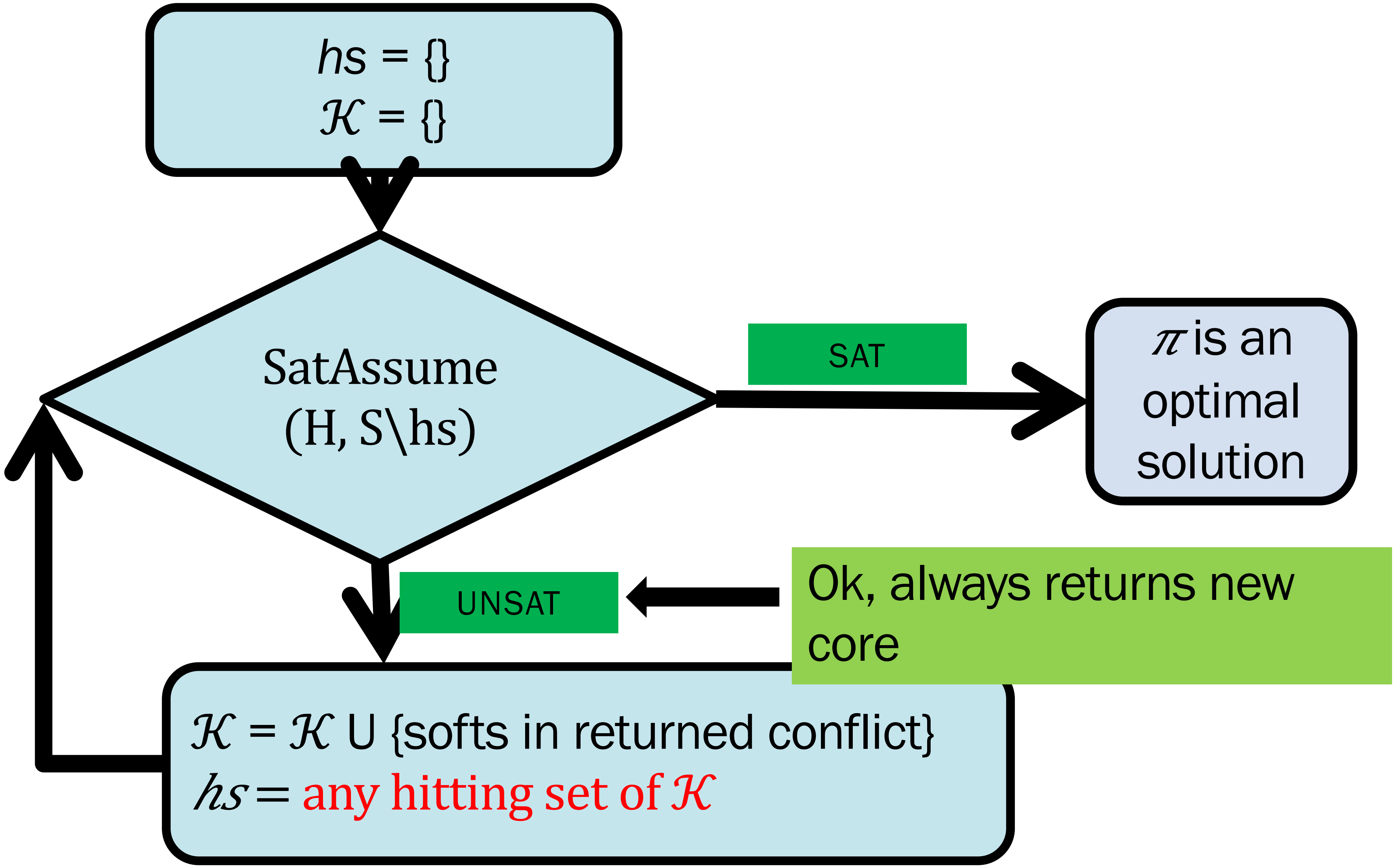

hs = {}
$\mathcal{K}$ = {}
SatAssume (H, S\hs)
SAT
π is an optimal solution
UNSAT
$\mathcal{K}$ = $\mathcal{K}$ U {softs in returned conflict}
hs = MCHS($\mathcal{K}$)

Use a non minimum cost hitting set instead.

The IHS Algorithm

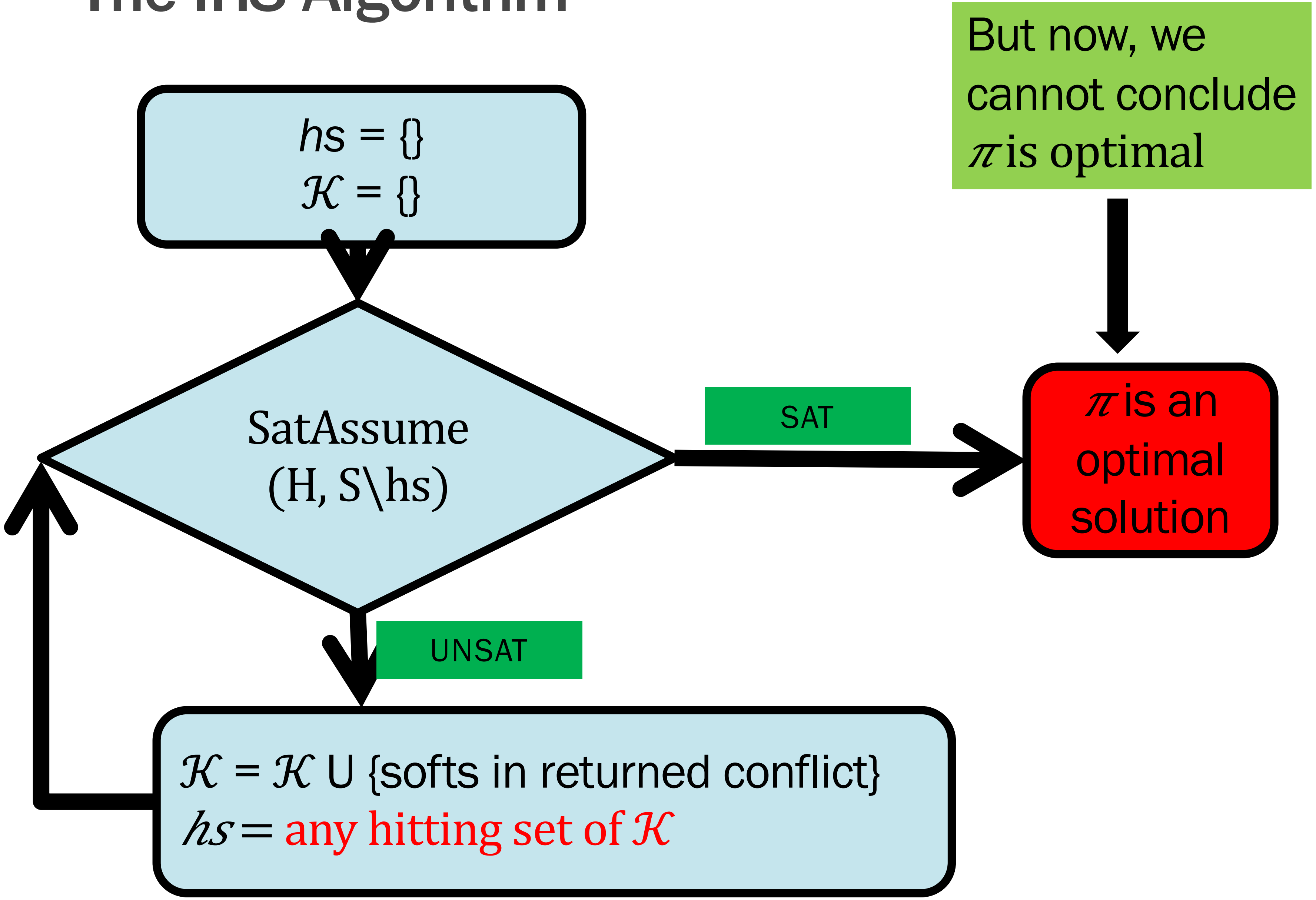

hs = {}
$\mathcal{K}$ = {}
SatAssume (H, S\hs)
SAT
π is an optimal solution
UNSAT
$\mathcal{K}$ = $\mathcal{K}$ U {softs in returned conflict}
hs = any hitting set of $\mathcal{K}$
Use a non minimum cost hitting set instead.

The IHS Algorithm



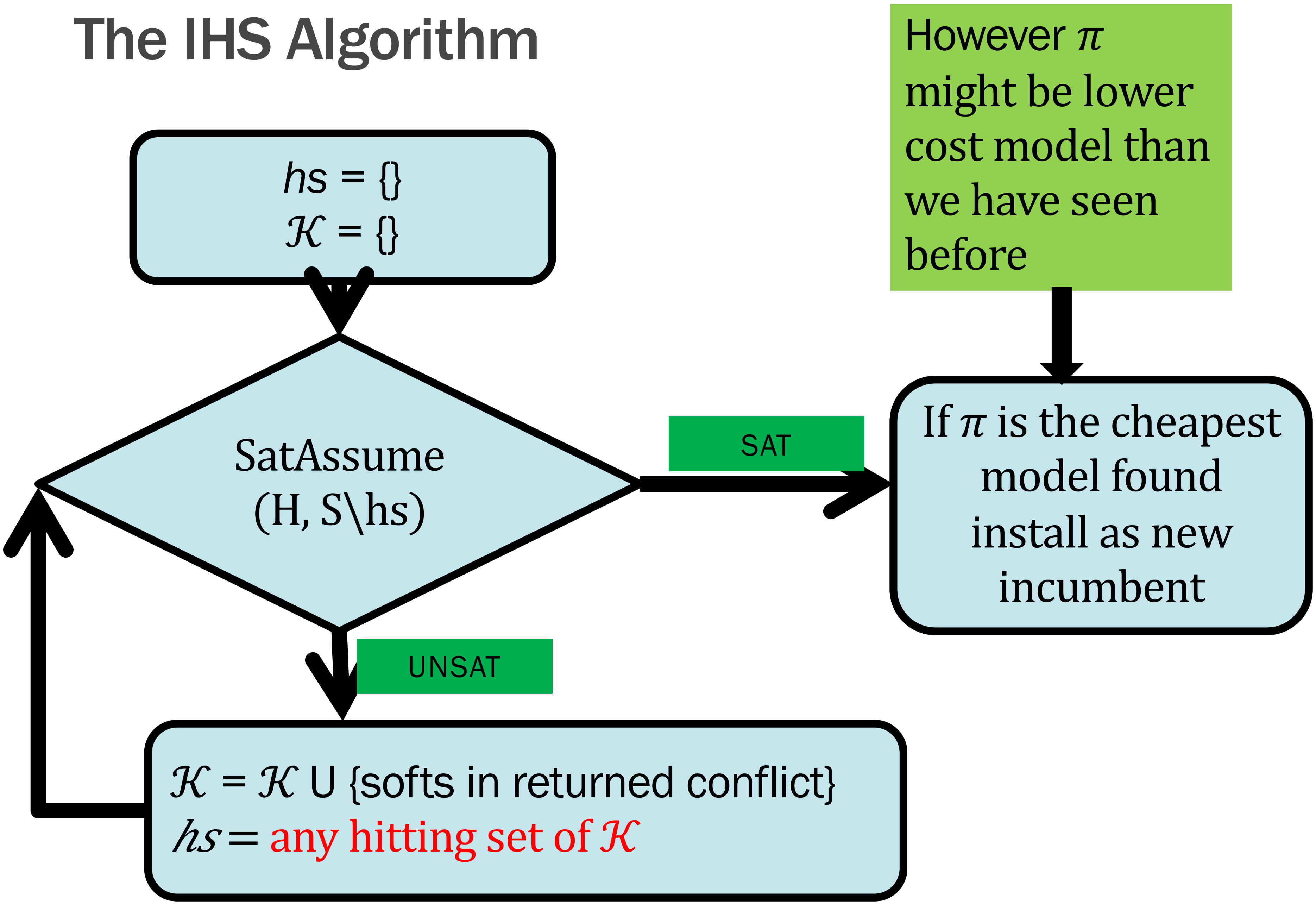
hs = {}
$\mathcal{K}$ = {}
SatAssume (H, S\hs)
SAT
π is an optimal solution
UNSAT
$\mathcal{K}$ = $\mathcal{K}$ U {softs in returned conflict}
$hs =$ any hitting set of $\mathcal{K}$
Ok, always returns new core

The IHS Algorithm



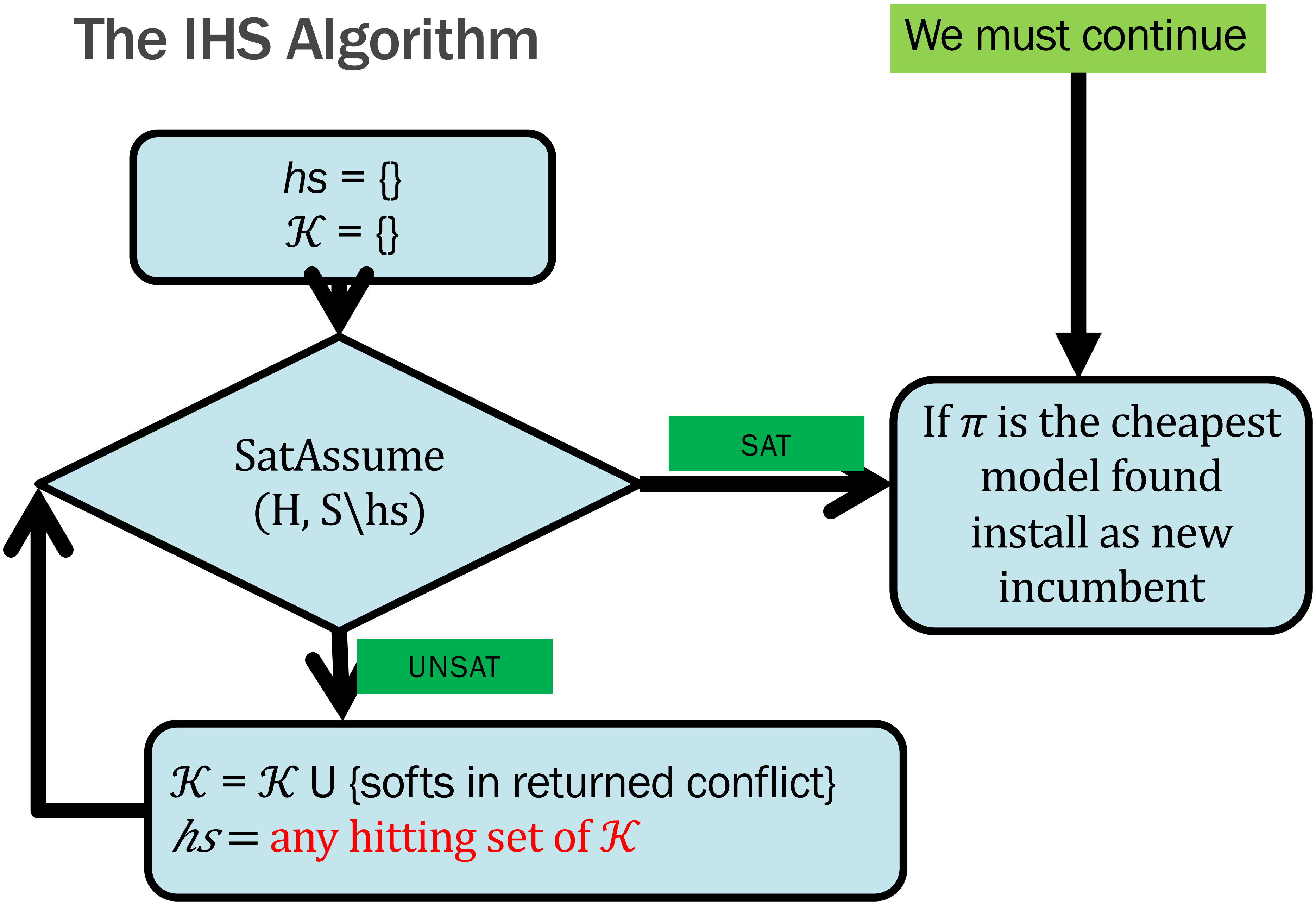
hs = {}
$\mathcal{K}$ = {}
SatAssume (H, S\hs)
SAT
UNSAT
π is an optimal solution
But now, we cannot conclude π is optimal
$\mathcal{K}$ = $\mathcal{K}$ U {softs in returned conflict}
$hs =$ any hitting set of $\mathcal{K}$

The IHS Algorithm

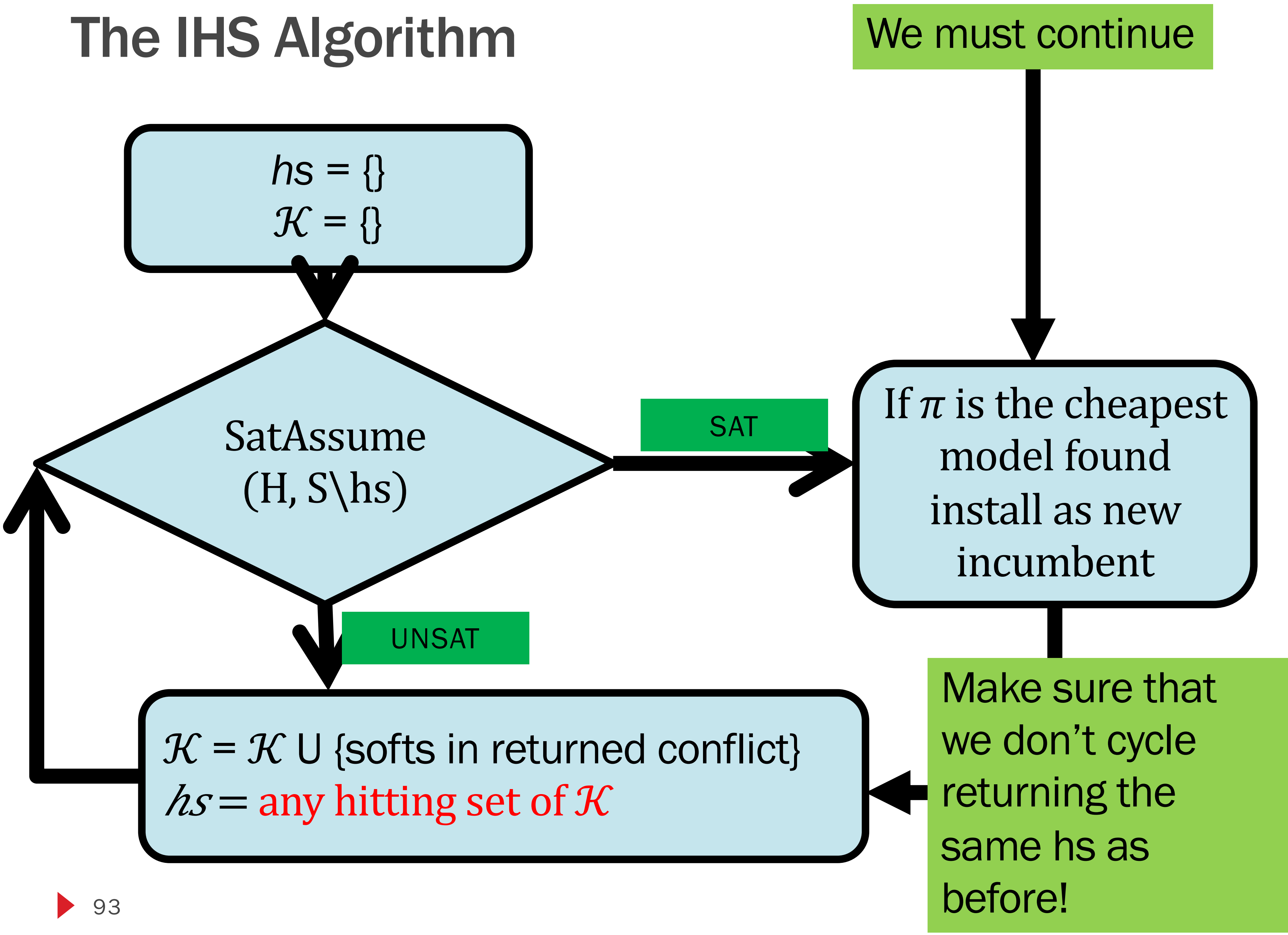


hs = {}
$\mathcal{K}$ = {}
SatAssume (H, S\hs)
SAT
UNSAT
If π is the cheapest model found install as new incumbent
However π might be lower cost model than we have seen before
$\mathcal{K} = \mathcal{K}$ U {softs in returned conflict}
hs = any hitting set of $\mathcal{K}$

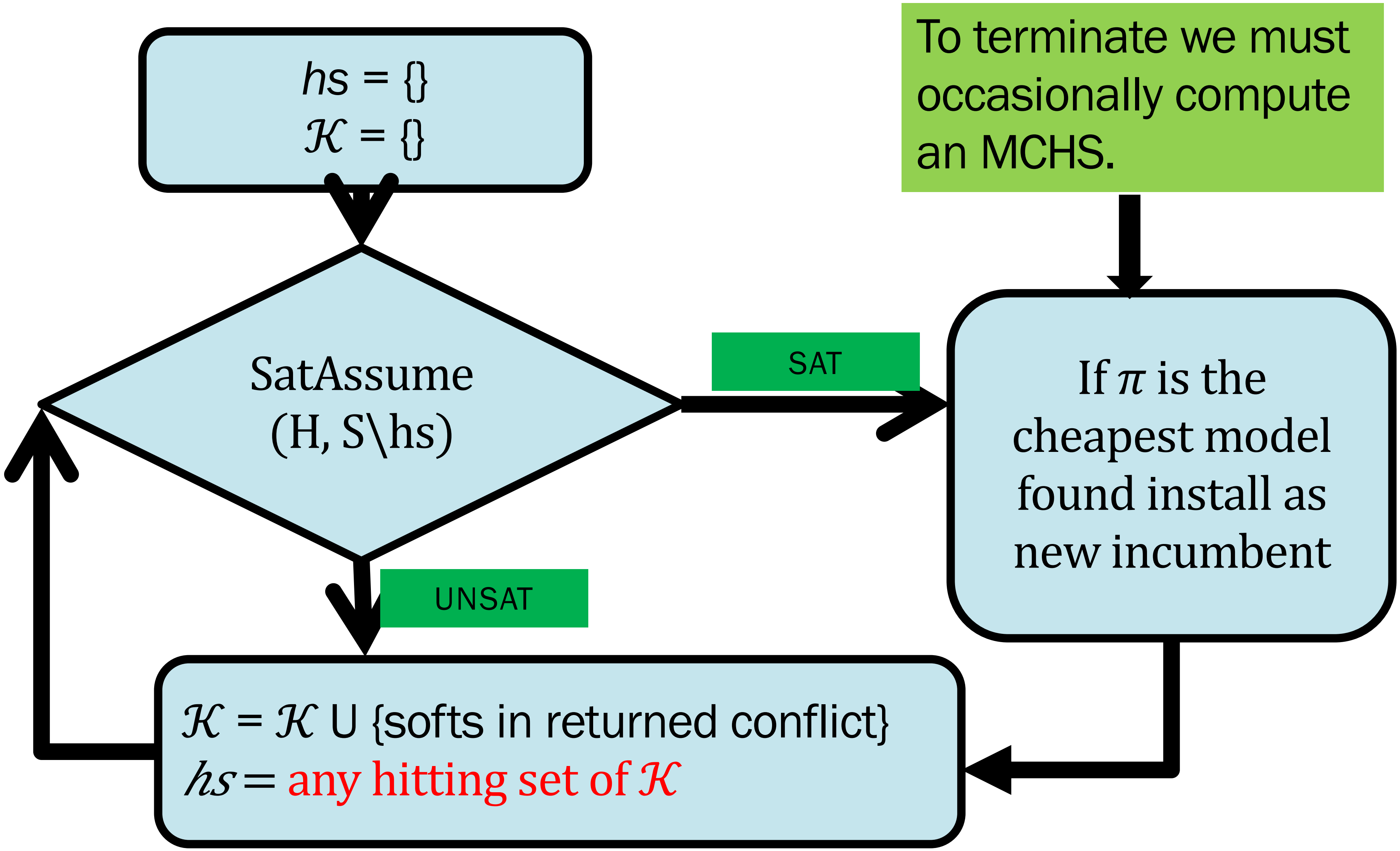
The IHS Algorithm

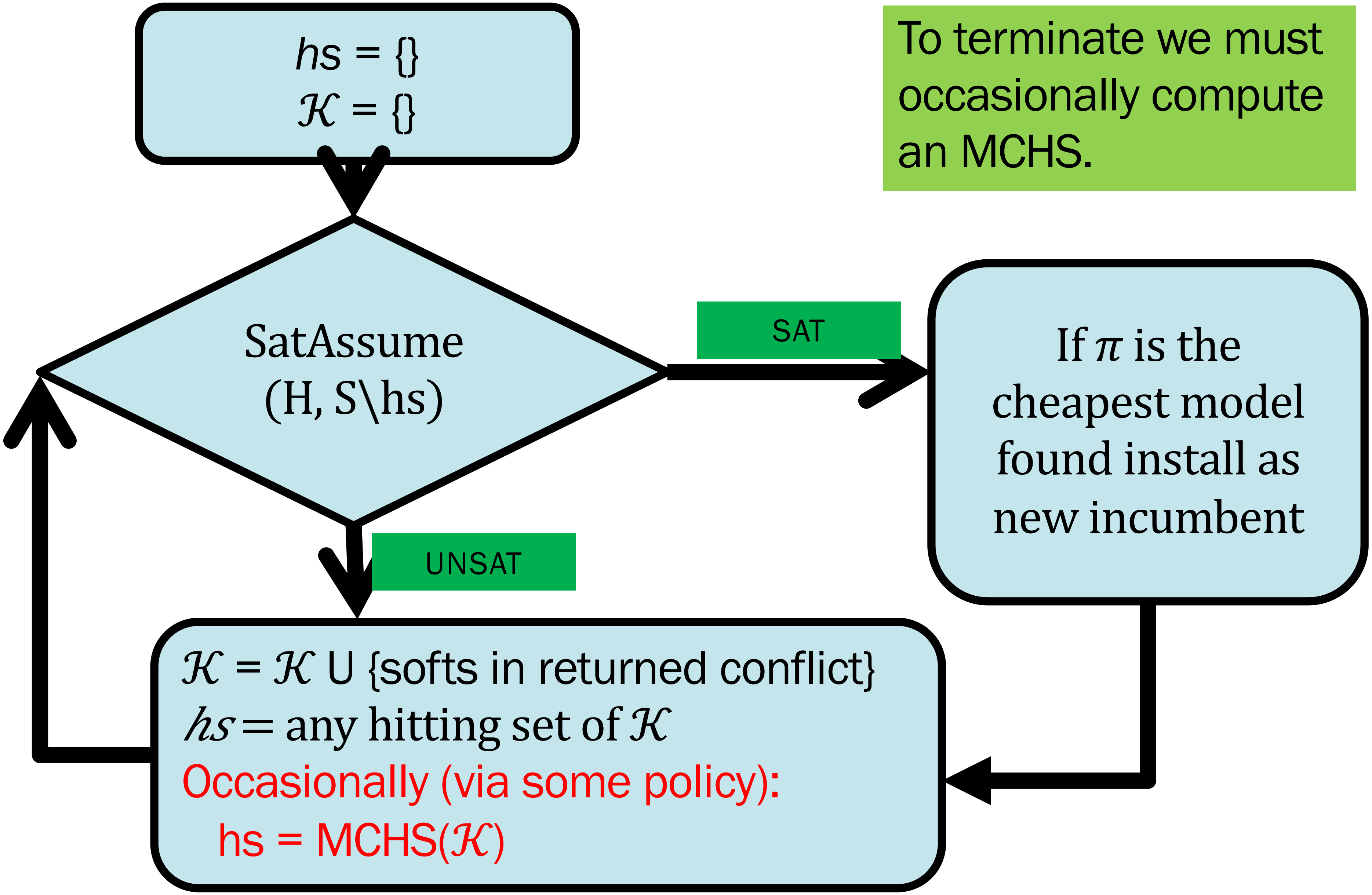

We must continue
hs = {}
$\mathcal{K}$ = {}
SatAssume (H, S\hs)
SAT
If π is the cheapest model found install as new incumbent
UNSAT
$\mathcal{K} = \mathcal{K}$ U {softs in returned conflict}
$hs =$ any hitting set of $\mathcal{K}$

The IHS Algorithm


We must continue
$hs = \{\}$
$\mathcal{K} = \{\}$
SatAssume (H, S\hs)
SAT
If π is the cheapest model found install as new incumbent
UNSAT
$\mathcal{K} = \mathcal{K} \cup$ {softs in returned conflict}
$hs =$ any hitting set of $\mathcal{K}$
Make sure that we don't cycle returning the same hs as before!

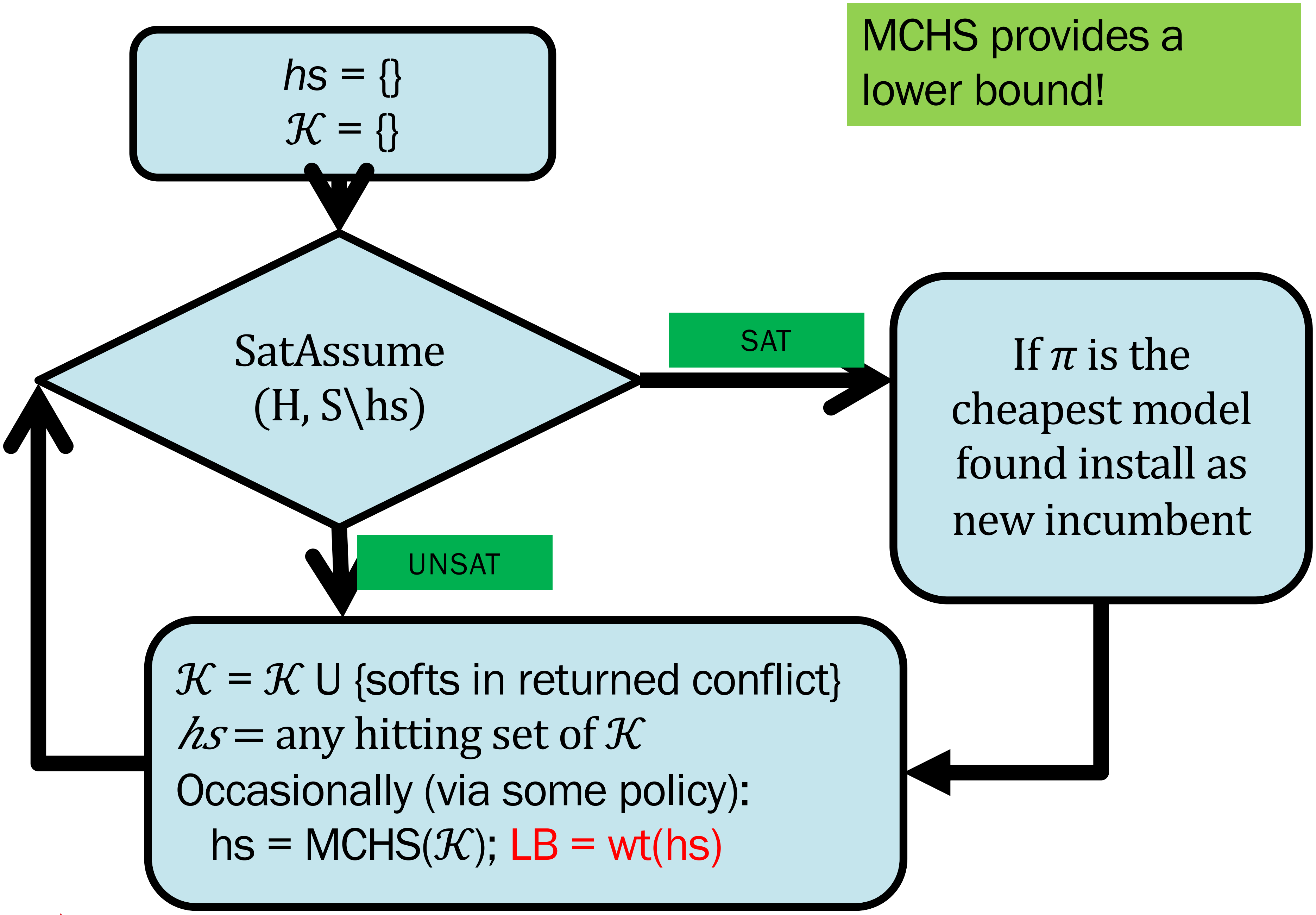
The IHS Algorithm


$hs = \{\}$
$\mathcal{K} = \{\}$
SatAssume (H, S\hs)
SAT
UNSAT
If π is the cheapest model found install as new incumbent
To terminate we must occasionally compute an MCHS.
$\mathcal{K} = \mathcal{K} \cup$ {softs in returned conflict}
$hs =$ any hitting set of $\mathcal{K}$


hs = {}
$\mathcal{K}$ = {}
SatAssume
(H, S\hs)
SAT
If π is the cheapest model found install as new incumbent
UNSAT
$\mathcal{K}$ = $\mathcal{K}$ U {softs in returned conflict}
hs = any hitting set of $\mathcal{K}$
Occasionally (via some policy):
hs = MCHS($\mathcal{K}$)

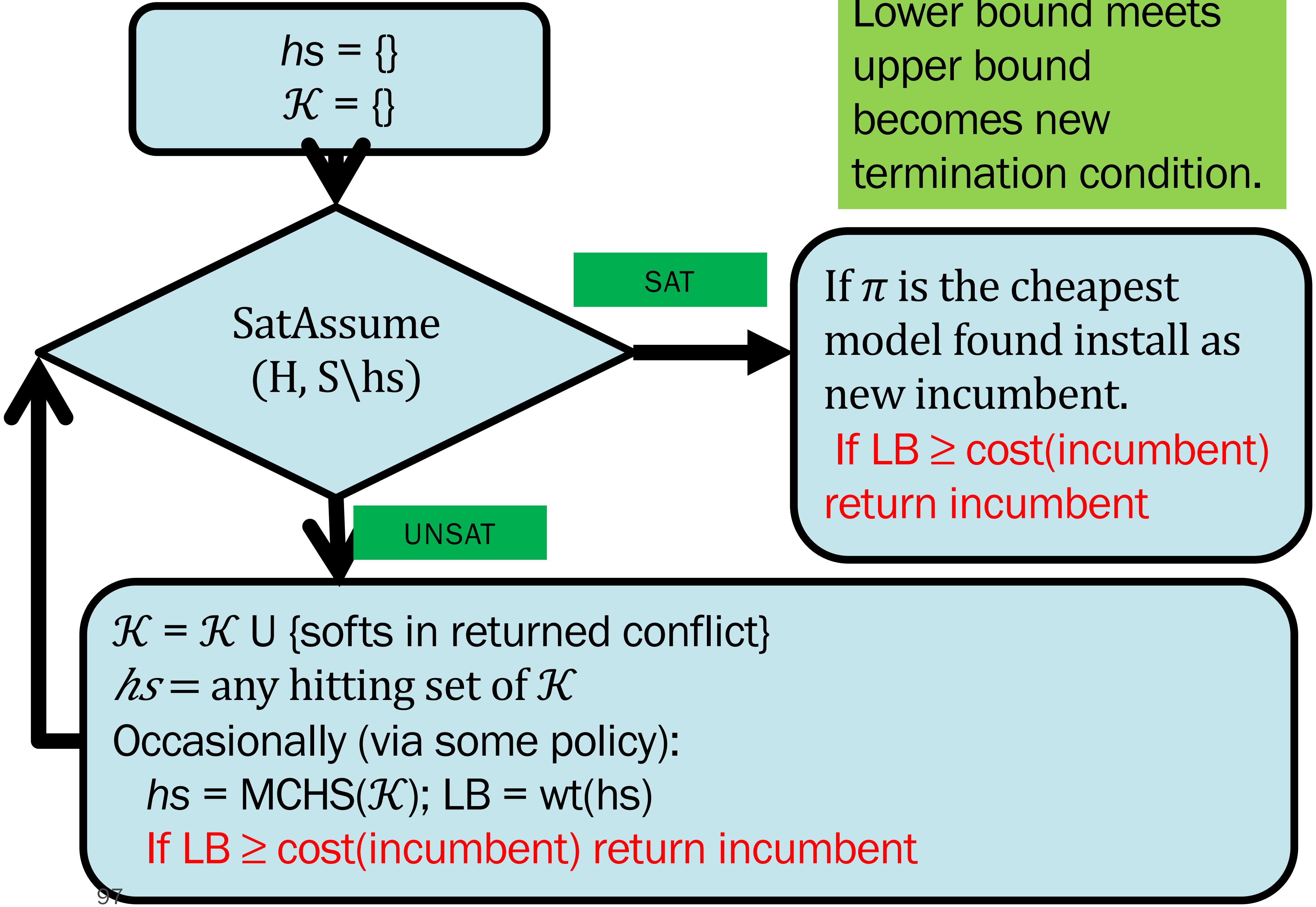
To terminate we must occasionally compute an MCHS.

MCHS provides a lower bound!



hs = {}
$\mathcal{K}$ = {}
SatAssume (H, S\hs)
SAT
UNSAT
If π is the cheapest model found install as new incumbent
$\mathcal{K} = \mathcal{K}$ U {softs in returned conflict}
hs = any hitting set of $\mathcal{K}$
Occasionally (via some policy):
hs = MCHS($\mathcal{K}$); LB = wt(hs)

Lower bound meets upper bound becomes new termination condition.



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

SatAssume (H, S\hs)

SAT

If π is the cheapest model found install as new incumbent.
If LB ≥ cost(incumbent) return incumbent

UNSAT

$\mathcal{K} = \mathcal{K}$ U {softs in returned conflict}
hs = any hitting set of $\mathcal{K}$
Occasionally (via some policy):
  hs = MCHS($\mathcal{K}$); LB = wt(hs)
  If LB ≥ cost(incumbent) return incumbent

IHS Algorithm

- As long at 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**

IHS on our Example.

- F = H U S
- H = $\sum_{i=1}^{2n} x_i < n$ Generalized to 2n unit softs.
- S = {(x_1), (x_2), …, (x_{2n})}
- Optimal solution has cost n+1.
- [Davies 2013] The IHS approach needs to accumulate at least $\binom{2n}{n}$ cores in $\mathcal{K}$ before wt(MCHS($\mathcal{K}$)) = n+1.
    - For any set of cores X of size with |X| < $\binom{2n}{n}$, $\mathrm{MCHS(X)} < \mathrm{n} + 1$
- So the IHS approach has to make an exponential number of calls to the SAT solver in this case.

Weighted Case (Intuition)

- H = $\left\{\begin{matrix} (\neg y_{11} & \dots & \neg y_{1k} & \neg z) \\ \vdots & \vdots & \vdots & \\ (\neg y_{m1} & \dots & \neg y_{mk} & \neg z) \end{matrix}\right\}$

- S = {(y_{11}), …, (y_{mk}), (z)}

- Weight of all softs is 2 except weight of wt(**z)=**2k-1.

- Every clause specifies a core. IHS would find k cores, all by unit prop.
- In the i-th iteration (1 ≤ i ≤ k-1) the MCHS would have weight 2i, and would consist of one y_{ij} from each of the i cores found so far.
- At step k, the MCHS would be computed to be {(z)}, with cost 2k-1.
- With hs = {(z)} we get SAT.

Weighted Case.

- H = $\left\{\begin{matrix} (\neg y_{11} & \dots & \neg y_{1k} & \neg z) \\ \vdots & \vdots & \vdots & \\ (\neg y_{m1} & \dots & \neg y_{mk} & \neg z) \end{matrix}\right\}$

- S = {(y_{11}), …, (y_{mk}), (z)}

- Weight of all softs is 2 except weight of wt(**z)=**2k-1.
- The sequence of sat approaches with clause cloning would do something more complex.
- Each UNSAT call would add a cardinality constraint over the variables in the core, and would clone z.
- So the formula gets harder to solve—not clear if this could lead to a non-polynomial separation.

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.
  - It would be good get some theoretical results here.

Implicit Hitting Set Solvers.

- As with the sequence of SAT approaches, it is not clearly understood how to guide the SAT solver to find good cores (cores that move the MCHS bound up more rapidly).

Thank you