Sudoku Programming Challenge

Andreas Klappenecker

Goal

The goal is to solve Sudoku using SAT solvers.

Study the impact of clause generation.

Try several different SAT solvers, explain how they work, and find out which one works best.

How it is done

- Form a team of up to 4 people
- Distribute roles (CNF generation, SAT solver integration, documentation, data collection and strategy optimization, etc.)
- Organize meetings
- Everything should be ready by Monday, Dec. 7

Deliverables

- Documentation that explains in detail the generation of clauses from a given Sudoku puzzle, the strategy of the SAT solvers (in detail - this is an algorithms class). Data that explains why some combinations are better than others.
- Program that implements these variations
- Solutions to the challenge problems that will be posted

Why do it?

- Participation score (minimal: documentation of approach including the description of the algorithms of several SAT solvers).
- Extra credit for fully working Sudoku solver (up to 4 points of the total 100 points of the entire course for each team mate).

Knuth's Solvers

Knuth has implemented the solvers described in his Chapter 7.2.2.2.

They are written in CWEB, a mix of C and TeX, in the literate programming style.

CWEAVE sat0.w   yields the documentation sat0.tex (tex sat0.tex)

CTANGLE sat0.w yields the C source code sat0.c (use gcc)

You need to install the Stanford Graph Base if you want to compile it.

Backtracking

$$F(x_1, x_2, x_3) = (x_1 \vee \bar{x}_2) \wedge (x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \qquad (1)$$

For example, consider again the formula F in (1). If we set $x_1 = 0$, F reduces to $\bar{x}_2 \wedge (x_2 \vee x_3)$, because the first clause $(x_1 \vee \bar{x}_2)$ loses its x_1, while the last two clauses contain $\bar{x}_1$ and are satisfied. It will be convenient to have a notation for this reduced problem; so let's write

$$F \,|\, \bar{x}_1 \;=\; \bar{x}_2 \wedge (x_2 \vee x_3). \qquad (54)$$

Similarly, if we set $x_1 = 1$, we obtain the reduced problem

$$F \,|\, x_1 \;=\; (x_2 \vee x_3) \wedge \bar{x}_3 \wedge (\bar{x}_2 \vee x_3). \qquad (55)$$

F is satisfiable if and only if we can satisfy either (54) or (55).

Algorithm A (*Satisfiability by backtracking*). Given nonempty clauses $C_1 \wedge \cdots \wedge C_m$ on $n > 0$ Boolean variables $x_1 \ldots x_n$, represented as above, this algorithm finds a solution if and only if the clauses are satisfiable. It records its current progress in an array $m_1 \ldots m_n$ of "moves," whose significance is explained below.

A1. [Initialize.] Set $a \leftarrow m$ and $d \leftarrow 1$. (Here a represents the number of active clauses, and d represents the depth-plus-one in an implicit search tree.)

A2. [Choose.] Set $l \leftarrow 2d$. If $\mathtt{C}(l) \le \mathtt{C}(l+1)$, set $l \leftarrow l+1$. Then set $m_d \leftarrow (l \mathbin{\&} 1) + 4[\mathtt{C}(l \oplus 1) = 0]$. (See below.) Terminate successfully if $\mathtt{C}(l) = a$.

A3. [Remove $\bar{l}$.] Delete $\bar{l}$ from all active clauses; but go to A5 if that would make a clause empty. (We want to ignore $\bar{l}$, because we're making l true.)

A4. [Deactivate l's clauses.] Suppress all clauses that contain l. (Those clauses are now satisfied.) Then set $a \leftarrow a - \mathtt{C}(l)$, $d \leftarrow d+1$, and return to A2.

A5. [Try again.] If $m_d < 2$, set $m_d \leftarrow 3 - m_d$, $l \leftarrow 2d + (m_d \mathbin{\&} 1)$, and go to A3.

A6. [Backtrack.] Terminate unsuccessfully if $d = 1$ (the clauses are unsatisfiable). Otherwise set $d \leftarrow d-1$ and $l \leftarrow 2d + (m_d \mathbin{\&} 1)$.

A7. [Reactivate l's clauses.] Set $a \leftarrow a + \mathtt{C}(l)$, and unsuppress all clauses that contain l. (Those clauses are now unsatisfied, because l is no longer true.)

A8. [Unremove $\bar{l}$.] Reinstate $\bar{l}$ in all the active clauses that contain it. Then go back to A5. ▮