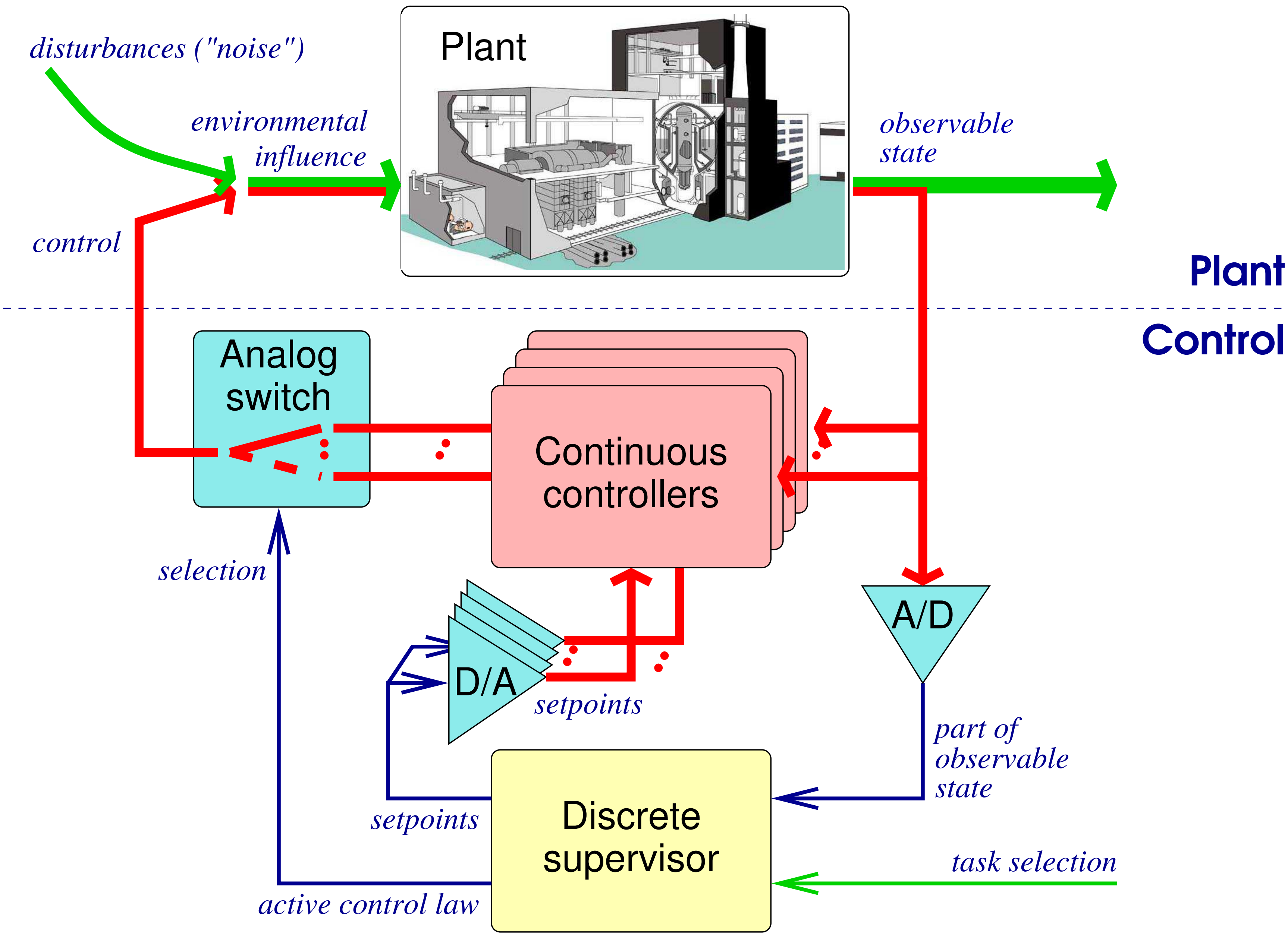
Satisfiability Solving in Arithmetic Domains

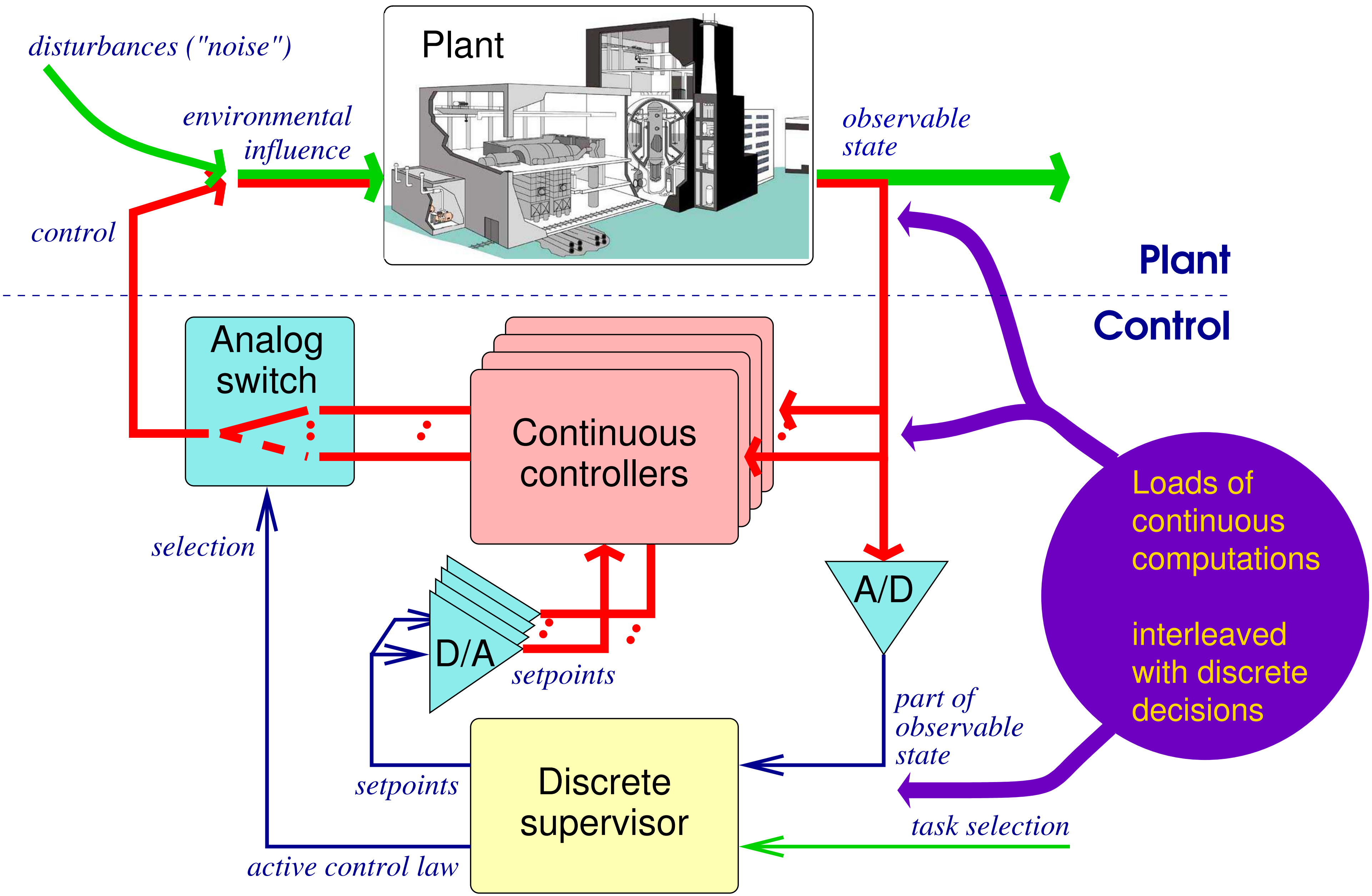
Martin Fränzle

Dept. of CS, Carl von Ossietzky Universität Oldenburg, Germany

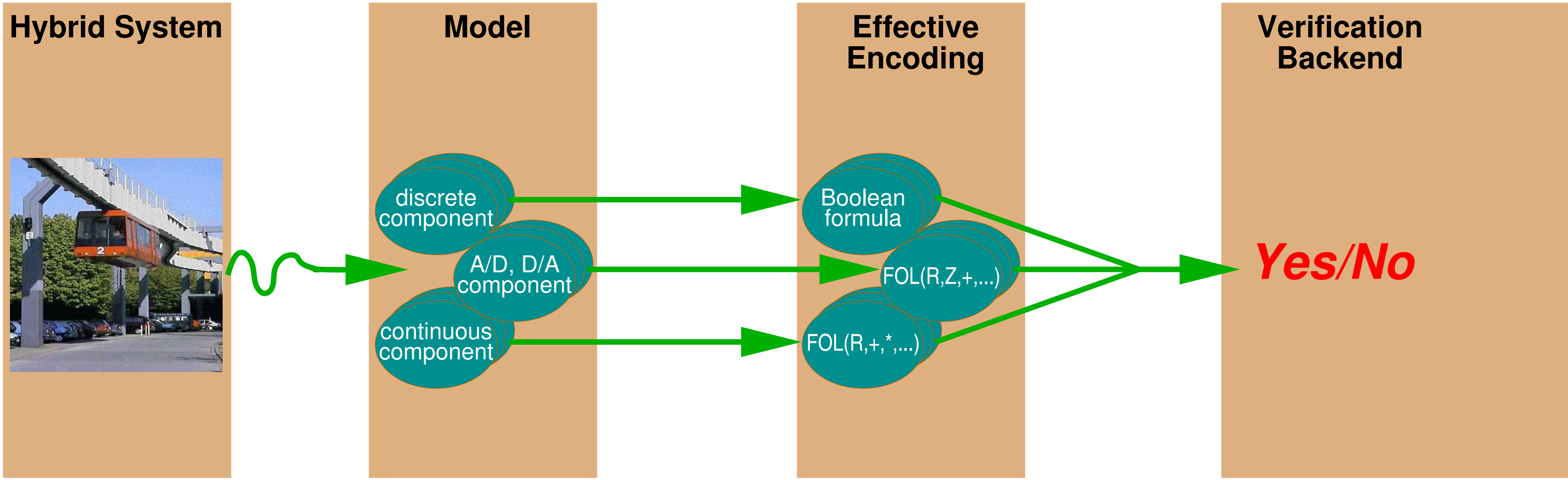
Hybrid Systems



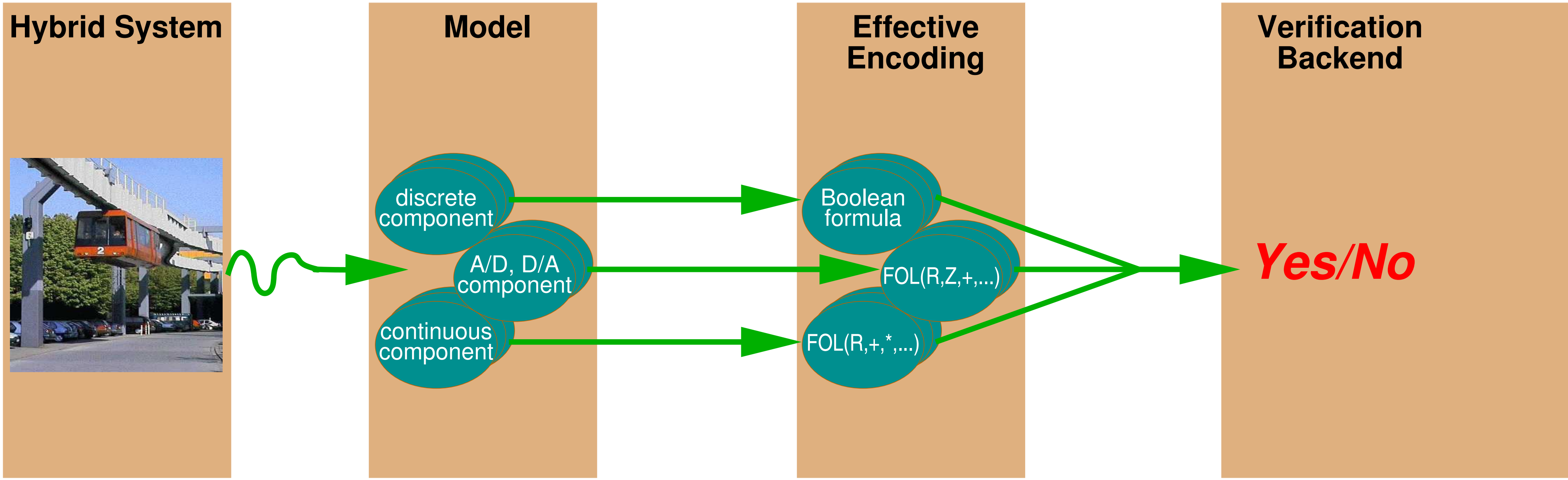
Hybrid Systems



Verification flow



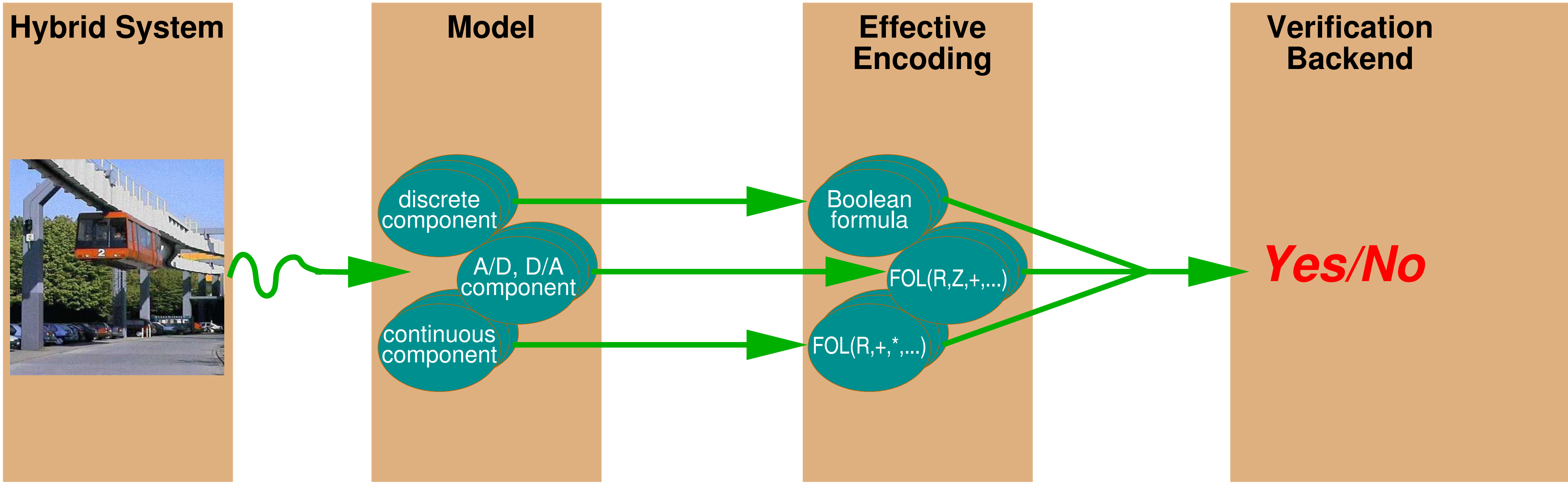
Verification flow



Formulae are

- extremely large arithmetic formulae with a rich Boolean structure
- over an undecidable domain, but approximation sufficient due to robustness of algorithms against manufacturing tolerances, rounding errors, . . .

Verification flow



Formulae are

- extremely large arithmetic formulae with a rich Boolean structure

  ⇒ Lazy Theorem Proving

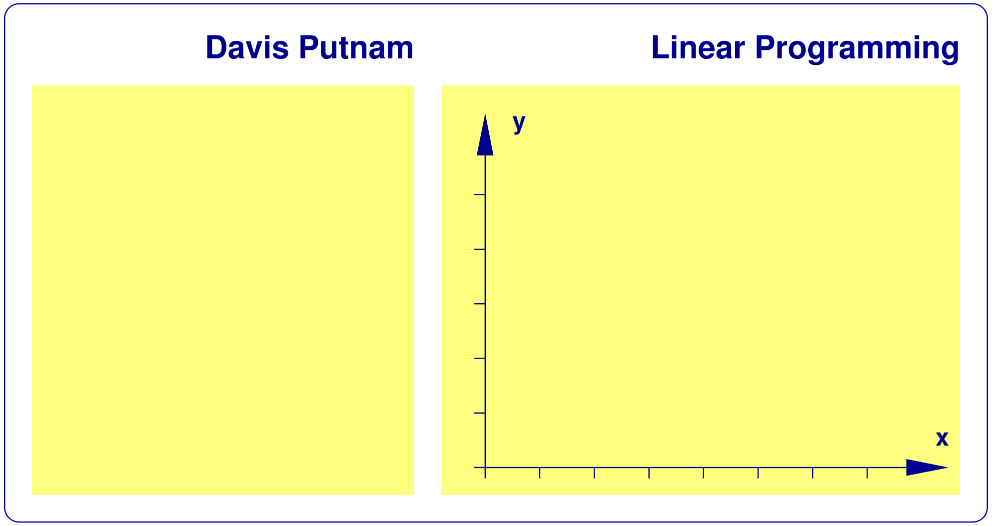
- over an undecidable domain, but approximation sufficient due to robustness of algorithms against manufacturing tolerances, rounding errors, . . .

  ⇒ Interval Constraint Solving

Satisfiability solving for decidable theories:

Lazy theorem proving & DPLL(T)

The Lazy TP Scheme: LinSAT



Input formula:

$$
\begin{aligned}
\Phi = {} & (\overline{e} \longrightarrow C \wedge D) \\
& \wedge \left(\overline{f} \longrightarrow A \wedge B\right) \\
& \wedge \left(\overline{f} \vee g \vee e\right) \\
& \wedge \left(\overline{g} \vee \overline{f}\right) \\
& \wedge (e \longrightarrow (C \vee D) \wedge g) \\
& \wedge (A \longrightarrow (4x - 2y \geq 9)) \\
& \wedge (B \longrightarrow (2x - 4y \leq -7)) \\
& \wedge (C \longrightarrow (x + y \leq 5)) \\
& \wedge (D \longrightarrow (x \leq 7))
\end{aligned}
$$

Backtrack search

1. traversing possible truth-value assignments of Boolean part
2. incrementally (de-)constructing a *conjunctive* arithmetic constraint system
3. querying external solver to determine consistency of arithm. constr. syst.

The Lazy TP Scheme: LinSAT

Davis Putnam

$2e + C + D \geq 2$

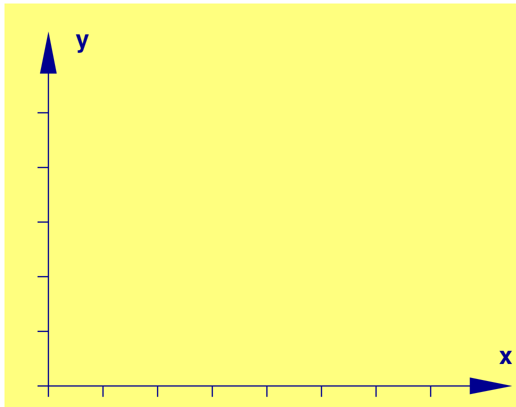
$2f + A + B \geq 2$

$\overline{f} + g + e \geq 1$

$\overline{g} + \overline{f} \geq 1$

$3\overline{e} + 2g + C + D \geq 3$

Linear Programming



Input formula:

$$\Phi = (\overline{e} \rightarrow C \wedge D)$$
$$\wedge\ (\overline{f} \rightarrow A \wedge B)$$
$$\wedge\ (\overline{f} \vee g \vee e)$$
$$\wedge\ (\overline{g} \vee \overline{f})$$
$$\wedge\ (e \rightarrow (C \vee D) \wedge g)$$
$$\wedge\ (A \rightarrow (4x - 2y \geq 9))$$
$$\wedge\ (B \rightarrow (2x - 4y \leq -7))$$
$$\wedge\ (C \rightarrow (x + y \leq 5))$$
$$\wedge\ (D \rightarrow (x \leq 7))$$

Backtrack search

1. traversing possible truth-value assignments of Boolean part
2. incrementally (de-)constructing a *conjunctive* arithmetic constraint system
3. querying external solver to determine consistency of arithm. constr. syst.

The Lazy TP Scheme: LinSAT

Davis Putnam

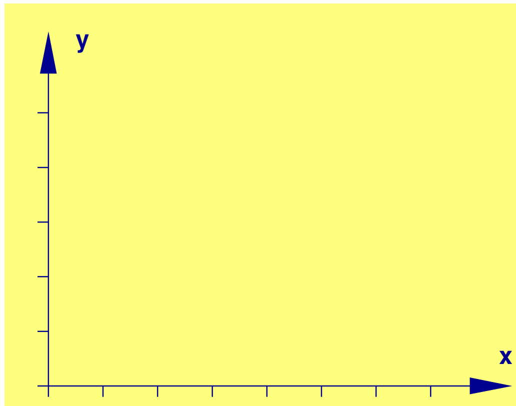
$C + D \geq 2$

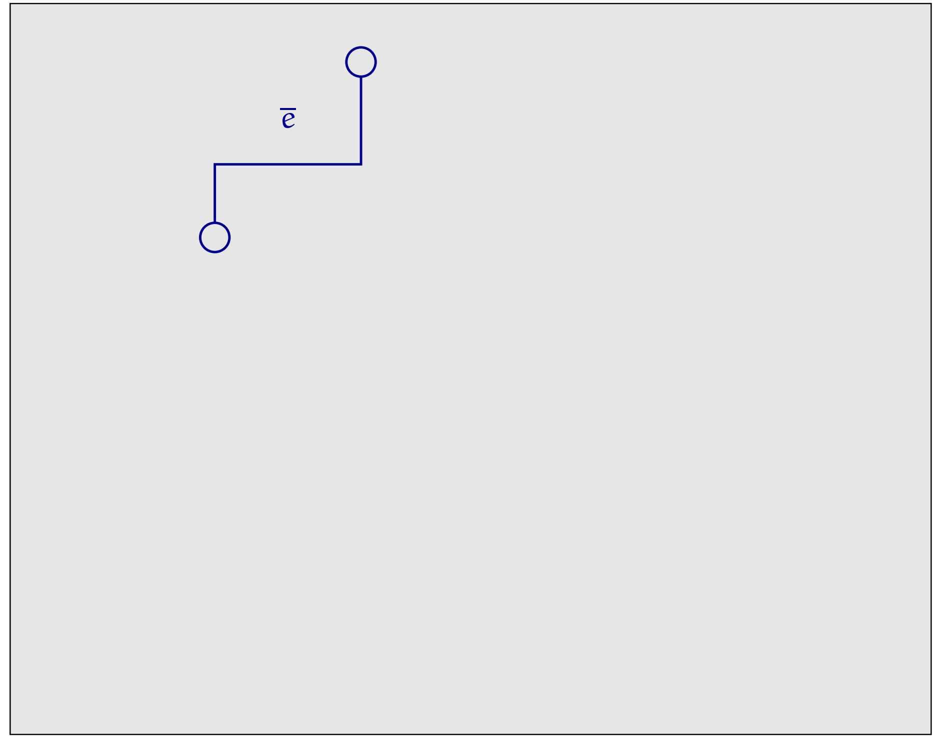
$2f + A + B \geq 2$

$\overline{f} + g \geq 1$

$\overline{g} + \overline{f} \geq 1$

Linear Programming





Backtrack search

1. traversing possible truth-value assignments of Boolean part
2. incrementally (de-)constructing a *conjunctive* arithmetic constraint system
3. querying external solver to determine consistency of arithm. constr. syst.

The Lazy TP Scheme: LinSAT

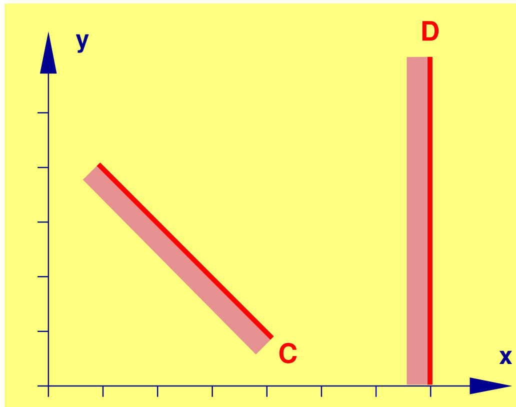
Davis Putnam

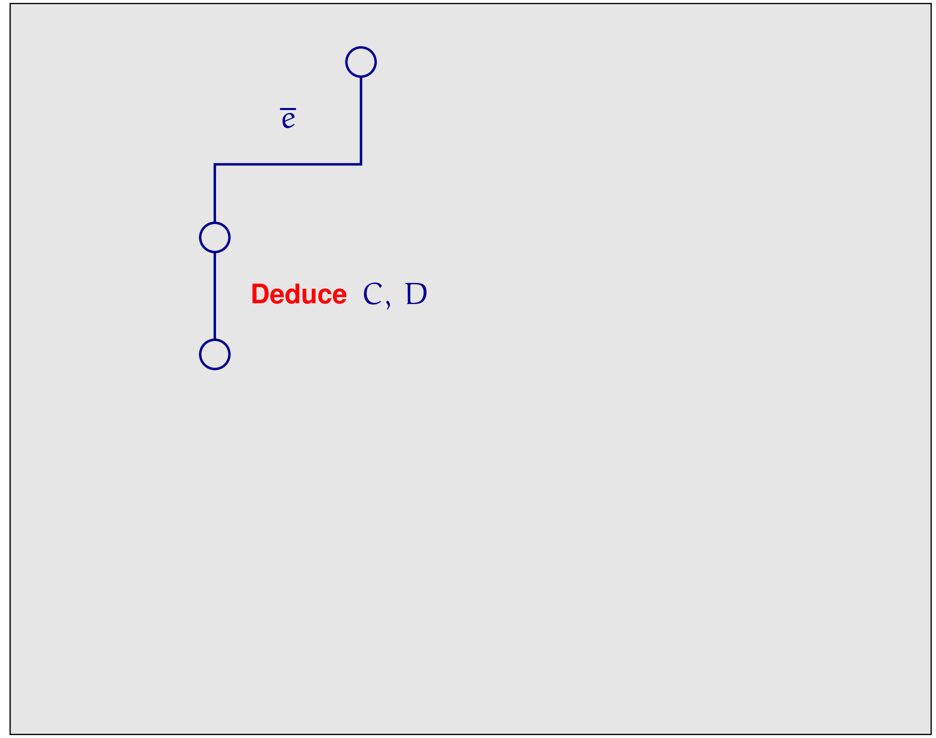
$2f + A + B \geq 2$

$\overline{f} + g \geq 1$

$\overline{g} + \overline{f} \geq 1$

Linear Programming

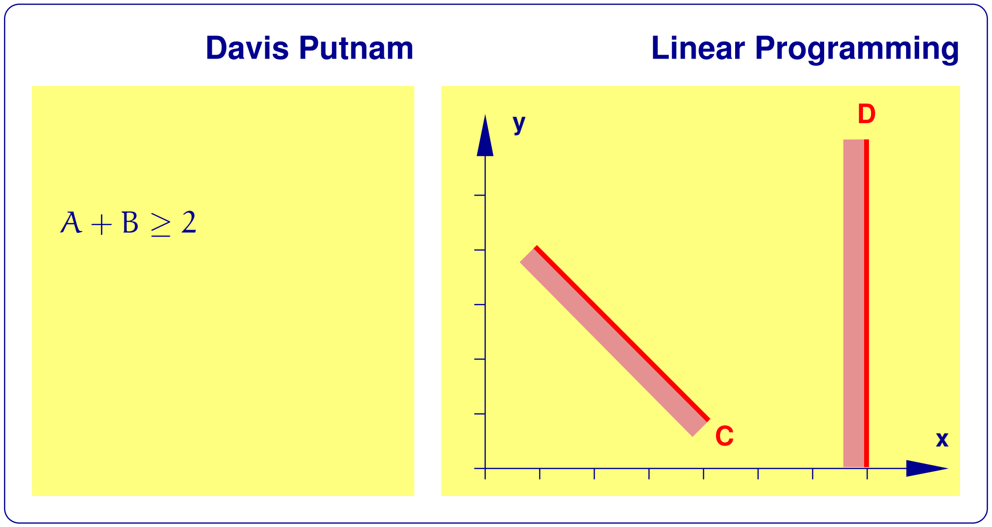


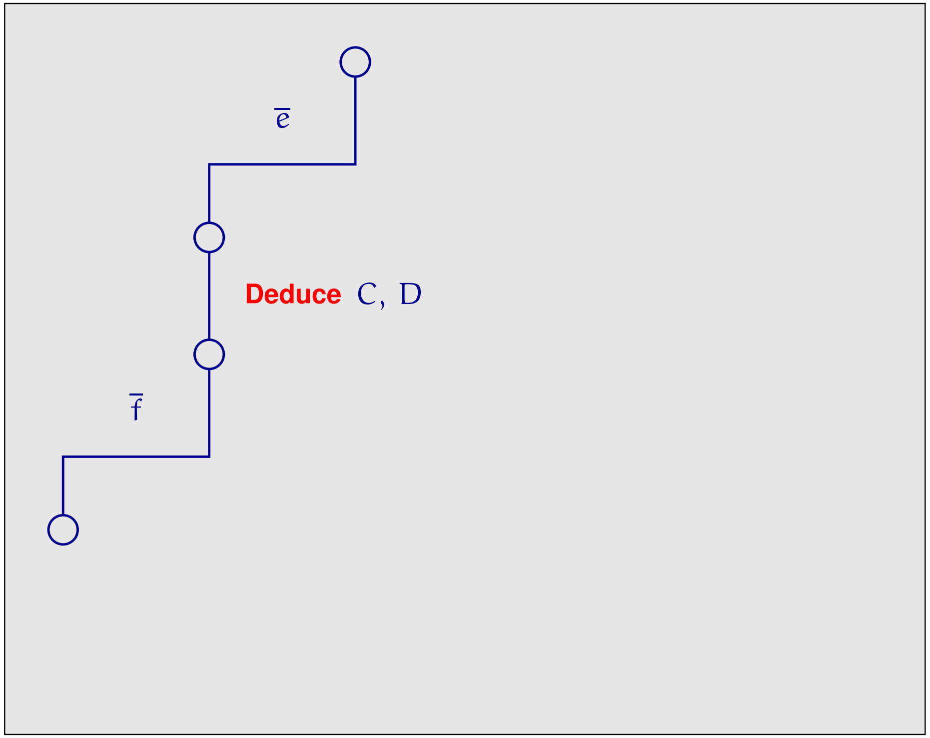


Backtrack search

1. traversing possible truth-value assignments of Boolean part
2. incrementally (de-)constructing a *conjunctive* arithmetic constraint system
3. querying external solver to determine consistency of arithm. constr. syst.

The Lazy TP Scheme: LinSAT

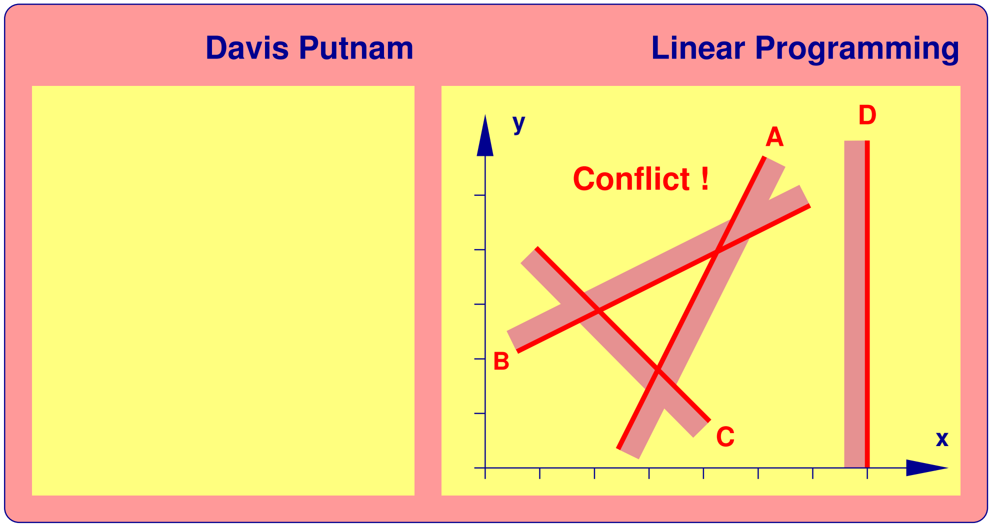


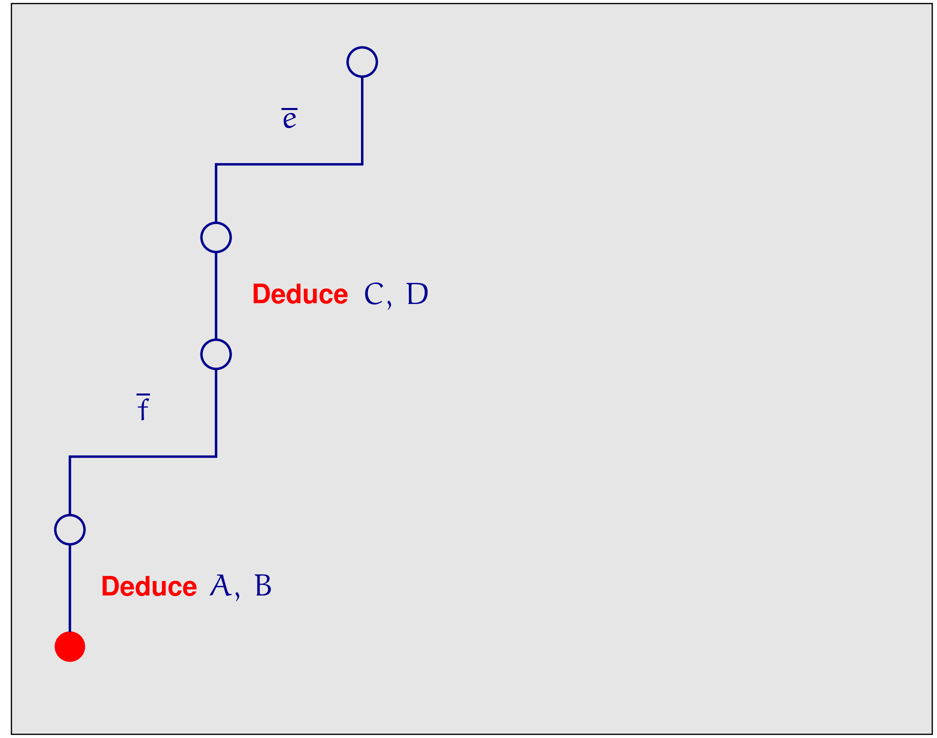


Backtrack search

1. traversing possible truth-value assignments of Boolean part
2. incrementally (de-)constructing a *conjunctive* arithmetic constraint system
3. querying external solver to determine consistency of arithm. constr. syst.

The Lazy TP Scheme: LinSAT





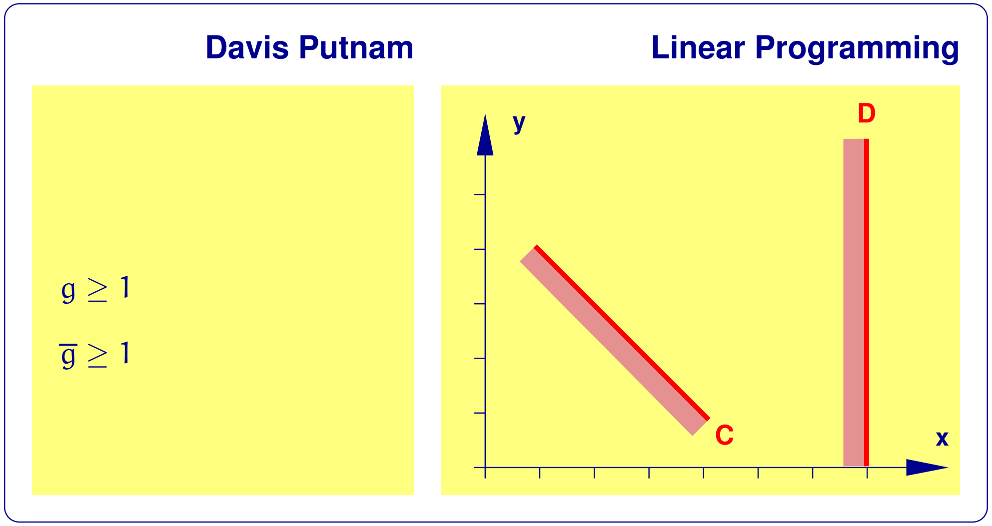
Irreducible infeasible subsystem is $\{A, B, C\}$

Learned conflict clause: $\overline{A} + \overline{B} + \overline{C} \geq 1$

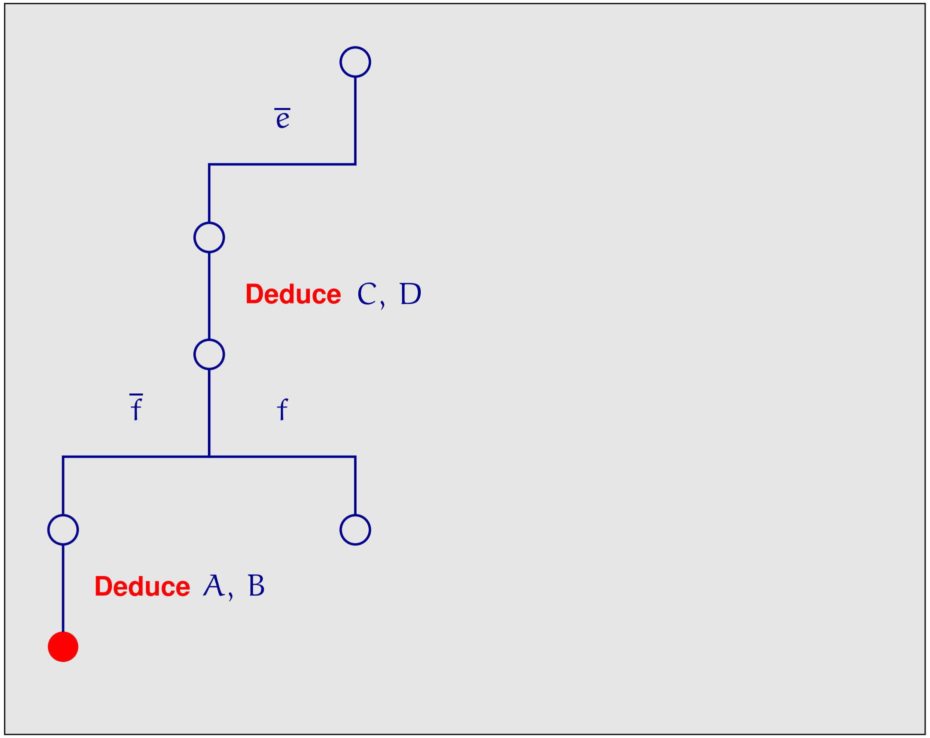
Backtrack search

1. traversing possible truth-value assignments of Boolean part
2. incrementally (de-)constructing a *conjunctive* arithmetic constraint system
3. querying external solver to determine consistency of arithm. constr. syst.

The Lazy TP Scheme: LinSAT



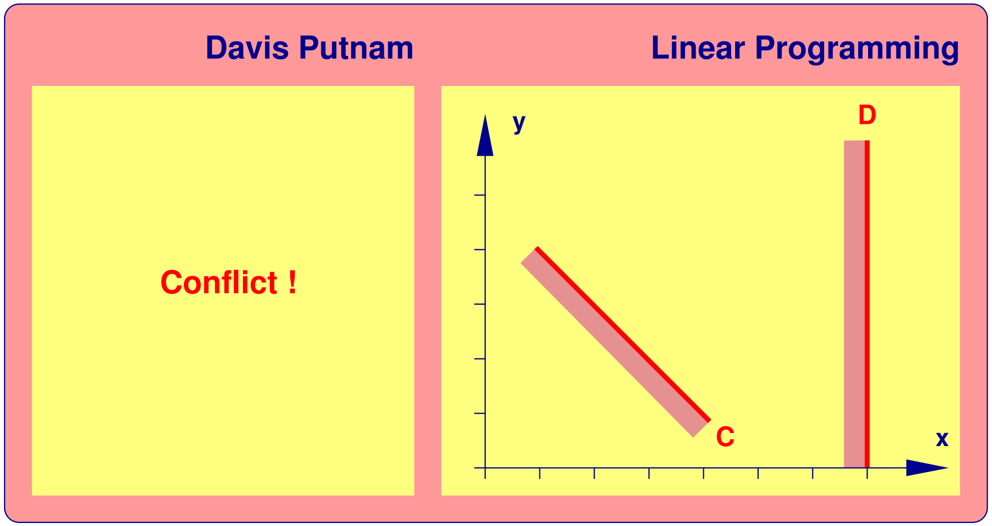
Learned conflict clause: $\overline{A} + \overline{B} + \overline{C} \geq 1$

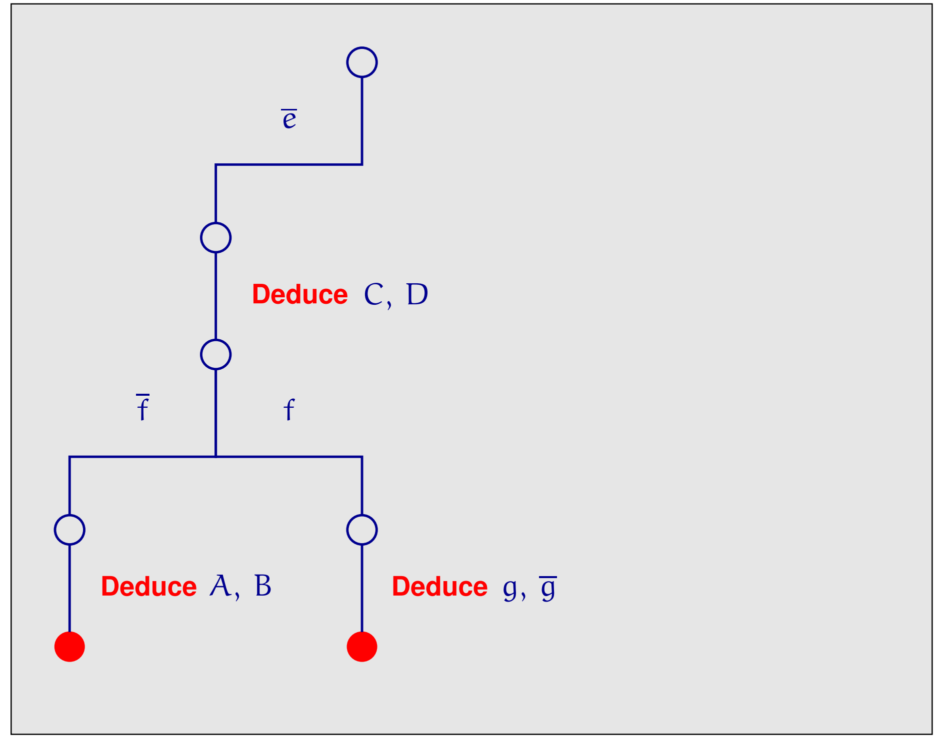


Backtrack search

1. traversing possible truth-value assignments of Boolean part
2. incrementally (de-)constructing a *conjunctive* arithmetic constraint system
3. querying external solver to determine consistency of arithm. constr. syst.

The Lazy TP Scheme: LinSAT



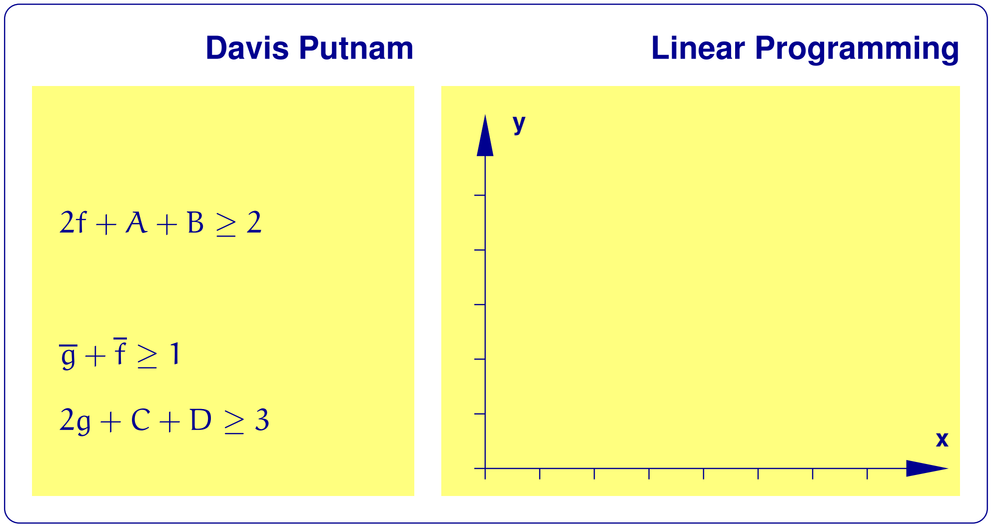


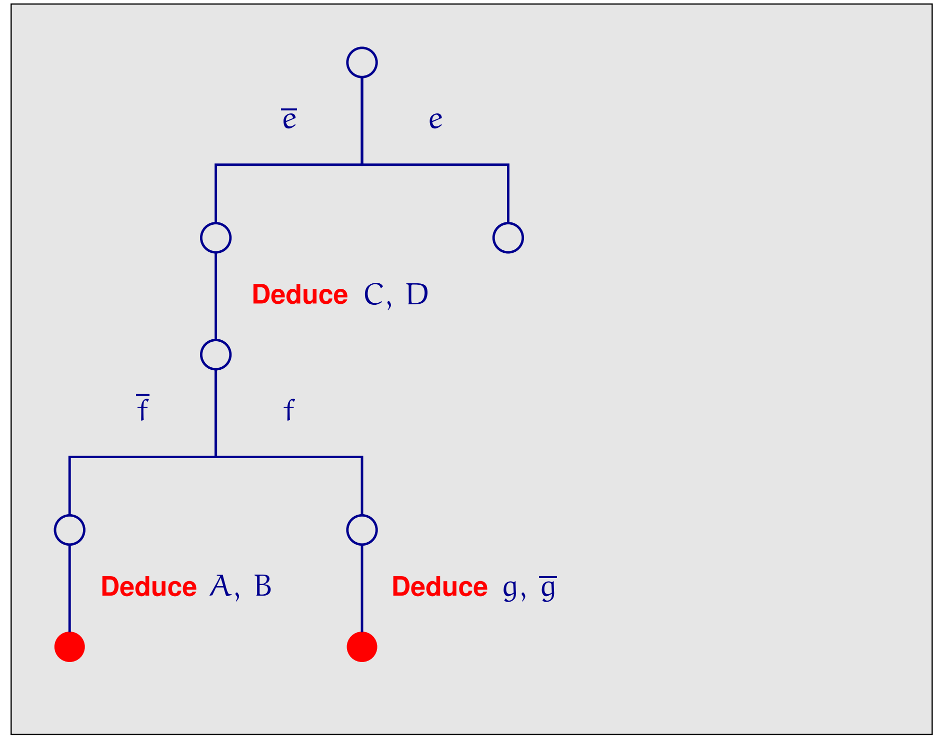
Learned conflict clause: $\overline{A} + \overline{B} + \overline{C} \geq 1$

Backtrack search

1. traversing possible truth-value assignments of Boolean part
2. incrementally (de-)constructing a *conjunctive* arithmetic constraint system
3. querying external solver to determine consistency of arithm. constr. syst.

The Lazy TP Scheme: LinSAT



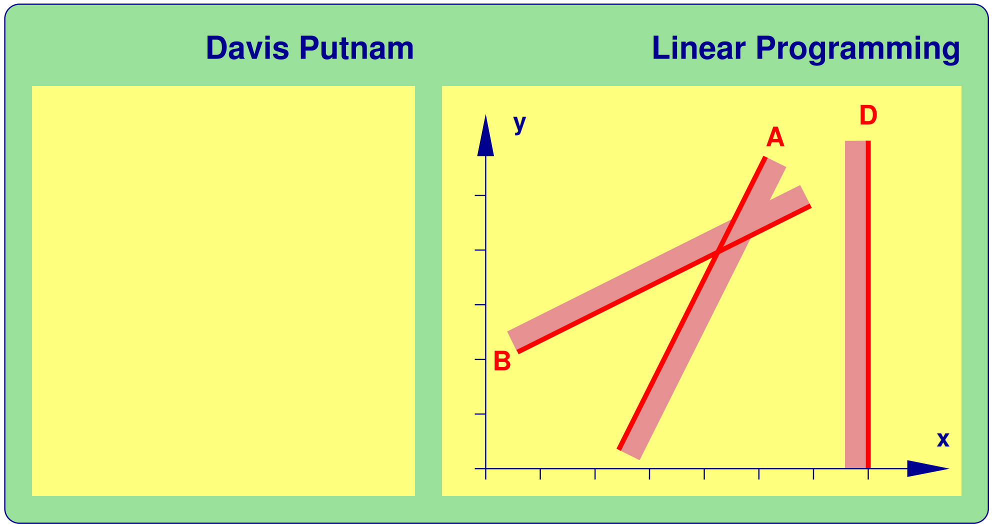


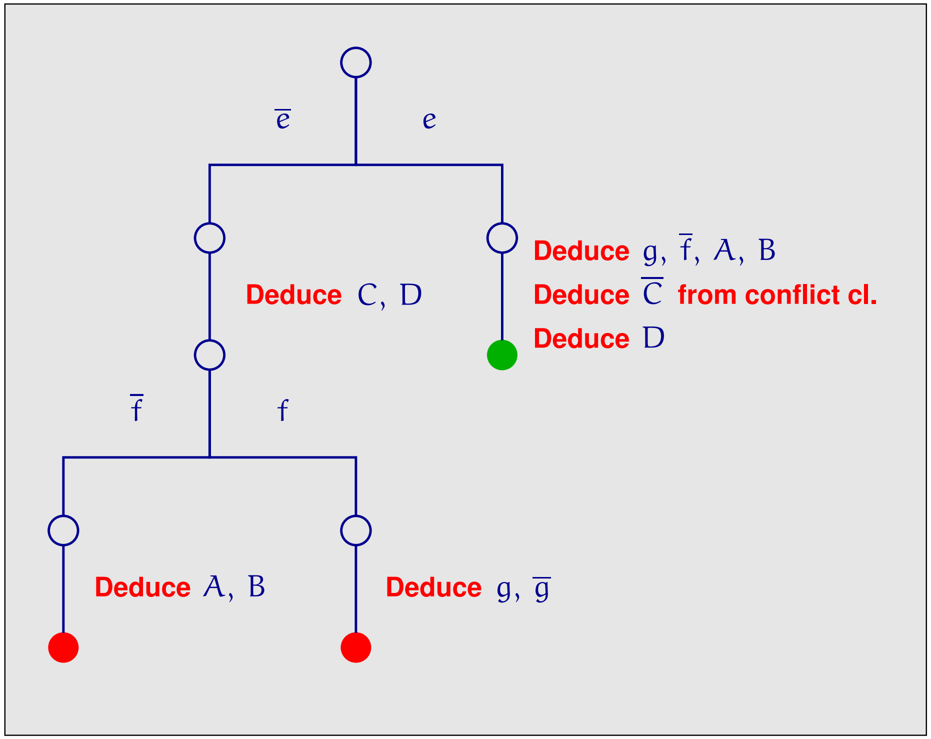
Learned conflict clause: $\overline{A} + \overline{B} + \overline{C} \geq 1$

Backtrack search

1. traversing possible truth-value assignments of Boolean part
2. incrementally (de-)constructing a *conjunctive* arithmetic constraint system
3. querying external solver to determine consistency of arithm. constr. syst.

The Lazy TP Scheme: LinSAT





Learned conflict clause: $\overline{A} + \overline{B} + \overline{C} \geq 1$

Backtrack search

1. traversing possible truth-value assignments of Boolean part
2. incrementally (de-)constructing a *conjunctive* arithmetic constraint system
3. querying external solver to determine consistency of arithm. constr. syst.

Deciding the conjunctive $\mathcal{T}$-problems

For $\mathcal{T}$ being linear arithmetic over $\mathbb{R}$, this can be done by linear programming:

$$\bigwedge_{i=1}^{n} \sum_{j=1}^{m} A_{i,j} x_j \leq b_j \quad \text{iff} \quad A\mathbf{x} \leq \mathbf{b}$$

$\leadsto$ Solving LP

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^T \mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \end{aligned}$$

with arbitrary $\mathbf{c}$ provides consistency information.

Deciding the conjunctive $\mathcal{T}$-problems

For $\mathcal{T}$ being linear arithmetic over $\mathbb{R}$, this can be done by linear programming:

$$\bigwedge_{i=1}^{n} \sum_{j=1}^{m} A_{i,j} x_j \leq b_j \quad \text{iff} \quad A\mathbf{x} \leq \mathbf{b}$$

$\rightsquigarrow$ Solving LP

$$\begin{array}{rl} \text{maximize} & \mathbf{c}^T \mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \end{array}$$

with arbitrary $\mathbf{c}$ provides consistency information.

To cope with systems C containing *strict* inequations $\sum_{j=1}^{m} A_{i,j} x_j < b_j$, one

- introduces a slack variable ε,
- replaces $\sum_{j=1}^{m} A_{i,j} x_j < b_j$ by $\sum_{j=1}^{m} A_{i,j} x_j + \varepsilon \leq b_j$,
- solves the resultant LP L, maximizing the objective function ε

$\rightsquigarrow$ C is satisfiable iff L is satisfiable with optimum solution > 0.

Extracting reasons for $\mathcal{T}$-conflicts

Goal: In case that the original constraint system

$$C = \left(\begin{array}{ccc} & \bigwedge_{i=1}^{k} & \sum_{j=1}^{n} \mathbf{A}_{i,j}\mathbf{x}_j \leq \mathbf{b}_i \\ \wedge & \bigwedge_{i=k+1}^{n} & \sum_{j=1}^{n} \mathbf{A}_{i,j}\mathbf{x}_j < \mathbf{b}_i \end{array} \right)$$

is infeasible, we want a subset $I \subseteq \{1, \ldots, n\}$ such that

- the subsystem $C|_I$ of the constraint system containing only the conjuncts from I also is infeasible,
- yet the subsystem is *irreducible* in the sense that any proper subset J of I designates a feasible system $C|_J$.

Such an irreducible infeasible subsystem (IIS) is a prime implicant of all the possible reasons for failure of the constraint system C.

Extracting IIS

Provided constraint system C contains only non-strict inequations,

- extraction of IIS can be reduced to finding extremal solutions of a dual system of linear inequations, similar to Farkas' Lemma (Gleeson & Ryan 1990; Pfetsch, 2002)
- to keep the objective function bounded, one can use dual LP

$$\begin{array}{ll} \text{maximize} & \mathbf{w}^T\mathbf{y} \\ \text{subject to} & \begin{array}{rcl} \mathbf{A}^T\mathbf{y} & = & 0 \\ \mathbf{b}^T\mathbf{y} & = & 1 \\ \mathbf{y} & \geq & 0 \end{array} \\ \text{where} & \mathbf{w}_i = \begin{cases} -1 & \text{if } b_i \leq 0, \\ 0 & \text{if } b_i > 0 \end{cases} \end{array}$$

- choice of **w** guarantees boundedness of objective function

$\Longrightarrow$ optimal solution exists whenever the LP is feasible.

! For such a solution, $I = \{i \mid \mathbf{y}_i \neq 0\}$ is an IIS.

Extensions & Optimizations

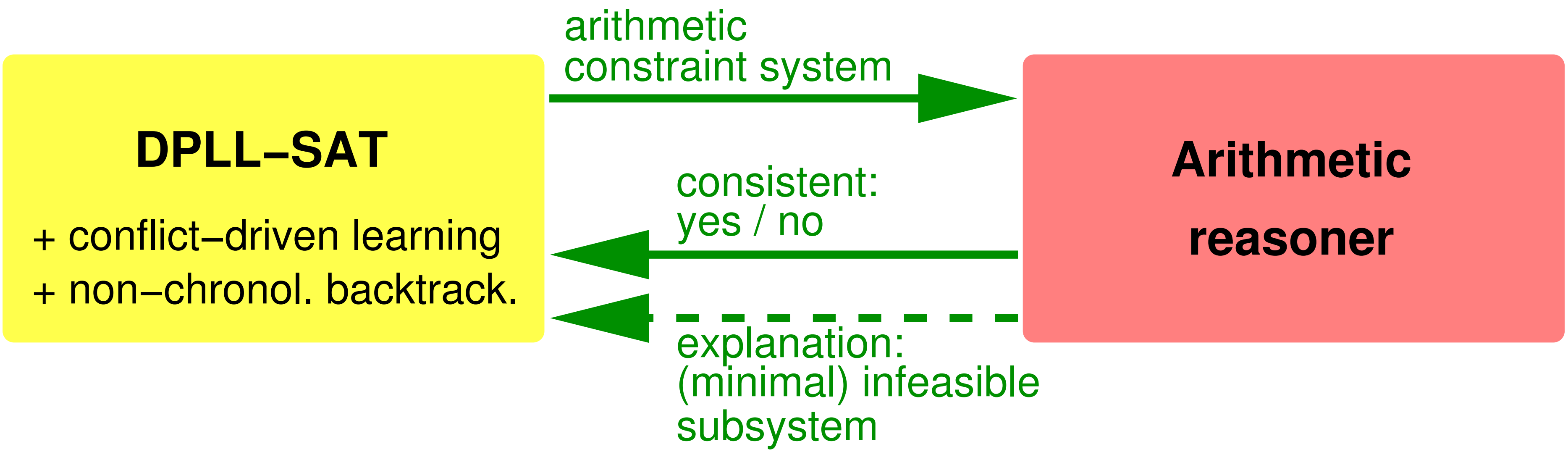
DPLL(T): If the T solver can itself do fwd. inference, it cannot only prune the search tree through conflict detection, but also through constraint propagation:

1. SAT solver assigns truth values to subset $\mathrm{C} \subset A$ of the set A of constraints occurring in the input formula,
2. T solver finds them to be consistent *and* to imply a truth value assignment to further T constraints $\mathrm{D} \subseteq A \setminus \mathrm{C}$,
3. these truth-value assignments are performed in the SAT solver store before resuming SAT solving.

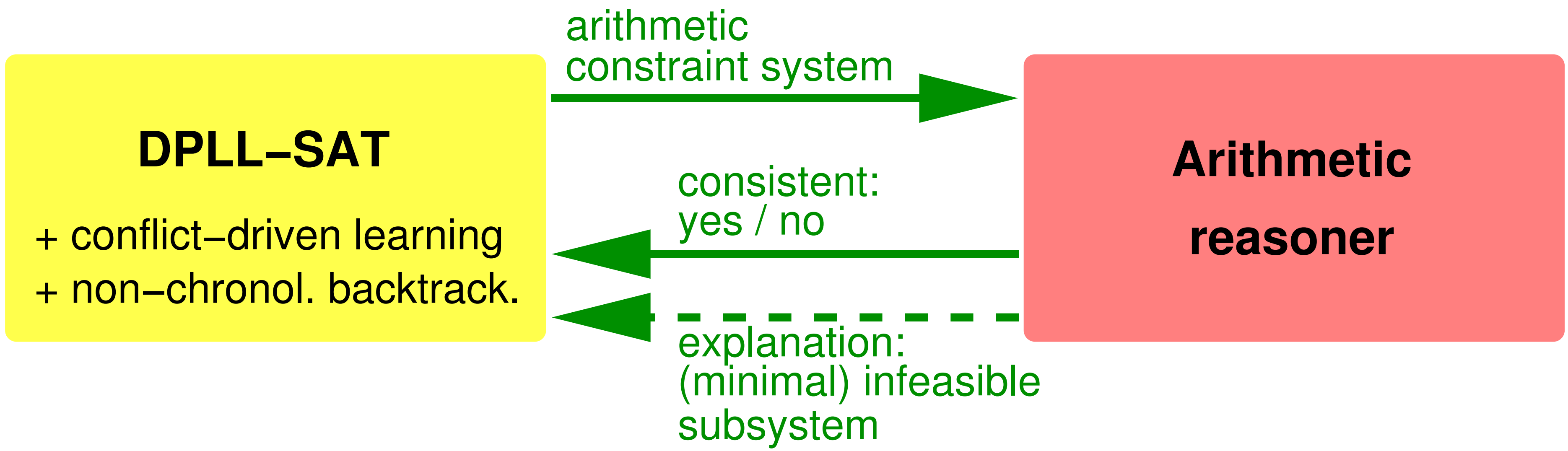
Satisfiability solving in undecidable arithmetic domains

iSAT algorithm (AVACS consortium 2006–)

Classical Lazy TP Layout



Classical Lazy TP Layout



Problems with extending it to richer arithmetic domains:

- undecidability: answer of arithmetic reasoner no longer two-valued; don't know cases arise
- explanations: how to generate (nearly) minimal infeasible subsystems of undecidable constraint systems?

Algorithmic basis:

Interval constraint propagation
(Hull consistency version)

Interval Constraint Solving (1)

- Complex constraints are rewritten to “triplets” (primitive constraints):

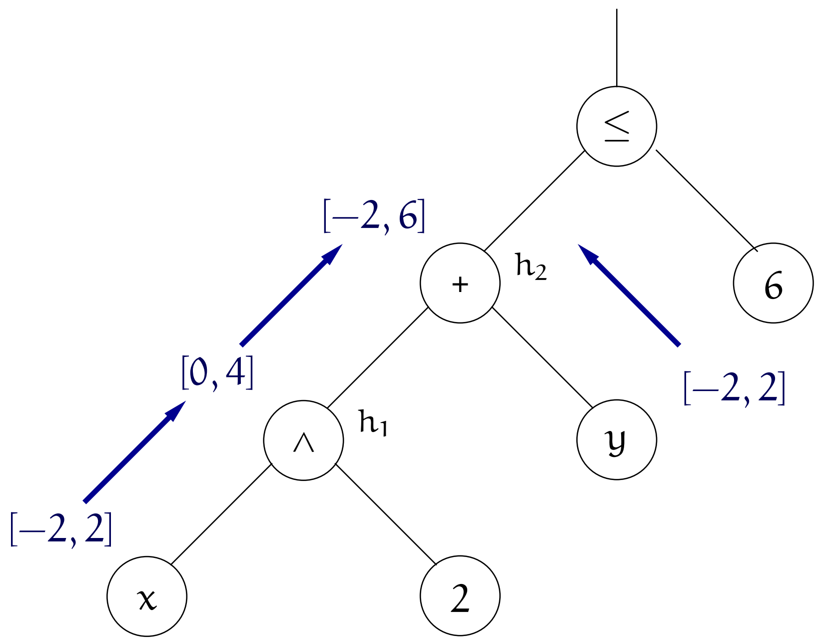
$$x^2 + y \leq 6 \quad \rightsquigarrow \quad \begin{array}{rll} c_1 : & & h_1 \triangleq x \wedge 2 \\ c_2 : & \wedge & h_2 \triangleq h_1 + y \\ & \wedge & h_2 \leq 6 \end{array}$$

Interval Constraint Solving (1)

- Complex constraints are rewritten to “triplets” (primitive constraints):

$$x^2 + y \leq 6 \quad \rightsquigarrow \quad \begin{array}{rll} c_1: & & h_1 \triangleq x \wedge 2 \\ c_2: & \wedge & h_2 \triangleq h_1 + y \\ & \wedge & h_2 \leq 6 \end{array}$$

- “Forward” interval propagation yields justification for constraint satisfaction:



$$x \in [-2, 2] \wedge y \in [-2, 2]$$

$$\Downarrow$$

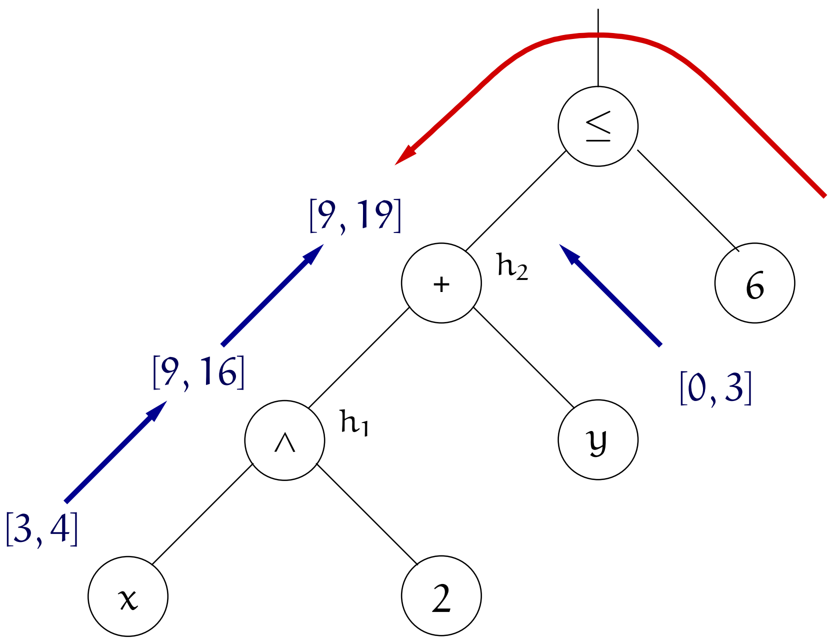
$h_2 \leq 6$ is satisfied in box

Interval Constraint Solving (1)

- Complex constraints are rewritten to “triplets” (primitive constraints):

$$x^2 + y \le 6 \quad \rightsquigarrow \quad \begin{array}{rcl} c_1: & & h_1 \triangleq x \wedge 2 \\ c_2: & \wedge & h_2 \triangleq h_1 + y \\ & \wedge & h_2 \le 6 \end{array}$$

- Interval propagation (fwd & bwd) yields witness for unsatisfiability:



$$x \in [3, 4]$$
$$\wedge \; y \in [0, 3]$$

$$\Downarrow$$

$h_2 \le 6$ is

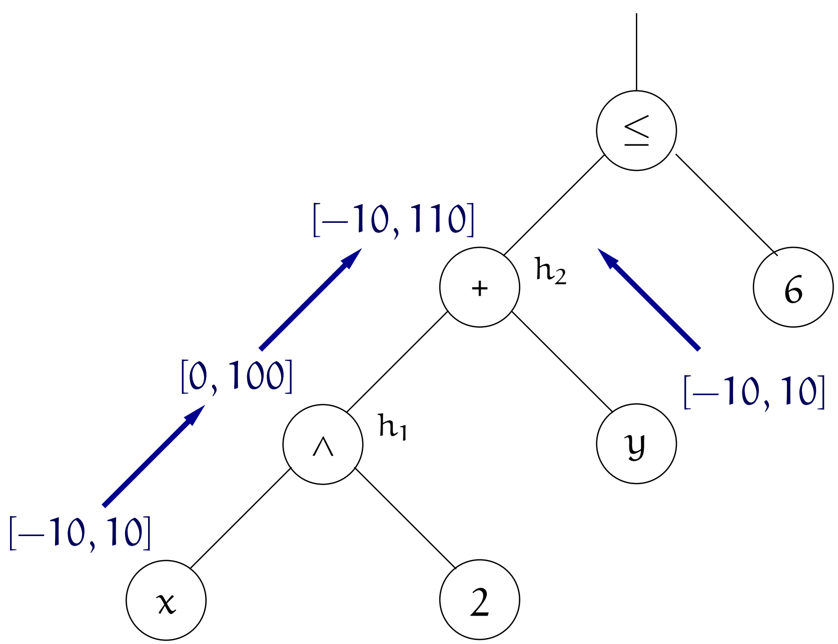
unsat. in box

Interval Constraint Solving (1)

- Complex constraints are rewritten to “triplets” (primitive constraints):

$$x^2 + y \leq 6 \quad \rightsquigarrow \quad \begin{array}{rll} c_1: & & h_1 \hat{=} x \wedge 2 \\ c_2: & \wedge & h_2 \hat{=} h_1 + y \\ & \wedge & h_2 \leq 6 \end{array}$$

- Interval prop. (fwd & bwd until fixpoint is reached) yields contraction of box:



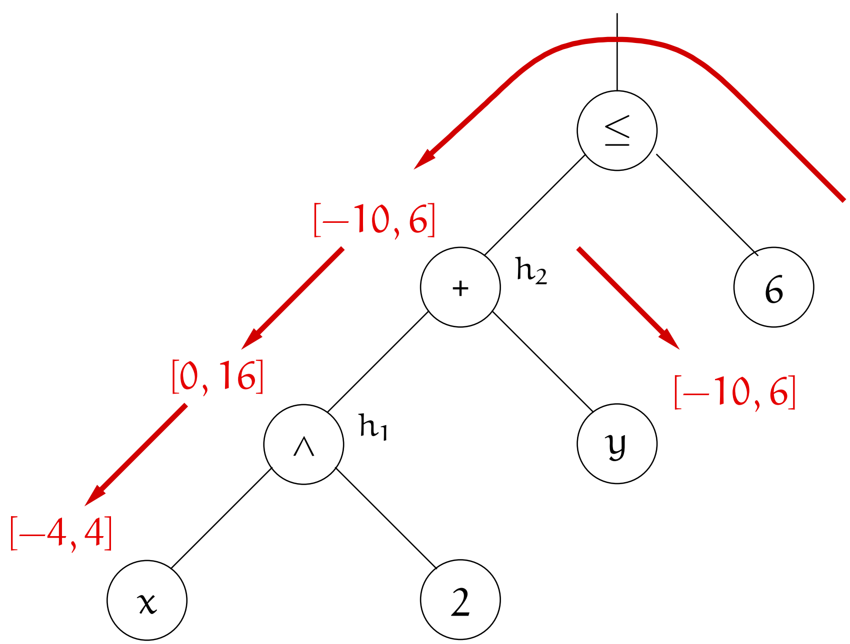
$$\begin{array}{r} x \in [-10, 10] \\ \wedge \; y \in [-10, 10] \end{array}$$

Interval Constraint Solving (1)

- Complex constraints are rewritten to “triplets” (primitive constraints):

$$x^2 + y \leq 6 \quad \rightsquigarrow \quad \begin{array}{lll} c_1 : & & h_1 \triangleq x \wedge 2 \\ c_2 : & \wedge & h_2 \triangleq h_1 + y \\ & \wedge & h_2 \leq 6 \end{array}$$

- Interval prop. (fwd & bwd until fixpoint is reached) yields contraction of box:



$$\begin{array}{r} x \in [-10, 10] \\ \wedge\ y \in [-10, 10] \end{array}$$

$$\Downarrow$$

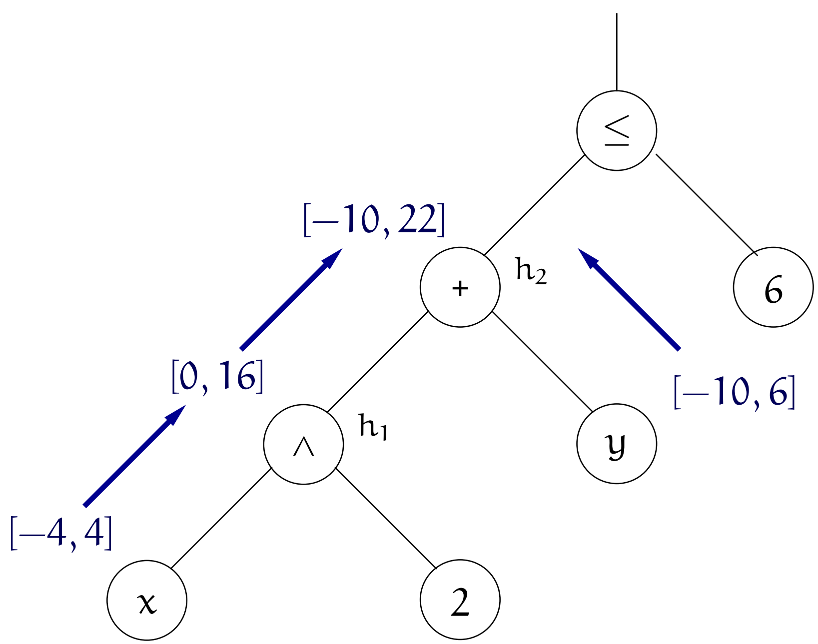
$$\begin{array}{r} x \in [-4, 4] \\ \wedge\ y \in [-10, 6] \end{array}$$

Interval Constraint Solving (1)

- Complex constraints are rewritten to "triplets" (primitive constraints):

$$x^2 + y \le 6 \quad \rightsquigarrow \quad \begin{array}{rll} c_1: & & h_1 \triangleq x \wedge 2 \\ c_2: & \wedge & h_2 \triangleq h_1 + y \\ & \wedge & h_2 \le 6 \end{array}$$

- Interval prop. (fwd & bwd until fixpoint is reached) yields contraction of box:



Constraint is not satisfied by the contracted box!

$$\begin{array}{rl} & x \in [-4, 4] \\ \wedge & y \in [-10, 6] \end{array}$$

Interval contraction

Backward propagation yields rectangular overapproximation of non-rectangular pre-images.

Thus, interval contraction provides a highly incomplete deduction system:

$$\begin{array}{rl} & x \in [0,\infty) \\ \wedge & h \hat{=} x \cdot y \\ \wedge & h > 5 \end{array} \Longrightarrow \begin{array}{rl} & x \in (0,\infty) \\ \wedge & y \in (0,\infty) \end{array} \Longrightarrow h \in (0,\infty) \not\Longrightarrow h > 5$$

Interval contraction

Backward propagation yields rectangular overapproximation of non-rectangular pre-images.

Thus, interval contraction provides a highly incomplete deduction system:

$$\begin{array}{ll} & x \in [0,\infty) \\ \wedge & h \hat{=} x \cdot y \\ \wedge & h > 5 \end{array} \implies \begin{array}{ll} & x \in (0,\infty) \\ \wedge & y \in (0,\infty) \end{array} \implies h \in (0,\infty) \not\Longrightarrow h > 5$$

$\leadsto$ enhance through branch-and-prune approach.

Schema of Interval-CP based CS Alg.

Given: Constraint set $C = \{c_1, \ldots, c_n\}$,
initial box (= cartesian product of intervals) B in $\mathbb{R}^{|\mathsf{free}(C)|}$

Goal: Find box $B' \subseteq B$ containing satisfying valuations throughout *or* show non-existence of such B'.

Alg.:
1. $L := \{B\}$
2. If $L \neq \emptyset$ then take some box $b :\in L$, otherwise report "unsatisfiable" and stop.
3. Use contraction to determine a sub-box $b' \subseteq b$.
4. If $b' = \emptyset$ then set $L := L \setminus \{b\}$, goto 2.
5. Use forward interval propagation to determine whether all constraints are satisfied throughout b'; if so then report b' as satisfying and stop.
6. If $b' \subset b$ then set $L := L \setminus \{b\} \cup \{b'\}$, goto 2.
7. Split b into subboxes b_1 and b_2, set $L := L \setminus \{b\} \cup \{b_1, b_2\}$, goto 2.

Schema of Interval-CP based CS Alg. / DPLL

Given: Constraint / clause set $C = \{c_1, \ldots, c_n\}$,
initial box (= cartesian product of intervals) B in $\mathbb{R}^{|\text{free}(C)|}$ / $\mathbb{B}^{|\text{free}(C)|}$

Goal: Find box $B' \subseteq B$ containing satisfying valuations throughout
or show non-existence of such B'.

Alg.:

1. $L := \{B\}$
2. If $L \neq \emptyset$ then take some box $b :\in L$,
   otherwise report "unsatisfiable" and stop.
3. Use contraction to determine a sub-box $b' \subseteq b$. (Unit Prop.)
4. If $b' = \emptyset$ then set $L := L \setminus \{b\}$, goto 2.
5. Use forward interval propagation to determine whether all constraints are satisfied throughout b'; if so then report b' as satisfying and stop.
6. If $b' \subset b$ then set $L := L \setminus \{b\} \cup \{b'\}$, goto 2.
7. Split b into subboxes b_1 and b_2, set $L := L \setminus \{b\} \cup \{b_1, b_2\}$, goto 2.

Schema of Interval-CP based CS Alg. / DPLL

Given: Constraint / clause set $C = \{c_1, \dots, c_n\}$,
initial box (= cartesian product of intervals) B in $\mathbb{R}^{|\mathsf{free}(C)|}$ / $\mathbb{B}^{|\mathsf{free}(C)|}$

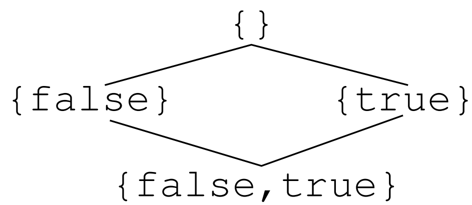
Goal: Find box $B' \subseteq B$ containing satisfying valuations throughout *or* show non-existence of such B'.

Alg.:

1. $L := \{B\}$
2. If $L \neq \emptyset$ then take some box $b :\in L$, (LIFO) otherwise report "unsatisfiable" and stop.
3. Use contraction to determine a sub-box $b' \subseteq b$. (Unit Prop.)
4. If $b' = \emptyset$ then set $L := L \setminus \{b\}$, goto 2.
5. Use forward interval propagation to determine whether all constraints are satisfied throughout b'; if so then report b' as satisfying and stop.
6. If $b' \subset b$ then set $L := L \setminus \{b\} \cup \{b'\}$, goto 2.
7. Split b into subboxes b_1 and b_2, set $L := L \setminus \{b\} \cup \{b_1, b_2\}$, goto 2.

Observation

DPLL-SAT and interval-CP based CS are inherently similar:

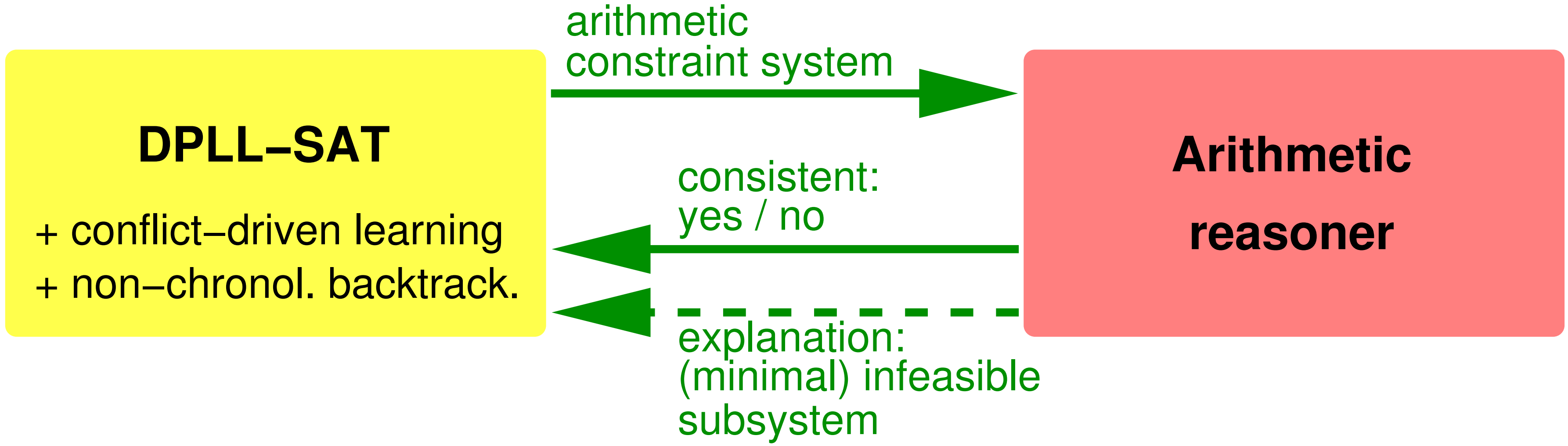
	DPLL-SAT	**Interval-based CS**
Propagation:	contraction in lattice  of Boolean intervals	contraction in lattice of intervals over $\mathbb{R}$
Split:	split of Boolean interval $[\texttt{false}, \texttt{true}]$	split of interval over $\mathbb{R}$

This suggests a tighter integration than lazy TP:
common algorithms should be shared, others should be lifted to both domains.

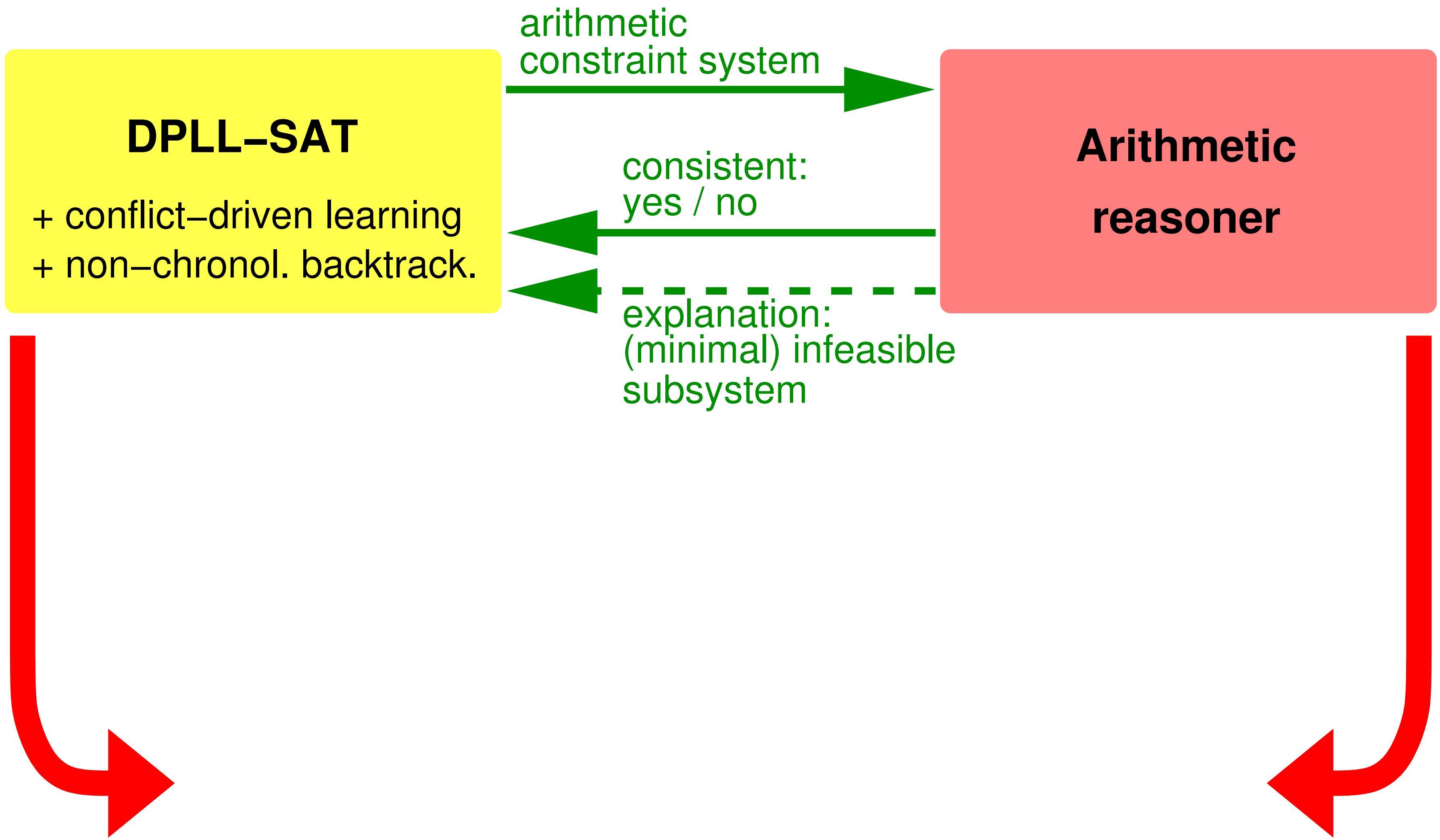
iSAT algorithm

Tight integration of DPLL and ICP

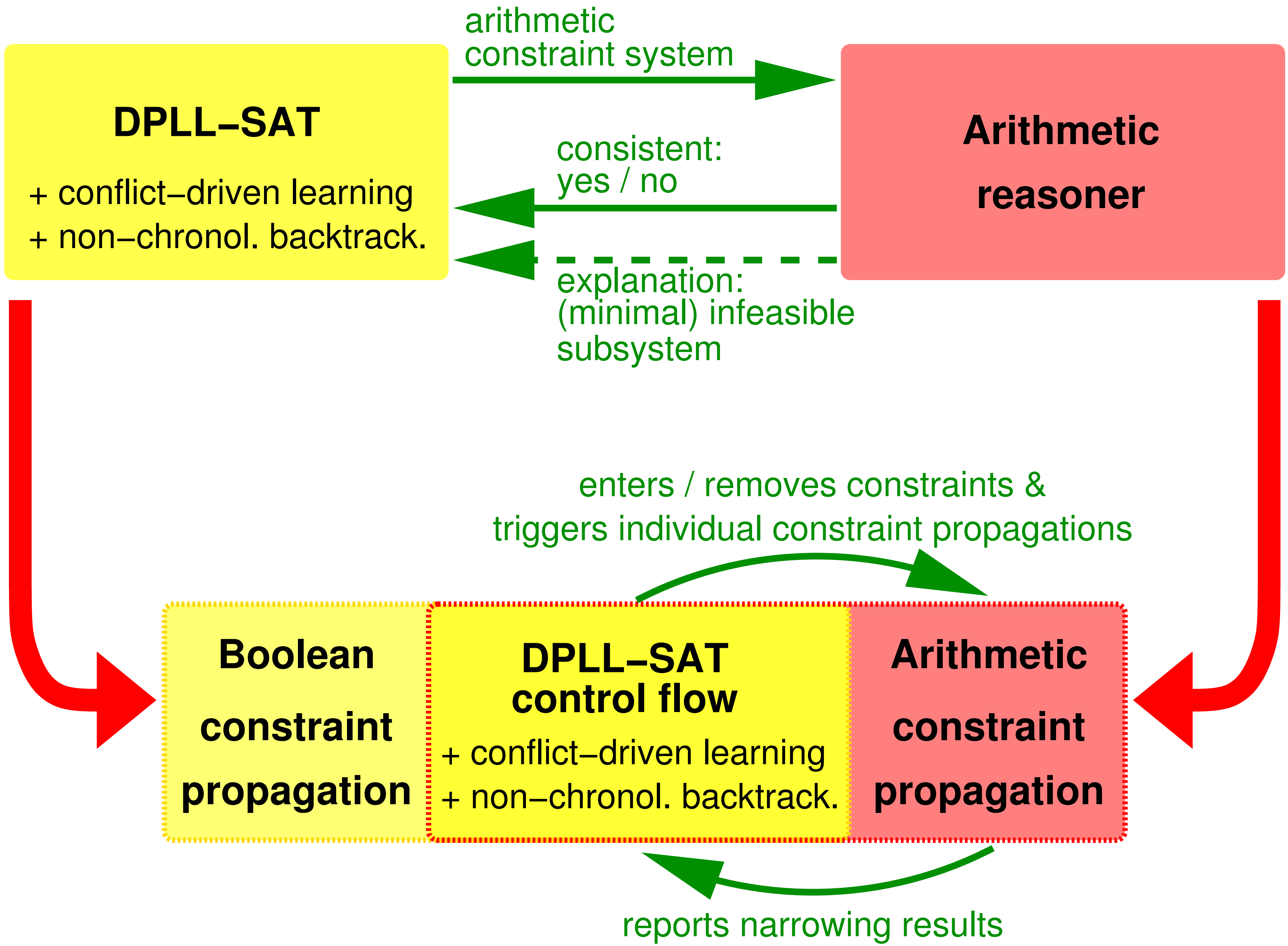
Lazy TP: Tightening the Interaction



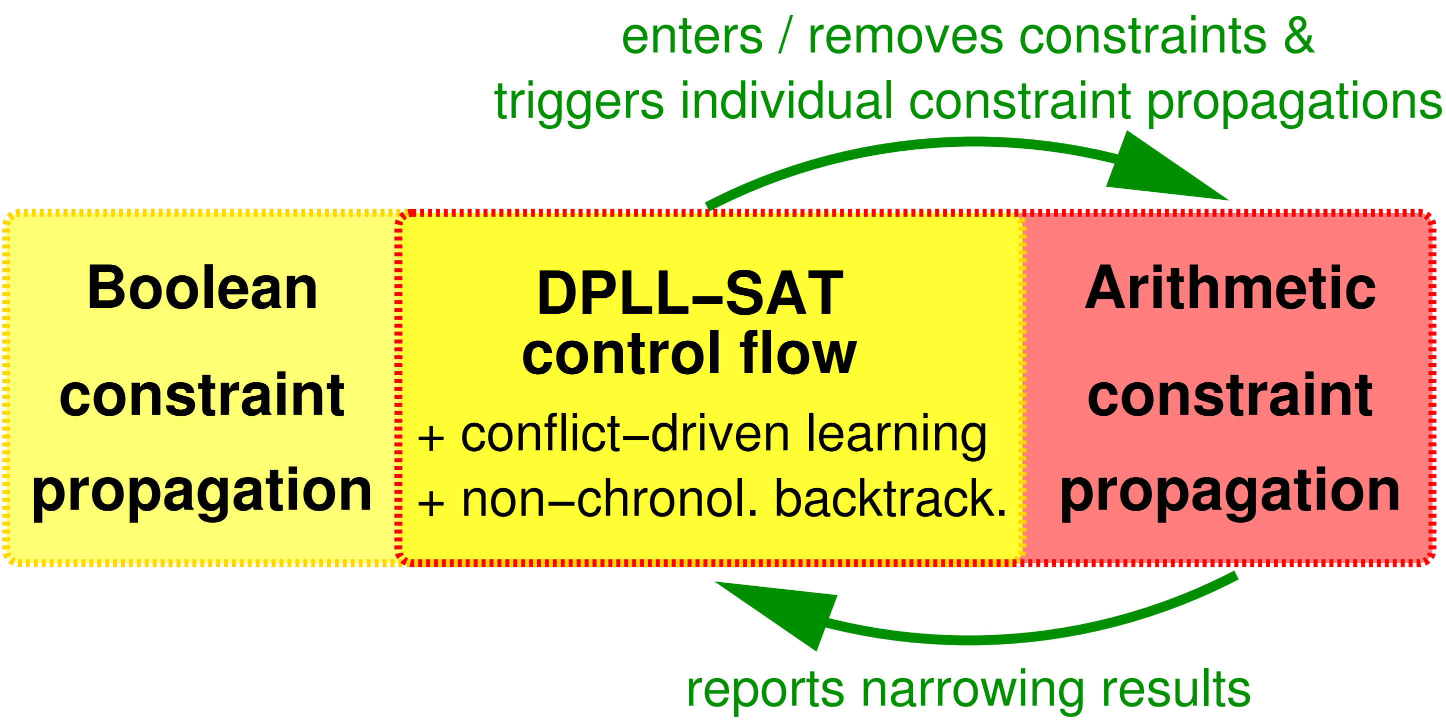
Lazy TP: Tightening the Interaction



Lazy TP: Tightening the Interaction



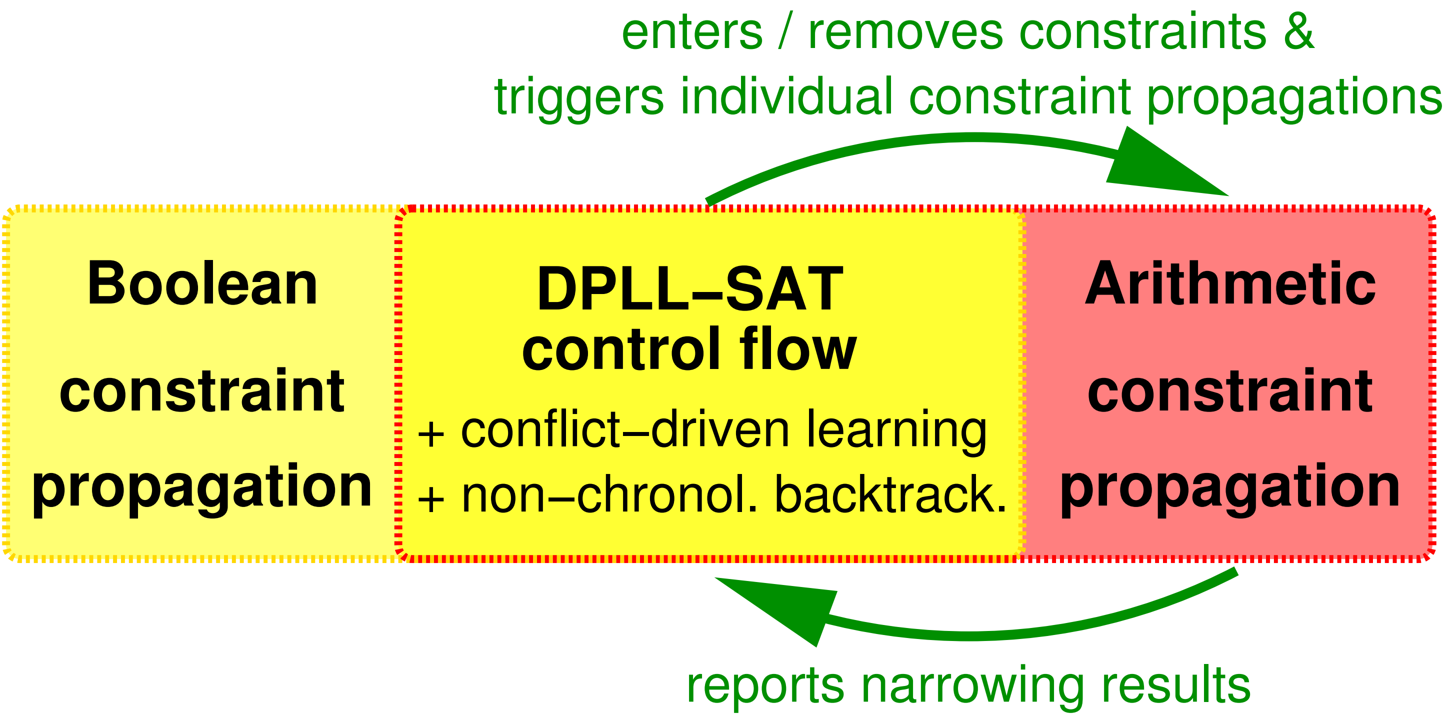
Properties of Modified Layout



- SAT engine has introspection into CP
- thus can keep track of inferences *and their reasons*

can use recent SAT mechanisms for generalizing reasons of conflicts and learning them, thus pruning the search tree

Properties of Modified Layout



- SAT engine has introspection into CP
- thus can keep track of inferences *and their reasons*
- 🙂 can use recent SAT mechanisms for generalizing reasons of conflicts and learning them, thus pruning the search tree
- 🙁 preoccupation towards depth-first search (inherited from DPLL)

The CP Mechanisms

Interpretation of variables: Each variable x is interpreted by *two* intervals $x\uparrow \supseteq x\downarrow$:

Interval	denotes	CP mechanisms
$x\uparrow$	justifying interval	Fwd. propagation among $.\uparrow$ intervals (wrt. some order)
$x\downarrow$	implied interval	Fwd. and bwd. narrowing among $.\downarrow$ intervals

Constraint propagation:

- $h \leq \mathrm{const}$: Narrow $h\downarrow$ to $h\downarrow' := h\downarrow \cap [\mathrm{const}, \infty)$.
- $x = y \oplus z$: Apply the contractors of all reshufflings.

Conflicts: Materialize by contracting a $.\downarrow$ interval to $\emptyset$.

Constraint satisfaction: Shows by $h\uparrow$ satisfying the constraint.

DPLL on a search lattice $\mathbb{D}$

1. Start from the most general assignment $\sigma_\perp \equiv \perp$.
2. (Propagation) If there is a not yet satisfied clause containing exactly one elementary formula ϕ with value $\neq \mathtt{false}$ then enqueue $\mathrm{contract}(\phi)$. Repeat 2 if possible.
3. (Perform updates) If implication queue non-empty then dequeue $\mathrm{contract}(\phi)$ and perform it. If this assigns $\top$ to some entailed variable then backtrack (if applicable, otherwise return "unsatisfiable"). If all clauses become true, report "satisfiable". Enqueue $\mathrm{contract}(\psi)$ for all affected atoms ψ and repeat 3 unless queue empty. Thereafter proceed with 2, if applicable.
4. (Split) Select an arbitrary variable x with non-maximal (in $\mathbb{D} \setminus \{\top\}$) value occurring in an unsatisfied elementary formula in an unsatisfied clause.
   Take $x', x'' \in \mathbb{D} \setminus \{\top\}$ s.t. $x := x' \sqcap x''$. Enqueue $x := x'$. Store alternative $x := x''$ as backtrack alternative. Goto 3.

Optimizations inherited from DPLL:

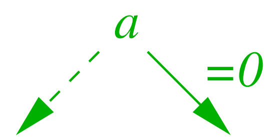
- **conflict-driven learning**
- **non-chronological backtracking**
- **watched literal scheme**
- **restarts**

Conflict-Driven Learning in DPLL: Example

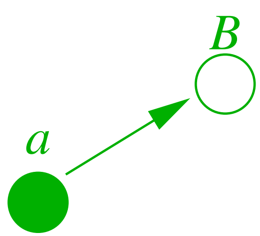
$A + B$
$A + c + D$
$A + d + E$
$c + F + G$
$d + F + g$
$f + G$
$f + g$
$X + y$
$e + Y + Z$

Conflict-Driven Learning in DPLL: Example

$A + B$
$A + c + D$
$A + d + E$
$c + F + G$
$d + F + g$
$f + G$
$f + g$
$X + y$
$e + Y + Z$

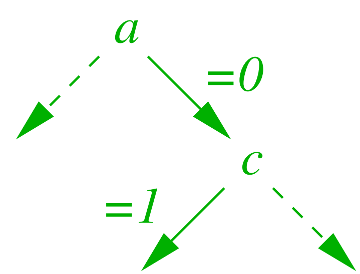


$b = 1$



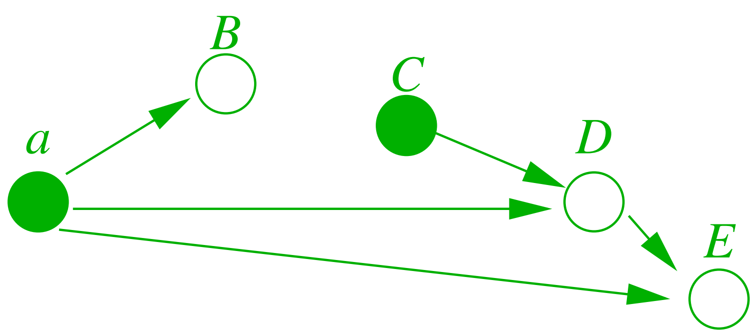
Conflict-Driven Learning in DPLL: Example

$A + B$
$A + c + D$
$A + d + E$
$c + F + G$
$d + F + g$
$f + G$
$f + g$
$X + y$
$e + Y + Z$



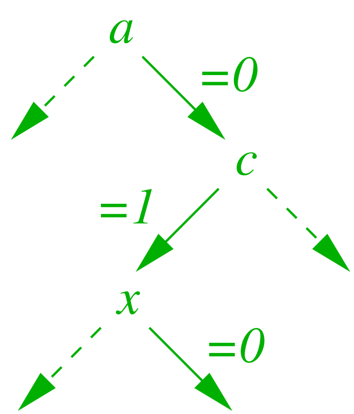
$b = 1$

$d = 1, e = 1$



Conflict-Driven Learning in DPLL: Example

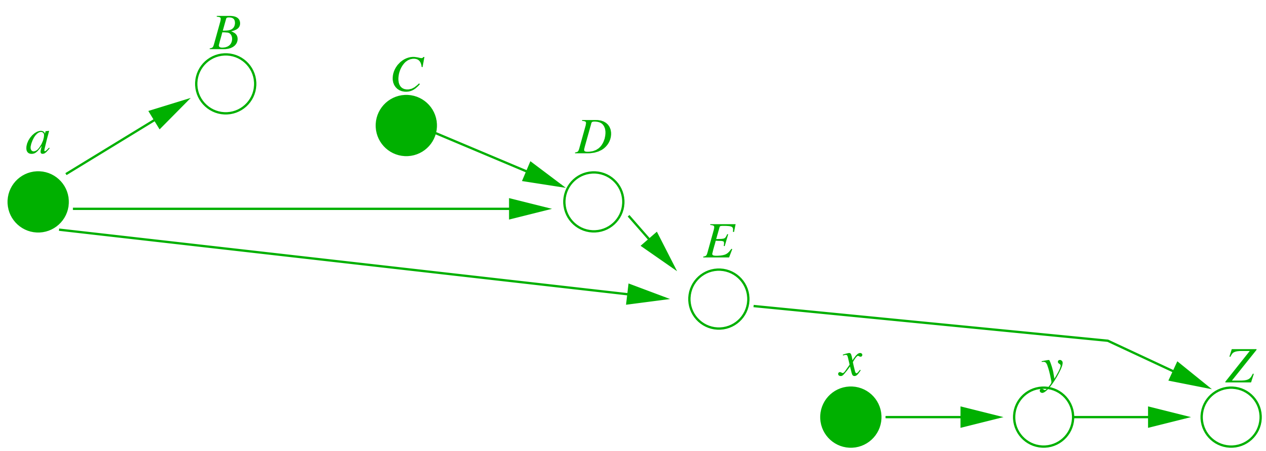
$A + B$
$A + c + D$
$A + d + E$
$c + F + G$
$d + F + g$
$f + G$
$f + g$
$X + y$
$e + Y + Z$



$b = 1$

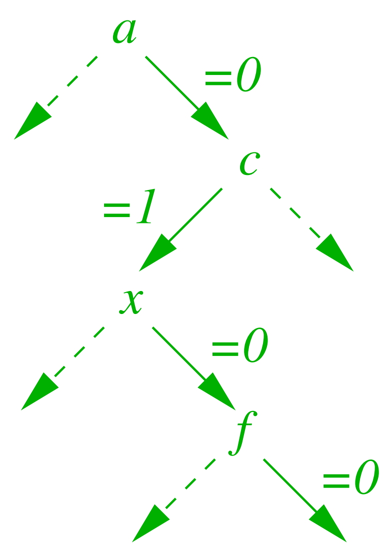
$d = 1, e = 1$

$y = 0, z = 1$



Conflict-Driven Learning in DPLL: Example

$A + B$
$A + c + D$
$A + d + E$
$c + F + G$
$d + F + g$
$f + G$
$f + g$
$X + y$
$e + Y + Z$

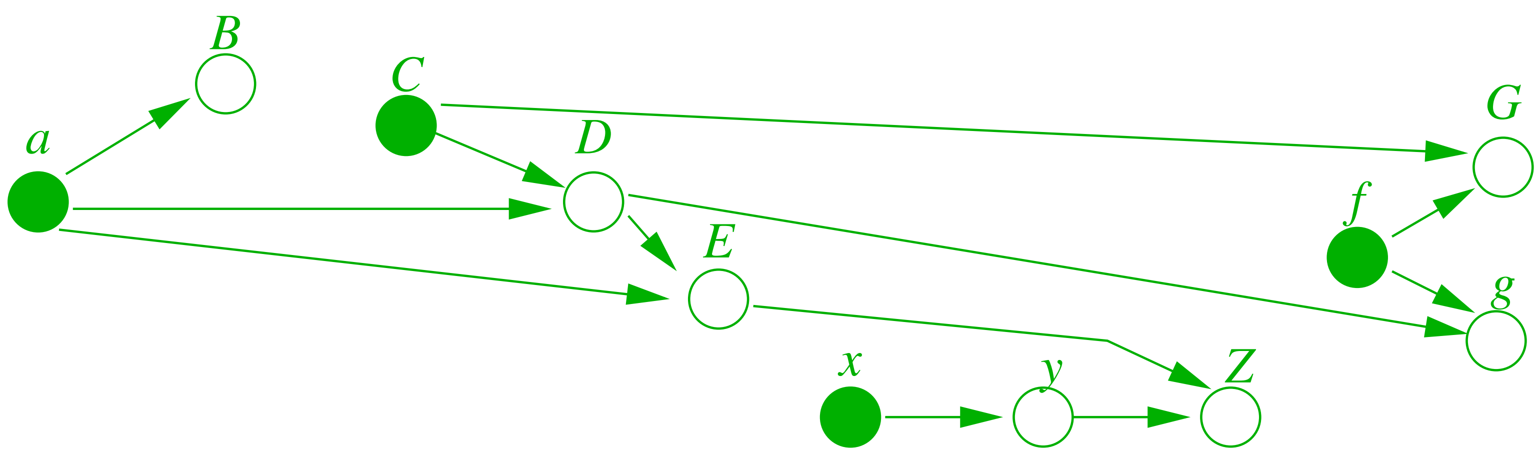


$b = 1$

$d = 1, e = 1$

$y = 0, z = 1$

$g = 1, g = 0$



Conflict-Driven Learning in DPLL: Example

$A + B$

$A + c + D$

$A + d + E$

$c + F + G$

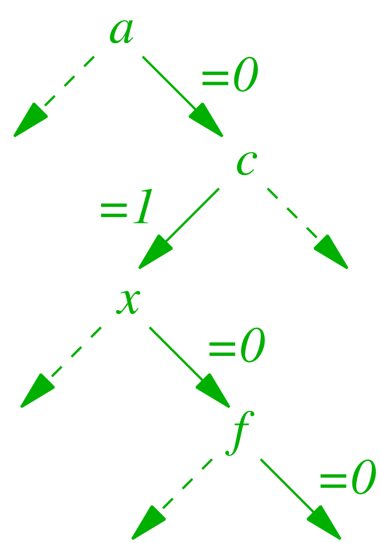
$d + F + g$

$f + G$

$f + g$

$X + y$

$e + Y + Z$

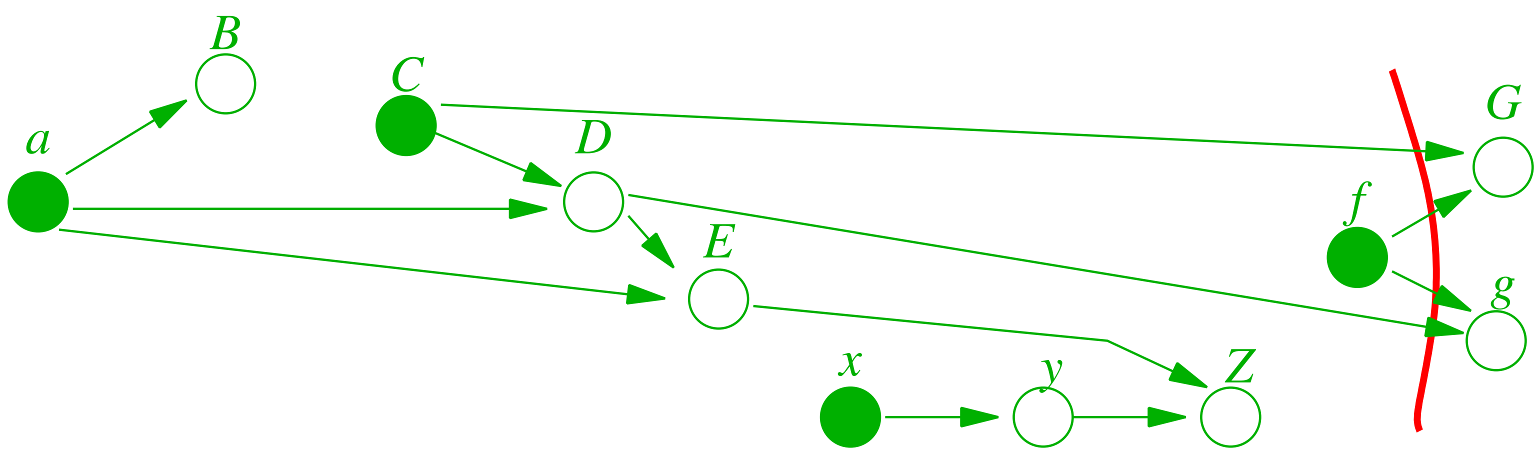


$b = 1$

$d = 1, e = 1$

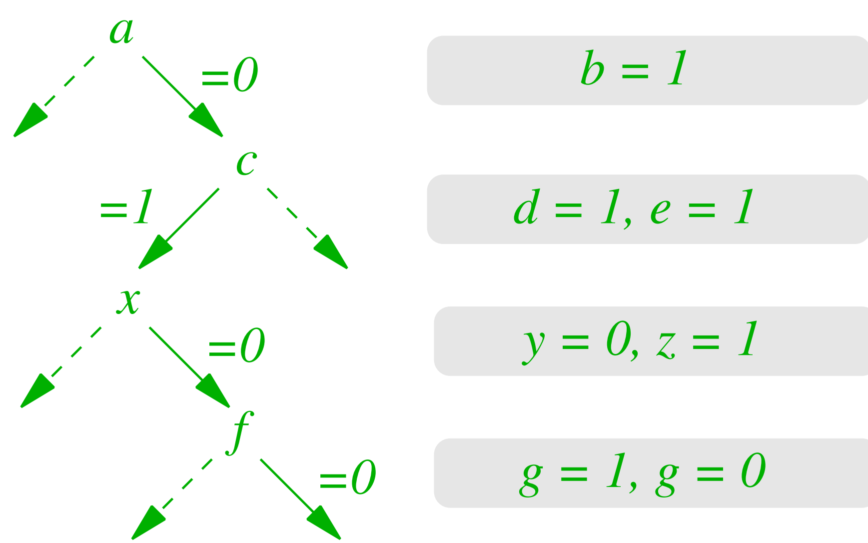
$y = 0, z = 1$

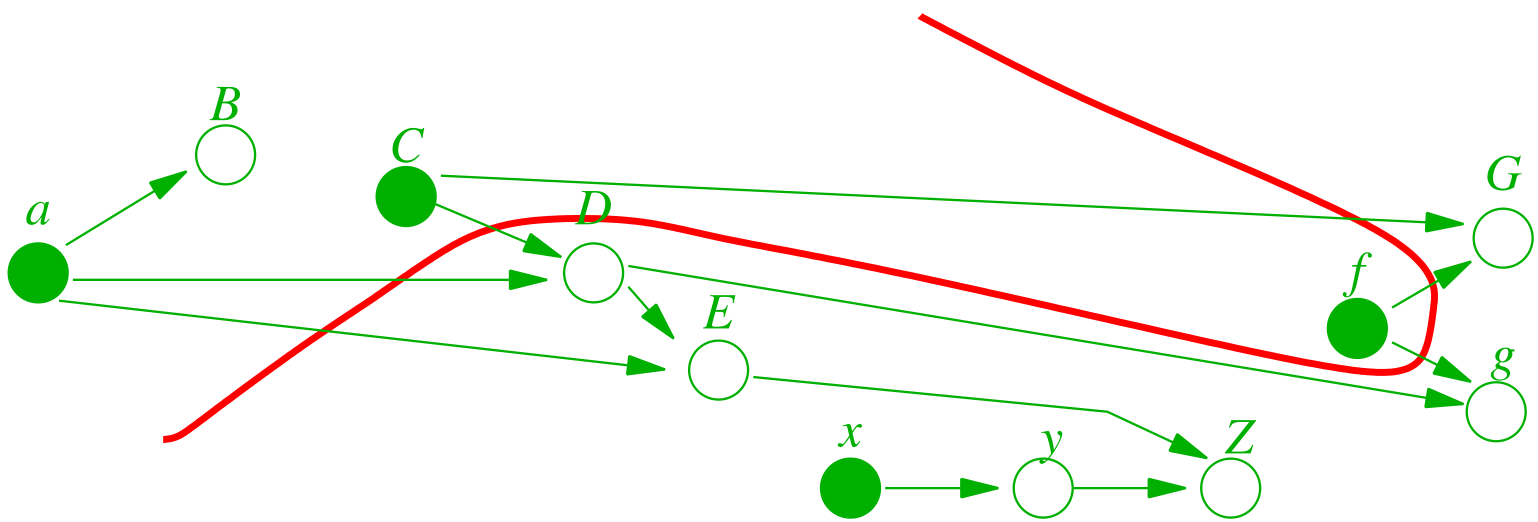
$g = 1, g = 0$



Conflict-Driven Learning in DPLL: Example

$A + B$
$A + c + D$
$A + d + E$
$c + F + G$
$d + F + g$
$f + G$
$f + g$
$X + y$
$e + Y + Z$





Conflict-Driven Learning in DPLL: Example

$A + B$

$A + c + D$

$A + d + E$

$c + F + G$

$d + F + g$

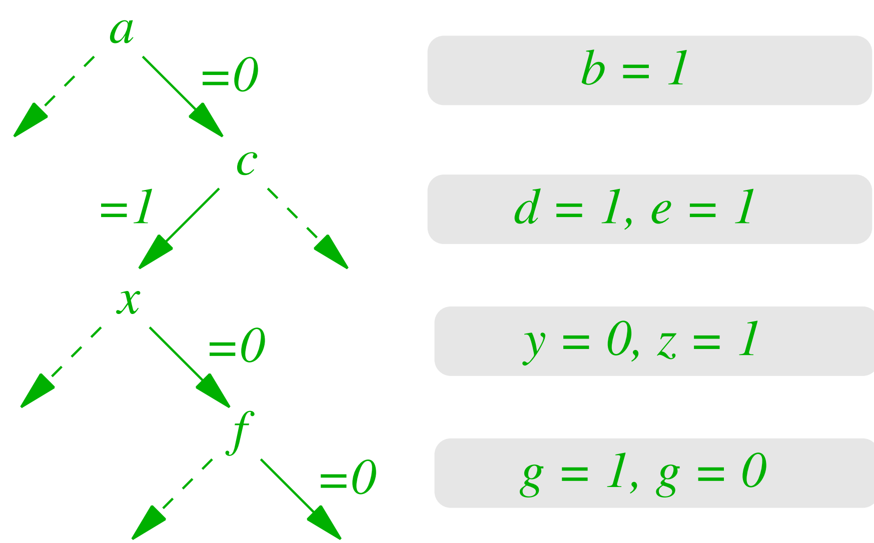
$f + G$

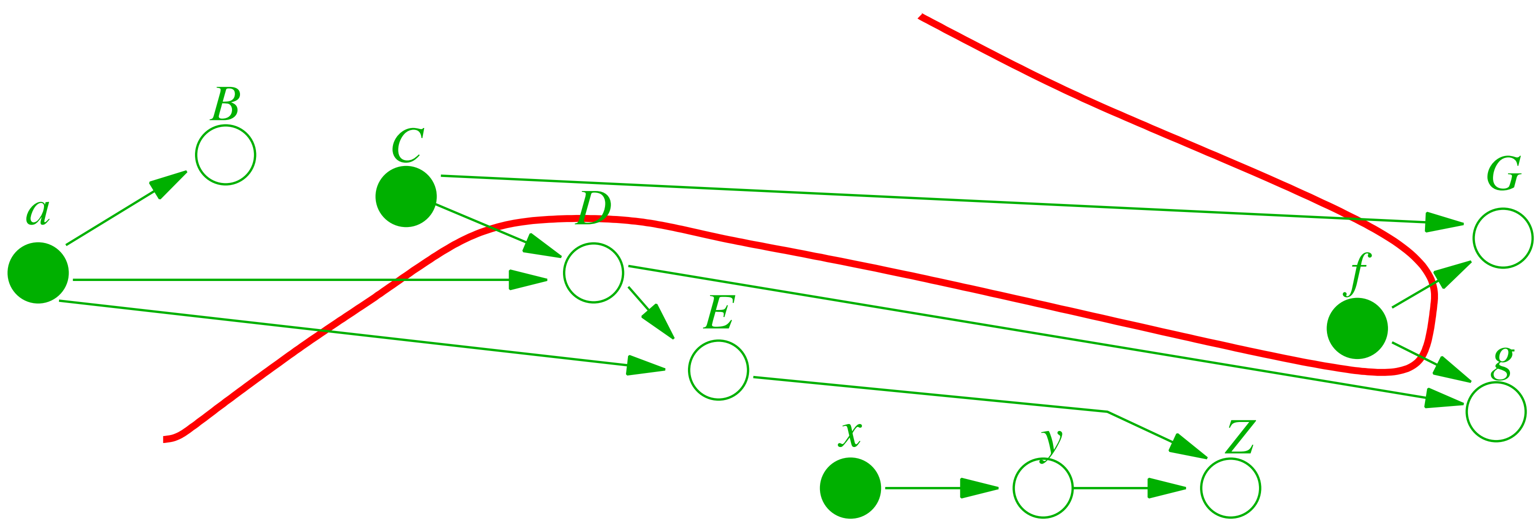
$f + g$

$X + y$

$e + Y + Z$

$A + c + F$





Conflict-Driven Learning in DPLL: Example

$A + B$

$A + c + D$

$A + d + E$

$c + F + G$

$d + F + g$

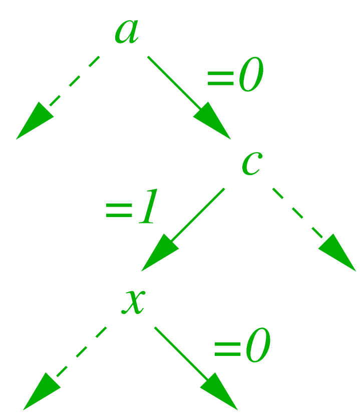
$f + G$

$f + g$

$X + y$

$e + Y + Z$

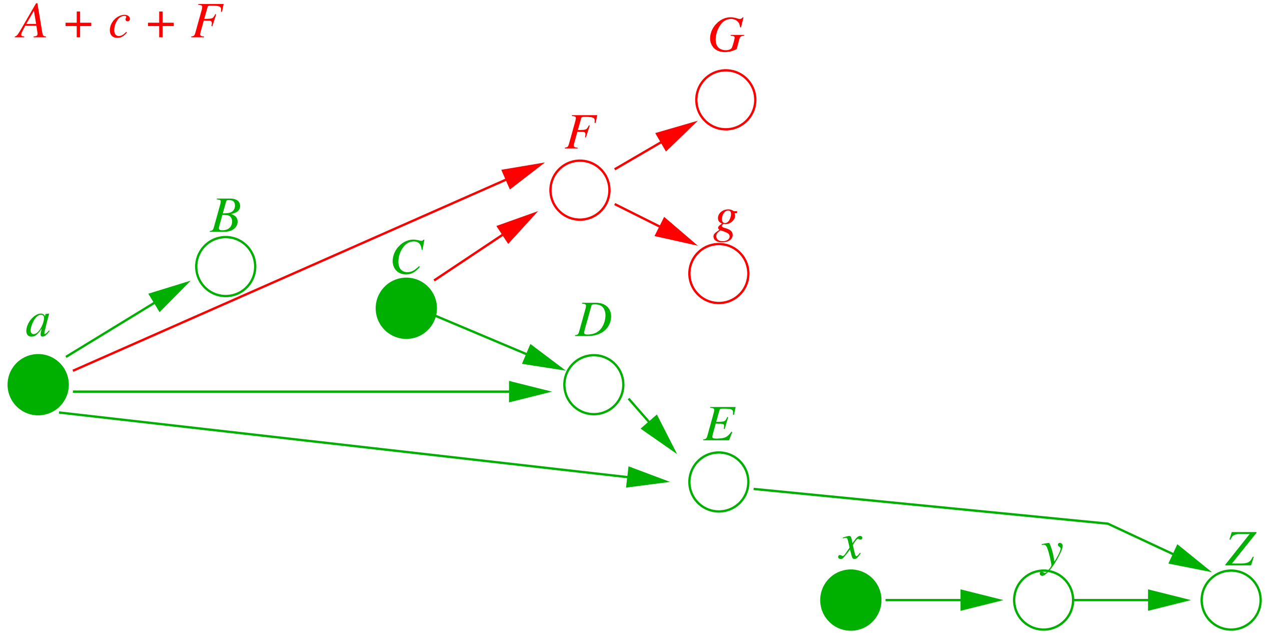
$A + c + F$



$b = 1$

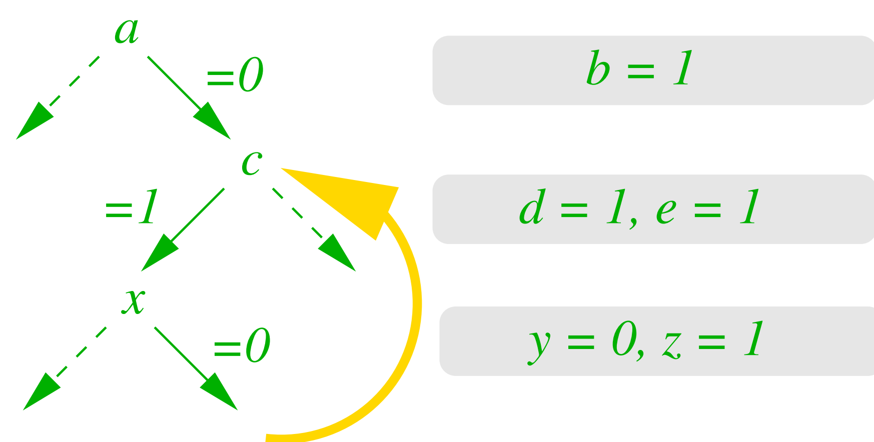
$d = 1, e = 1$

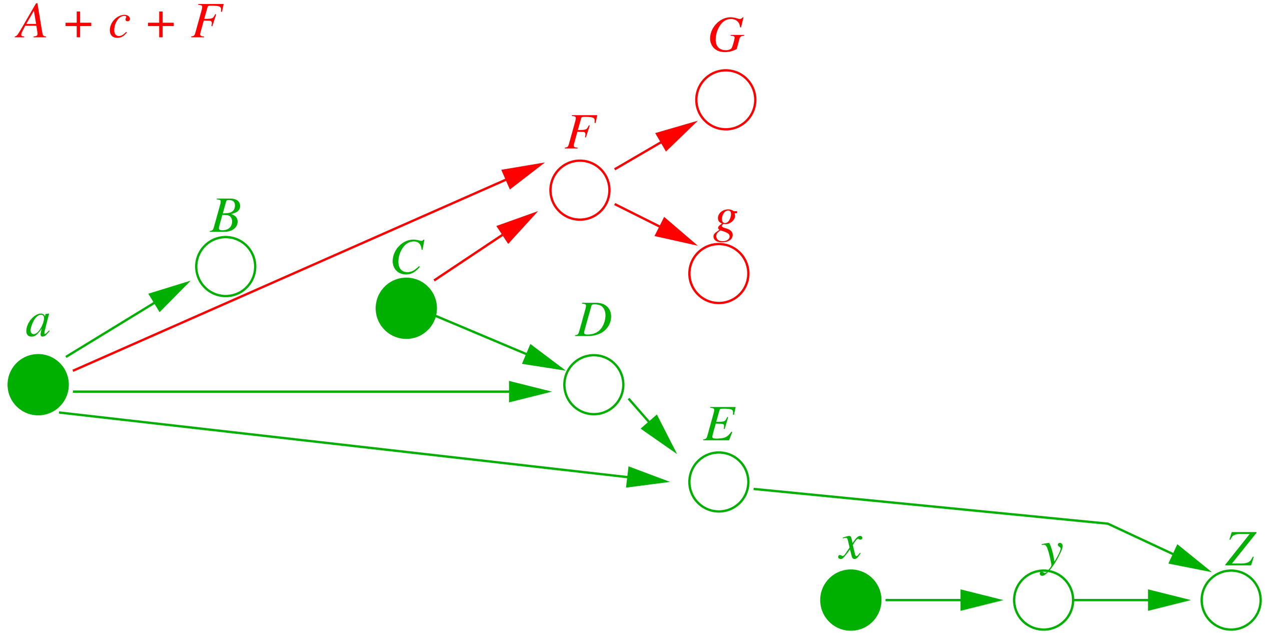
$y = 0, z = 1$



Conflict-Driven Learning in DPLL: Example

$A + B$
$A + c + D$
$A + d + E$
$c + F + G$
$d + F + g$
$f + G$
$f + g$
$X + y$
$e + Y + Z$
$A + c + F$





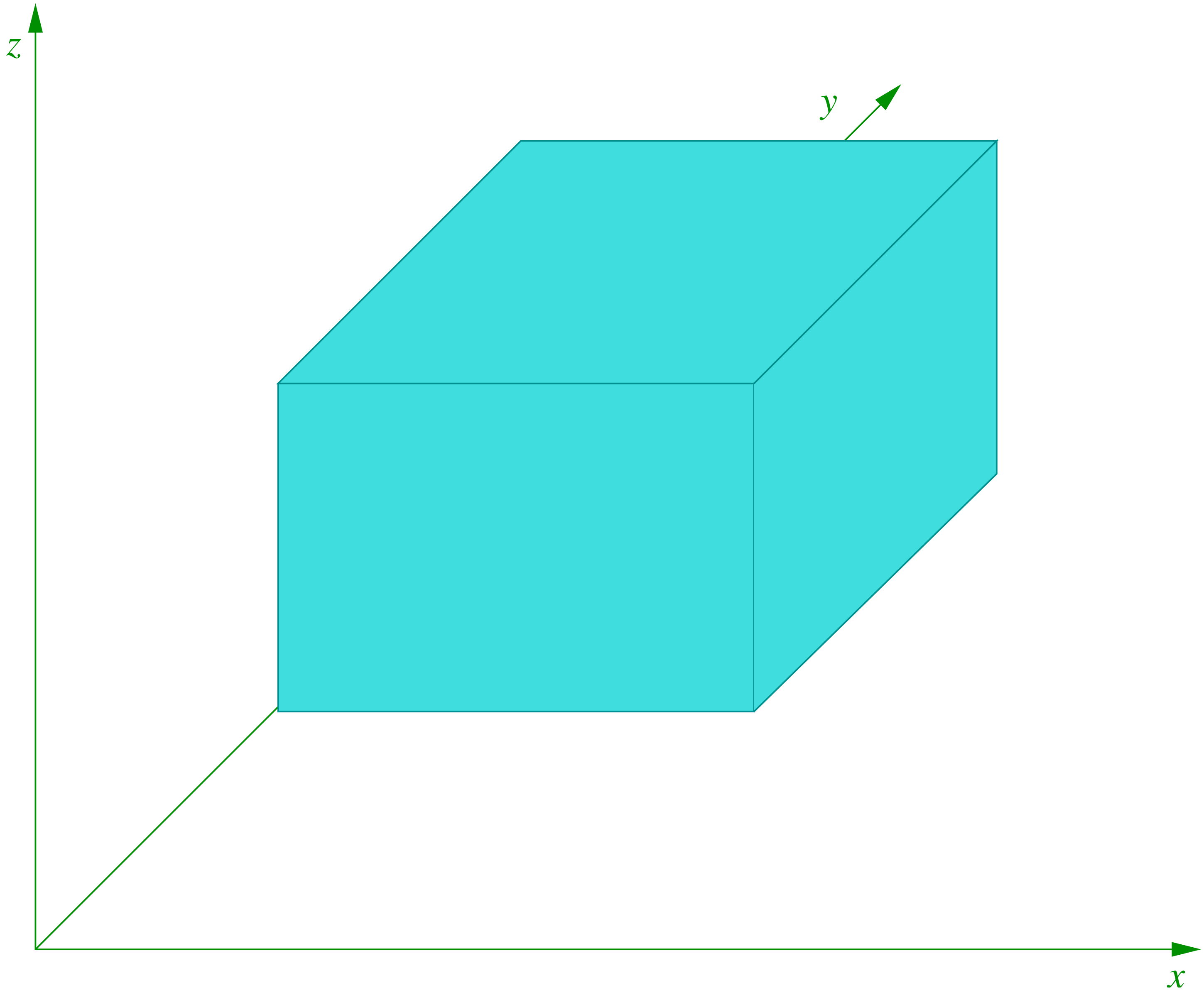
Conflict-driven learning in multi-valued case

Works like a charme w/o fundamental modifications:

- Decision variables coincide to interval splits; the assigned values to asserted bounds $x \geq c$, $x > c$, $x < c$, $x \leq c$;
- Implications correspond to contractions;
- Reasons to sets of asserted atoms giving rise to a contraction.

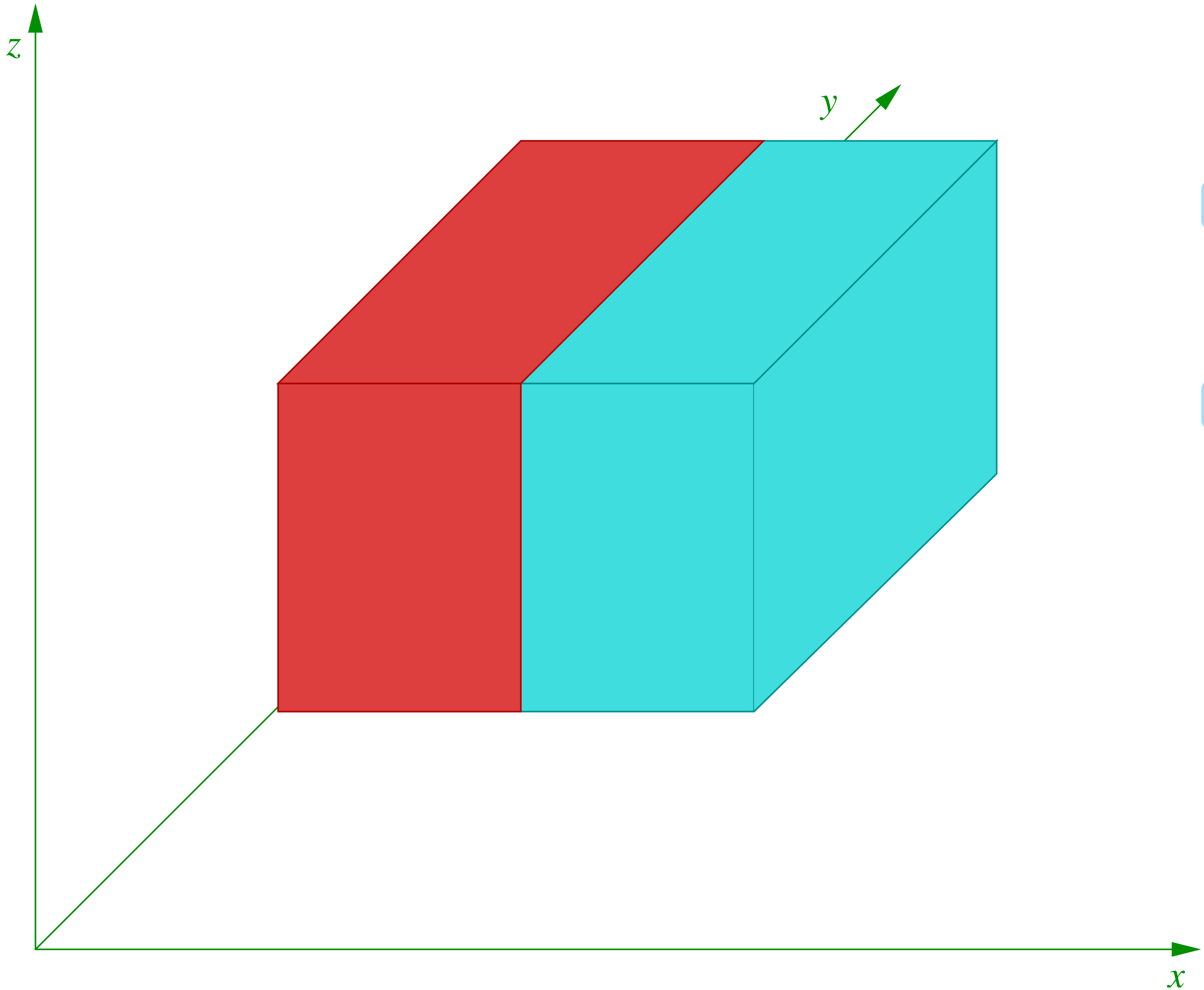
Through embedding into SAT, we get conflict-driven learning and non-chronological backtracking for free!

Learning: Principle



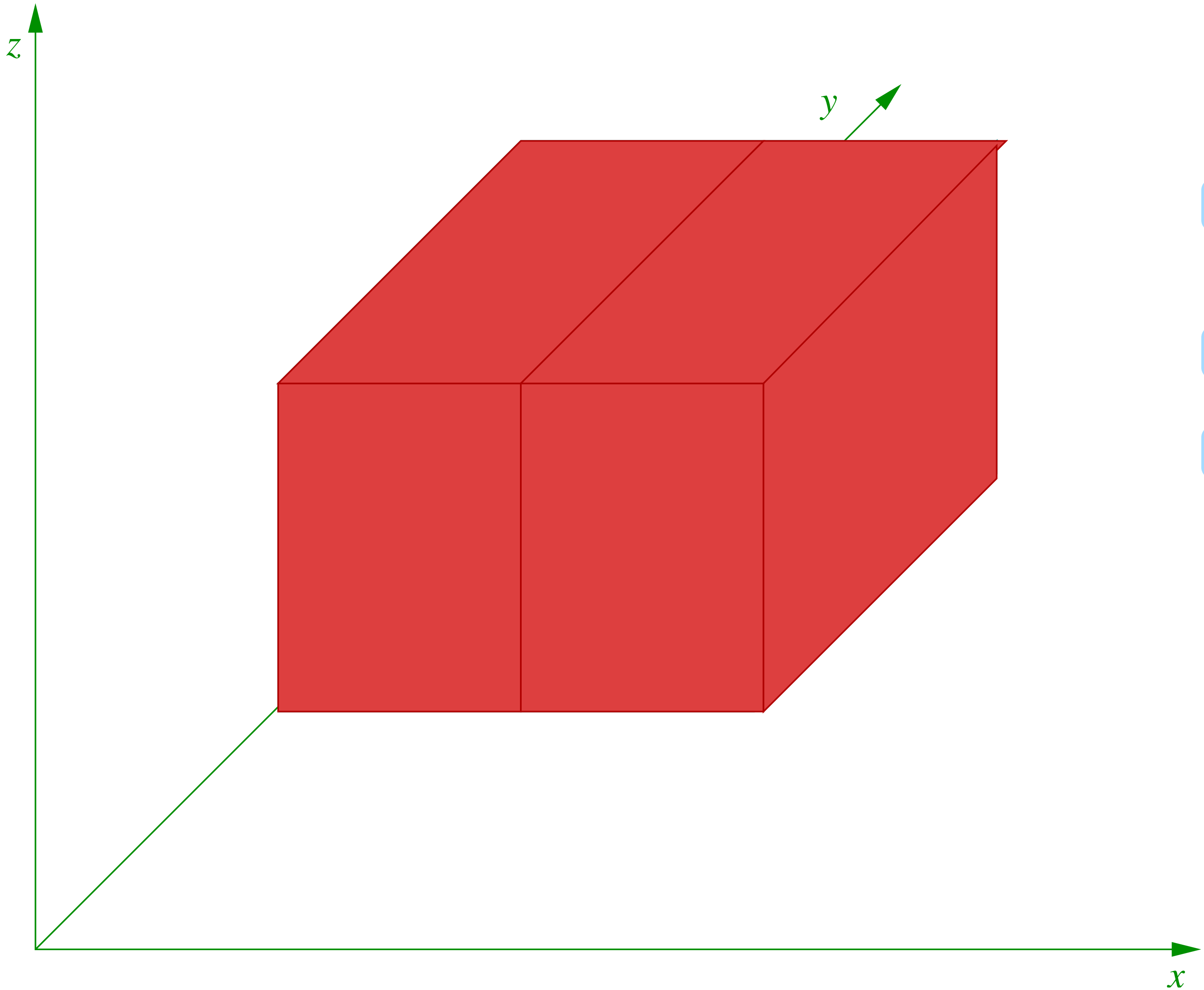
$x > 0$
$x < 3$
$y > 2$
$y < 5$
$z > 0$
$z < 2$
$x > y$
$z = x*y$
$y = z+x$
...

Learning: Principle



$x > 0$ $x > 2$
$x < 3$
$y > 2$
$y < 5$
$z > 0$
$z < 2$
$x > y$
$z = x*y$
$y = z+x$
...

Learning: Principle



$x > 0$

$x > 2$

$x < 3$

$y > 2$

$y < 5$

$z > 0$

$z < 2$

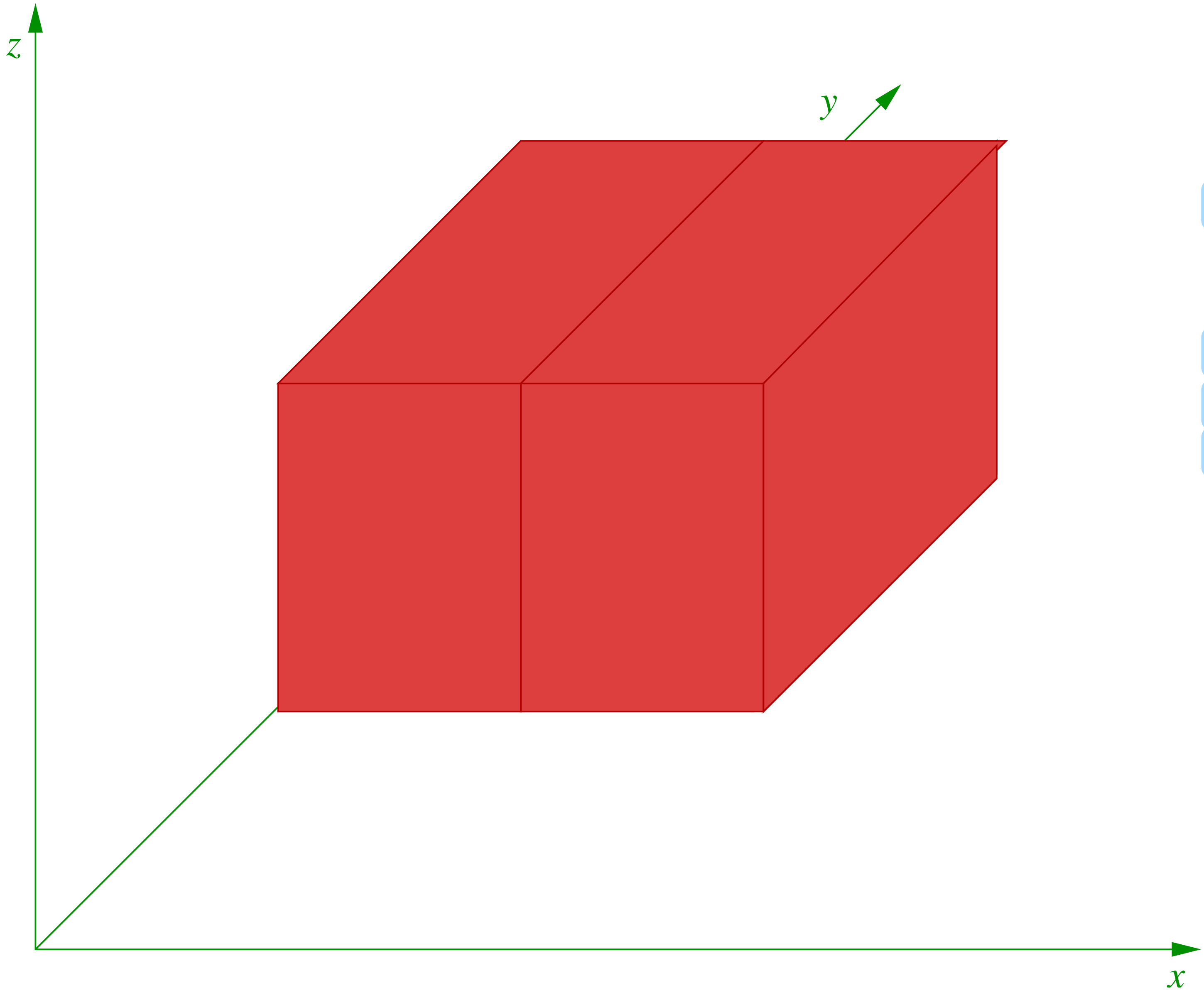
$x > y$

$z = x*y$

$y = z+x$

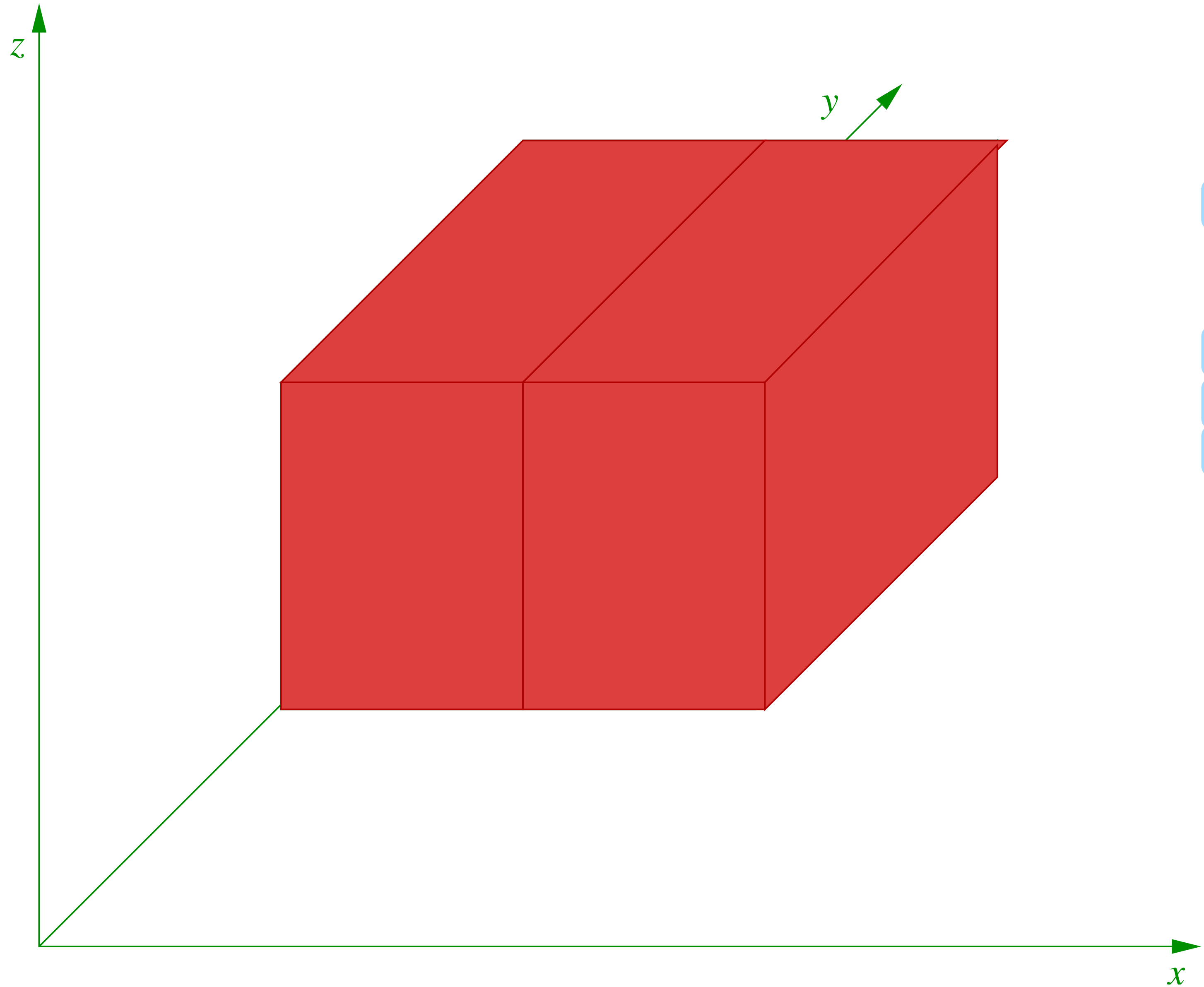
...

Learning: Principle



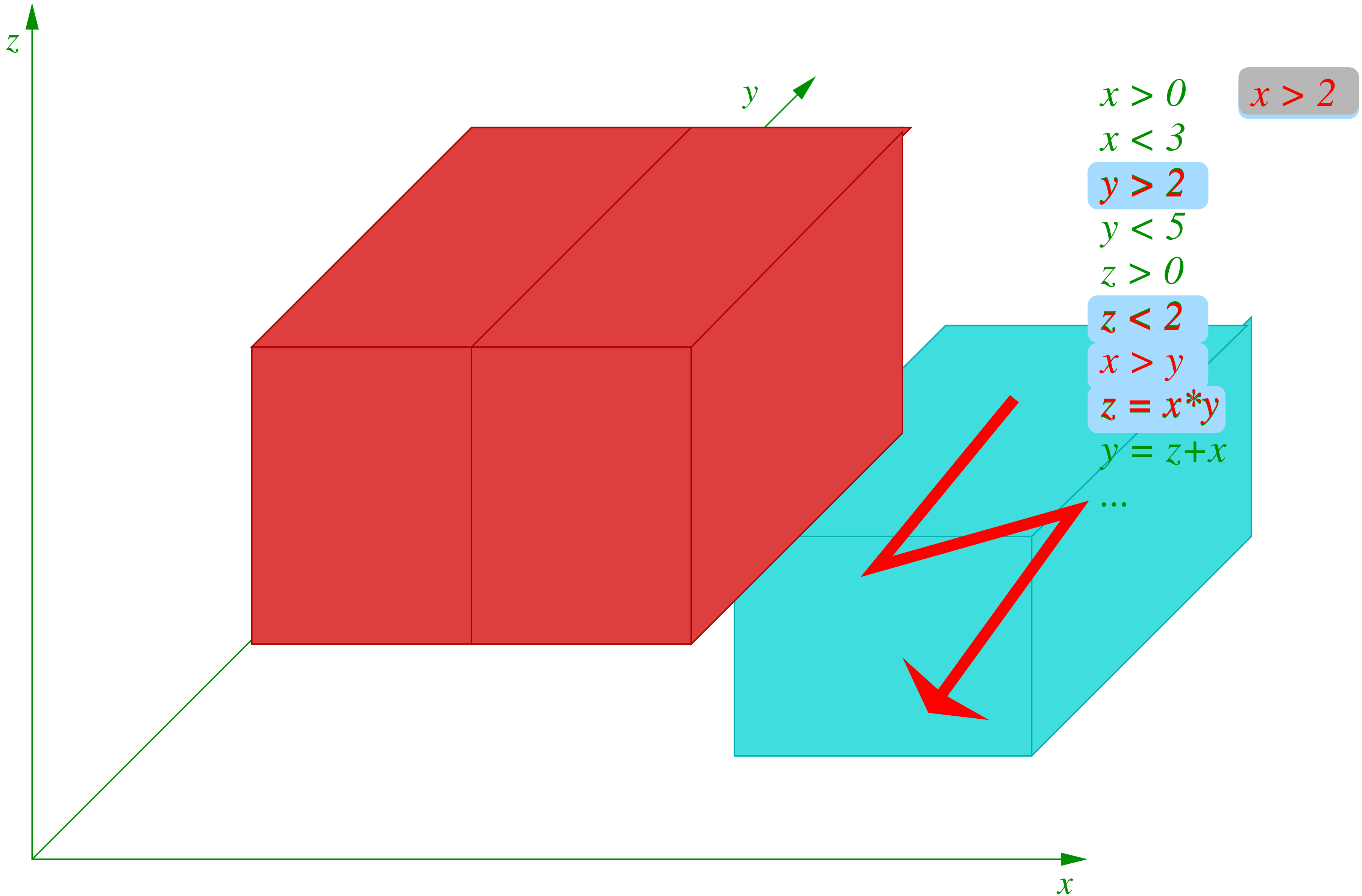
$x > 0$ $x > 2$
$x < 3$
$y > 2$
$y < 5$
$z > 0$
$z < 2$
$x > y$
$z = x*y$
$y = z+x$
...

Learning: Principle



$x > 0$ $x > 2$
$x < 3$
$y > 2$
$y < 5$
$z > 0$
$z < 2$
$x > y$
$z = x*y$
$y = z+x$
...

Learning: Principle



Refutes other candidate boxes and constraint combinations immediately.

Optimizations for DP:

Watched literal scheme

Watched literals

Boolean SAT: Within each (not yet satisfied) clause, watch two unassigned literals:

$$
\begin{array}{ccccccc}
x & \vee & y & \vee & \neg z & \vee & a \\
\texttt{true} & & \texttt{false} & & \uparrow & & \uparrow \\
 & & & & \text{watch} & & \text{watch}
\end{array}
$$

Clause needs only be visited if one of the watched lieterals gets assigned with wrong polarity. Otherwise clause either satisfied or still satisfiable.

Watched literals

Lattice-SAT: Within each clause, watch two undecided elementary formulae:

$x \geq 4$	$\vee$	$z + x \hat{=} y$	$\vee$	$y \geq 1$	$\vee$	$a > 4$
$(2, 3)$		$(0, 1), (2, 3), (1, 3)$		$(1, 3)$		$(2, 5)$
`false`		$\uparrow$		`true`		$\uparrow$
		watch				watch

Clause needs only be visited if a variable in the observed parts becomes assigned:

- visit if a's upper bound is reduced
  (would suffice to visit if reduced to 4 or below)
- visit if x's, y's, or z's interval is narrowed
  (would actually suffice to visit if $(z \oplus x) \cap y$ becomes empty)

Enforcing termination

Enforcing termination

- SAT on an infinitely deep lattice may digress into an infinite sequence of splits.

Enforcing termination

- SAT on an infinitely deep lattice may digress into an infinite sequence of splits.
- This can be avoided if splitting depth within a SAT-solver run is bounded a priori:
  1. Select a bound on splitting depth,
  2. run lattice-SAT and learn a pseudo-conflict closing the branch whenever current search path has reached maximum number of splits,
  3. report any solution thus found or any certificate of unsatisfiability thus found (sound results due to monotonicity!),
  4. if problem remained unsolved then
     (a) reopen closed branches through deletion of *pseudo*-conflicts,
     (b) restart SAT with larger splitting depth.

Enforcing termination

- SAT on an infinitely deep lattice may digress into an infinite sequence of splits.

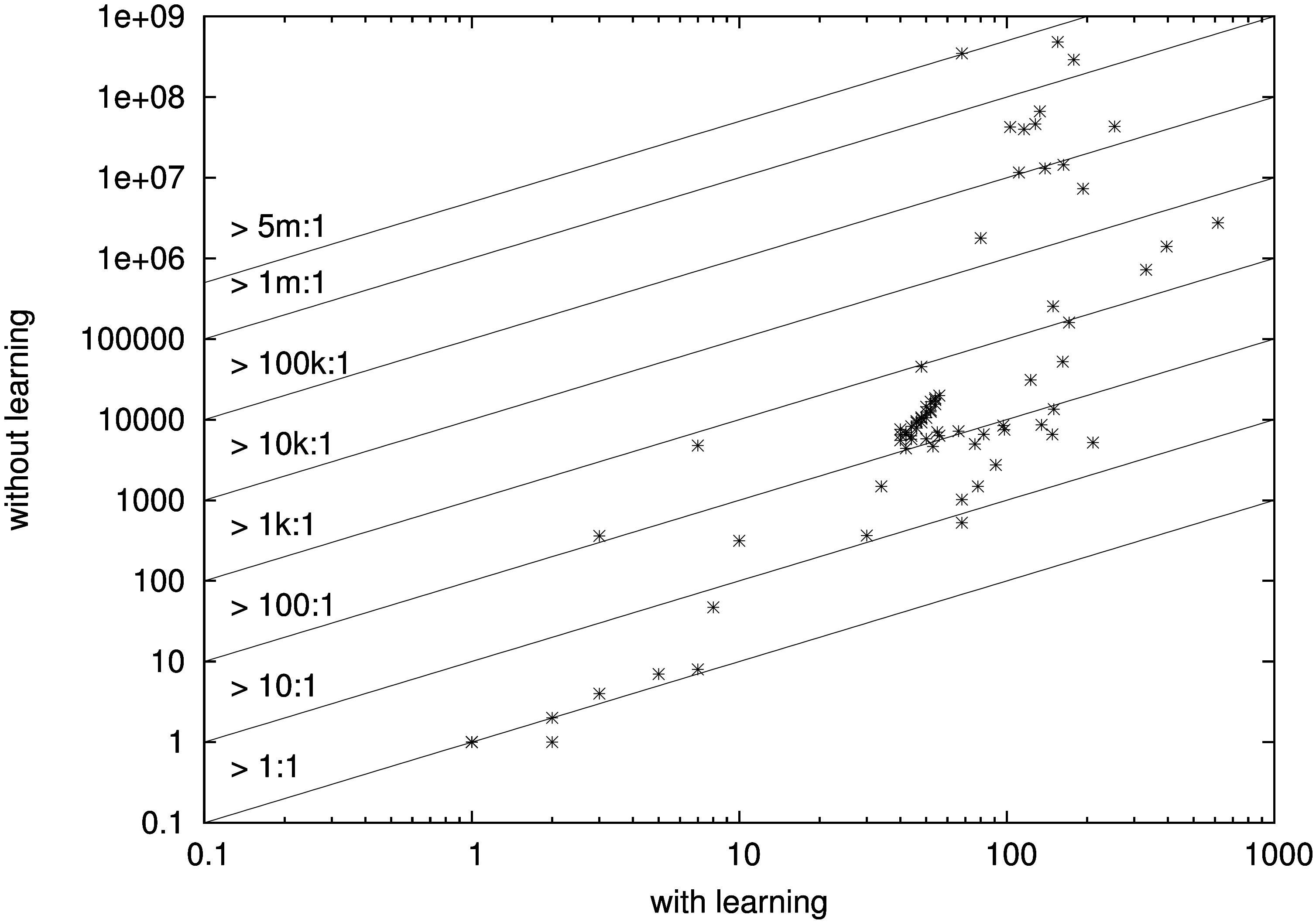
- This can be avoided if splitting depth within a SAT-solver run is bounded a priori:
  1. Select a bound on splitting depth,
  2. run lattice-SAT and learn a pseudo-conflict closing the branch whenever current search path has reached maximum number of splits,
  3. report any solution thus found or any certificate of unsatisfiability thus found (sound results due to monotonicity!),
  4. if problem remained unsolved then
     (a) reopen closed branches through deletion of *pseudo*-conflicts,
     (b) restart SAT with larger splitting depth.

- Due to conflict-driven learning, restarts do never reexplore paths already solved with lower splitting depth!

iSAT in practice:

Benchmark results

The impact of learning: no. of conflicts



Examples:
BMC of

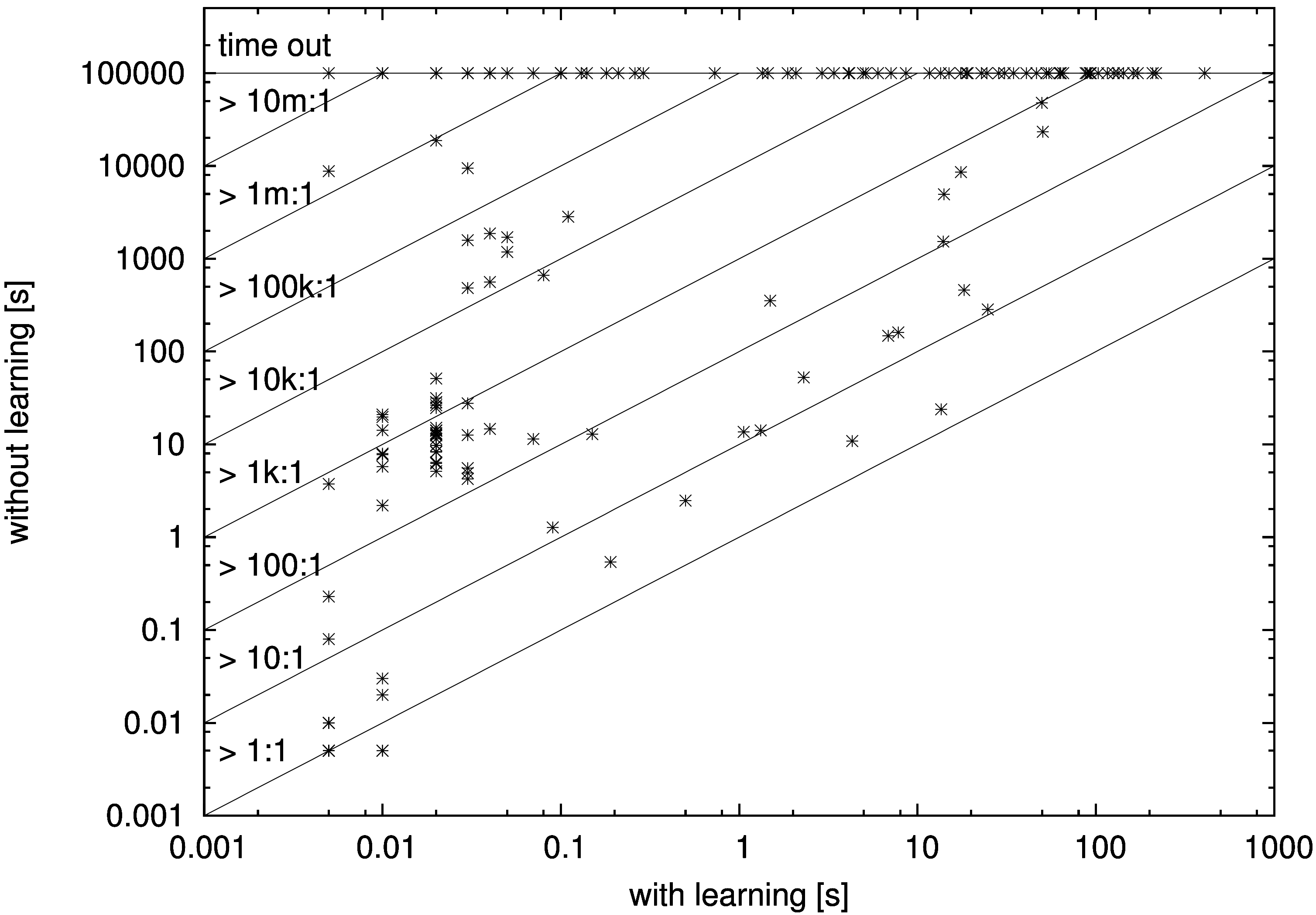
- train ctrl.
- bounc. ball
- gingerbread map
- oscillatory logistic map

Intersect. of geometric bodies

Size: Limited to some 100 var.s by solver *without learning*

$\Longrightarrow$ enormous pruning of search space already on small examples

The impact of learning: runtime



[2.5 GHz AMD Opteron, 4 GByte physical memory, Linux]

Examples:
BMC of

- train ctrl.
- bounc. ball
- gingerbread map
- oscillatory logistic map

Intersect. of geometric bodies

Size:
Up to 2400 var.s, $\gg 10^3$ Boolean connectives.

iSAT in practice

Formula syntax

Constraint solving: single formula mode

```
DECL
  int [1, 100] a, b, c;

EXPR
  a*a + b*b = c*c;
```

- Two sections:
  1. Variable declarations (keyword “DECL”)
  2. Constraint (keyword “EXPR”)
- Variables can be bounded integers (“int”), bounded reals (“float”), or Booleans (“boole”)
- integers and reals come with declarations of bounded ranges:

```
int [-17, 123] a, b;
int [13,54045] c;
float [-9999.9999,3.1415927] alpha,omega;
```

iSAT: type consistency

- `boole` is identified with `int [0,1]`
- `float`s and `int`s can be freely mixed within constraints,
- constraint evaluation is always in (safely outward rounding) float arithmetic,
- the restriction to `int` only confines the search lattice:
  - interval split: $[a, b] \quad \rightsquigarrow \quad [a, z] \cup [z+1, b]$ with $z \in \mathbb{Z}$,
  - strengthened propagation: $\ldots \quad \rightsquigarrow \quad [a, b] \quad \rightsquigarrow \quad [\lceil a \rceil, \lfloor b \rfloor]$.

Sample constraints

```
x + y * 2 >= 5 + 2 * y;
x / y > 10 xor !a;
abs(nrt(x,5)) < 2.545;
```

Note that any type of fixedpoint equation is possible:

- `(a + x / y) = x`

and that type constraints can (voluntarily or accidentially) destroy referential transparency:

- `(a + x / y) = x` for `float[...]x, y` vs.
- `i = x / y; (a + i) = x` for `float[...]x, y; int[...]i`.

Interpreting the results

iSAT (a.k.a. HySAT 2) returns

unsatisfiable: all possible interval assignments, and hence all possible real-valued assignments, have been refuted,

candidate solution box found: an interval box has been found which is free of conflict and sufficiently small (below the selected minimum split width).

The publicly available version of iSAT

- does not yet contain a check for actual existence of a satisfying solution in the candidate box,
- for real-valued problems, safer information may be obtained by restarting with smaller bounds on interval width in splitting and on progress in deduction,
- for integer-valued problems, “candidate solution box found” can generally be identified with “solution found”.