MINISAT

SAT ALGORITHMS AND APPLICATIONS

Niklas Sörensson
nik@cs.chalmers.se

Chalmers University of Technology and Göteborg University

Empirically Successful Classical Automated Reasoning
a CADE-20 Workshop
22nd - 23th July, 2005

MINISAT

MINISAT is a SAT solver with the following features:

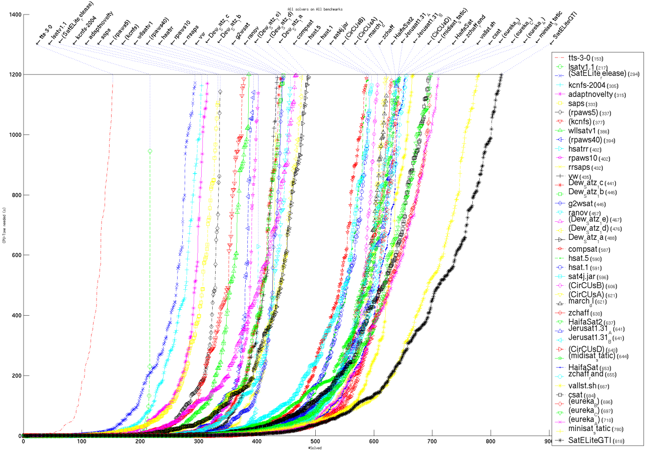
- Simple, well documented implementation suitable for educational purposes
- Incremental SAT-solving
- Well-defined interface for general boolean constraints

Empirically Successful

SAT competition 2005 results:

- MINISAT won all industrial categories
- MINISAT also performed best overall

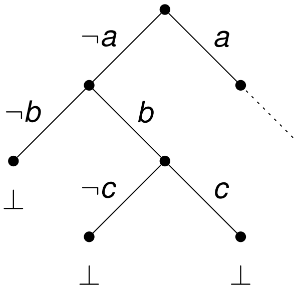
Empirically Successful



Overview

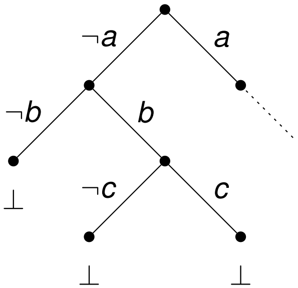
- Algorithms for SAT
- Applications

DPLL SAT solving



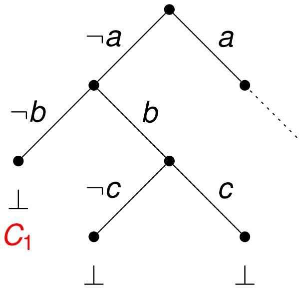
- Branching
- Unit propagation
- Backtracking

DPLL SAT solving



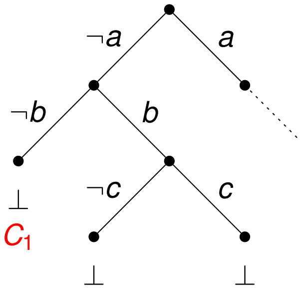
- Branching
- Unit propagation
- Backtracking
- Learning

DPLL SAT solving



- Branching
- Unit propagation
- Backtracking
- Learning

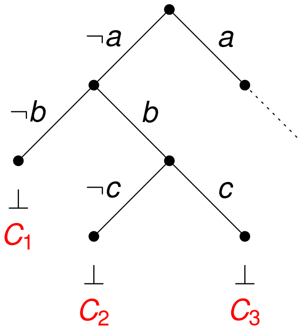
DPLL SAT solving



- Branching
- Unit propagation
- Backtracking
- Learning

Analyze the conflict to infer a clause C_1 that is a logical consequence of the problem

DPLL SAT solving



- Branching
- Unit propagation
- Backtracking
- Learning

Analyze the conflict to infer a clause C_1 that is a logical consequence of the problem

Propagation & Conflicts

Unit propagation

Is realized by particular datastructures rather than explicit inferences. No clauses are changed.

Propagation & Conflicts

Unit propagation

Is realized by particular datastructures rather than explicit inferences. No clauses are changed.

Conflict

A conflict is a situation where at least one conflicting clause is falsified by unit propagation

Conflict Clause

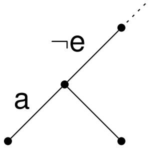
- Infered by conflict analysis
- Helps prune future parts of the search space
- Actually drives backtracking

Requirements

- Consequence of the clause set
- Falsified by current assignment
- Contains exactly one literal implied by last assumption

Conflict Analysis Example

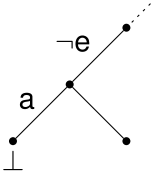
⋮	⋮
e=F	*assumption*
f=F	¬f ∨ e
g=F	¬g ∨ f
h=F	¬h ∨ g
a=T	*assumption*
b=T	b ∨ ¬a ∨ e
c=T	c ∨ e ∨ f
d=T	d ∨ ¬b ∨ h



Conflict Analysis Example

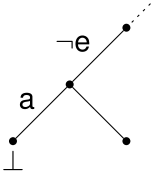
⋮	⋮
e=F	*assumption*
f=F	¬f ∨ e
g=F	¬g ∨ f
h=F	¬h ∨ g
a=T	*assumption*
b=T	b ∨ ¬a ∨ e
c=T	c ∨ e ∨ f
d=T	d ∨ ¬b ∨ h

¬b ∨ ¬c ∨ ¬d



Conflict Analysis Example

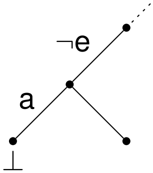
⋮	⋮
e=F	*assumption*
f=F	¬f ∨ e
g=F	¬g ∨ f
h=F	¬h ∨ g
a=T	*assumption*
b=T	b ∨ ¬a ∨ e
c=T	c ∨ e ∨ f
d=T	d ∨ ¬b ∨ h



¬b ∨ ¬c ∨ ¬d

Conflict Analysis Example

⋮	⋮
e=F	*assumption*
f=F	$\neg f \lor e$
g=F	$\neg g \lor f$
h=F	$\neg h \lor g$
a=T	*assumption*
b=T	$b \lor \neg a \lor e$
c=T	$c \lor e \lor f$
d=T	$d \lor \neg b \lor h$

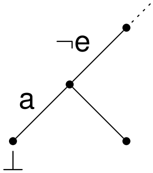


$\neg b \lor \neg c \lor \neg d$

$\neg b \lor \neg c \lor h$

Conflict Analysis Example

⋮	⋮
e=F	*assumption*
f=F	$\neg f \lor e$
g=F	$\neg g \lor f$
h=F	$\neg h \lor g$
a=T	*assumption*
b=T	$b \lor \neg a \lor e$
c=T	$c \lor e \lor f$
d=T	$d \lor \neg b \lor h$

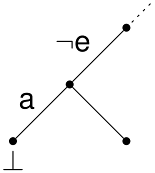


$\neg b \lor \neg c \lor \neg d$

$\neg b \lor \neg c \lor h$

Conflict Analysis Example

⋮	⋮
e=F	*assumption*
f=F	$\neg f \vee e$
g=F	$\neg g \vee f$
h=F	$\neg h \vee g$
a=T	*assumption*
b=T	$b \vee \neg a \vee e$
c=T	$c \vee e \vee f$
d=T	$d \vee \neg b \vee h$



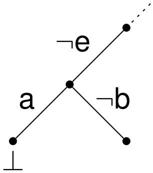
$\neg b \vee \neg c \vee \neg d$

$\neg b \vee \neg c \vee h$

$\neg b \vee h \vee e \vee f$

Conflict Analysis Example

⋮	⋮
e=F	*assumption*
f=F	¬f ∨ e
g=F	¬g ∨ f
h=F	¬h ∨ g
a=T	*assumption*
b=T	b ∨ ¬a ∨ e
c=T	c ∨ e ∨ f
d=T	d ∨ ¬b ∨ h



¬b ∨ ¬c ∨ ¬d
¬b ∨ ¬c ∨ h
¬b ∨ h ∨ e ∨ f

Conflict Analysis Algorithm

1. Begin with conflicting clause
2. Resolve on the most recently propagated literal, using the clause that caused the propagation as side clause
3. Repeat until the candidate clause contains exactly one literal implied by the last assumption

Alternative Conflict Analyses

- Traditional Conflict Analysis is minimal in the number of derivations
- Balance between time spent and usefulness of the conflict clause
- Is a shorter clause always better?

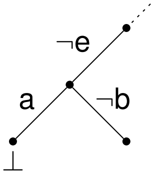
Subsumption Resolution

$$\frac{a \vee C \quad \neg a \vee D}{D} \quad C \subseteq D$$

Conclusion clause subsumes one of the antecedents

Basic Conflict Minimizing Example

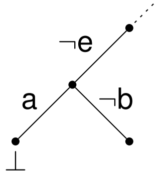
⋮	⋮
e=F	*assumption*
f=F	$\neg f \vee e$
g=F	$\neg g \vee f$
h=F	$\neg h \vee g$
a=T	*assumption*
b=T	$b \vee \neg a \vee e$
c=T	$c \vee e \vee f$
d=T	$d \vee \neg b \vee h$



$\neg b \vee \neg c \vee \neg d$
$\neg b \vee \neg c \vee h$
$\neg b \vee h \vee e \vee f$

Basic Conflict Minimizing Example

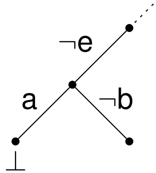
⋮	⋮
e=F	*assumption*
f=F	¬f ∨ e
g=F	¬g ∨ f
h=F	¬h ∨ g
a=T	*assumption*
b=T	b ∨ ¬a ∨ e
c=T	c ∨ e ∨ f
d=T	d ∨ ¬b ∨ h



¬b ∨ ¬c ∨ ¬d
¬b ∨ ¬c ∨ h
¬b ∨ h ∨ e ∨ f

Basic Conflict Minimizing Example

⋮	⋮
$e{=}F$	*assumption*
$f{=}F$	$\neg f \vee e$
$g{=}F$	$\neg g \vee f$
$h{=}F$	$\neg h \vee g$
$a{=}T$	*assumption*
$b{=}T$	$b \vee \neg a \vee e$
$c{=}T$	$c \vee e \vee f$
$d{=}T$	$d \vee \neg b \vee h$



$\neg b \vee \neg c \vee \neg d$

$\neg b \vee \neg c \vee h$

$\neg b \vee h \vee e \vee f$

$\neg b \vee h \vee e$

Basic Conflict Minimizing

- Start from ordinary conflict clause
- Apply subsumption resolution greedily
- Works because there are no cyclic dependencies
- Also uses reason clauses from other levels

Very cheap, almost for free.

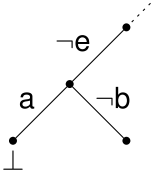
Generalized Subsumption Resolution

$$\frac{C_1 \quad D_1}{C_2}, \quad \frac{C_2 \quad D_2}{C_3}, \quad \ldots, \quad \frac{C_{n-1} \quad D_{n-1}}{C_n}$$

where C_n subsumes C_1

Full Conflict Minimizing Example

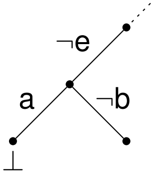
⋮	⋮
e=F	*assumption*
f=F	¬f ∨ e
g=F	¬g ∨ f
h=F	¬h ∨ g
a=T	*assumption*
b=T	b ∨ ¬a ∨ e
c=T	c ∨ e ∨ f
d=T	d ∨ ¬b ∨ h



¬b ∨ ¬c ∨ ¬d
¬b ∨ ¬c ∨ h
¬b ∨ h ∨ e ∨ f

Full Conflict Minimizing Example

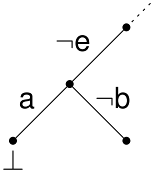
⋮	⋮
e=F	*assumption*
f=F	¬f ∨ e
g=F	¬g ∨ f
h=F	¬h ∨ g
a=T	*assumption*
b=T	b ∨ ¬a ∨ e
c=T	c ∨ e ∨ f
d=T	d ∨ ¬b ∨ h



¬b ∨ ¬c ∨ ¬d
¬b ∨ ¬c ∨ h
¬b ∨ h ∨ e ∨ f

Full Conflict Minimizing Example

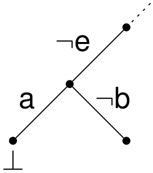
⋮	⋮
e=F	*assumption*
f=F	¬f ∨ e
g=F	¬g ∨ f
h=F	¬h ∨ g
a=T	*assumption*
b=T	b ∨ ¬a ∨ e
c=T	c ∨ e ∨ f
d=T	d ∨ ¬b ∨ h



¬b ∨ ¬c ∨ ¬d
¬b ∨ ¬c ∨ h
¬b ∨ h ∨ e ∨ f
¬b ∨ e ∨ f ∨ g

Full Conflict Minimizing Example

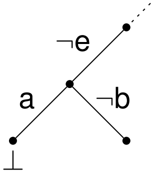
⋮	⋮
e=F	*assumption*
f=F	¬f ∨ e
g=F	¬g ∨ f
h=F	¬h ∨ g
a=T	*assumption*
b=T	b ∨ ¬a ∨ e
c=T	c ∨ e ∨ f
d=T	d ∨ ¬b ∨ h



¬b ∨ ¬c ∨ ¬d
¬b ∨ ¬c ∨ h
¬b ∨ h ∨ e ∨ f
¬b ∨ e ∨ f ∨ g

Full Conflict Minimizing Example

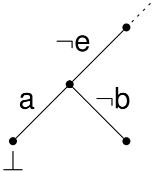
⋮	⋮
e=F	*assumption*
f=F	¬f ∨ e
g=F	¬g ∨ f
h=F	¬h ∨ g
a=T	*assumption*
b=T	b ∨ ¬a ∨ e
c=T	c ∨ e ∨ f
d=T	d ∨ ¬b ∨ h



¬b ∨ ¬c ∨ ¬d
¬b ∨ ¬c ∨ h
¬b ∨ h ∨ e ∨ f
¬b ∨ e ∨ f ∨ g
¬b ∨ e ∨ f

Full Conflict Minimizing Example

⋮	⋮
$e = F$	*assumption*
$f = F$	$\neg f \lor e$
$g = F$	$\neg g \lor f$
$h = F$	$\neg h \lor g$
$a = T$	*assumption*
$b = T$	$b \lor \neg a \lor e$
$c = T$	$c \lor e \lor f$
$d = T$	$d \lor \neg b \lor h$



$\neg b \lor \neg c \lor \neg d$

$\neg b \lor \neg c \lor h$

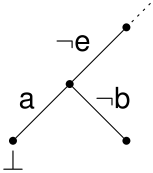
$\neg b \lor h \lor e \lor f$

$\neg b \lor e \lor f \lor g$

$\neg b \lor e \lor f$

Full Conflict Minimizing Example

⋮	⋮
e=F	*assumption*
f=F	¬f ∨ e
g=F	¬g ∨ f
h=F	¬h ∨ g
a=T	*assumption*
b=T	b ∨ ¬a ∨ e
c=T	c ∨ e ∨ f
d=T	d ∨ ¬b ∨ h



¬b ∨ ¬c ∨ ¬d
¬b ∨ ¬c ∨ h
¬b ∨ h ∨ e ∨ f
¬b ∨ e ∨ f ∨ g
¬b ∨ e ∨ f
¬b ∨ e

Full Conflict Minimizing

- Search for applications of generalized subsumption
- Caching of subresults is neccessary to make it efficient

Is relatively expensive, but simplifies a lot

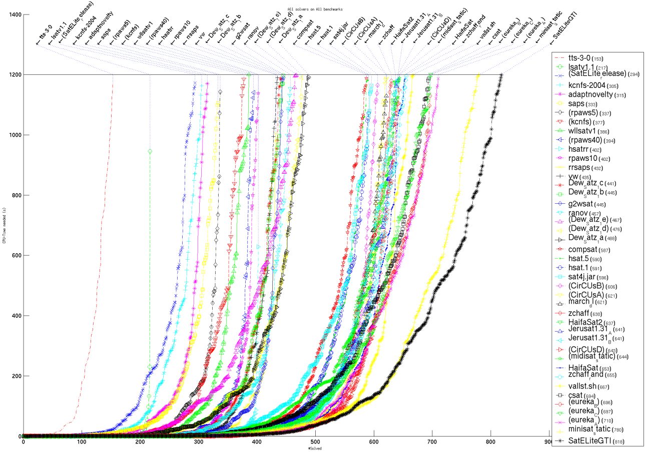
Effect of Minimization

	w10_45.cnf		fifo8_200.cnf		f2clk_30.cnf	
	LPC	Time	LPC	Time	LPC	Time
Orig	47.0	4.90	37.9	182.0	174.1	33.273
Basic	31.3 (7.8%)	4.28	36.2 (11.5%)	194.9	92.1 (12.8%)	29.482
Full	13.2 (4.6%)	3.84	16.5 (29.6%)	158.1	39.0 (42.1%)	31.628

SatELite

- A pre-processing tool developed by Niklas Een
- Uses simplification techniques
  - Variable elimination
  - Subsumption
  - Subsumption resolution
- Works well for simplifying formulas translated from circuits

Effects of using SatElite



Incremental SAT

- Allows to solve a sequence of related problems
- Between runs clauses can be added or unit clauses can be retracted
- Learned clauses are valid in later runs

Safety Property Verification

Symbolic Finite State Machine

- $T(s, s')$ *(transition relation)*
- $I(s)$ *(initial states)*

Can a state satisfying a property $\neg P(s)$ be reached?

Temporal Induction

Temporal Induction

$$I(s_0) \wedge \neg P(s_0) \qquad \textit{(base 1)}$$

Temporal Induction

$$I(s_0) \wedge \neg P(s_0) \qquad \textit{(base 1)}$$

$$P(s_1) \wedge T(s_1, s_0) \wedge \neg P(s_0) \qquad \textit{(ind. 1)}$$

Temporal Induction

$$I(s_0) \land \neg P(s_0) \qquad \textit{(base 1)}$$

$$P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \qquad \textit{(ind. 1)}$$

$$I(s_1) \land P(s_1) \land T(s_1, s_0) \land \neg P(s_0) \qquad \textit{(base 2)}$$

Temporal Induction

$$I(s_0) \wedge \neg P(s_0) \qquad \textit{(base 1)}$$

$$P(s_1) \wedge T(s_1, s_0) \wedge \neg P(s_0) \qquad \textit{(ind. 1)}$$

$$I(s_1) \wedge P(s_1) \wedge T(s_1, s_0) \wedge \neg P(s_0) \qquad \textit{(base 2)}$$

$$P(s_2) \wedge T(s_2, s_1) \wedge P(s_1) \wedge T(s_1, s_0) \wedge \neg P(s_0) \qquad \textit{(ind. 2)}$$

Temporal Induction

$$I(s_0) \wedge \neg P(s_0) \qquad \textit{(base 1)}$$

$$P(s_1) \wedge T(s_1, s_0) \wedge \neg P(s_0) \qquad \textit{(ind. 1)}$$

$$I(s_1) \wedge P(s_1) \wedge T(s_1, s_0) \wedge \neg P(s_0) \qquad \textit{(base 2)}$$

$$P(s_2) \wedge T(s_2, s_1) \wedge P(s_1) \wedge T(s_1, s_0) \wedge \neg P(s_0) \qquad \textit{(ind. 2)}$$

$$I(s_2) \wedge P(s_2) \wedge T(s_2, s_1) \wedge P(s_1) \wedge T(s_1, s_0) \wedge \neg P(s_0) \qquad \textit{(base 3)}$$

Temporal Induction

$$I(s_0) \wedge \neg P(s_0) \qquad \textit{(base 1)}$$

$$P(s_1) \wedge T(s_1, s_0) \wedge \neg P(s_0) \qquad \textit{(ind. 1)}$$

$$I(s_1) \wedge P(s_1) \wedge T(s_1, s_0) \wedge \neg P(s_0) \qquad \textit{(base 2)}$$

$$P(s_2) \wedge T(s_2, s_1) \wedge P(s_1) \wedge T(s_1, s_0) \wedge \neg P(s_0) \qquad \textit{(ind. 2)}$$

$$I(s_2) \wedge P(s_2) \wedge T(s_2, s_1) \wedge P(s_1) \wedge T(s_1, s_0) \wedge \neg P(s_0) \qquad \textit{(base 3)}$$

$$\vdots$$

Coding Unique States

For completeness all states in a path is required to be different

Coding Unique States

For completeness all states in a path is required to be different

Problem

Encoding the unique states requirement gives a formula quadratic in number of states

Coding Unique States

For completeness all states in a path is required to be different

Problem

Encoding the unique states requirement gives a formula quadratic in number of states

Solution

Overhead avoided by lazy addition of uniqueness constraints

Paradox - a First Order Model Finder

- Searches for models with a finite domain.
- Uses MiniSat in an incremental way
- Won the SAT* category of CASC 2003 and 2004

First Order Model Finding Using SAT

- When searching for models of size k it is enough to choose the first k natural numbers as the domain.

First Order Model Finding Using SAT

- When searching for models of size k it is enough to choose the first k natural numbers as the domain.
- In an interpretation each ground atom is identified with a propositional variable. Example:

$$\mathrm{P}(0), \quad \mathrm{P}(1), \quad \mathrm{f}(0) = 0, \quad \mathrm{f}(0) = 1, \quad \mathrm{f}(1) = 0, \quad \mathrm{f}(1) = 1$$

First Order Model Finding Using SAT

- When searching for models of size k it is enough to choose the first k natural numbers as the domain.
- In an interpretation each ground atom is identified with a propositional variable. Example:

$$\mathrm{P}(0), \quad \mathrm{P}(1), \quad \mathrm{f}(0)=0, \quad \mathrm{f}(0)=1, \quad \mathrm{f}(1)=0, \quad \mathrm{f}(1)=1$$

- To transform a first order problem to SAT we need to get rid of complex terms, and variables. This is done by function flattening, and ground instantiation.

Function flattening

Shallow Literals are of one of the following forms:

1. $P(x_1, \ldots, x_m)$, or $\neg P(x_1, \ldots, x_m)$,
2. $f(x_1, \ldots, x_n) = y$, or $f(x_1, \ldots, x_n) \neq y$,
3. $x = y$.

Rewrite rules

$$
\begin{array}{lll}
1. & C[t] & \longrightarrow & t \neq x \lor C[x] \\
2. & x \neq y \lor C[x, y] & \longrightarrow & C[x, x]
\end{array}
$$

Terminates with a shallow clause set, if applied exhaustively

Ground Instantiation

Ground instantiation of a clause set F is done for the particular domain $D = \{1, \ldots, k\}$

1. Instances $F\sigma$ with respect to D

Ground Instantiation

Ground instantiation of a clause set F is done for the particular domain $D = \{1, \ldots, k\}$

1. Instances $F\sigma$ with respect to D
2. Functionality constraints: each function has at most one value in D for each input

   Example: $\mathrm{f}(1) \neq 1 \vee \mathrm{f}(1) \neq 2$

Ground Instantiation

Ground instantiation of a clause set F is done for the particular domain $D = \{1, \ldots, k\}$

1. Instances $F\sigma$ with respect to D
2. Functionality constraints: each function has at most one value in D for each input

   Example: $\mathrm{f}(1) \neq 1 \vee \mathrm{f}(1) \neq 2$
3. Totality constraints: each function has at least one value in D for each input

   Example: $\mathrm{f}(1) = 1 \vee \mathrm{f}(1) = 2 \vee \mathrm{f}(1) = 3, \quad$ for $D = \{1, 2, 3\}$

Non Ground Splitting

Grounding is expontential in the number of variables in a clause. Number of variables can be reduced by splitting.

Non Ground Splitting

Grounding is expontential in the number of variables in a clause. Number of variables can be reduced by splitting.

Example: $P(x,y) \vee Q(y,z) \longrightarrow$

1. $P(x,y) \vee S(y)$
2. $Q(y,z) \vee \neg S(y)$

Non Ground Splitting

Grounding is expontential in the number of variables in a clause. Number of variables can be reduced by splitting.

Example: $P(x, y) \vee Q(y, z) \longrightarrow$

1. $P(x, y) \vee S(y)$
2. $Q(y, z) \vee \neg S(y)$

- Optimally splitting a clause is NP hard
- We use a simple greedy heuristic that works well

Summary

Summary

- Relatively expensive techiques pay off in computing conflict clauses

Summary

- Relatively expensive techiques pay off in computing conflict clauses
- Variable elimination based preprocessing helps with industrial problems

Summary

- Relatively expensive techiques pay off in computing conflict clauses
- Variable elimination based preprocessing helps with industrial problems
- Incremental SAT allows to solve a sequence of related problems more efficiently

Summary

- Relatively expensive techiques pay off in computing conflict clauses
- Variable elimination based preprocessing helps with industrial problems
- Incremental SAT allows to solve a sequence of related problems more efficiently
- Use incremental architecture for lazy encoding of otherwise infeasible (or expensive) constraints