SAT, SMT and Applications

Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University
Linz, Austria

Invited Talk

LPNMR'09

10th International Conference on
Logic Programming and Nonmonotonic Reasoning

Potsdam, Germany

Thursday, September 17, 2009

- propositional logic:
  - variables **tie** **shirt**
  - negation $\neg$ (not)
  - disjunction $\vee$ disjunction (or)
  - conjunction $\wedge$ conjunction (and)

- three conditions / clauses:
  - clearly one should not wear a **tie** without a **shirt** $\neg\textbf{tie} \vee \textbf{shirt}$
  - not wearing a **tie** nor a **shirt** is impolite $\textbf{tie} \vee \textbf{shirt}$
  - wearing a **tie** and a **shirt** is overkill $\neg(\textbf{tie} \wedge \textbf{shirt}) \equiv \neg\textbf{tie} \vee \neg\textbf{shirt}$

- is the formula $(\neg\textbf{tie} \vee \textbf{shirt}) \wedge (\textbf{tie} \vee \textbf{shirt}) \wedge (\neg\textbf{tie} \vee \neg\textbf{shirt})$ satisfiable?

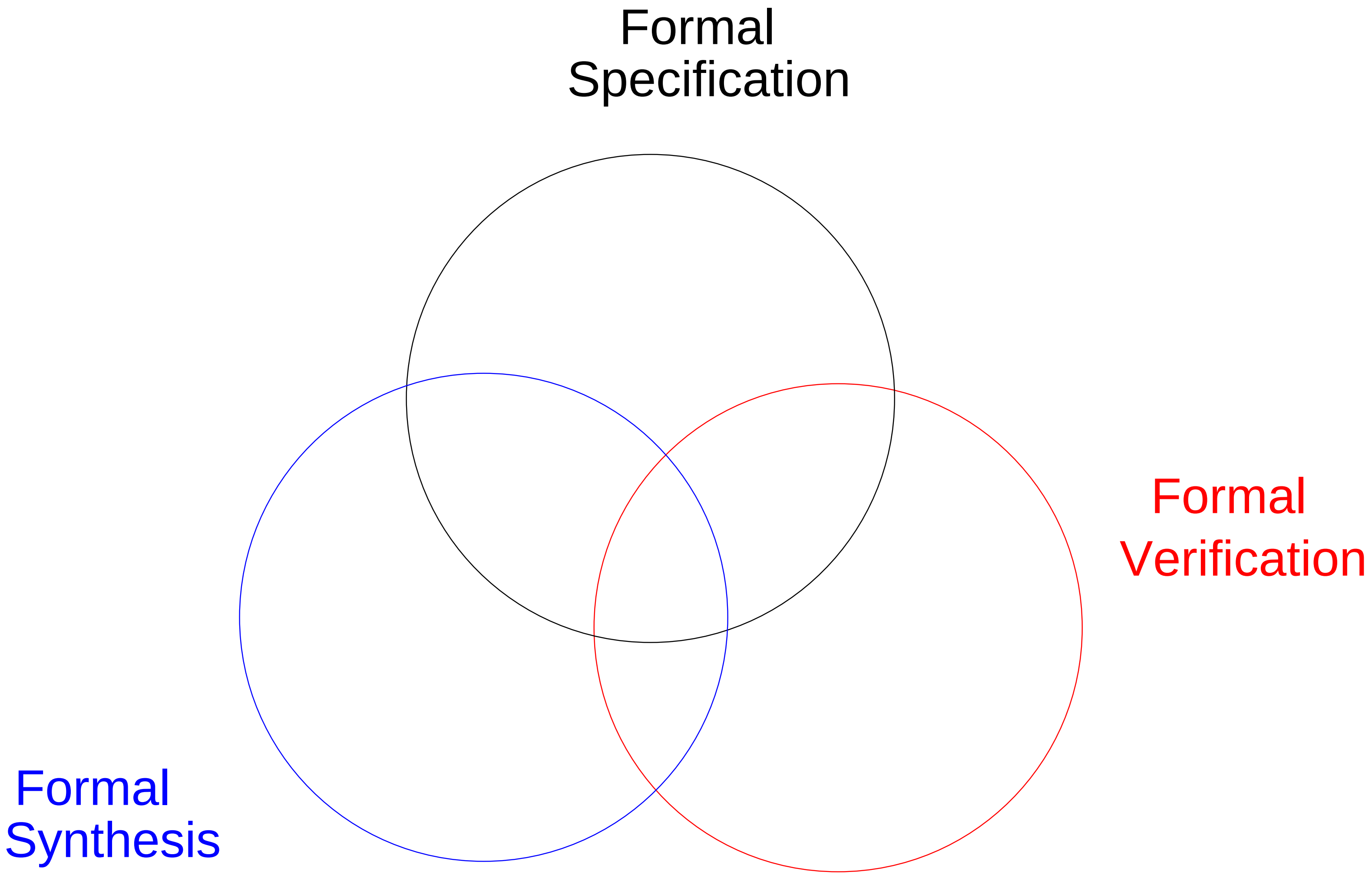
What is SAT?

- a class of rather low-level kind of problems:
  - propositional variables only, e.g. either hold (true) or not (false)
  - logic operators $\neg$, $\vee$, $\wedge$, actually restricted to conjunctive normal form (CNF)
  - but no quantifiers such as “for all such things”, or “there is one such thing”
  - can we find an assignment of the variables to true or false, such that a set of clauses is satisfied simultaneously
- theory: it is **the** standard NP complete problem [Cook’70]
- encoding: how to get your problem into CNF
- simplifying: how can the problem or the CNF be simplified (preprocessing)
- solving: how to implement fast solvers

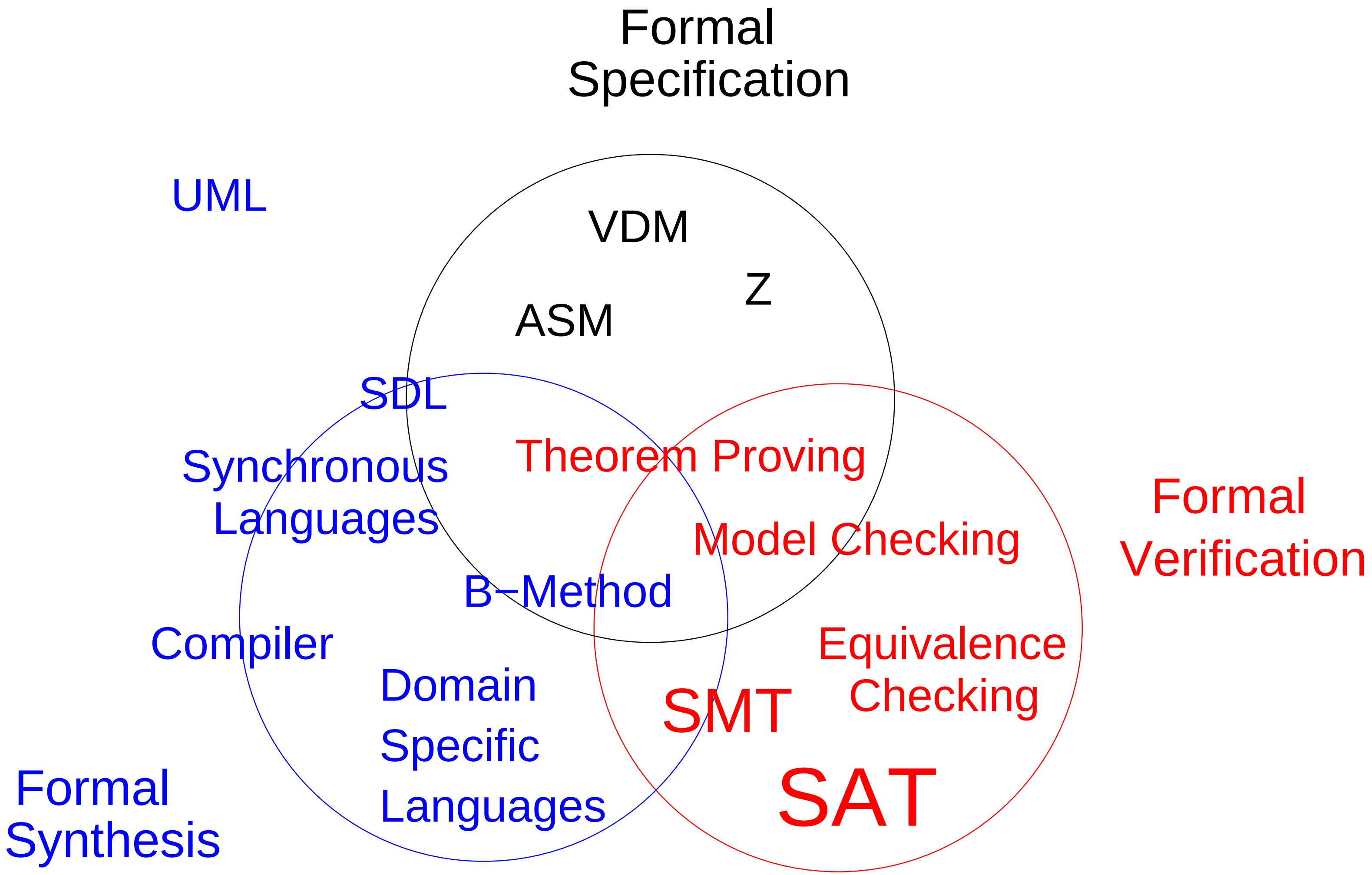
What is SMT?

- satisfiability solving for first order formulae
  - interpreted over fixed theories
  - usually without quantifiers
  - fully automatic decision procedures which also can provide models
- theories of interest
  - equality, uninterpreted functions
  - real / integer arithmetic
  - bit-vectors
  - arrays
- particularly important are bit-vectors and arrays for HW/SW verification

Formal Methods in Computer Science



Formal Methods and Satisfiability (SAT)



Applications of SAT and SMT

- bounded model checking in electronic design automation (EDA)
  - routinely used for falsification in all major design houses
  - unbounded extensions also use SAT technology
- SAT as working horse in static software verification
- static device driver verification at Microsoft (SLAM, SDV)
  - predicate abstraction with SMT solvers
  - spurious counter example checking
- software configuration, e.g. Eclipse IDE ships with SAT4J
- cryptanalysis and solving other combinatorial problems

A Short SAT Solver History

up to and including 2nd version of MiniSAT and SatELite

- Davis and Putnam procedure
  - DP: elimination procedure [DavisPutnam'60]
  - DPLL: splitting [DavisLogemannLoveland'62]
- modern SAT solvers are mostly based on DPLL
  - learning: GRASP [MarquesSilvaSakallah'96], RelSAT [BayardoSchrag'97]
  - watched literals, VSIDS, mChaff [MoskewiczMadiganZhaoZhangMalik-DAC'01]
  - improved heuristics: MiniSAT [EénSörensson-SAT'03] actually version from 2005
- preprocessing is still a hot topic:
  - currently fastest solvers use SatELite style preprocessing [EénBiere'05] DP
  - our solver PrecoSAT won the industrial category of last SAT competition

[EénSörensson-SAT'03]

- originally was a clean and compact reimplementation of
  - Satzoo, Satnik which are based on
  - zChaff, Limmat which in turn are based on
  - mChaff [MoskewiczMadiganZhaoZhangMalik-DAC'01] which is based on
  - Grasp [MarquesSilvaSakallah96] which in turn is based on?
  - triggered by the success of Satzoo in the SAT competition 2003
- bridged gap between (complicated) implementations and high-level descriptions
- made it possible to understand the inner workings of a SAT solver for a broad audience

Improved MiniSAT 1.13 / 1.14

[EénSörensson'05]

- very competitive performance in 2005
  - nobody could catch up until 2007
  - faster than any other solver in the competition
  - except for SateELiteGTI which however used MiniSAT as backend
- replaced "zChaff" as **quasi standard** open source SAT solver
- improvements:
  - careful tuning of some magic constants (clause reduction ratio, restart intervals)
  - precise decision scheduler
  - learned clause minimization
- most newer SAT solvers are based on MiniSAT technology

Variable State Independent Decaying Sum (VSIDS)

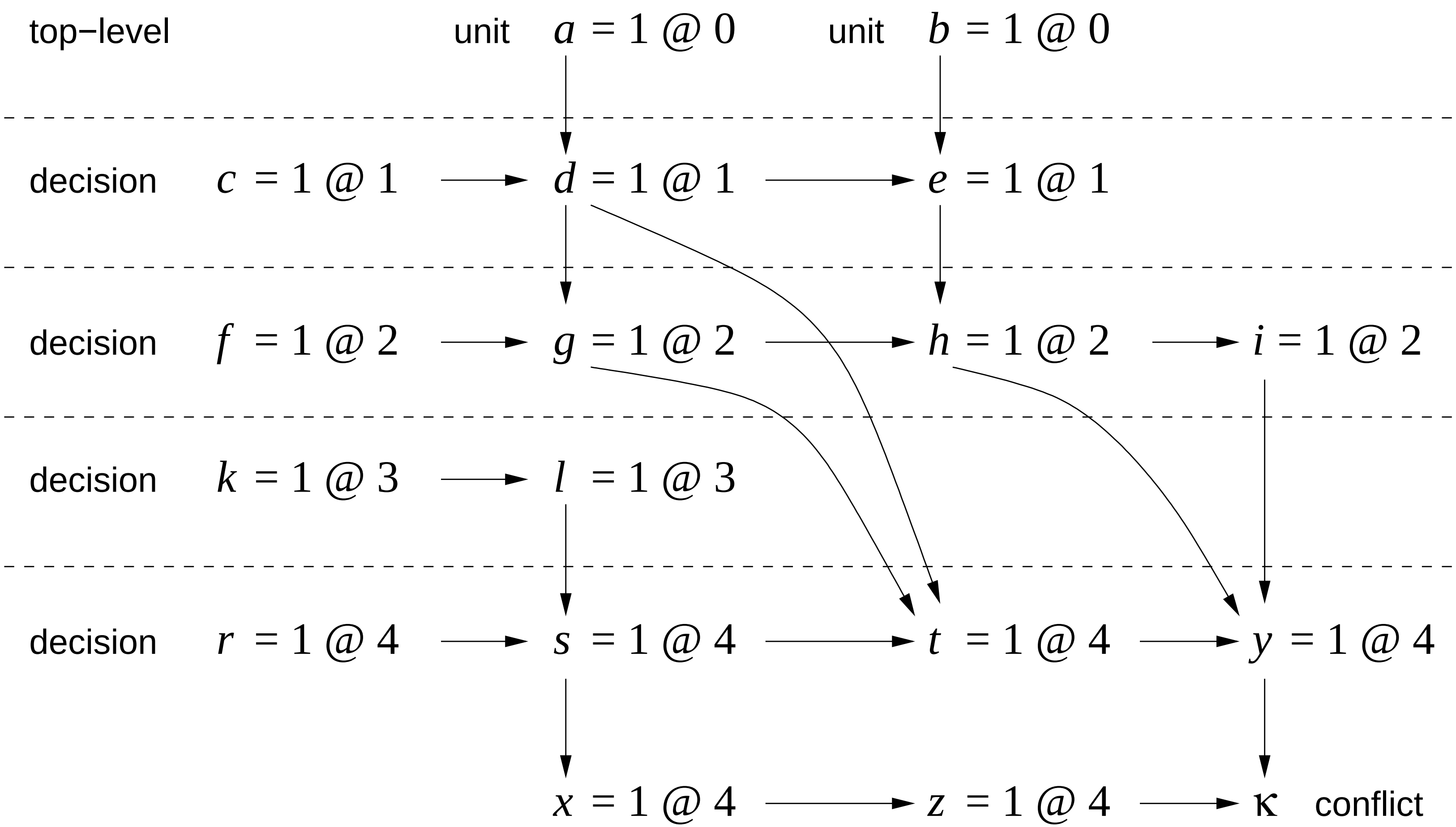
mChaff, zChaff, [MoskewiczMadiganZhaoZhangMalik-DAC'01]

- which variable to pick?
  - alternatives are to **statically** "schedule" variables in the same order
  - or pick one that **greedily** satisfies the largest number of unsatisfied clauses
  - or monitor **dynamically** involvement of variables in conflicts
- variable state independent **decaying** sum (VSIDS) heuristic:
  - involvement of variable in conflict increments its score
  - score is divided by two at every 256th conflict
  - always pick variable with largest score
- VSIDS localizes search and thus finds short proofs

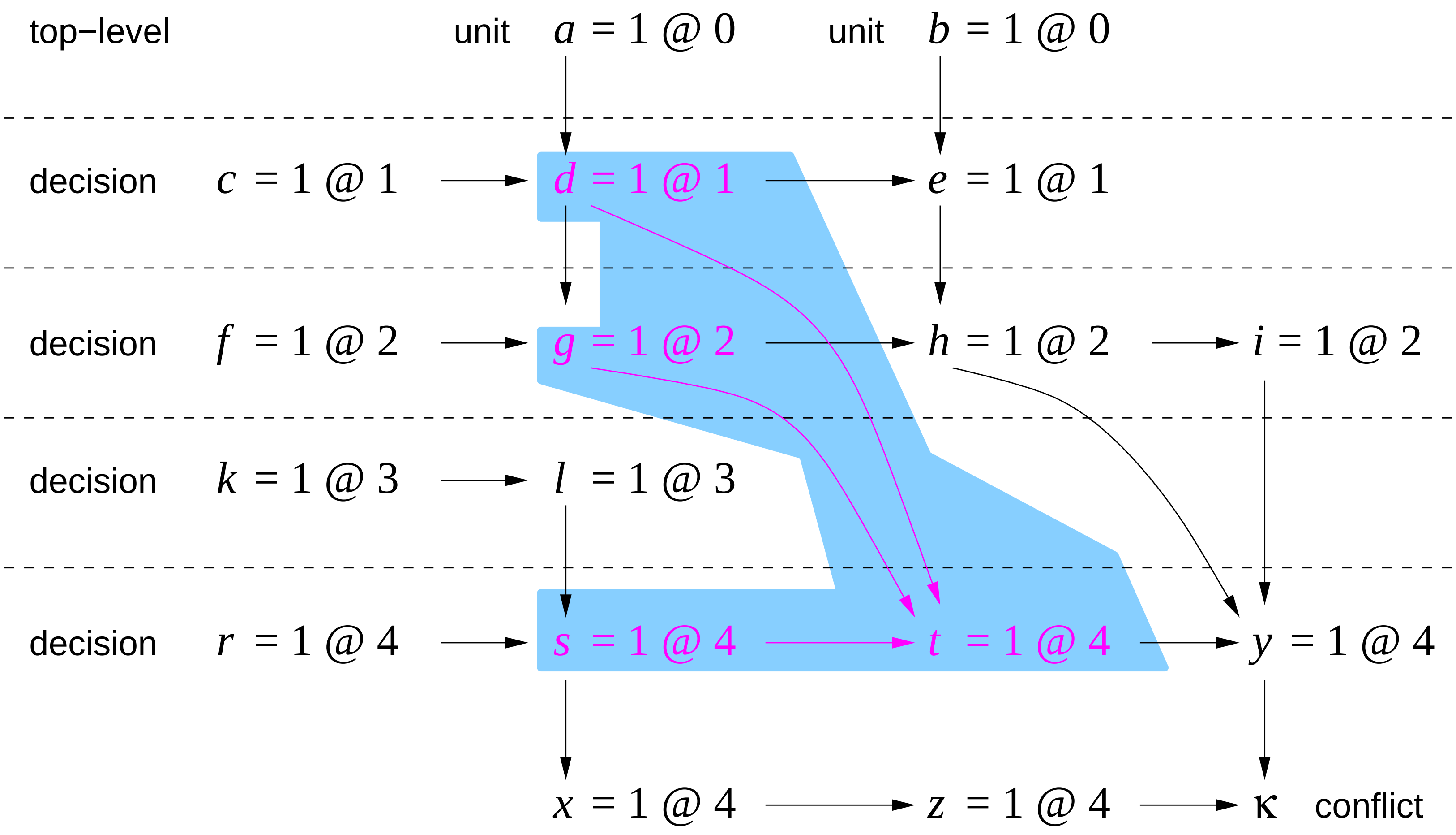
Precise Variable Scheduler

- searching through all variables at every decision point is too expensive
- zChaff's **imprecise** solution:
  - sort variables every 256 conflicts
  - pick first unassigned variable in this sorted list (may not have largest score)
- Jerusat's, NanoSAT's **slow** solution:
  - priority queue of unassigned variables (updates are logarithmic)
  - decisions usually force 2 orders of magnitude more assignments
- Niklas Sörensson's **precise** and **fast** solution:
  - keep assigned variables in priority queue, remove them if they have largest score
  - add variables back while backtracking

Implication Graph

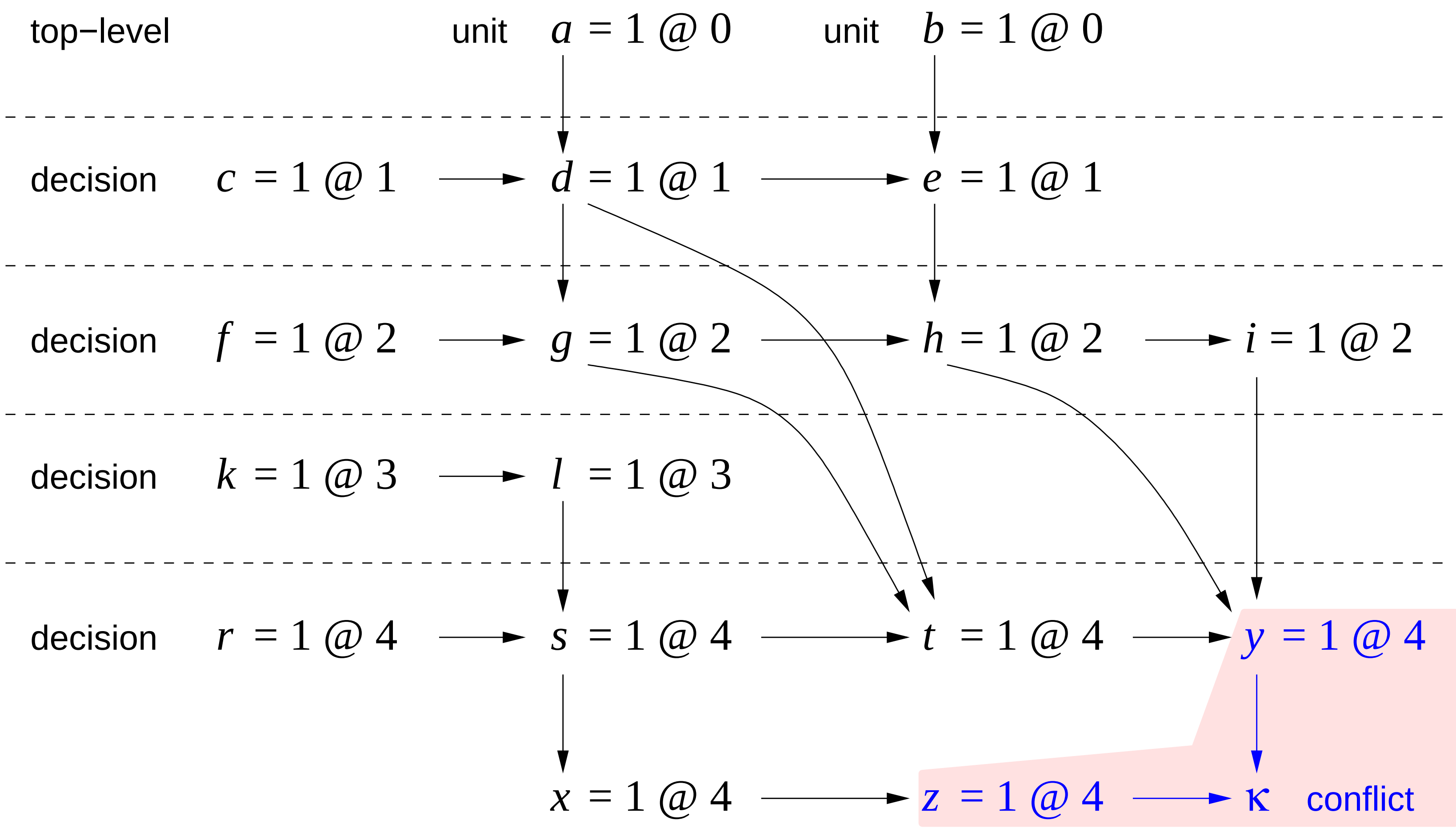


Antecedents / Reasons



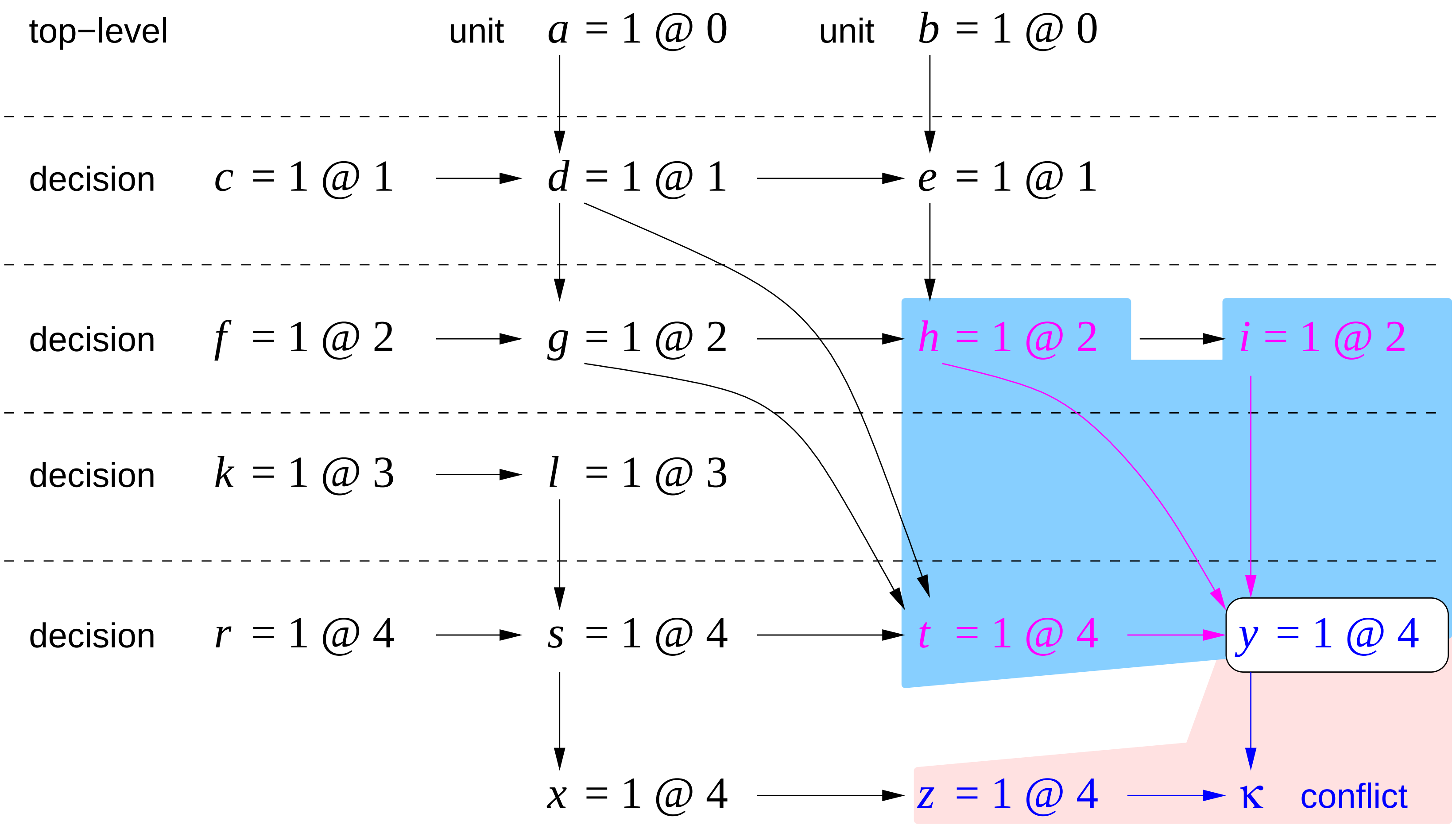
$$d \wedge g \wedge s \rightarrow t \qquad \equiv \qquad (\overline{d} \vee \overline{g} \vee \overline{s} \vee t)$$

Conflicting Clauses



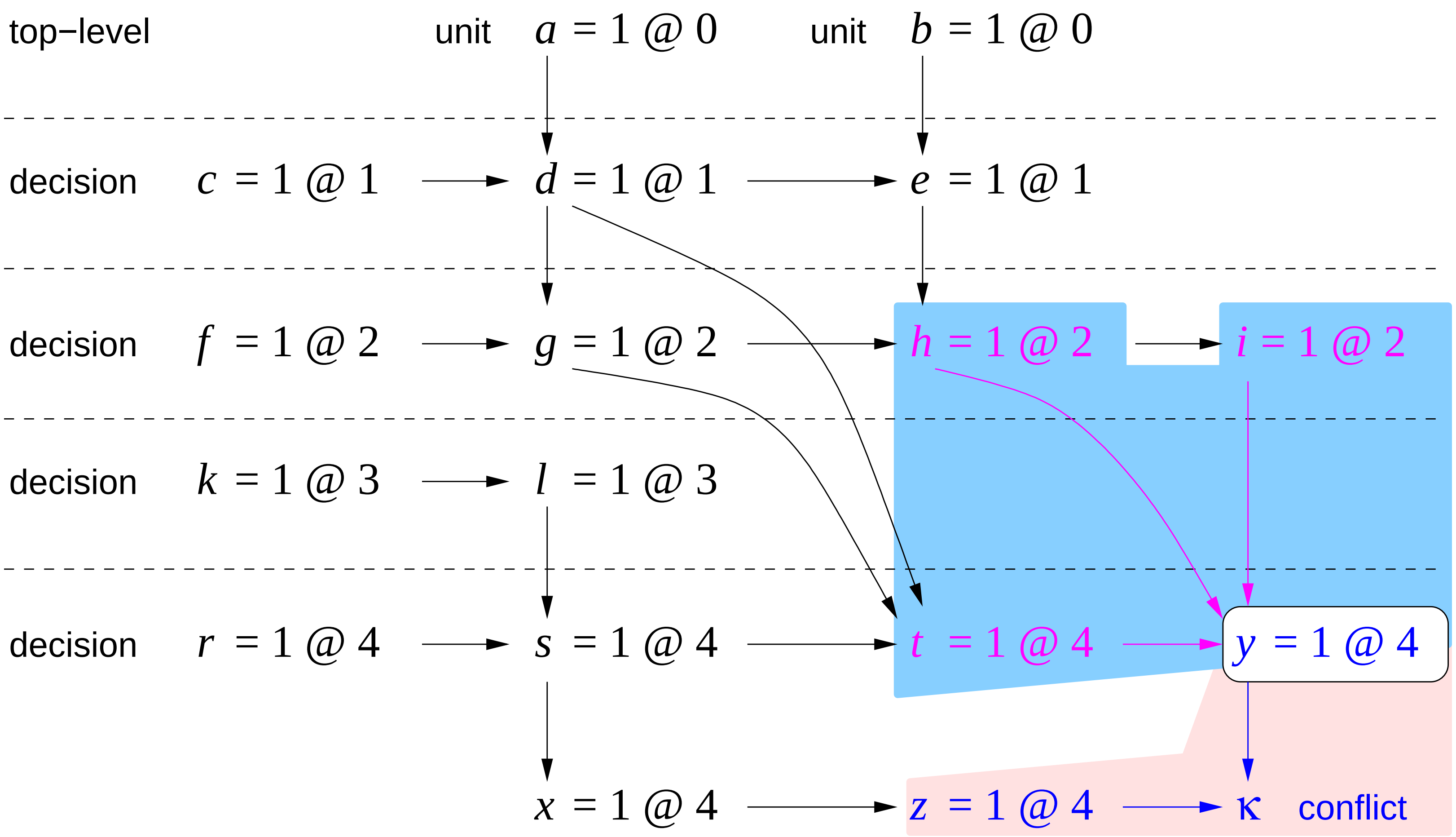
$$\neg(y \wedge z) \quad \equiv \quad (\bar{y} \vee \bar{z})$$

Resolving Antecedents 1st Time



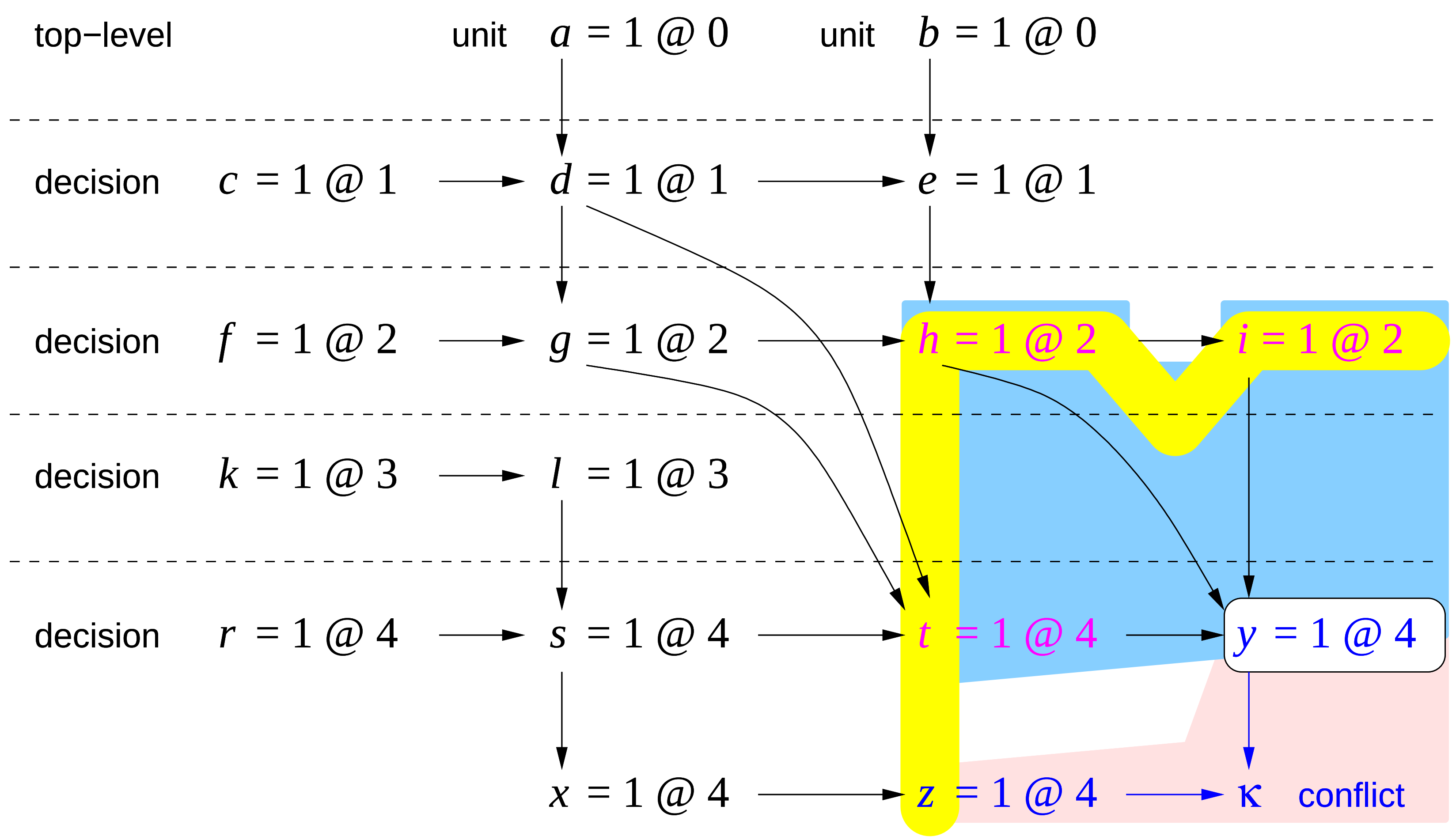
$(\overline{h} \vee \overline{i} \vee \overline{t} \vee y)$ $(\overline{y} \vee \overline{z})$

Resolving Antecedents 1st Time



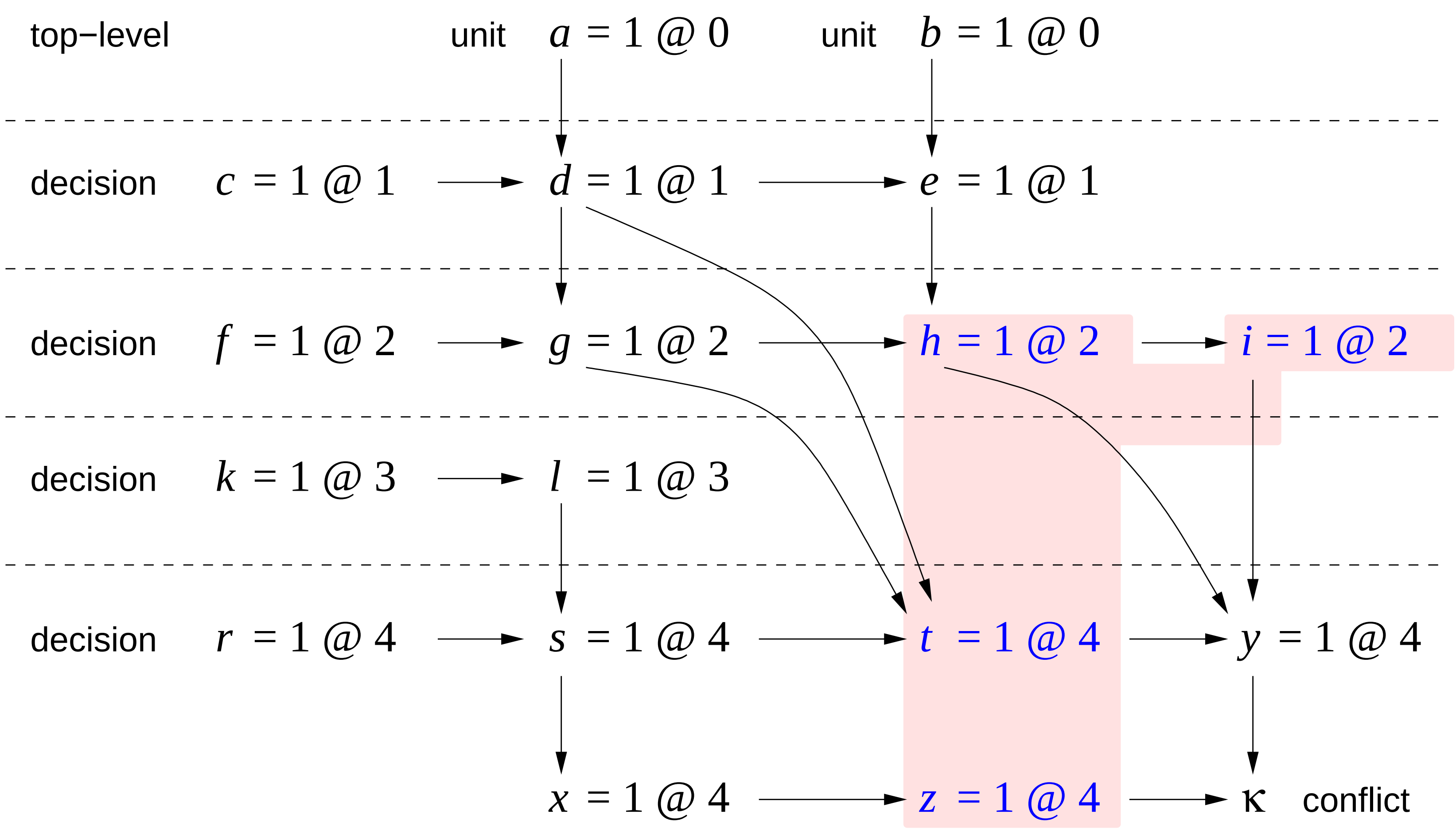
$$\frac{(\bar{h} \vee \bar{i} \vee \bar{t} \vee y) \qquad (\bar{y} \vee \bar{z})}{(\bar{h} \vee \bar{i} \vee \bar{t} \vee \bar{z})}$$

Resolvents = Cuts = Potential Learned Clauses



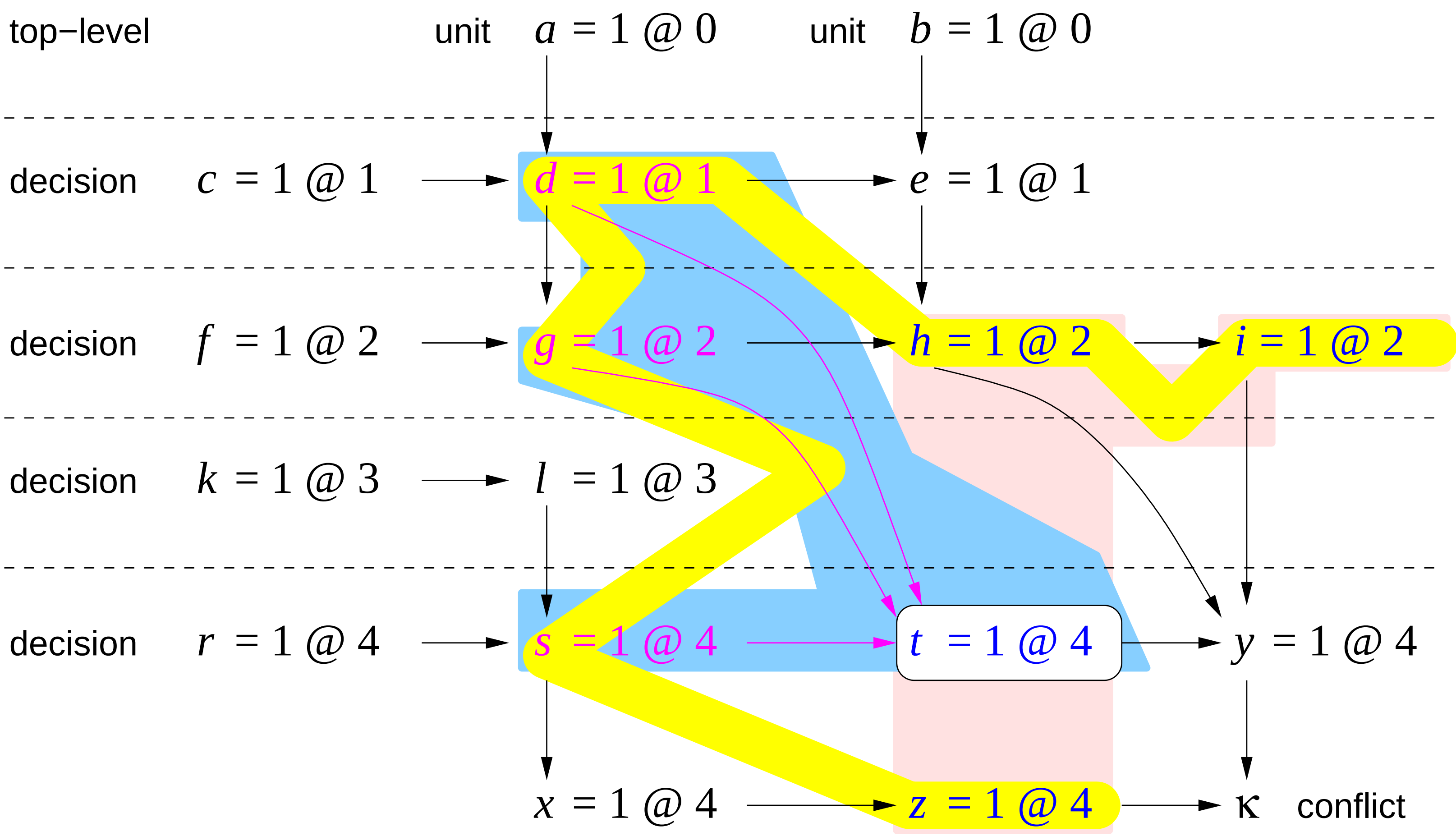
$$\frac{(\bar{h} \vee \bar{i} \vee \bar{t} \vee y) \qquad (\bar{y} \vee \bar{z})}{(\bar{h} \vee \bar{i} \vee \bar{t} \vee \bar{z})}$$

Potential Learned Clause After 1 Resolution



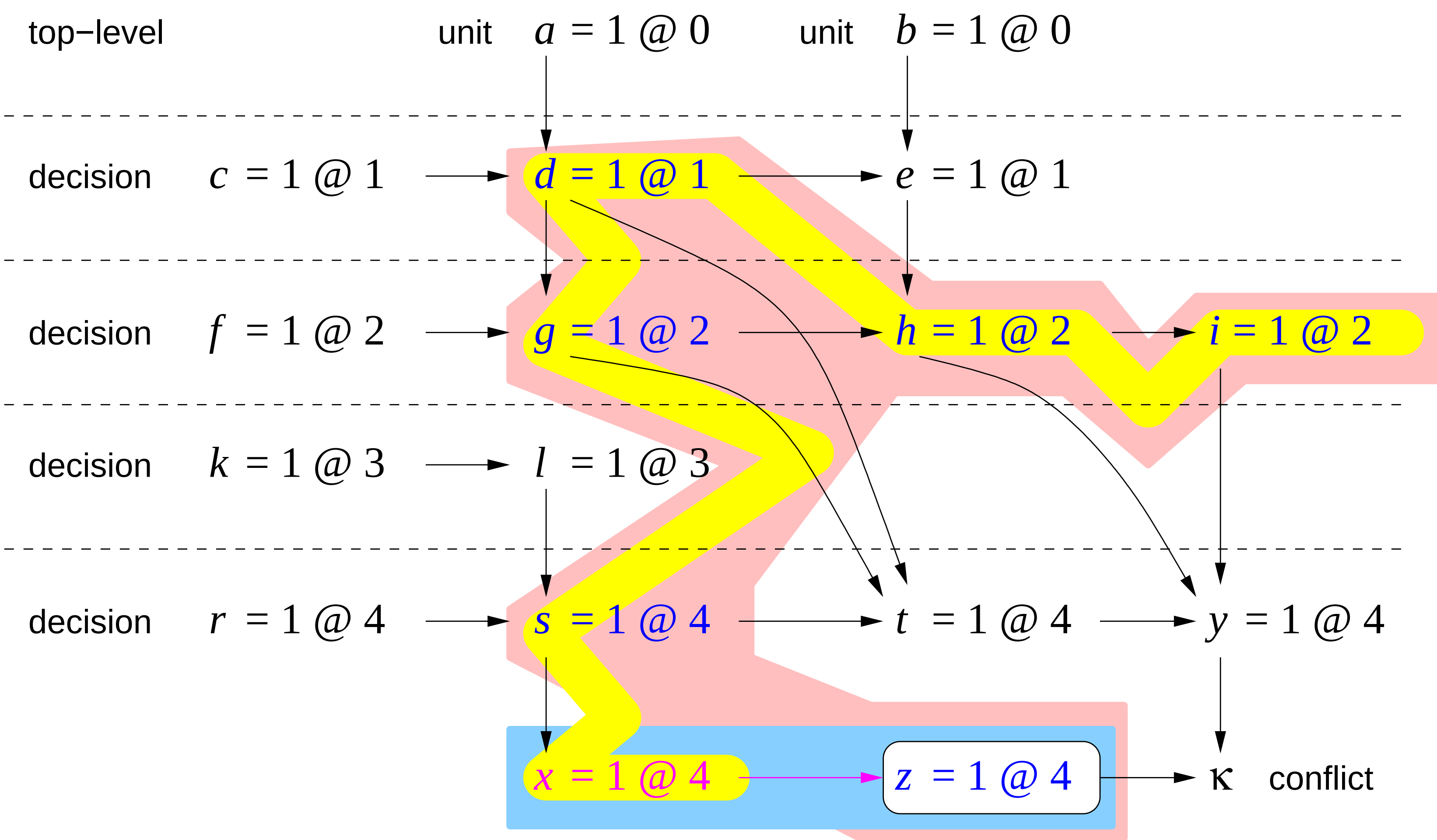
$$(\overline{h} \vee \overline{i} \vee \overline{t} \vee \overline{z})$$

Resolving Antecedents 2nd Time



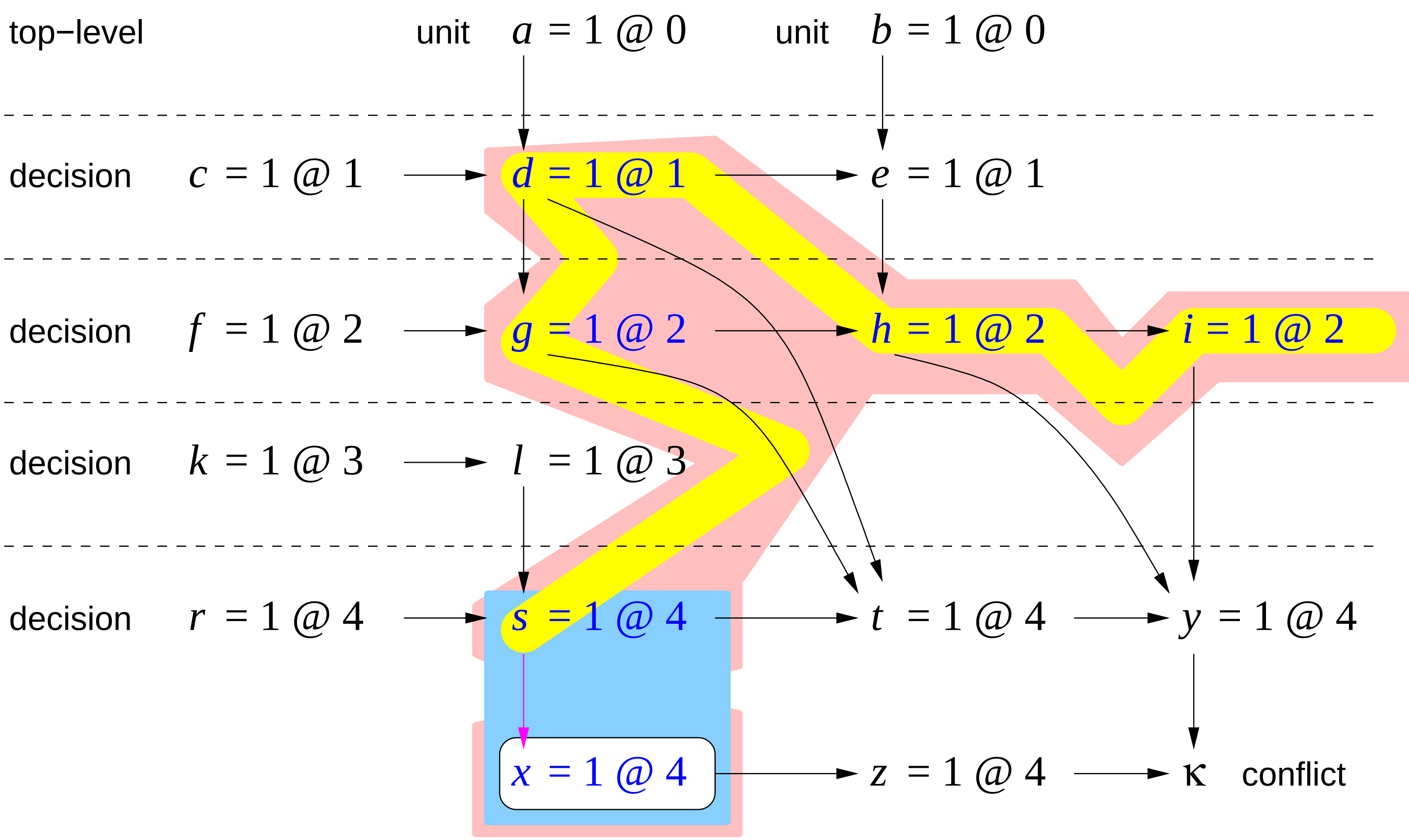
$$\frac{(\overline{d} \vee \overline{g} \vee \overline{s} \vee t) \qquad (\overline{h} \vee \overline{i} \vee \overline{t} \vee \overline{z})}{(\overline{d} \vee \overline{g} \vee \overline{s} \vee \overline{h} \vee \overline{i} \vee \overline{z})}$$

Resolving Antecedents 3rd Time



$$\frac{(\bar{x} \vee z) \qquad (\bar{d} \vee \bar{g} \vee \bar{s} \vee \bar{h} \vee \bar{i} \vee \bar{z})}{(\bar{x} \vee \bar{d} \vee \bar{g} \vee \bar{s} \vee \bar{h} \vee \bar{i})}$$

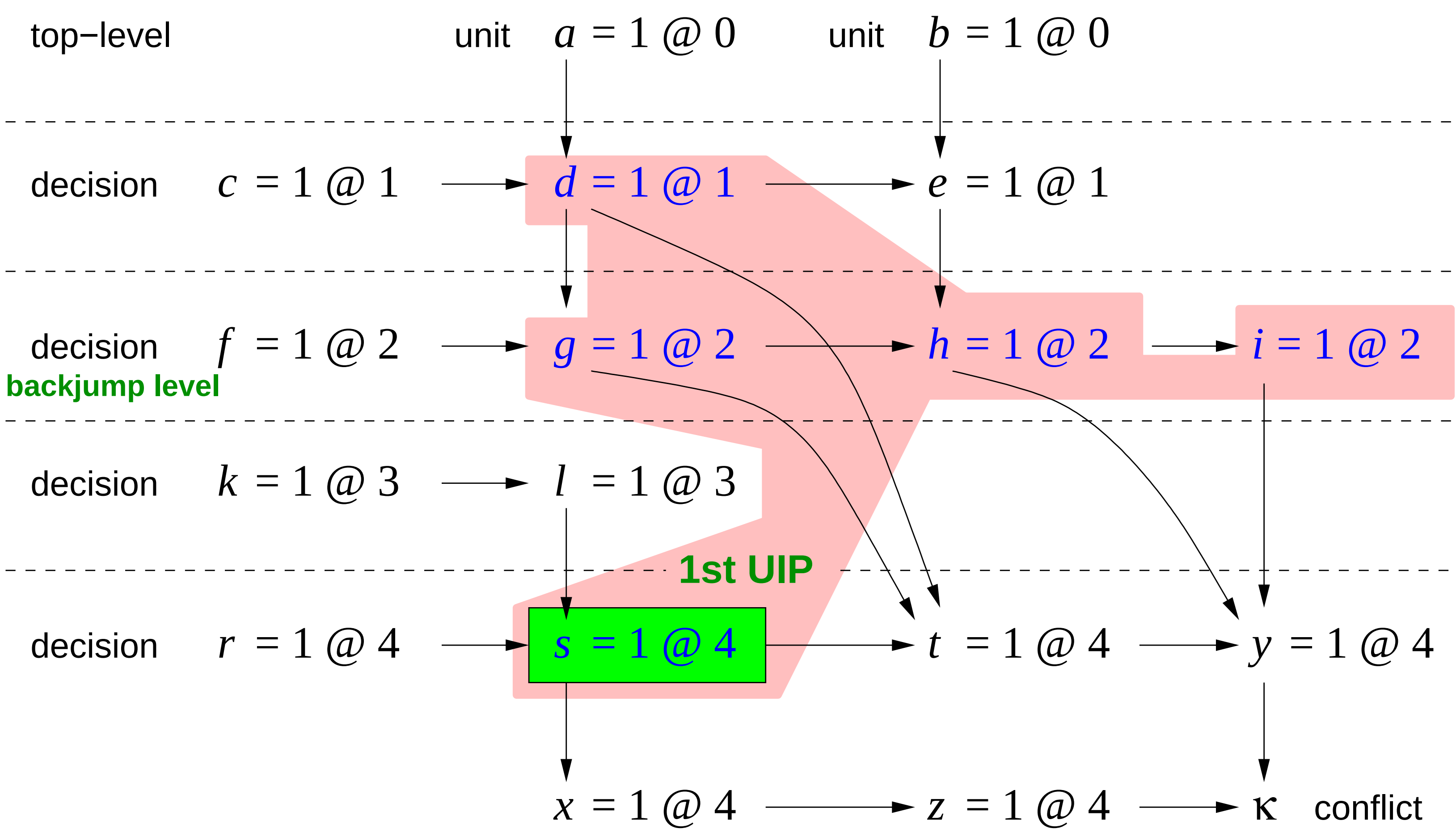
Resolving Antecedents 4th Time



$$\frac{(\bar{s} \vee x) \qquad (\bar{x} \vee \bar{d} \vee \bar{g} \vee \bar{s} \vee \bar{h} \vee \bar{i})}{(\bar{d} \vee \bar{g} \vee \bar{s} \vee \bar{h} \vee \bar{i})}$$

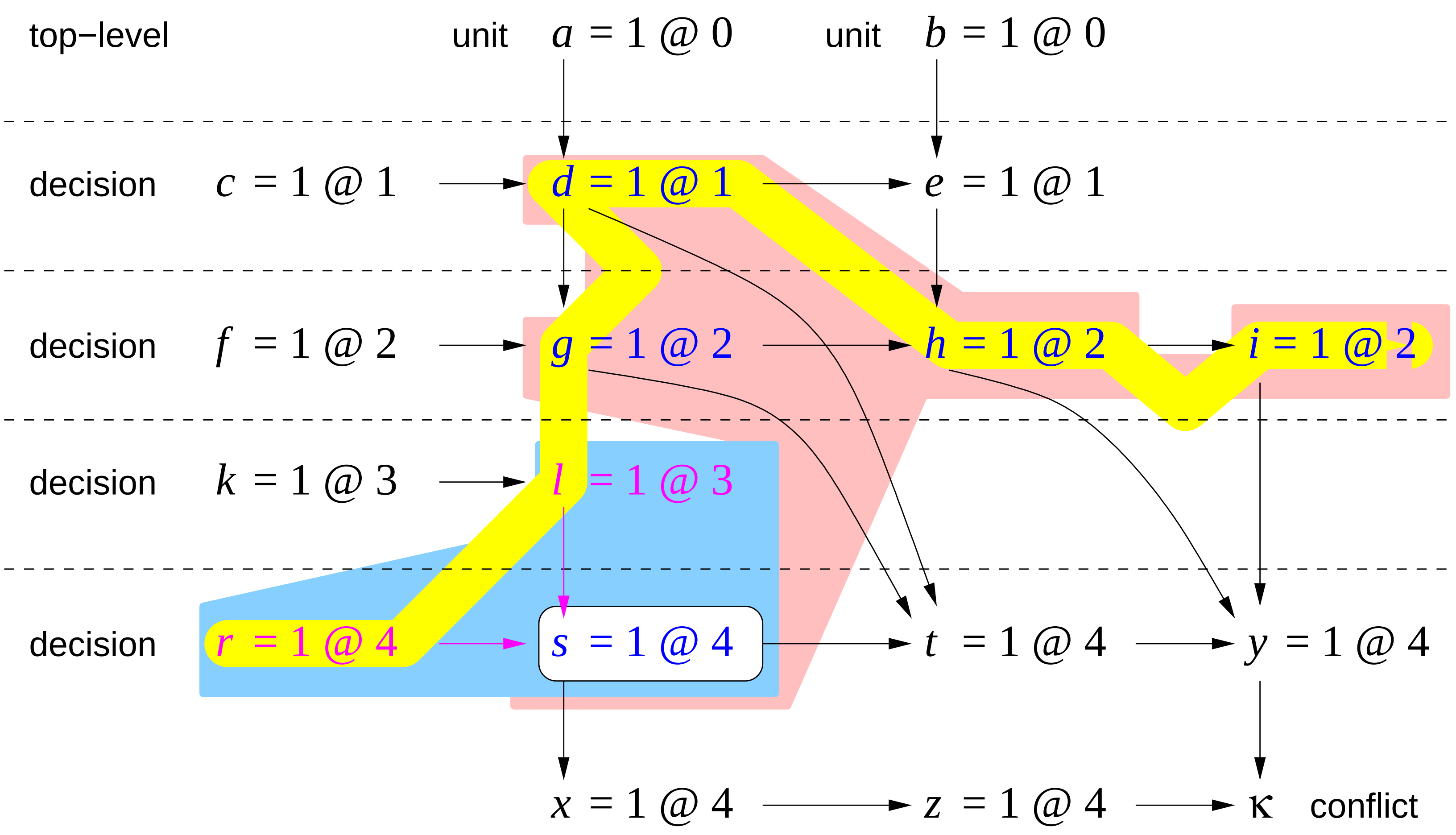
self subsuming resolution

1st UIP Clause after 4 Resolutions



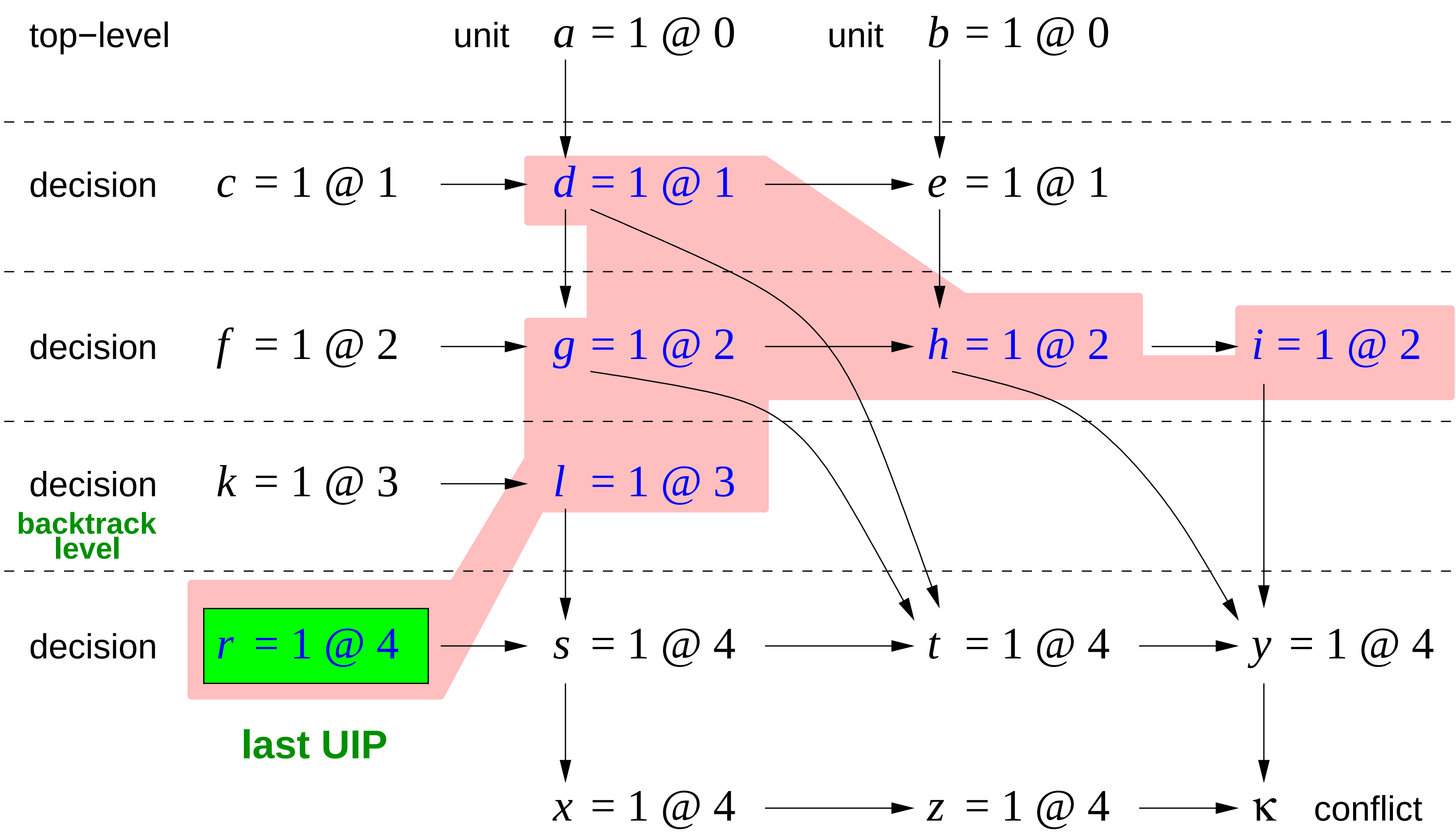
$$(\overline{d} \vee \overline{g} \vee \overline{s} \vee \overline{h} \vee \overline{i})$$

Resolving Antecedents 5th Time



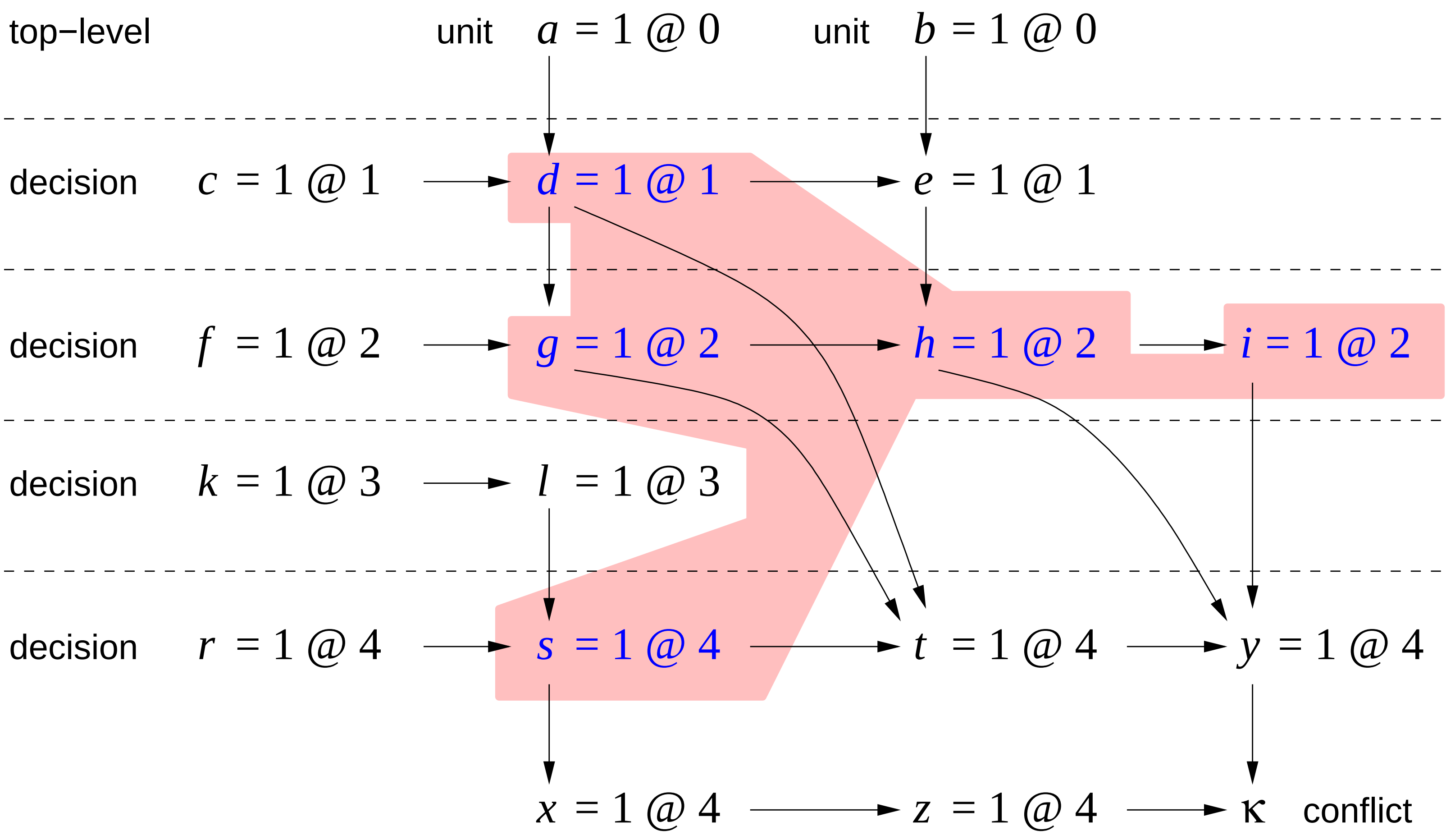
$$\frac{(\bar{l} \vee \bar{r} \vee s) \qquad (\bar{d} \vee \bar{g} \vee \bar{s} \vee \bar{h} \vee \bar{i})}{(\bar{l} \vee \bar{r} \vee \bar{d} \vee \bar{g} \vee \bar{h} \vee \bar{i})}$$

Decision Learned Clause



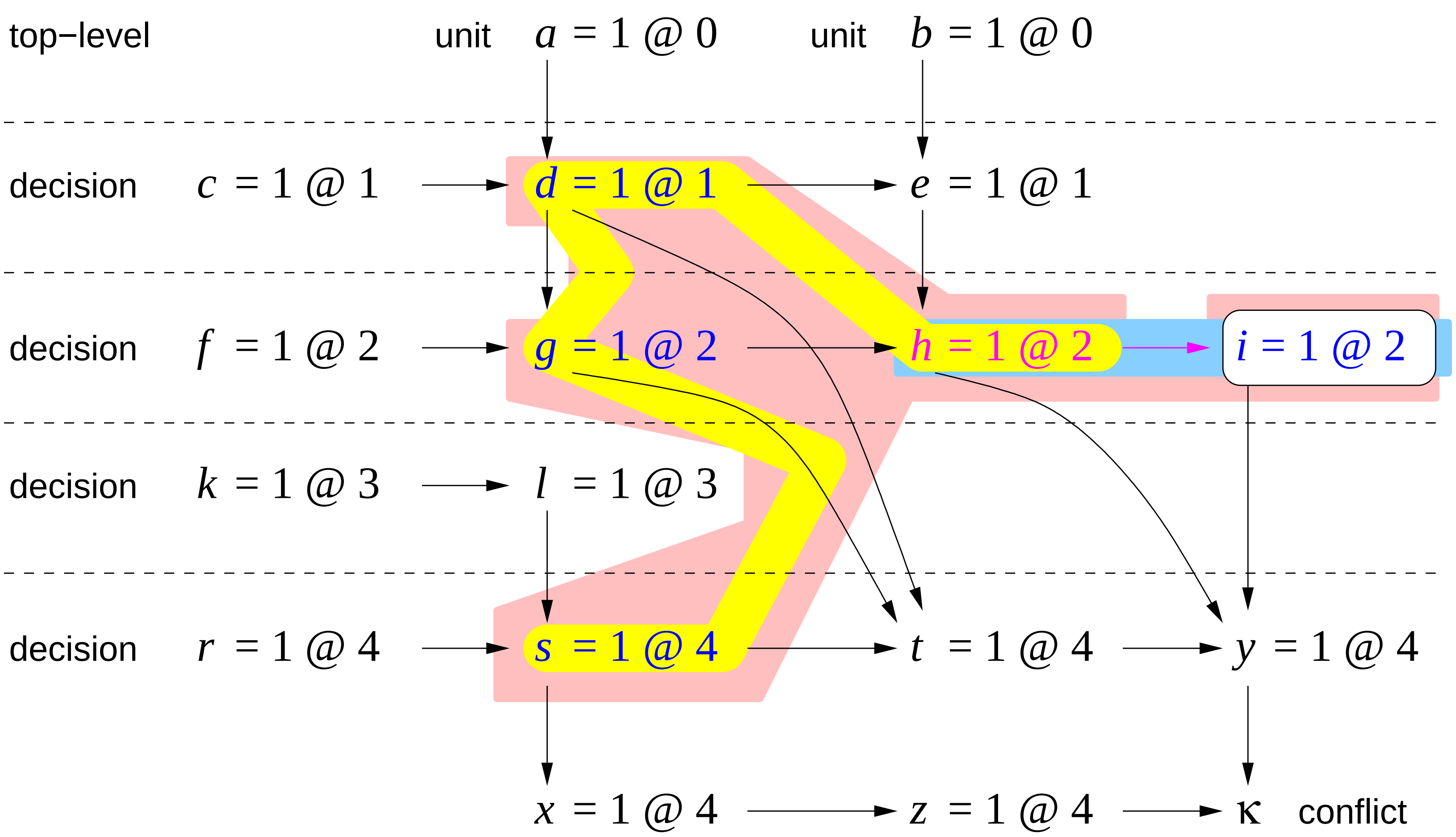
$$(\overline{d} \vee \overline{g} \vee \overline{l} \vee \overline{r} \vee \overline{h} \vee \overline{i})$$

1st UIP Clause after 4 Resolutions



$$(\overline{d} \vee \overline{g} \vee \overline{s} \vee \overline{h} \vee \overline{i})$$

Locally Minimizing 1st UIP Clause

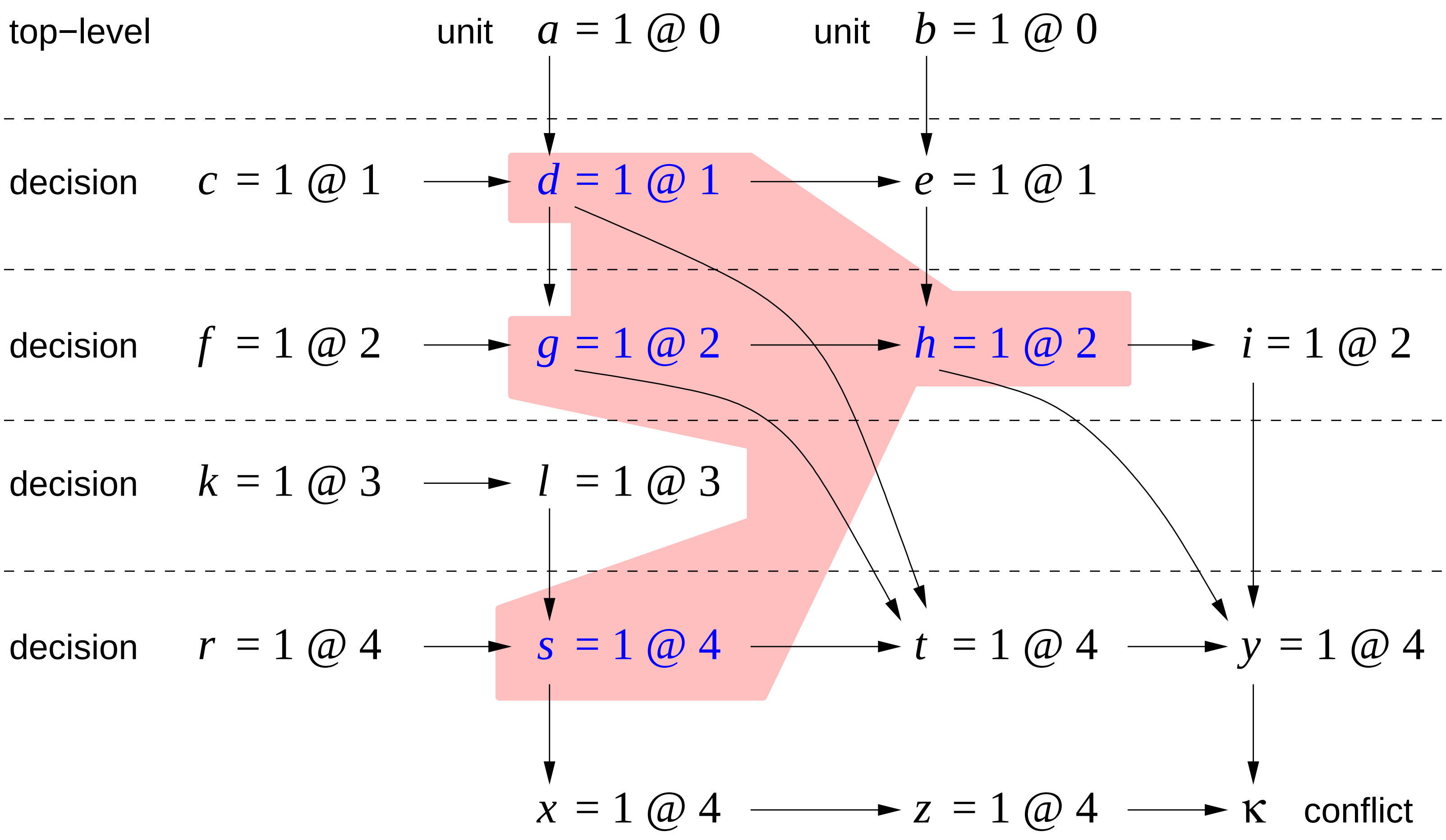


$$\frac{(\bar{h} \vee i) \qquad (\bar{d} \vee \bar{g} \vee \bar{s} \vee \bar{h} \vee \bar{i})}{(\bar{d} \vee \bar{g} \vee \bar{s} \vee \bar{h})}$$

self subsuming resolution

Locally Minimized Learned Clause

[SörenssonBiere-SAT'09]



$$(\overline{d} \vee \overline{g} \vee \overline{s} \vee \overline{h})$$

Local Minimization Algorithm

[BeameKautzSabharwal-JAIR'04] is an independent variation

[SörenssonBiere-SAT'09]

Two step algorithm:

1. mark all variables in 1st UIP clause

2. remove literals with all antecedent literals also marked

Correctness:

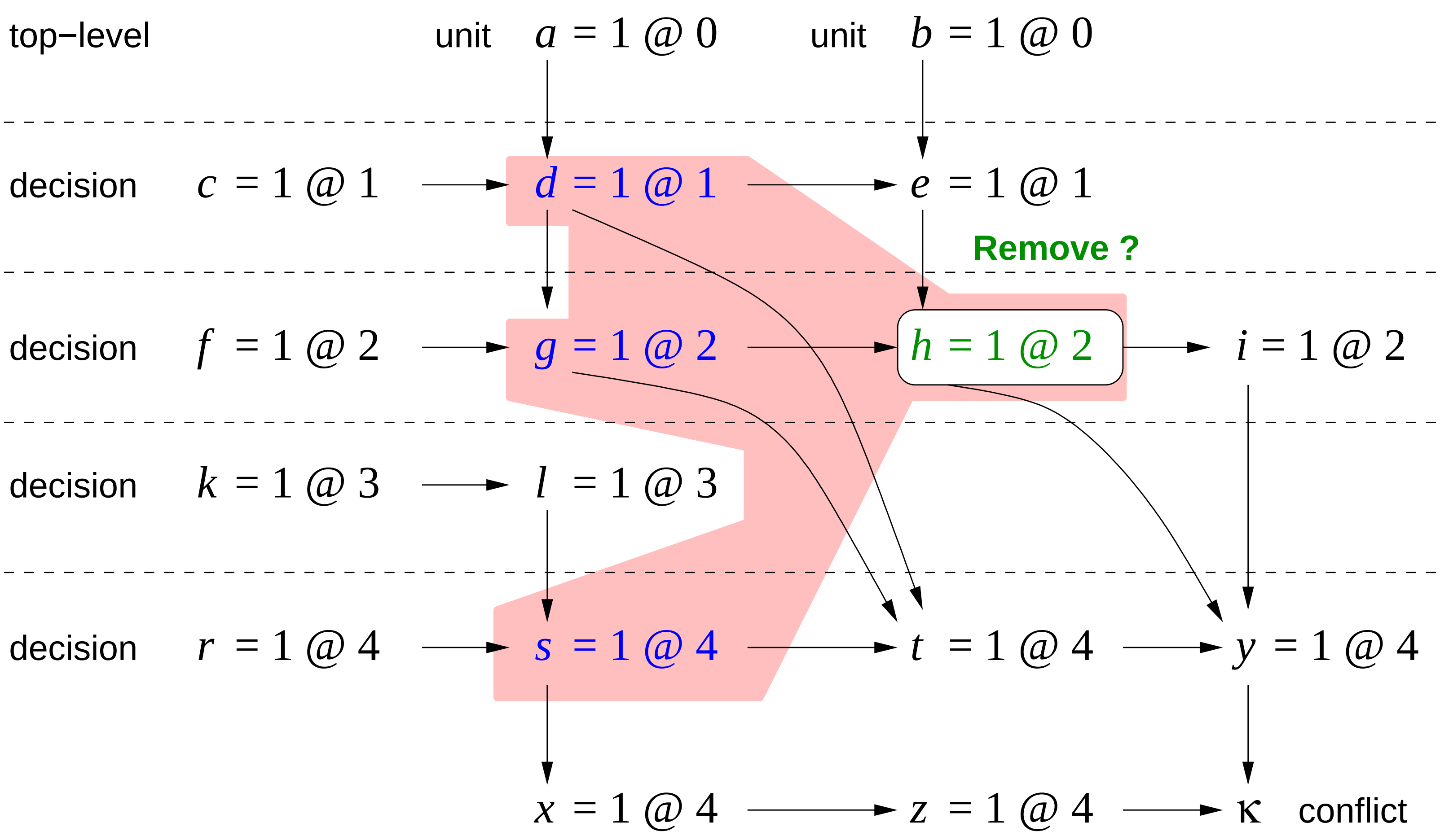
- removal of literals in step 2 are self subsuming resolution steps.

- implication graph is acyclic.

Confluence: produces a unique result.

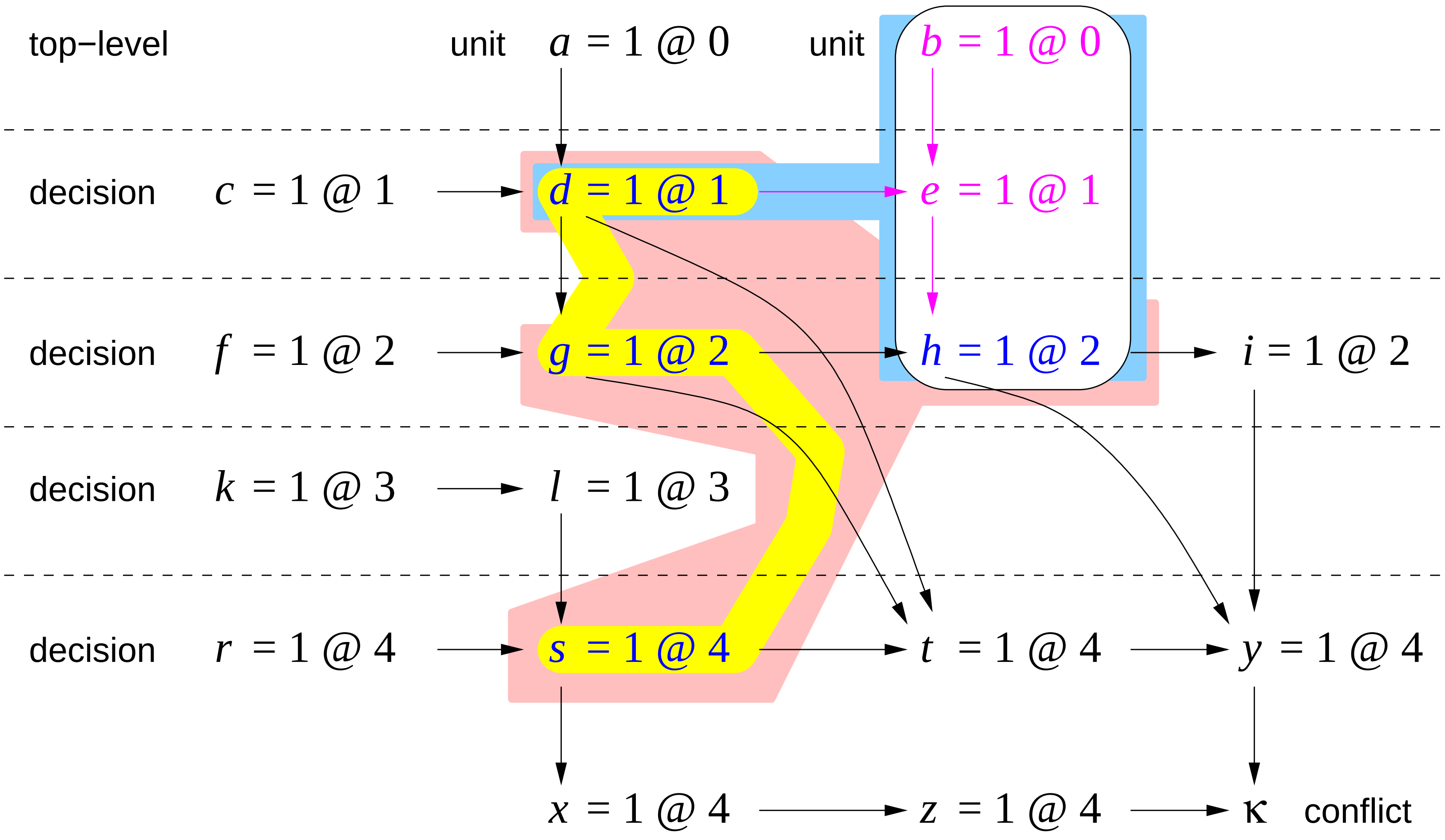
Minimizing Locally Minimized Learned Clause Further?

[SörenssonBiere-SAT'09]



$$(\overline{d} \vee \overline{g} \vee \overline{s} \vee \overline{h})$$

Recursively Minimizing Learned Clause



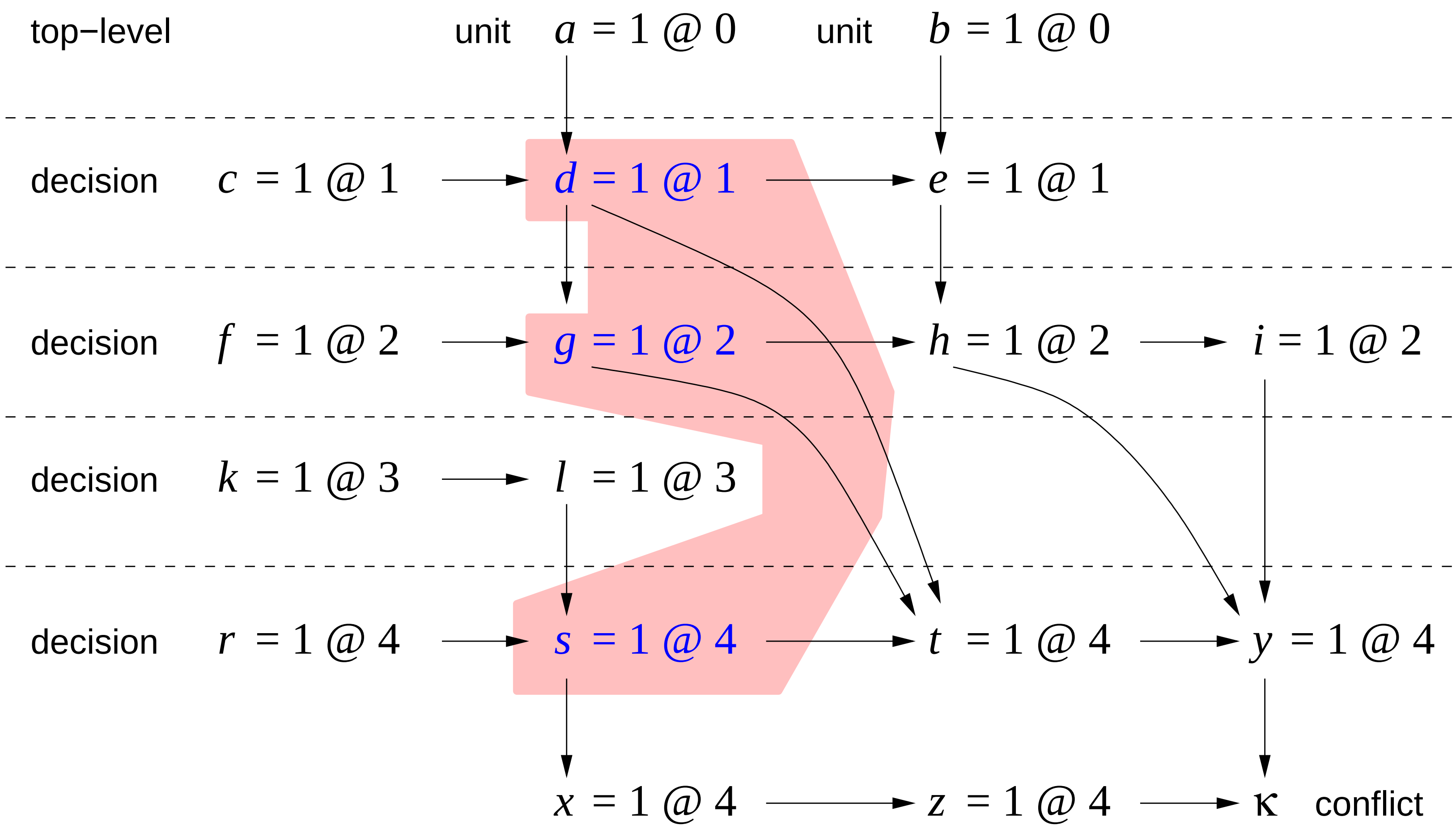
$$(\bar{e} \vee \bar{g} \vee h) \qquad (\bar{d} \vee \bar{g} \vee \bar{s} \vee \bar{h})$$

$$(\bar{d} \vee \bar{b} \vee e) \qquad (\bar{e} \vee \bar{d} \vee \bar{g} \vee \bar{s})$$

$$(b) \qquad (\bar{b} \vee \bar{d} \vee \bar{g} \vee \bar{s})$$

$$(\bar{d} \vee \bar{g} \vee \bar{s})$$

Recursively Minimized Learned Clause



$$(\overline{d} \vee \overline{g} \vee \overline{s})$$

Recursive Minimization Algorithm

[MiniSAT 1.13]

Four step algorithm:

1. mark all variables in 1st UIP clause

2. for each candidate literal: search implication graph

3. start at antecedents of candidate literals

4. if search always terminates at marked literals remove candidate

Correctness and Confluence as in local version!!!

Optimization: terminate early with failure if new decision level is "pulled in"

Experiments on 100 SAT'08 Race Instances

		solved instances		time in hours		space in GB		out of memory		deleted literals
MiniSAT with preprocessing	recur	788	9%	170	11%	198	49%	11	89%	33%
	local	774	7%	177	8%	298	24%	72	30%	16%
	none	726		192		392		103		
MiniSAT without preprocessing	recur	705	13%	222	8%	232	59%	11	94%	37%
	local	642	3%	237	2%	429	24%	145	26%	15%
	none	623		242		565		196		
PicoSAT with preprocessing	recur	767	10%	182	13%	144	45%	10	60%	31%
	local	745	6%	190	9%	188	29%	10	60%	15%
	none	700		209		263		25		
PicoSAT without preprocessing	recur	690	6%	221	8%	105	63%	10	68%	33%
	local	679	5%	230	5%	194	31%	10	68%	14%
	none	649		241		281		31		
altogether	recur	2950	9%	795	10%	679	55%	42	88%	34%
	local	2840	5%	834	6%	1109	26%	237	33%	15%
	none	2698		884		1501		355		

10 runs for each configuration with 10 seeds for random number generator

Large Variance for Different Seeds

		MiniSAT with preprocessing					
		seed	solved	time	space	mo	del
1.	recur	8	82	16	19	1	33%
2.	recur	6	81	17	20	1	33%
3.	local	0	81	16	29	7	16%
4.	local	7	80	17	29	8	15%
5.	recur	4	80	17	20	1	33%
6.	recur	1	79	17	20	1	33%
7.	recur	9	79	17	20	1	34%
8.	local	5	78	18	29	7	16%
9.	local	1	78	17	29	6	16%
10.	recur	0	78	17	20	1	34%
11.	recur	5	78	17	19	1	33%
12.	local	3	77	18	31	7	16%
13.	local	8	77	18	30	8	16%
14.	recur	7	77	17	20	1	34%
15.	recur	3	77	17	20	1	34%
16.	recur	2	77	17	20	2	33%
17.	none	7	76	19	39	9	0%
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Conclusion on Mimizing Learned Clauses

- minimization is effective and efficient

- how to use clauses not in the implication graph

  [AudemardBordeauxHamadiJabbourSais-SAT'08] ...

- how to use intermediate resolvents

  [HanSomenzi-SAT'9] ...

- how to extract resolution proofs directly [VanGelder SAT'9]

More Recent Developments

see also [Biere-JSAT'08]

- **phase saving** of assigned variables as in RSAT [PipatsrisawatDarwiche'07]
  - initially pick phase according to number of occurrences
  - afterwards always pick last saved phase for decision variables
- **rapid restarts** [Luby...'93] as in TiniSAT [Huang'07]
  - uses ideas from stochastic local search
  - empirically works well for complete solvers as well
  - see peformance of RSAT and PicoSAT in SAT competition in 2007
- actually both ideas need to be **combined** to give an improvement
- ongoing work in SAT'08/SAT'09 on how to **schedule restarts** even better

New Normalized VSIDS (NVSIDS)

(consider only one variable)

[Biere-SAT'08]

feedback / punishment / I in PID: $0 < f < 1$

s old score s' new score

$$s' = \begin{cases} s \cdot f + (1-f) & \text{if variable is involved in current conflict} \\ s \cdot f & \text{if variable is NOT involved} \end{cases}$$

$$0 \leq \underbrace{s \cdot f}_{\text{decay in any case}} \leq s' \leq s \overbrace{\cdot f}^{\text{decay in any case}} \underbrace{+ (1-f)}_{\text{increment if involved}} \leq f + (1-f) = 1$$

MiniSAT, RSAT: $f = 0.95 \approx 1/1.05$ $(1-f) = 0.05$

PicoSAT: $f = 1/1.1 \approx 0.91$ $(1-f) = 0.09$

Exponential VSIDS (EVSIDS) as in MiniSAT

[Biere-SAT'08]

(consider only one variable)

$$\delta_k = \begin{cases} 1 & \text{if involved in } k\text{-th conflict} \\ 0 & \text{otherwise} \end{cases}$$

$$i_k = (1 - f) \cdot \delta_k$$

$$s_n = (\ldots(i_1 \cdot f + i_2) \cdot f + i_3) \cdot f \cdots) \cdot f + i_n = \sum_{k=1}^{n} i_k \cdot f^{n-k} = (1-f) \cdot \sum_{k=1}^{n} \delta_k \cdot f^{n-k} \quad \text{(NVSIDS)}$$

$$S_n = \frac{f^{-n}}{(1-f)} \cdot s_n = \frac{f^{-n}}{(1-f)} \cdot (1-f) \cdot \sum_{k=1}^{n} \delta_k \cdot f^{n-k} = \sum_{k=1}^{n} \delta_k \cdot f^{-k} \quad \text{(EVSIDS)}$$

What is Model Checking?

- mechanically check properties of models
- models:
  - finite automata, labelled transition systems
  - often requires automatic/manual abstraction techniques
- properties:
  - only interested in *partial properties*
  - specified in temporal logic: CTL, LTL, etc.
  - safety: something bad should not happen
  - liveness: something god should happen
- automatic generation of counterexamples

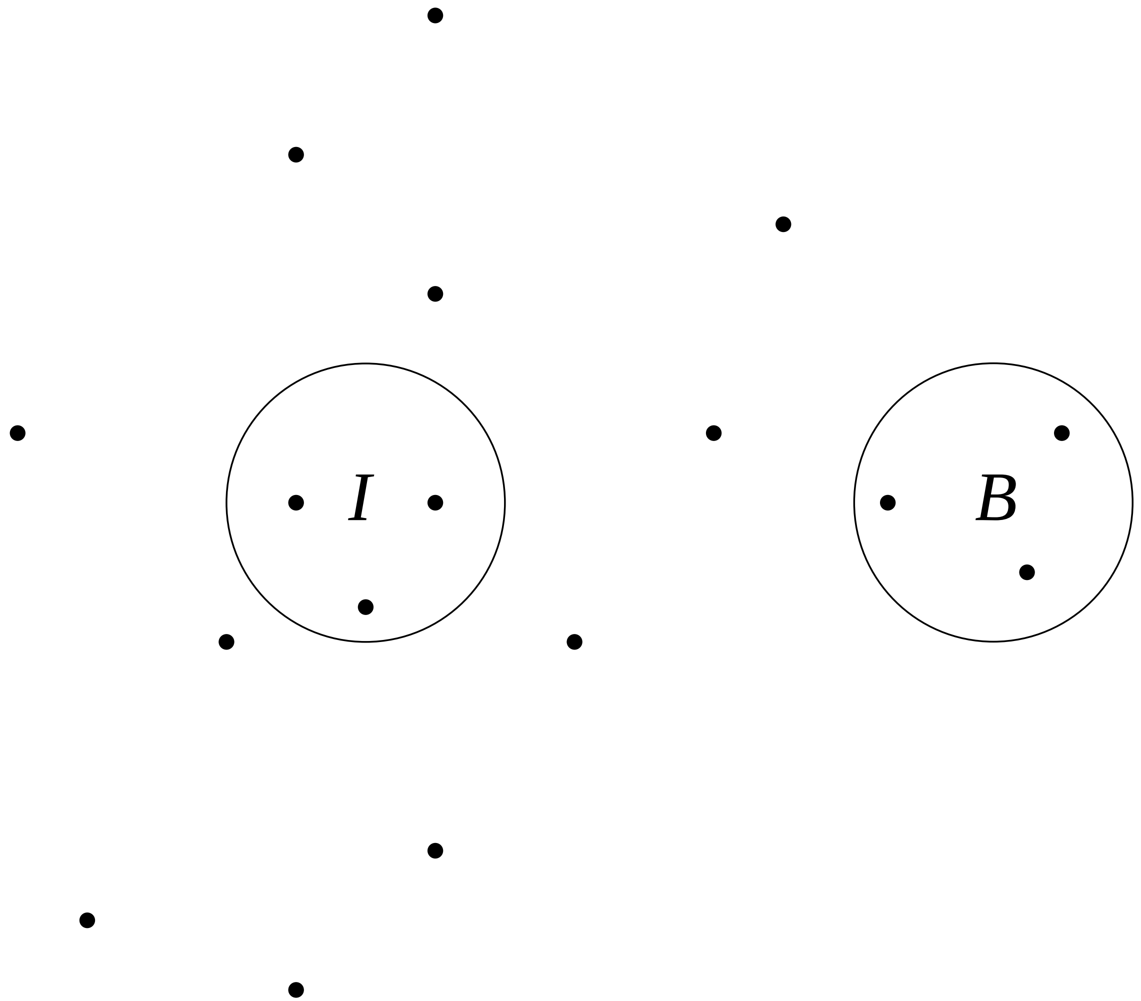
- set of states S, initial states I, transition relation T

- bad states B reachable from I via T?

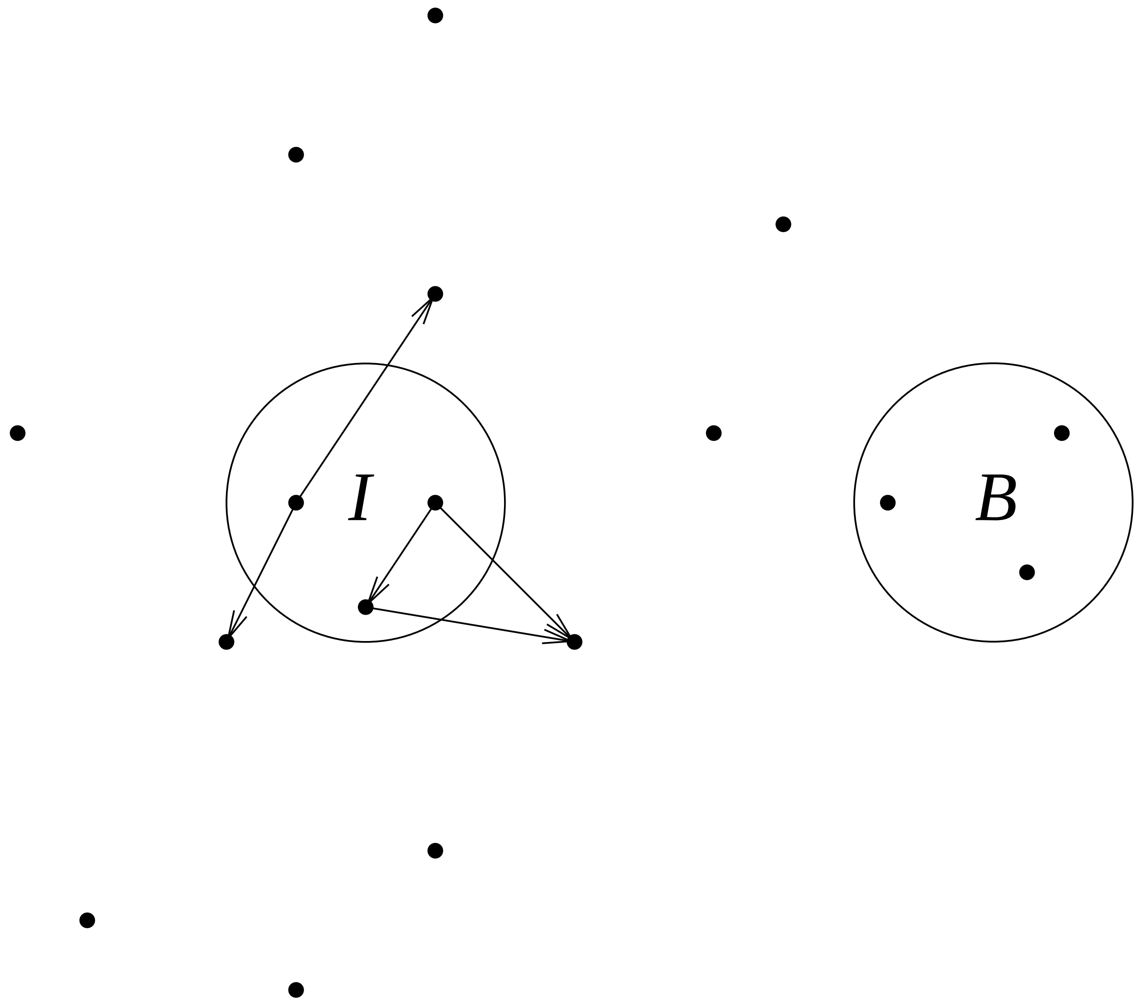
- symbolic representation of T (ciruit, program, parallel product)
  - avoid explicit matrix representations, because of the
  - state space explosion problem, e.g. n-bit counter: $|T| = O(n)$, $|S| = O(2^n)$
  - makes reachability PSPACE complete [Savitch'70]

- on-the-fly [Holzmann'81'] for protocols
  - restrict search to reachable states
  - simulate and hash reached concrete states

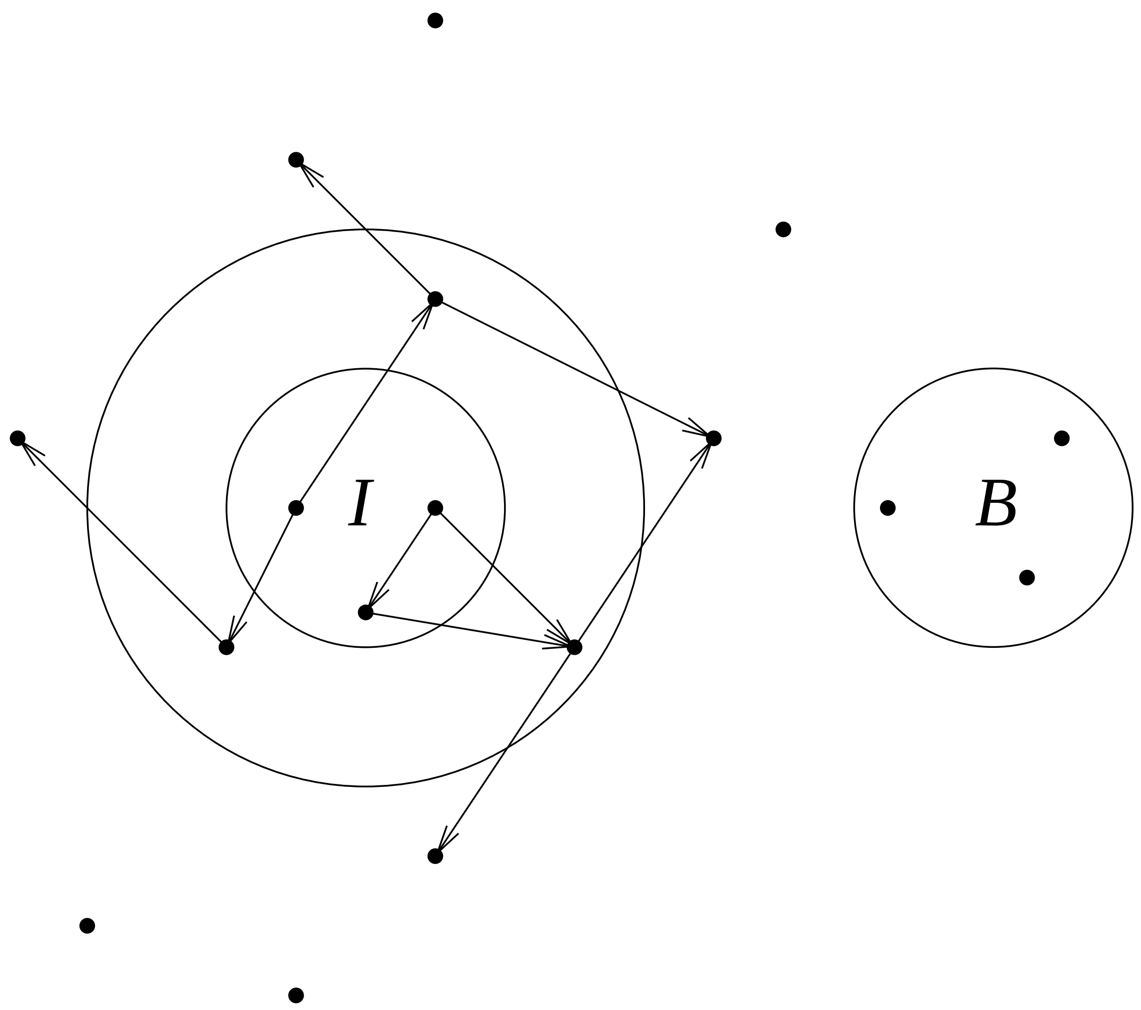
Forward Fixpoint: Initial and Bad States



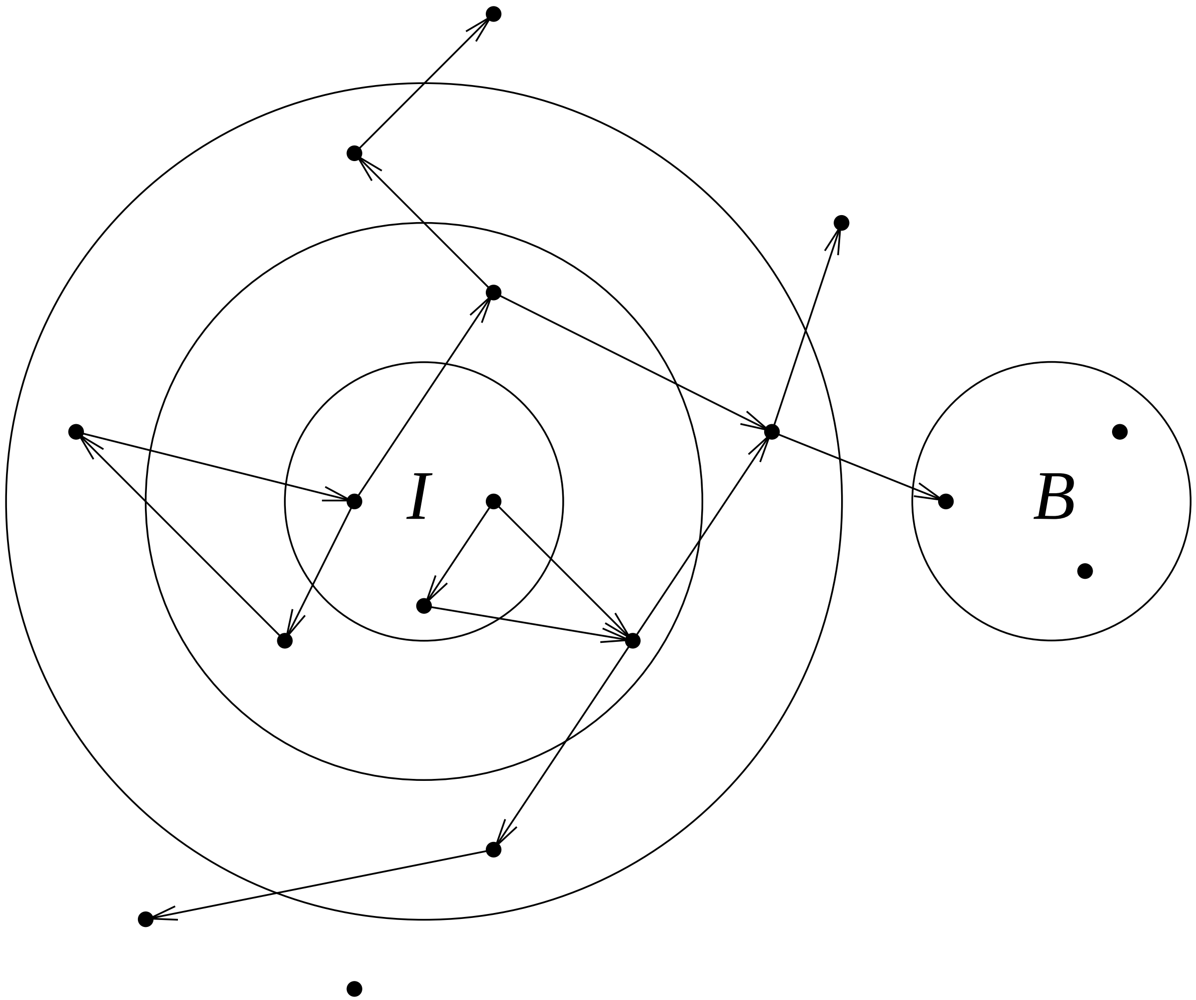
Forward Fixpoint: Step 1



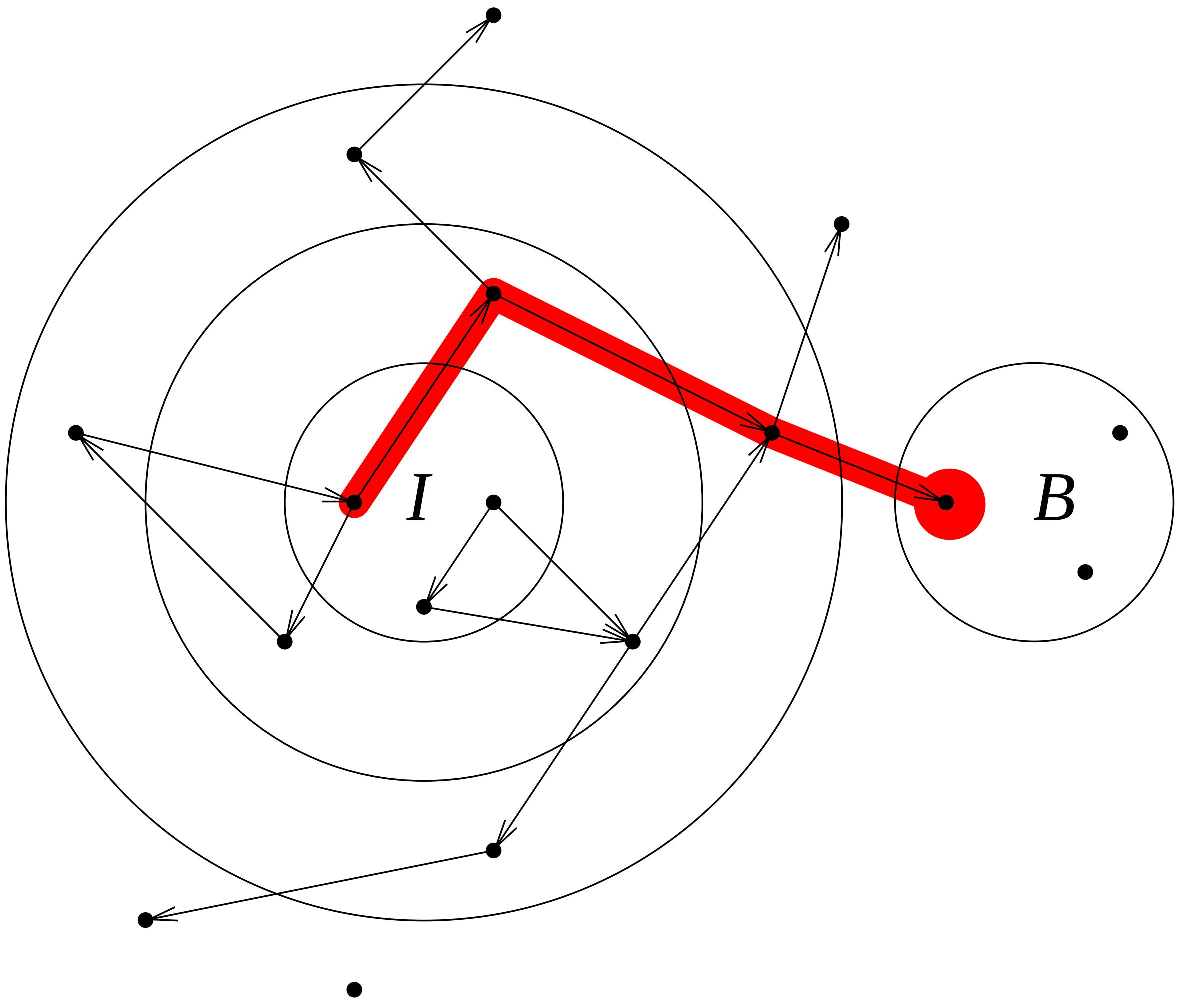
Forward Fixpoint: Step 2



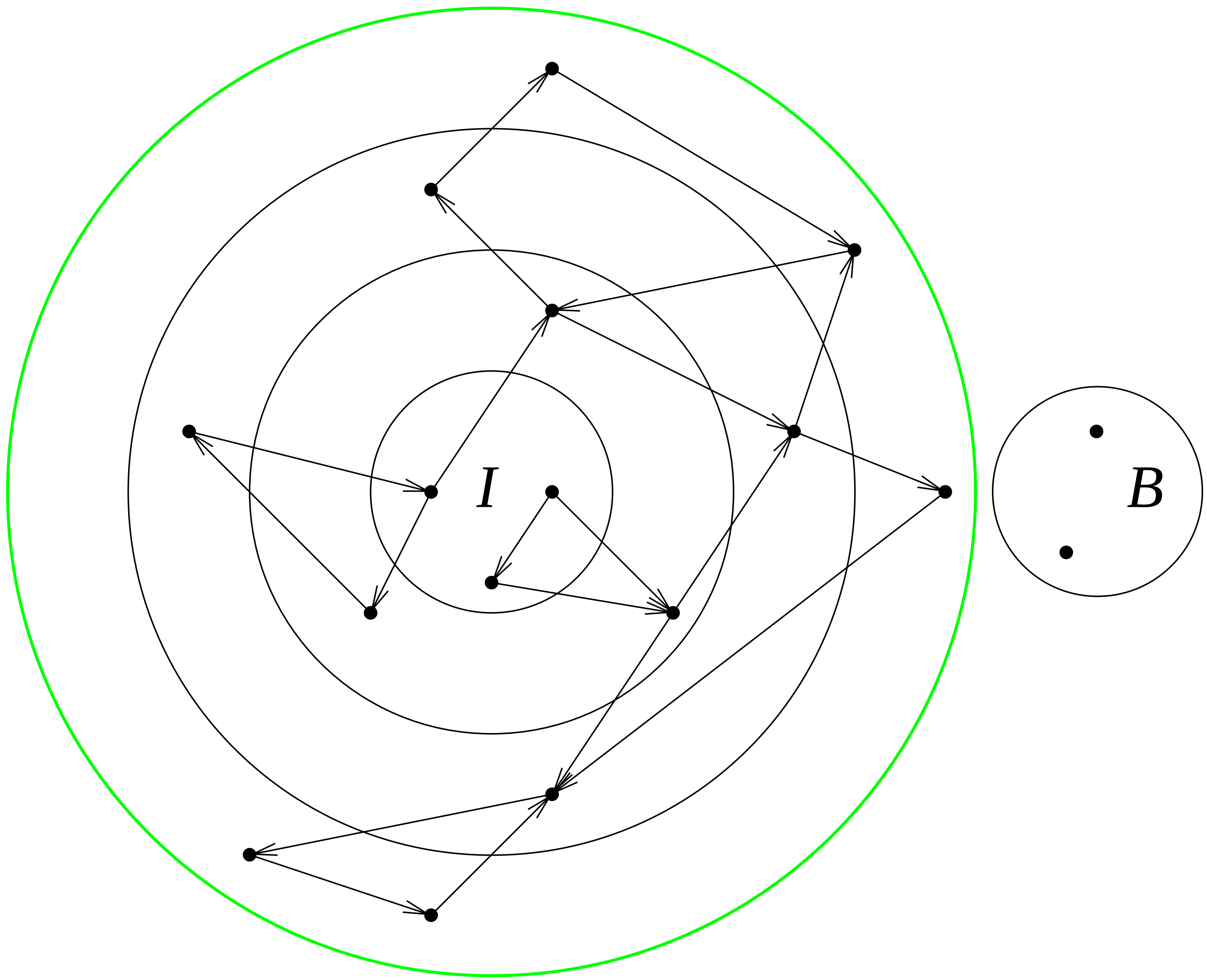
Forward Fixpoint: Step 3



Forward Fixpoint: Bad State Reached



Forward Fixpoint: Termination, No Bad State Reachable



initial states I, transition relation T, bad states B

<u>model-check</u>$^{\mu}_{\text{forward}}$ (I, T, B)

$S_C = \emptyset$; $S_N = I$;

while $S_C \neq S_N$ **do**

if $B \cap S_N \neq \emptyset$ **then**

return "found error trace to bad states";

$S_C = S_N$;

$S_N = S_C \cup Img(S_C)$;

done;

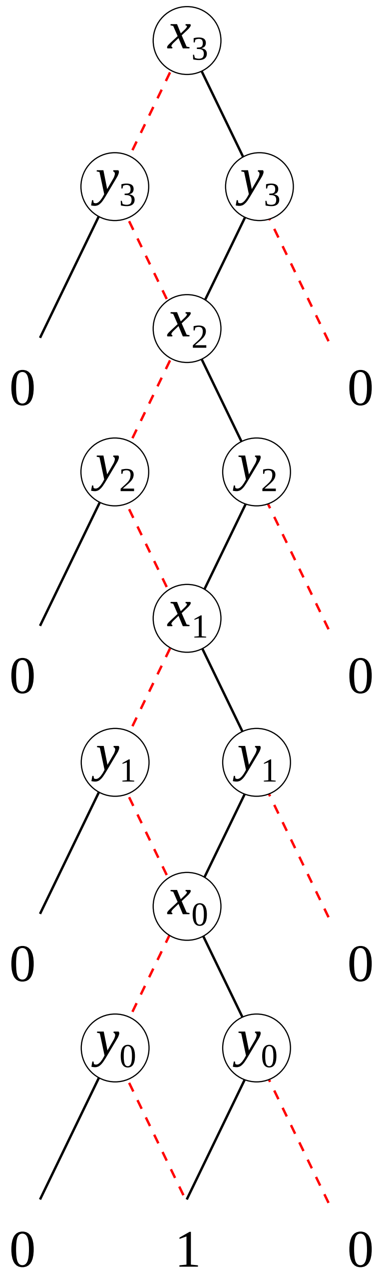
return "no bad state reachable";

Symbolic Model Checking

- work with symbolic representations of states
  - symbolic representations are potentially exponentially more succinct
  - favors BFS: next frontier set of states in BFS is calculated symbolically
- originally “symbolic” meant model checking with BDDs

  [CoudertMadre’89/’90,BurchClarkeMcMillanDillHwang’90,McMillan’93]
- Binary Decision Diagrams [Bryant’86]
  - canonical representation for boolean functions
  - BDDs have fast operations (but image computation is expensive)
  - often blow up in space
  - restricted to hundreds of variables

Linear Size BDD for Bit-Vector Comparison



boolean function/expression:

$$\bigwedge_{i=0}^{n-1} x_i = y_i$$

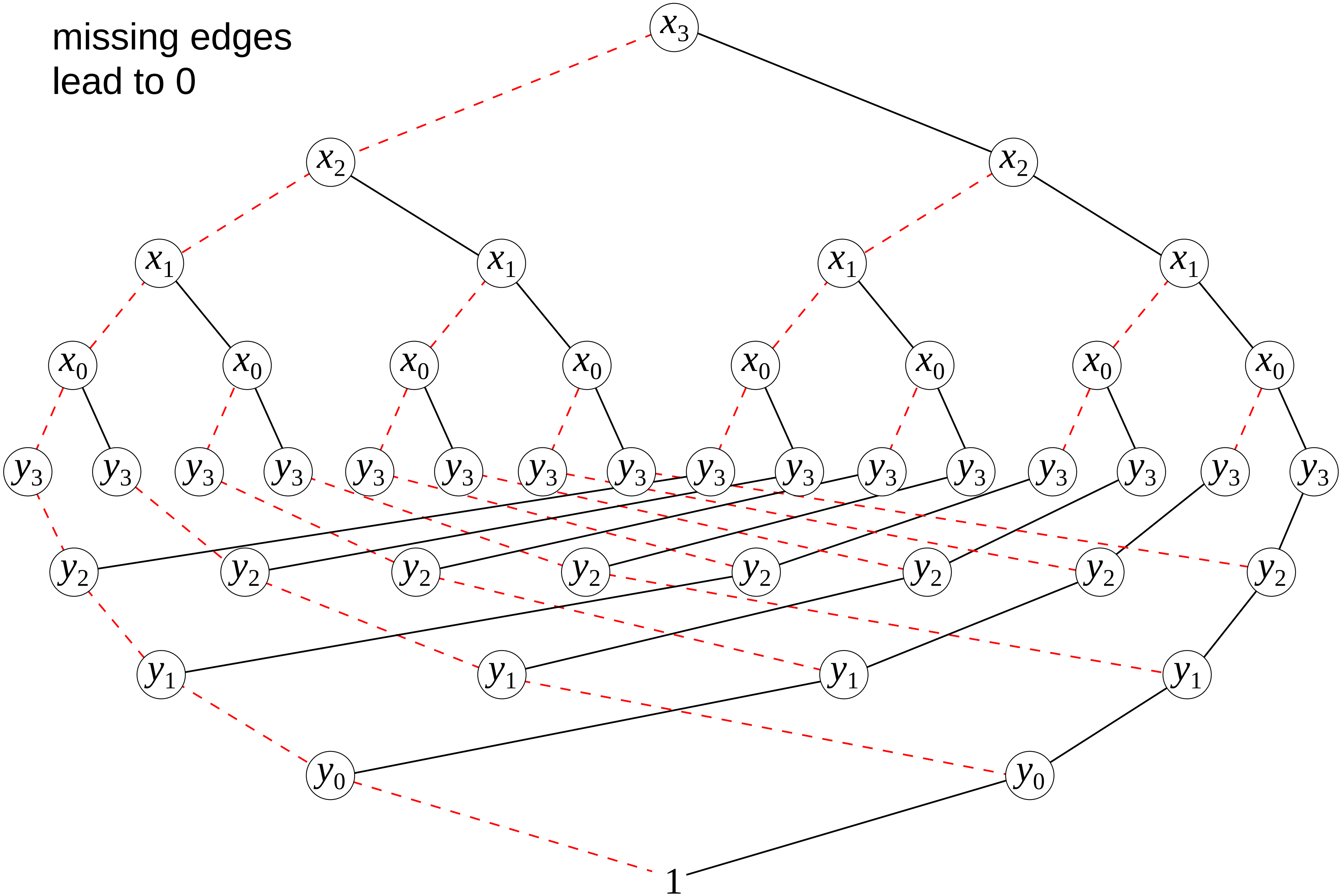
interleaved variable order:

$$x_3 > y_3 > x_2 > y_2 > x_1 > y_1 > x_0 > y_0$$

comparison of two n-bit-vectors needs $3 \cdot n$ inner nodes for the interleaved variable order

Exponential BDD for Bit-Vector Comparison

missing edges
lead to 0



Unrolling of Forward Least Fixpoint Algorithm

0: continue?	$S_C^0 \neq S_N^0$	$\exists s_0[I(s_0)]$
0: terminate?	$S_C^0 = S_N^0$	$\forall s_0[\neg I(s_0)]$
0: bad state?	$B \cap S_N^0 \neq \emptyset$	$\exists s_0[I(s_0) \wedge B(s_0)]$
1: continue?	$S_C^1 \neq S_N^1$	$\exists s_0, s_1[I(s_0) \wedge T(s_0,s_1) \wedge \neg I(s_1)]$
1: terminate?	$S_C^1 = S_N^1$	$\forall s_0, s_1[I(s_0) \wedge T(s_0,s_1) \rightarrow I(s_1)]$
1: bad state?	$B \cap S_N^1 \neq \emptyset$	$\exists s_0, s_1[I(s_0) \wedge T(s_0,s_1) \wedge B(s_1)]$
2: continue?	$S_C^2 \neq S_N^2$	$\exists s_0, s_1, s_2[I(s_0) \wedge T(s_0,s_1) \wedge T(s_1,s_2) \wedge \neg(I(s_2) \vee \exists t_0[I(t_0) \wedge T(t_0,s_2)])]$
2: terminate?	$S_C^2 = S_N^2$	$\forall s_0, s_1, s_2[I(s_0) \wedge T(s_0,s_1) \wedge T(s_1,s_2) \rightarrow I(s_2) \vee \exists t_0[I(t_0) \wedge T(t_0,s_2)]]$
2: bad state?	$B \cap S_N^1 \neq \emptyset$	$\exists s_0, s_1, s_2[I(s_0) \wedge T(s_0,s_1) \wedge T(s_1,s_2) \wedge B(s_2)]$

Falsification Part of Fixpoint Algorithm

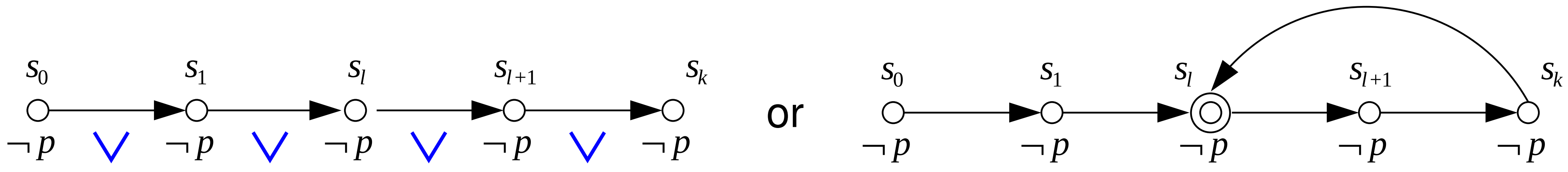
[BiereCimattiClarkeZhu-TACAS'99]

~~0: continue?~~	~~$S_C^0 \neq S_N^0$~~	~~$\exists s_0[I(s_0)]$~~
~~0: terminate?~~	~~$S_C^0 = S_N^0$~~	~~$\forall s_0[\neg I(s_0)]$~~
0: bad state?	$B \cap S_N^0 \neq \emptyset$	$\exists s_0[I(s_0) \wedge B(s_0)]$
~~1: continue?~~	~~$S_C^1 \neq S_N^1$~~	~~$\exists s_0, s_1[I(s_0) \wedge T(s_0, s_1) \wedge \neg I(s_1)]$~~
~~1: terminate?~~	~~$S_C^1 = S_N^1$~~	~~$\forall s_0, s_1[I(s_0) \wedge T(s_0, s_1) \rightarrow I(s_1)]$~~
1: bad state?	$B \cap S_N^1 \neq \emptyset$	$\exists s_0, s_1[I(s_0) \wedge T(s_0, s_1) \wedge B(s_1)]$
~~2: continue?~~	~~$S_C^2 \neq S_N^2$~~	~~$\exists s_0, s_1, s_2[I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge$~~ $\neg(I(s_2) \vee \exists t_0[I(t_0) \wedge T(t_0, s_2)])]$
~~2: terminate?~~	~~$S_C^2 = S_N^2$~~	~~$\forall s_0, s_1, s_2[I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \rightarrow$~~ ~~$I(s_2) \vee \exists t_0[I(t_0) \wedge T(t_0, s_2)]]$~~
2: bad state?	$B \cap S_N^1 \neq \emptyset$	$\exists s_0, s_1, s_2[I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge B(s_2)]$

Bounded Model Checking (BMC)

[BiereCimattiClarkeZhu-TACAS'99]

- look only for counter example made of k states (the bound)



- simple for safety properties p is invariantly true (e.g. $p = \neg B$)

$$I(s_0) \wedge T(s_0, s_1)) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$$

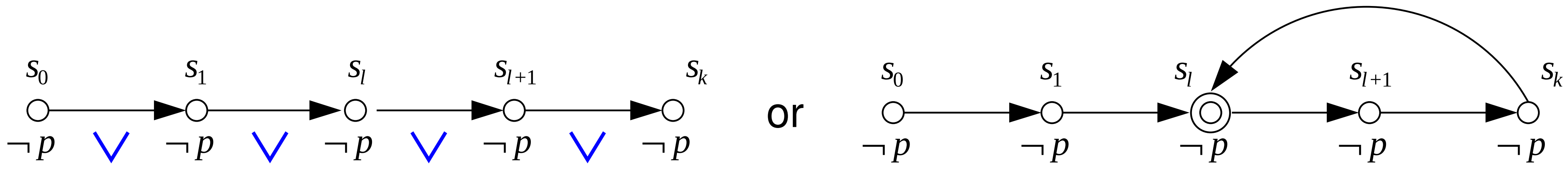
- harder for liveness properties p is eventually true

$$I(s_0) \wedge T(s_0, s_1)) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \bigwedge_{i=0}^{k} \neg p(s_i) \wedge \exists l\, T(s_k, s_l)$$

Bounded Model Checking (BMC)

[BiereCimattiClarkeZhu-TACAS'99]

- look only for counter example made of k states (the bound)



- simple for safety properties p is invariantly true (e.g. $p = \neg B$)

$$I(s_0) \wedge T(s_0, s_1)) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$$

- harder for liveness properties p is eventually true

$$I(s_0) \wedge T(s_0, s_1)) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \bigwedge_{i=0}^{k} \neg p(s_i) \wedge \bigvee_{l=0}^{k} T(s_k, s_l)$$

Bounded Model Checking State-of-the-Art

- increase in efficiency of SAT solvers [Grasp,zChaff,MiniSAT,SatELite,...]
- SAT more robust than BDDs in bug finding

  (shallow bugs are easily reached by explicit model checking or testing)
- better unbounded but still SAT based model checking algorithms
  - k-induction [SinghSheeranStalmarck'00]
  - interpolation [McMillan'03]
- 4th Intl. Workshop on Bounded Model Checking (BMC'06)
- other logics, better encodings, e.g. [LatvalaBiereHeljankoJuntilla-FMCAD'04]
- other models, e.g. C/C++/Verilog [Kröning...], hybrid automata [Audemard...-BMC'04]

Induction with SAT

[SinghSheeranStalmarck'00]

- more specifically k-induction
  - does there exist k such that the following formula is *unsatisfiable*

$$\overline{B(s_0)} \wedge \cdots \wedge \overline{B(s_{k-1})} \wedge T(s_0, s_1) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge B(s_k) \wedge \bigwedge_{0 \le i < j \le k} s_i \neq s_j$$

  - if *unsatisfiable* and ¬BMC(k) then bad state unreachable

- bound on k: length of longest cycle free path = reoccurrence diameter

- $k = 0$ check whether $\neg B$ tautological (propositionally)

- $k = 1$ check whether $\neg B$ inductive for T

[McMillan'03]

- SAT based technique to overapproximate frontiers $Img(S_C)$
  - currently most effective technique to show that bad states are unreachable
  - better than BDDs and k-induction in most cases [HWMCC'08]

- starts from a resolution proof refutation of a BMC problem with bound $k+1$

$$S_C(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_k, s_{k+1}) \wedge B(s_{k+1})$$

  - result is a characteristic function $f(s_1)$ over variables of the second state s_1
  - these states do not reach the bad state s_{k+1} in k steps
  - any state reachable from S_C satisfies f: $S_C(s_0) \wedge T(s_0, s_1) \Rightarrow f(s_1)$

- k is bounded by the diameter (exponentially smaller than longest cycle free path)

Simple Path Constraints

- bounded model checking: [BiereCimattiClarkeZhu'99]

$$I(s_1) \wedge T(s_1, s_2) \wedge \ldots \wedge T(s_{k-1}, s_k) \wedge \bigvee_{0 \leq i \leq k} B(s_i)$$ satisfiable?

- reoccurrence diameter checking: [BiereCimattiClarkeZhu'99]

$$T(s_1, s_2) \wedge \ldots \wedge T(s_{k-1}, s_k) \;\wedge \bigwedge_{1 \leq i < j \leq k} s_i \neq s_j$$ unsatisfiable?

- k-induction base case: [SheeranSinghStålmarck'00]

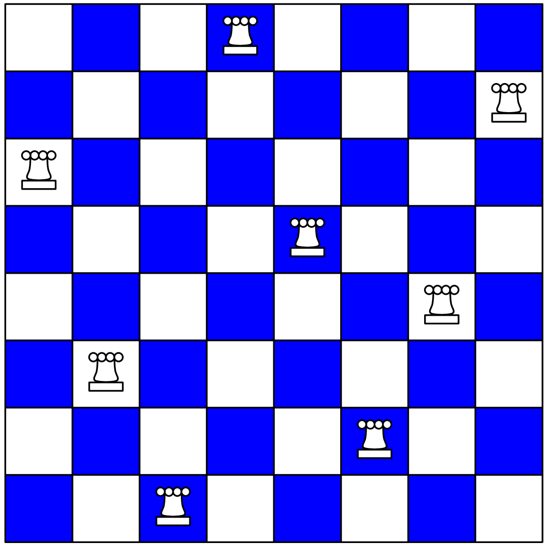
$$I(s_1) \wedge T(s_1, s_2) \wedge \ldots \wedge T(s_{k-1}, s_k) \;\wedge B(s_k) \wedge \bigwedge_{0 \leq i < k} \neg B(s_i)$$ satisfiable?

- k-induction induction step: [SheeranSinghStålmarck'00]

$$T(s_1, s_2) \wedge \ldots \wedge T(s_{k-1}, s_k) \;\wedge B(s_k) \wedge \bigwedge_{0 \leq i < k} \neg B(s_i) \;\wedge \bigwedge_{1 \leq i < j \leq k} s_i \neq s_j$$ unsatisfiable?

All Different Constraints (ADC)

- classical concept in constraint programming:
  - k variables over a domain of size m supposed to have different values
  - for instance k-queen problem



- propagation algorithms to establish arc-consistency
  - explicit propagators: [Régin'94]
    - $\mathrm{O}(k \cdot m)$ space
    - $\mathrm{O}(k^2 \cdot m^2)$ time
  - symbolic propagators: [GentNightingale'04] also [MarquesSilvaLynce'07]
    - one-hot CNF encoding with $\Omega(k \cdot m)$ boolean variables

- in model checking $k \ll m$ typically $k < 1000 \quad m = 2^n > 2^{100}$ n latches

Symbolic ADCs for Large Domains

[BiereBrummayer-FMCAD'08]

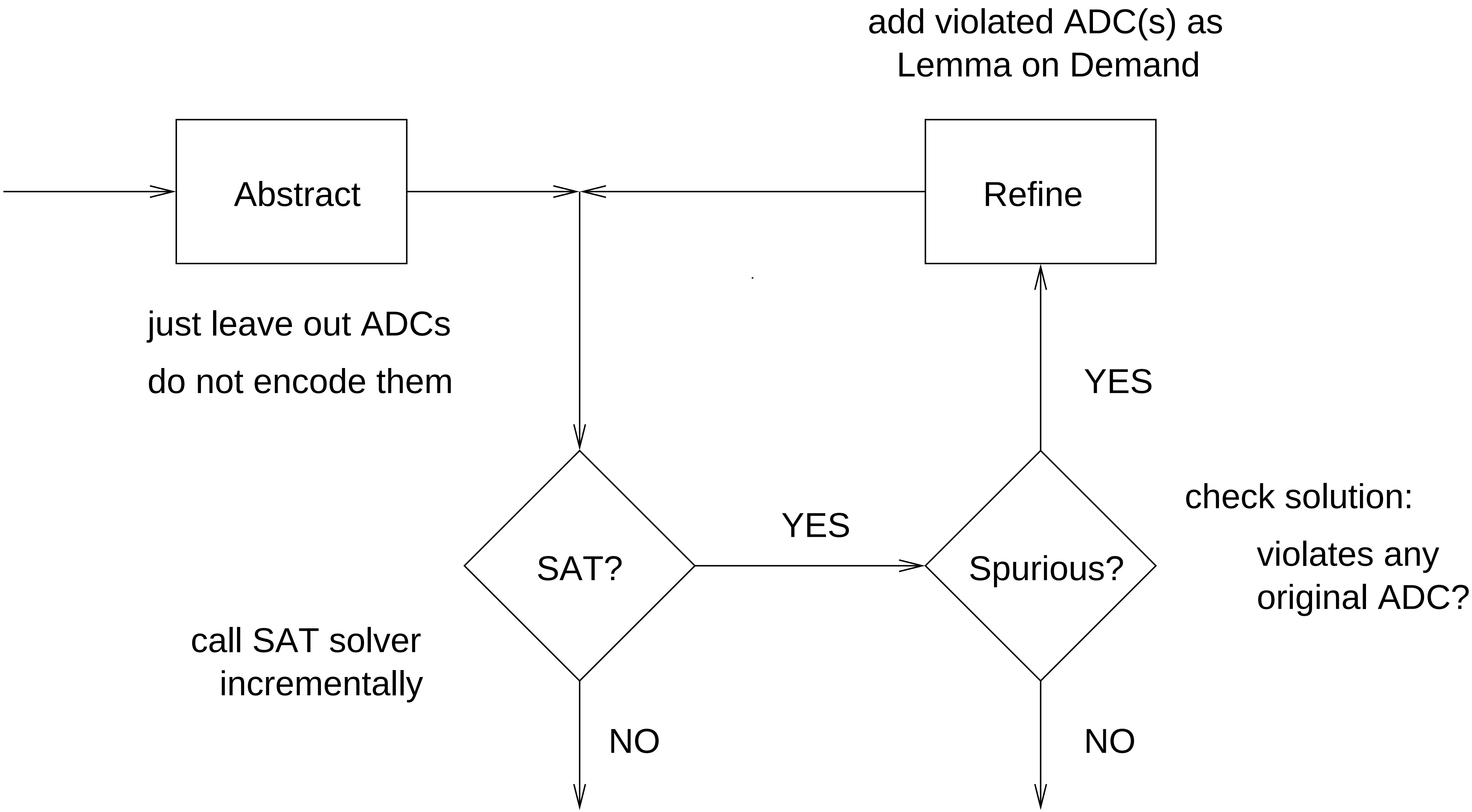
- encoding bit-vector inequalities directly:
  - let u, v be two n-bit vectors, $d_0, \ldots, d_{n-1}$ fresh boolean variables

    $u \neq v$ is equisatisfiable to $(d_0 \vee \cdots \vee d_{n-1}) \wedge \bigwedge_{j=0}^{n-1} (u_j \vee v_j \vee \overline{d_j}) \wedge (\overline{u_j} \vee \overline{v_j} \vee \overline{d_j})$
  - can be extended to encode Ackermann Constraints + McCarthy Axioms
  - either **eagerly** encode all $s_i \neq s_j$ quadratic in k
  - or **refine** adding bit-vector inequalities on demand [EénSörensson-BMC'03]

- natively handle ADCs within SAT solver: main contribution in FMCAD'08
  - similar to theory consistency checking in lazy SMT vs. "lemmas on demand"
  - can be extended to also perform theory propagation

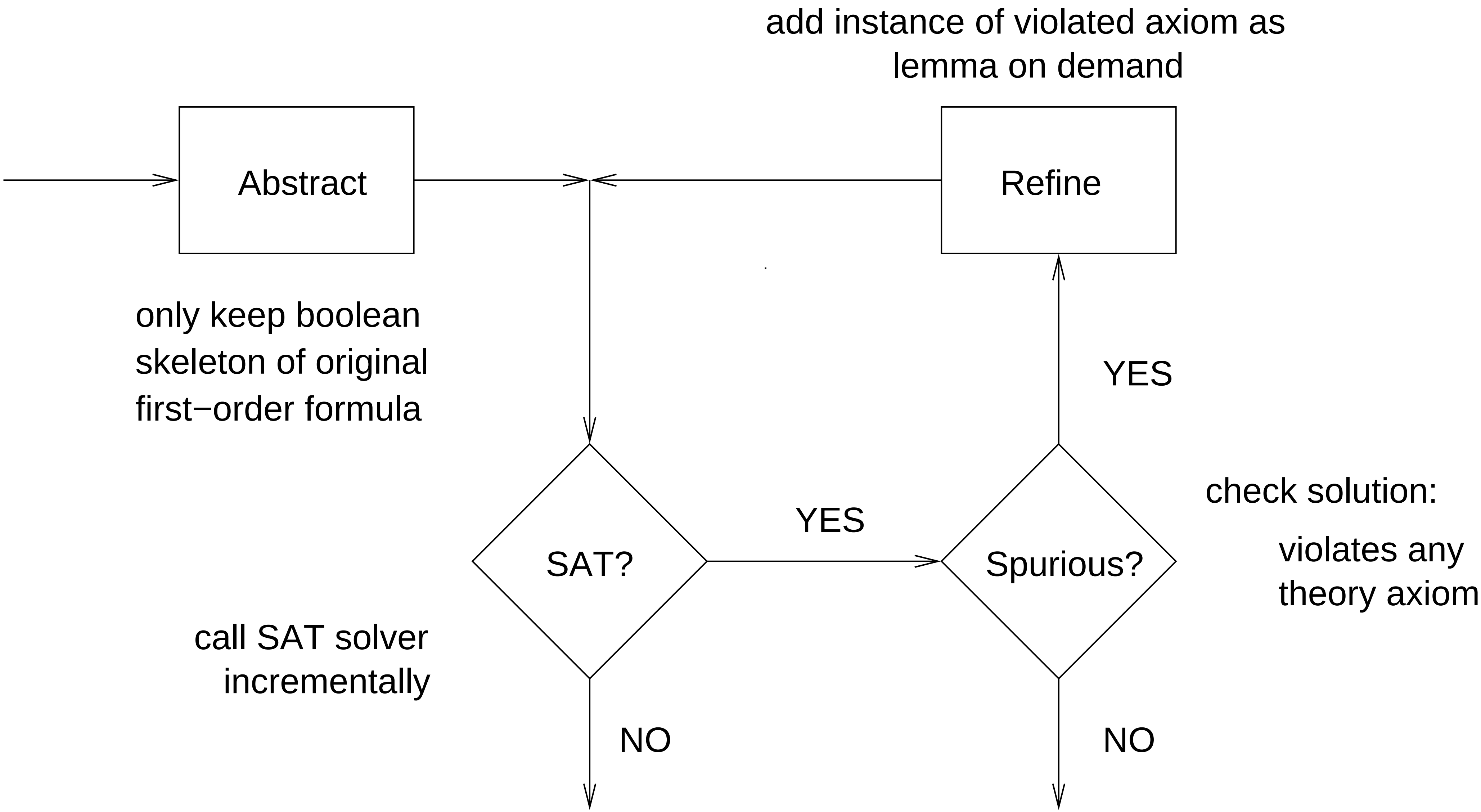
- sorting networks ineffective in our experience [KröningStrichman'03,JussilaBiere'06]

Lemmas on Demand for ADCs



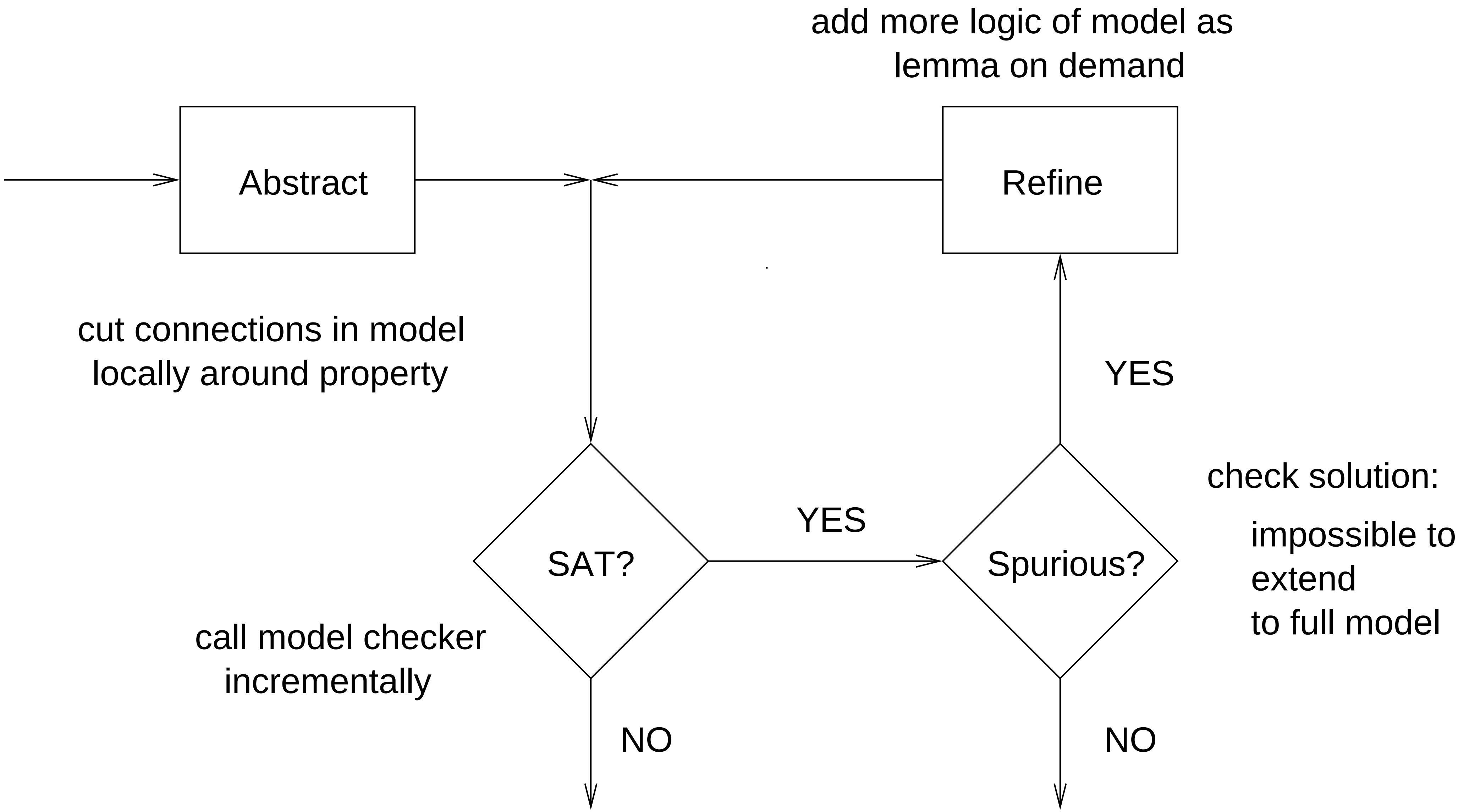
Lemmas on Demand for Satisfiability Modulo Theory (SMT)

[DeMouraRueß-SAT’02] [BarrettDillStump-CAV’02] . . .

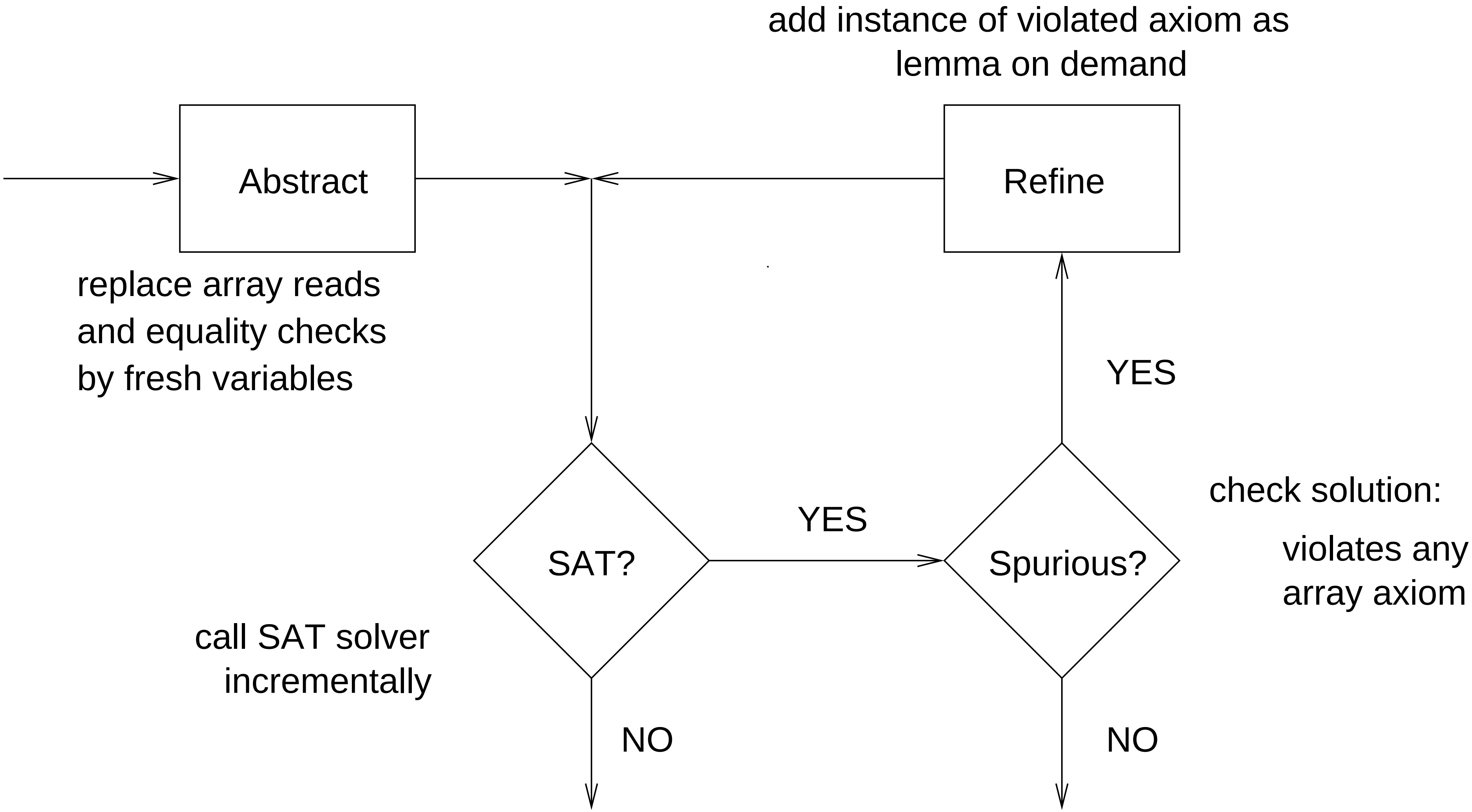


Localization / Counter Example Guided Abstraction Refinement

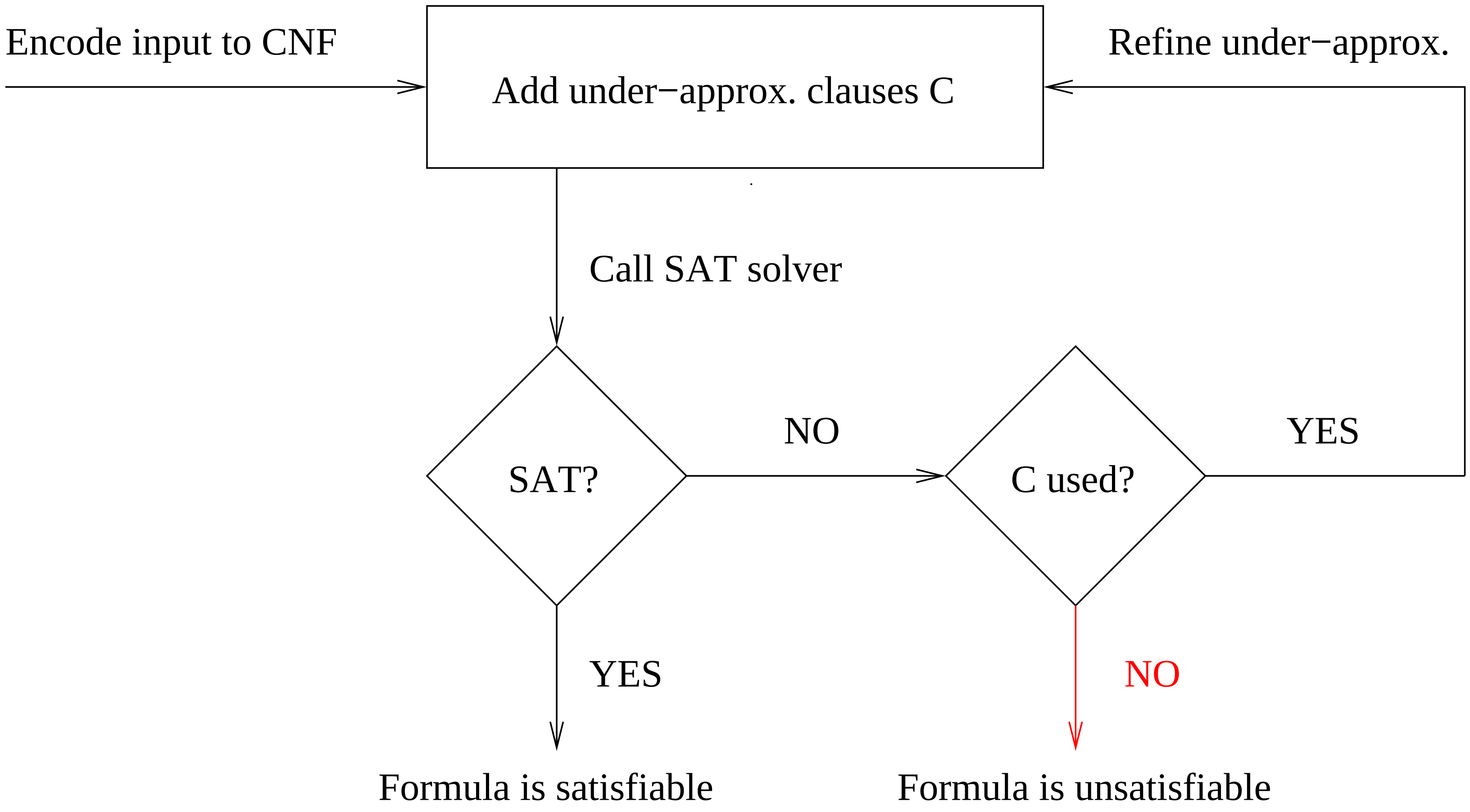
Localization [Kurshan'93], Predicate Abstraction [GrafSaidi'97],
SLAM [BallRajamani'01], CEGAR [ClarkeGrumbergJhaLuVeith'03]



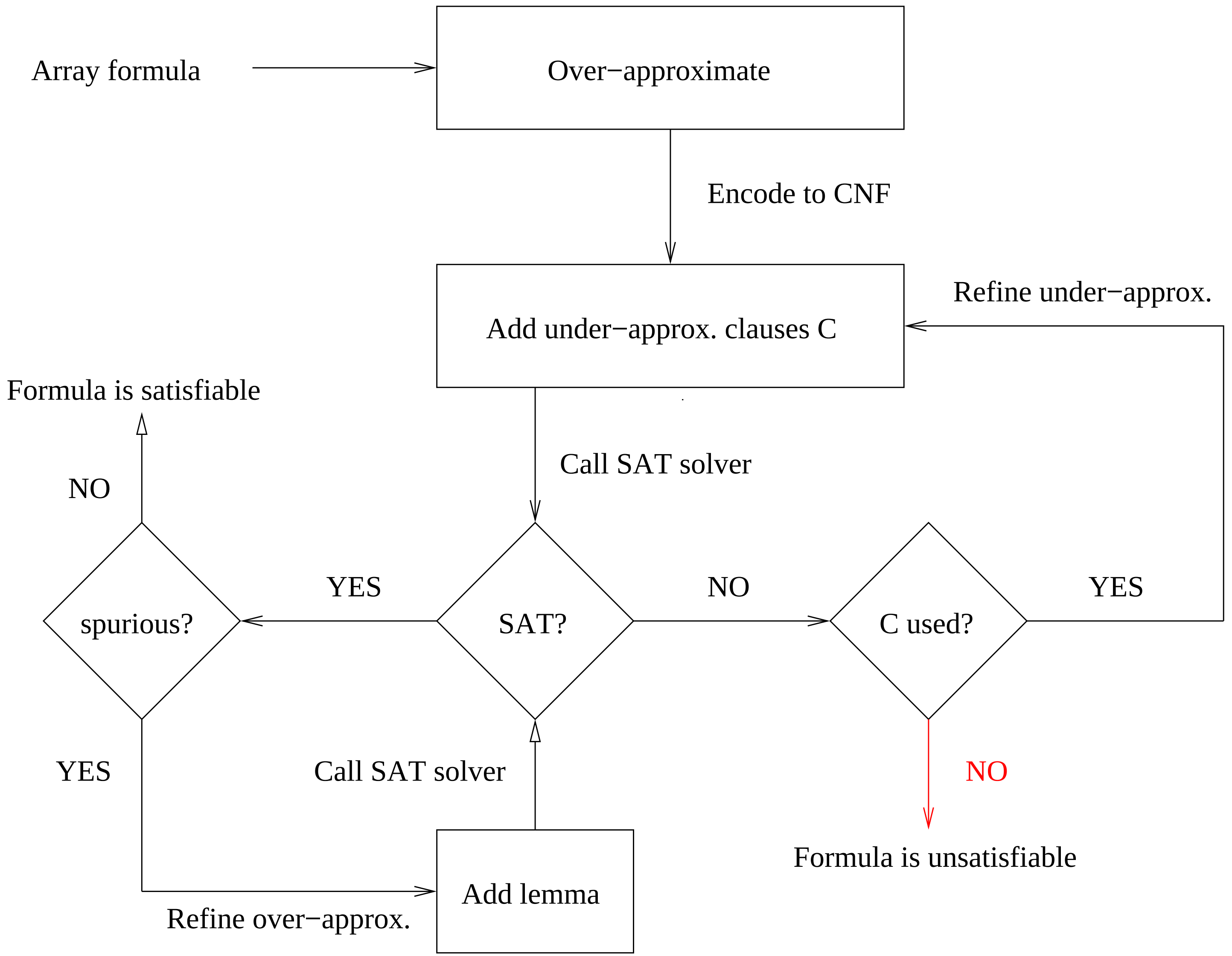
Lemmas on Demand for Extensional Arrays



Early Unsat Termination



Combining Over- and Under-Approximation

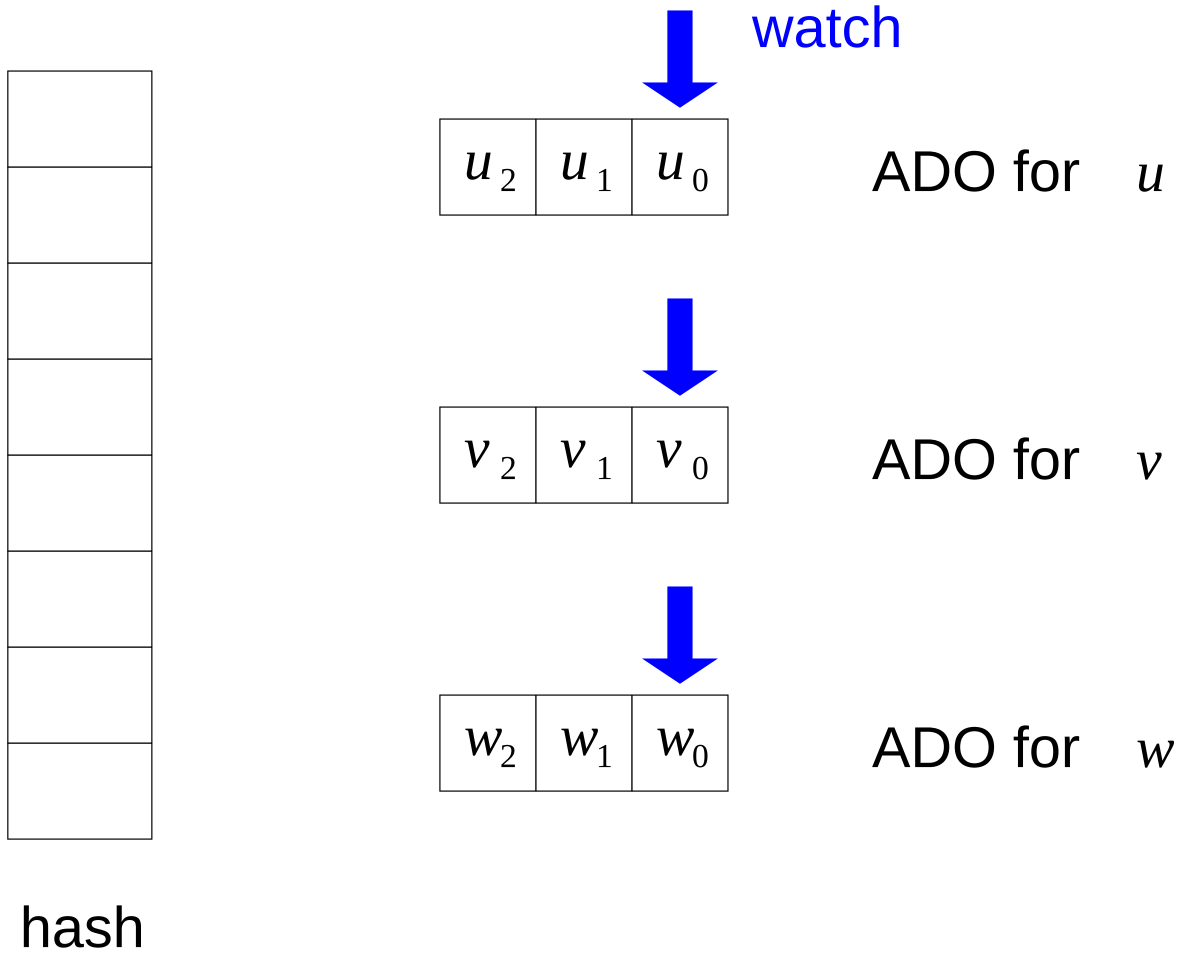


Lazy SMT

survey on lazy SMT [Sebastiani-JSAT'07]

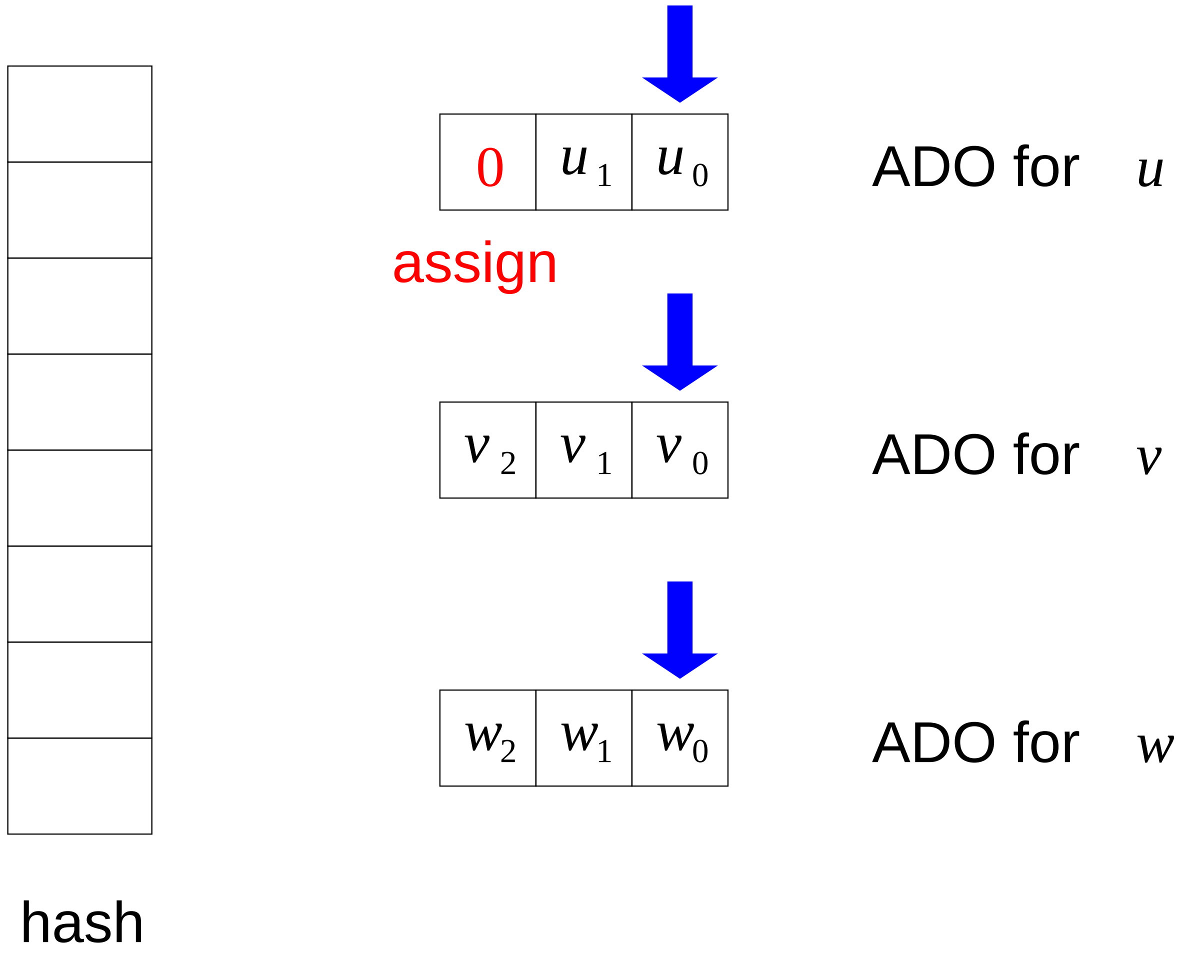
- Lemmas on Demand are as lazy as it gets
  - SAT solver enumerates full models of propositional skeleton
  - abstracted lemmas are added / learned on demand
  - theory solver checks consistency of conjunction of theory literals
- on-the-fly consistency checking
  - additionally theory solver checks consistency of partial model as well
- theory propagation
  - theory solver even deduces and notifies SAT solver about implied values of literals
- generic framework: DPLL(T) [NieuwenhuisOliverasTinelli-JACM'06]

All Different Objects (ADOs)

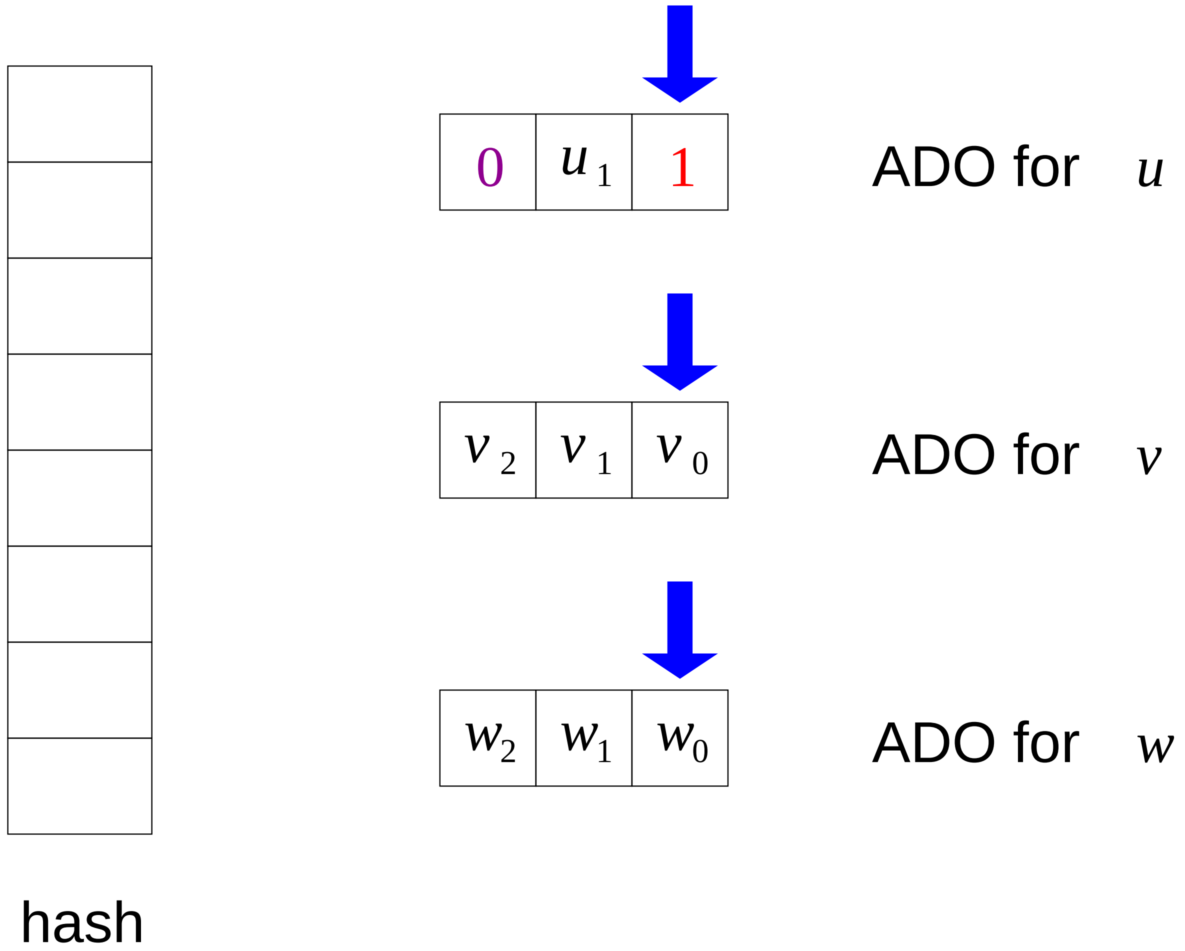


All Different Objects (ADOs)

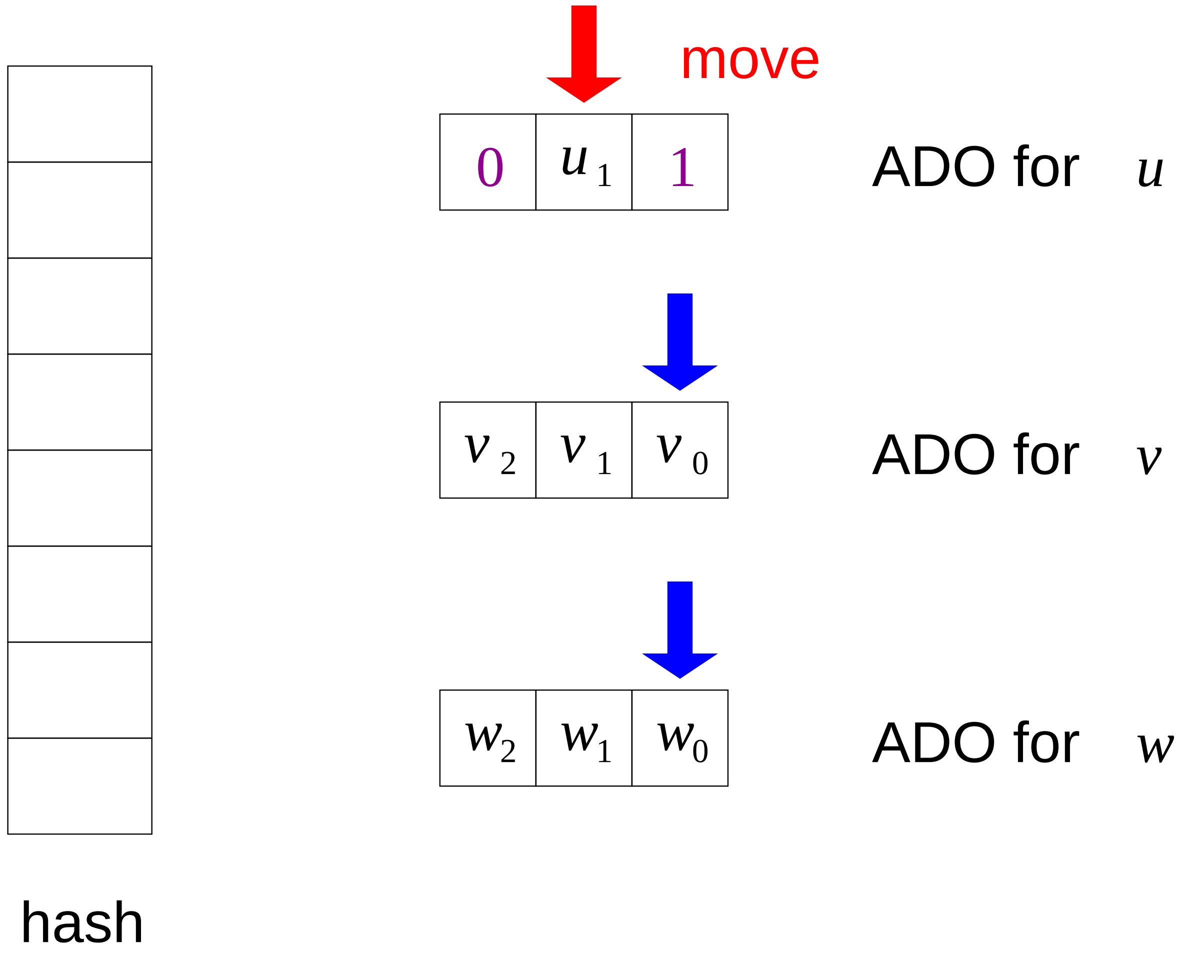
[BiereBrummayer-FMCAD'08]



All Different Objects (ADOs)

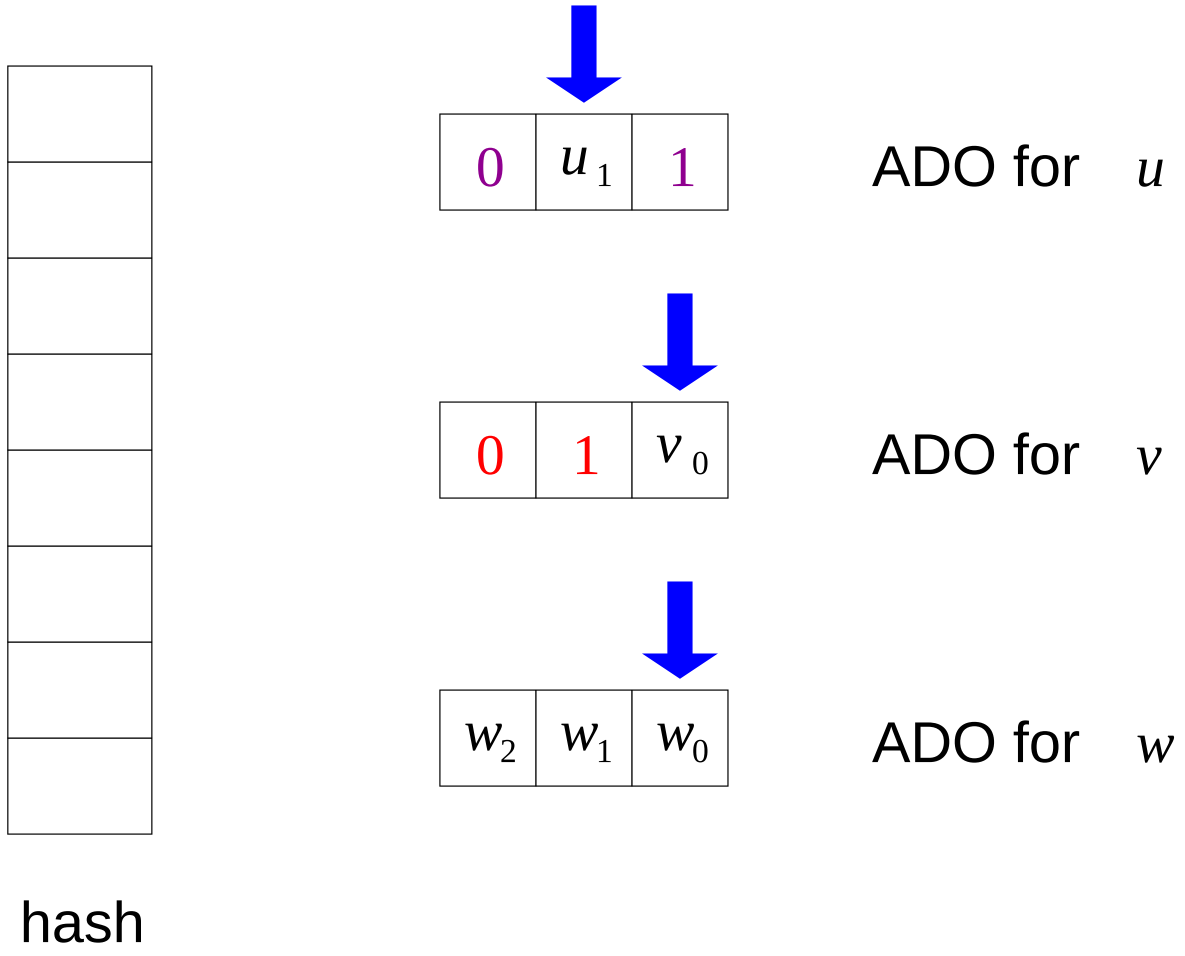


All Different Objects (ADOs)

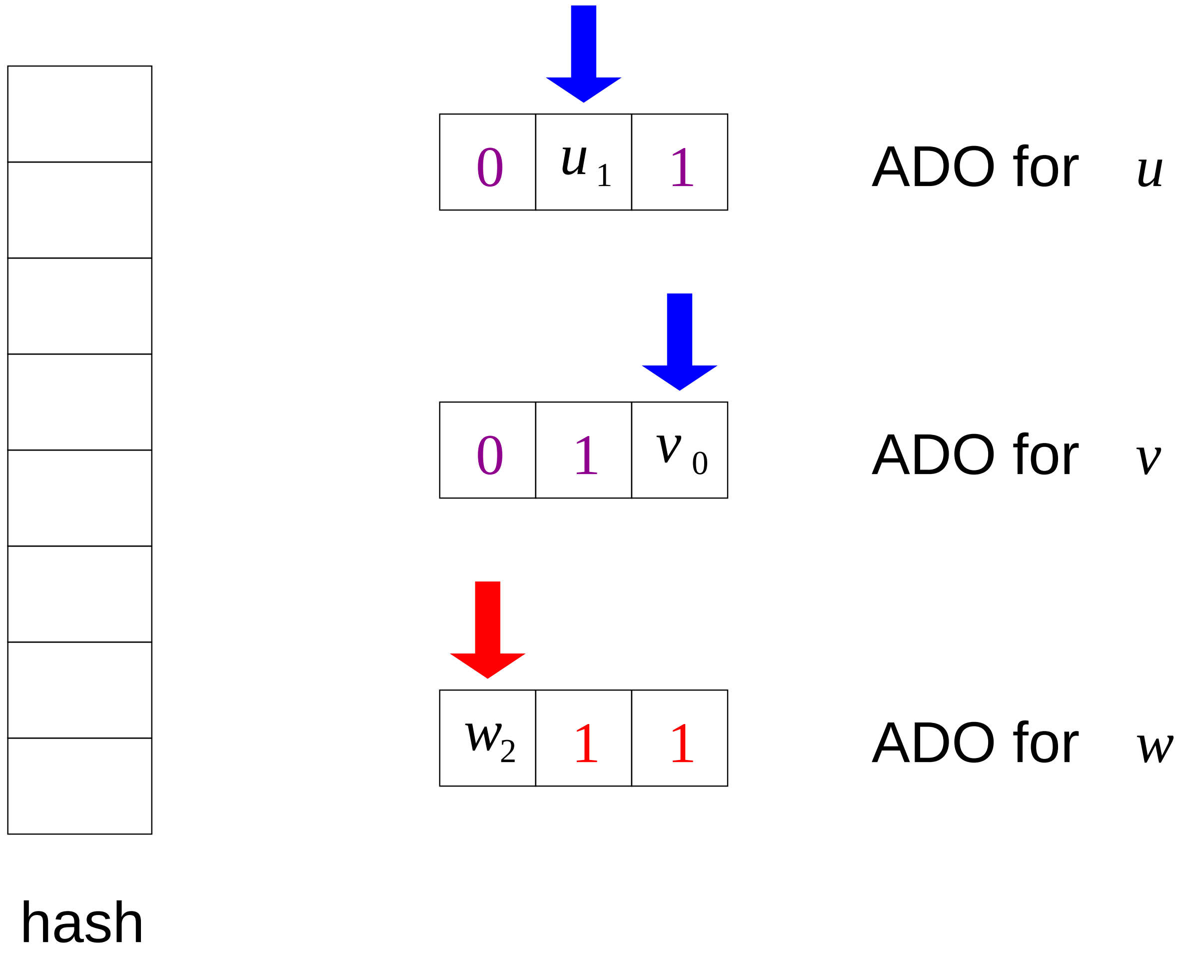


All Different Objects (ADOs)

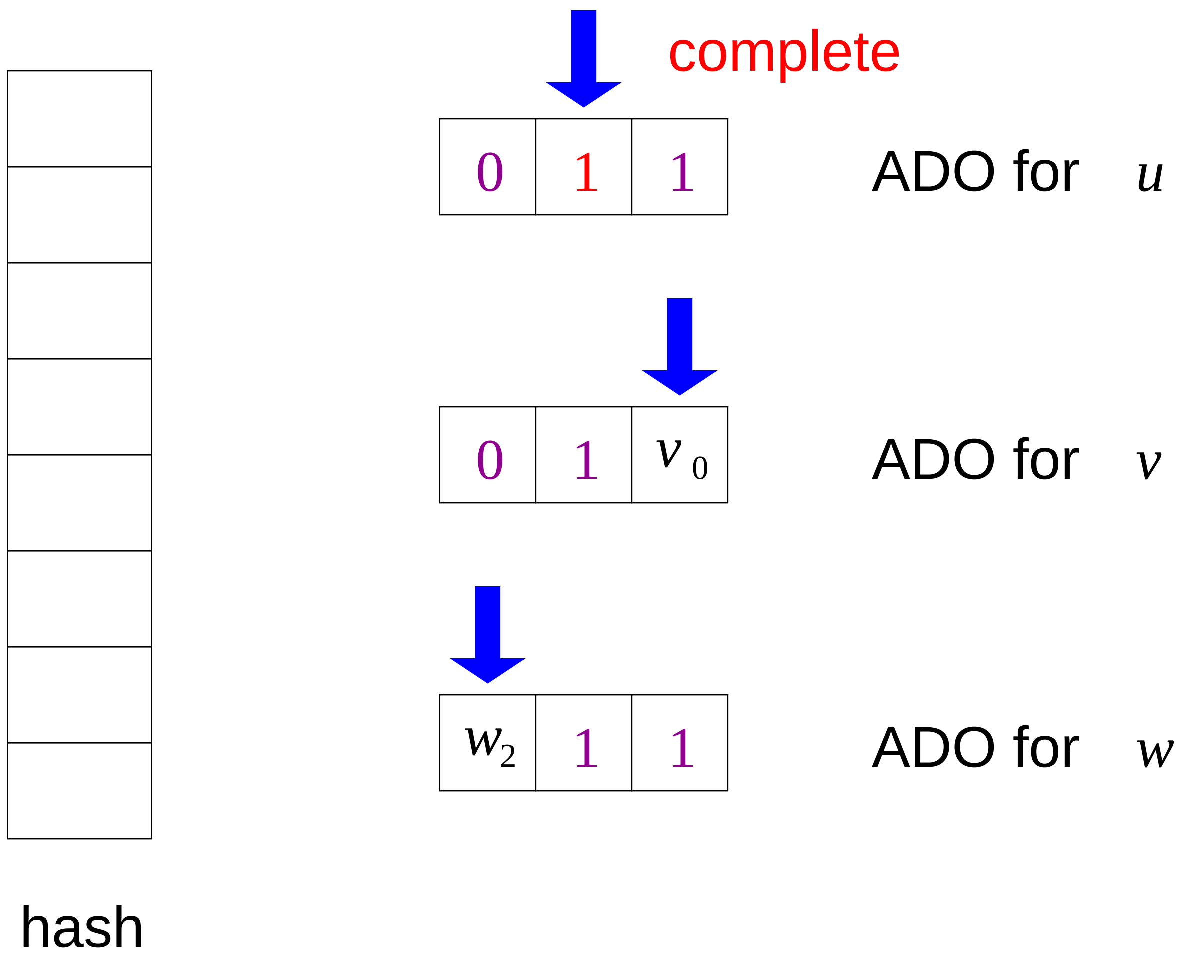
[BiereBrummayer-FMCAD'08]



All Different Objects (ADOs)

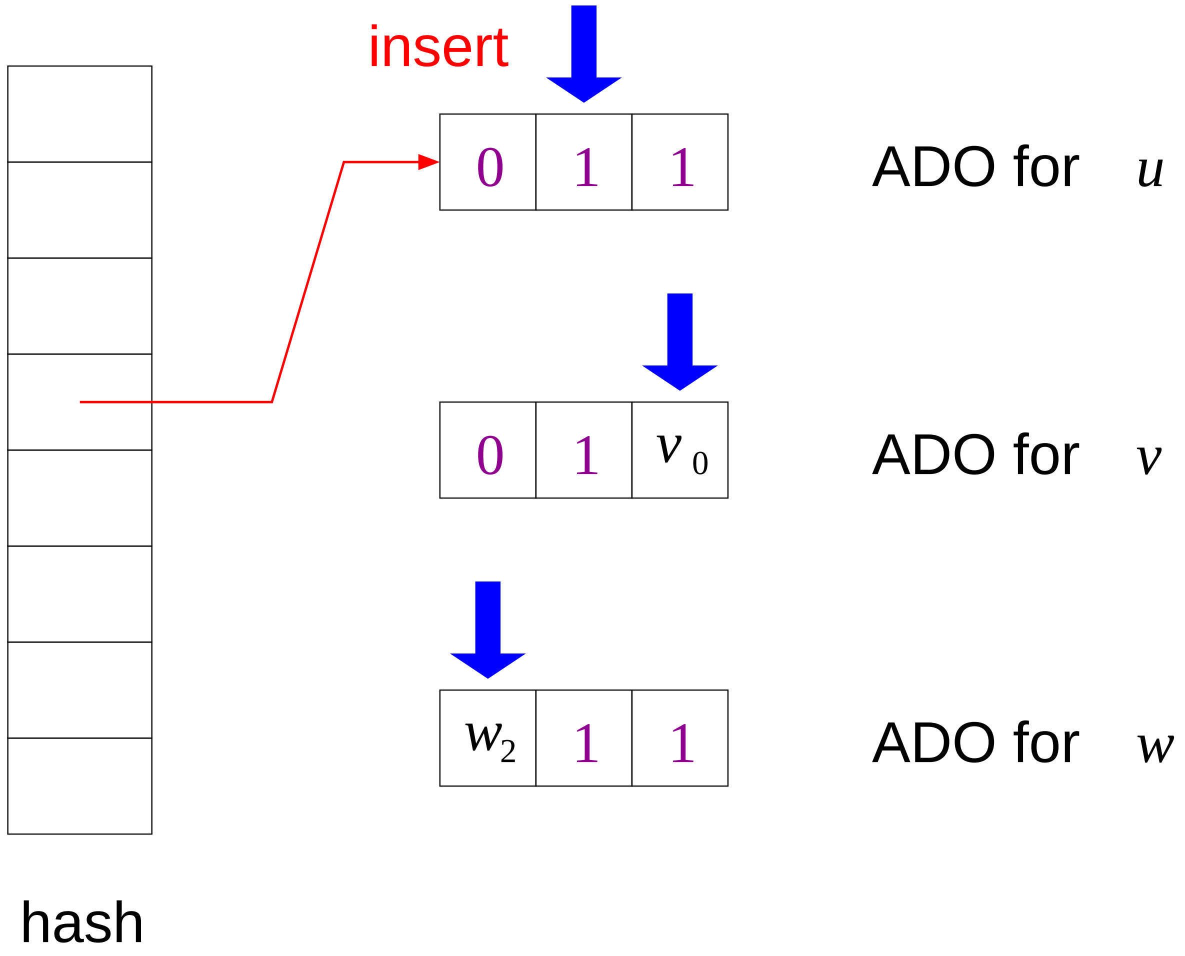


All Different Objects (ADOs)

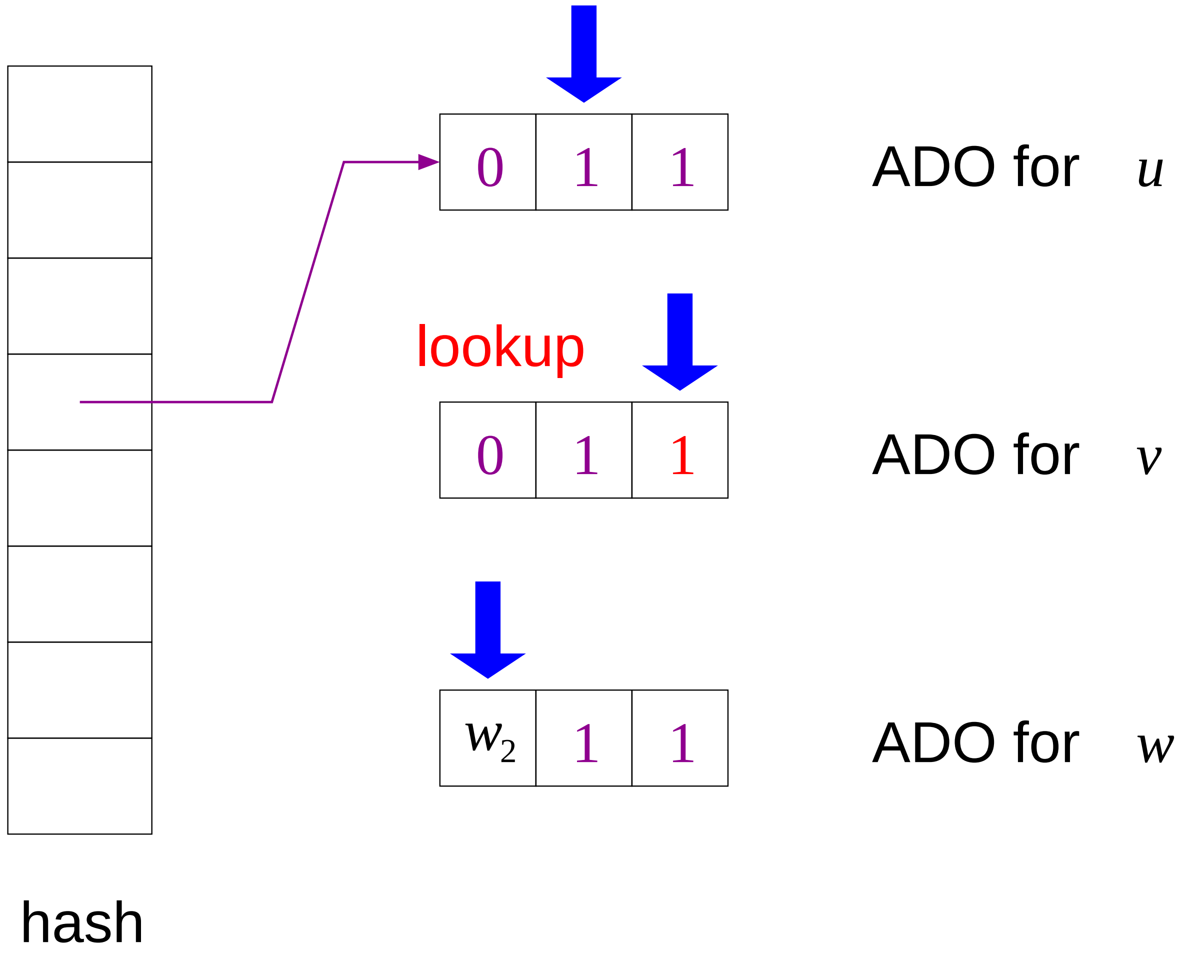


All Different Objects (ADOs)

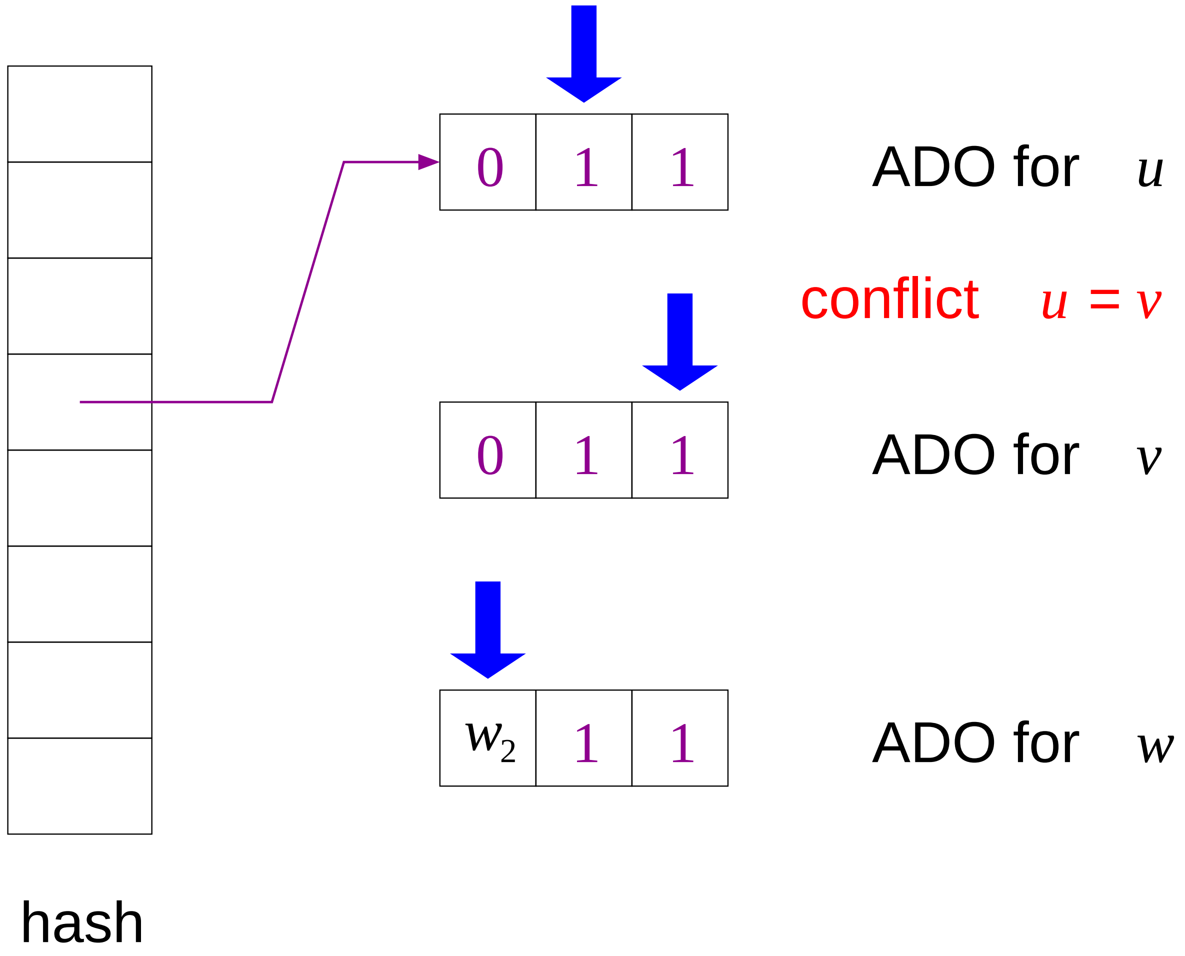
[BiereBrummayer-FMCAD'08]



All Different Objects (ADOs)



All Different Objects (ADOs)

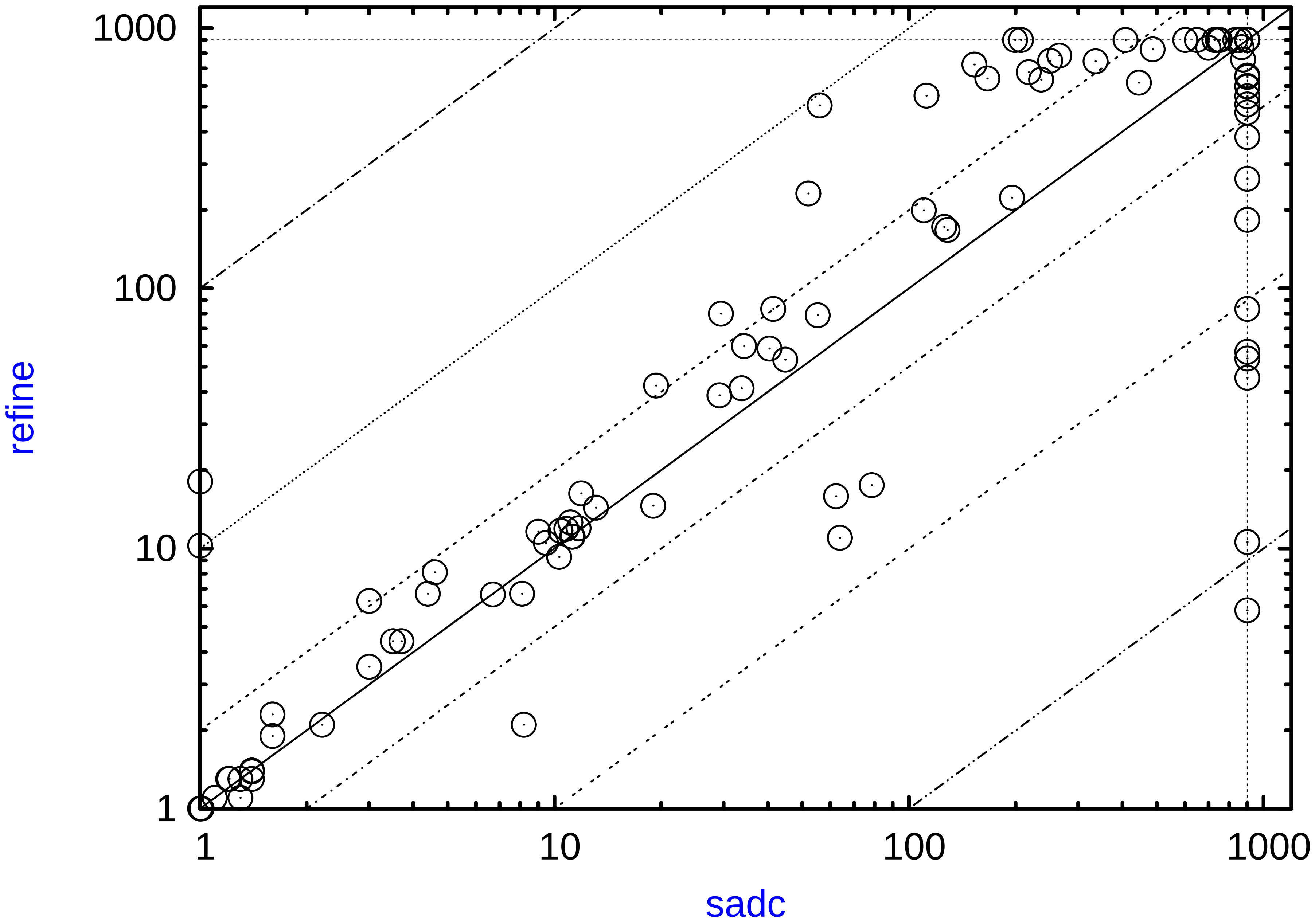


Implementation Details

- ADO key is calculated from concrete bit-vector
  - by for instance XOR'ing bits word by word
- ADOs watched by variables (not literals)
  - during backtracking all inserted ADOs need to be removed from hash table
  - save whether variable assignment forced ADO to be inserted
  - stack like insert/remove operations on hash table allow open addressing
- conflict analysis
  - all bits of the bit-vectors in conflict are followed
  - can be implemented by temporarily generating a pseudo clause

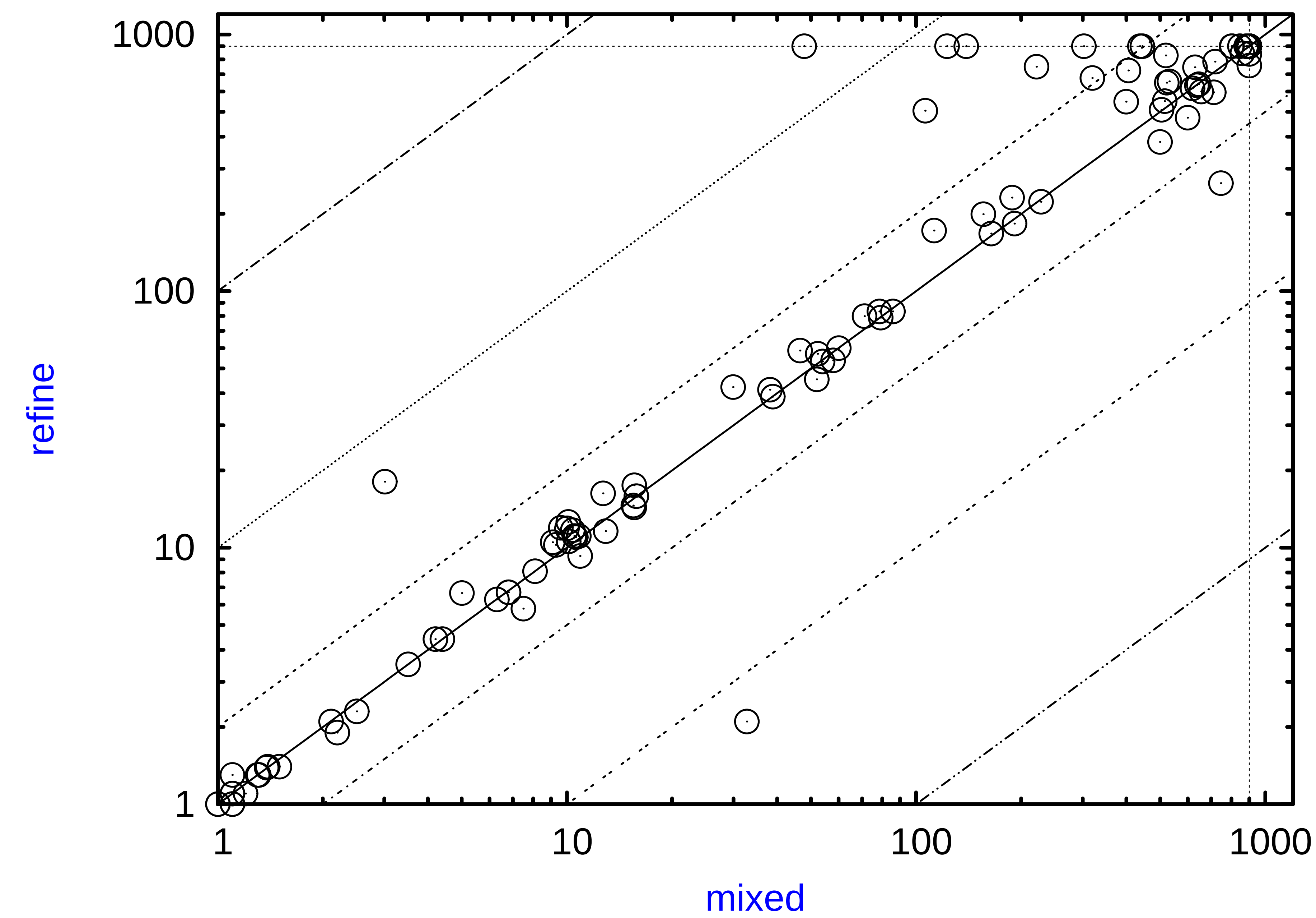
$$(u_2 \vee \overline{u}_1 \vee \overline{u}_0 \vee v_2 \vee \overline{v}_1 \vee \overline{v}_0)$$

Symbolic ADCs versus Refine



Mixed Approach versus Refine Only

[BiereBrummayer-FMCAD'08]

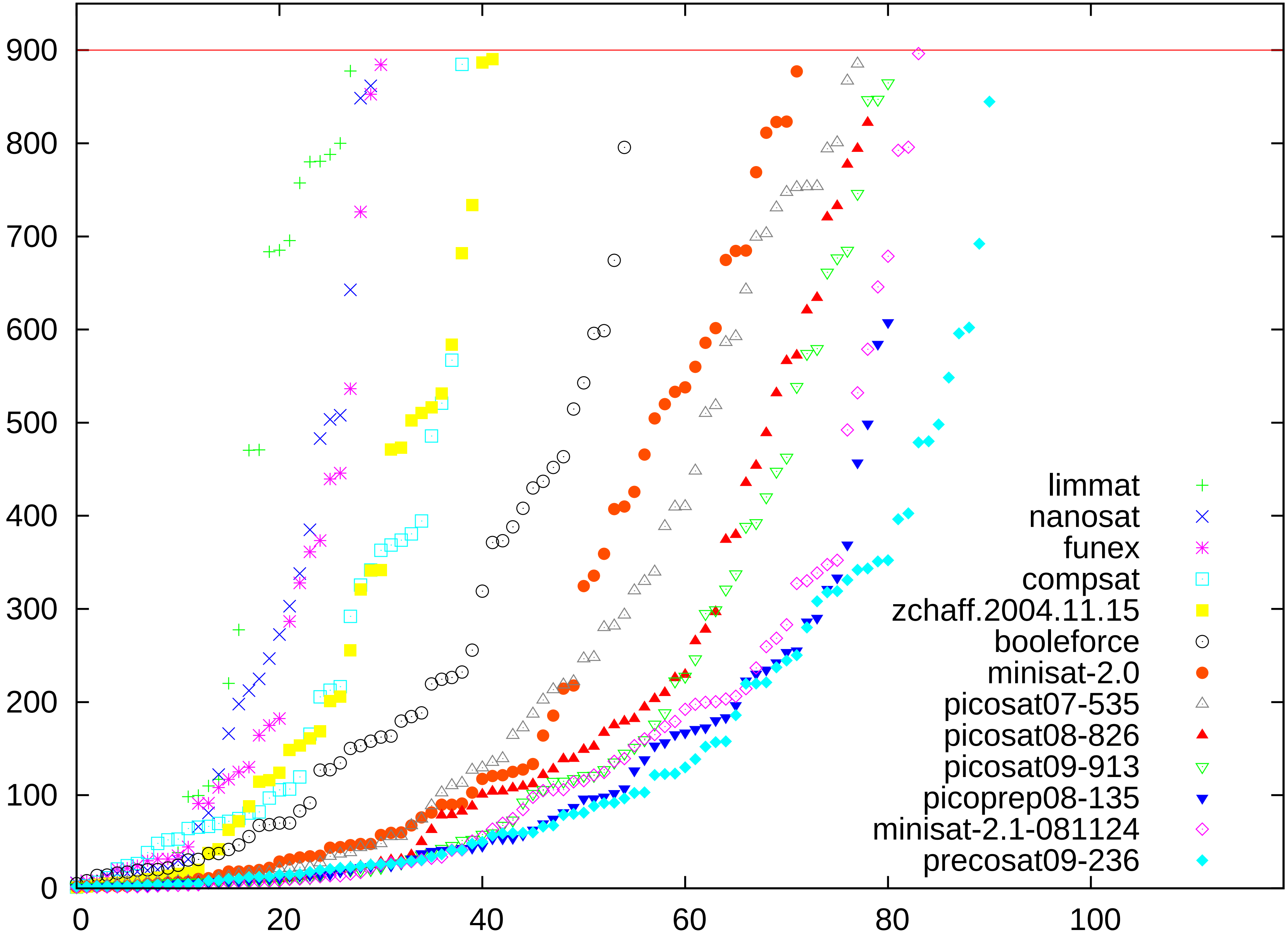


Conclusion on Symbolic All-Different Constraints

[BiereBrummayer-FMCAD'08]

- symbolic consistency checker for ADCs over bit-vectors
  - successfully applied to simple path constraints in model checking
  - similar to theory consistency checking in lazy SMT solvers
  - combination with eager refinement approach   lemmas on demand
- future work: ADC based BCP for bit-vectors
  - aka theory propagation in lazy SMT solvers
  - extensions to handle Ackermann constraints or even McCarthy axioms
  - one-way to get away from pure bit-blasting in BV

SAT Solver Progress (SAT'06 Race Instances)



- SAT and SMT have seen tremendous improvements in recent years

- many applications through the whole field of computer science

- still lots of opportunities for improvements:
  - parallel SAT solving
  - integration of new paradigms
  - portfolio and preprocessing (PrecoSAT as first attempt)
  - improved decision procedures for SW / HW verificiation
  - make quantified boolean formula (QBF) reasoning work