Practical SAT Solving (ST 2026) | Assignment 4

Ashlin Iser, Dominik Schreiber, Niccolò Rigi-Luperti

Algorithm Engineering & Scalable Automated Reasoning (KIT) 2026-06-16 – 2026-06-30

1 Variable Elimination (2×2 Points)

Let the formula S with gate encodings E_1 and E_2 be given. Apply variable elimination for gates for variables a and r. Give the clause sets after each elimination step. Try the following two strategies.

1. Eliminate variable a first, and then r if possible.
2. Eliminate variable r first, and then a if possible.

$$S = \{\underbrace{\{\neg x, \neg y, a\}, \{x, \neg a\}, \{y, \neg a\}}_{E_1}, \underbrace{\{\neg a, r\}, \{\neg z, r\}, \{a, z, \neg r\}}_{E_2}, \{a, z, r\}, \{\neg a, \neg r\}\}$$

2 Variable Elimination with Gate Encodings (3 Points)

For a gate encoding E with output x in a formula $F = E \cup R$, we simplified the resolvents $(E_x \cup R_x) \otimes (E_{\overline{x}} \cup R_{\overline{x}})$ by $S := (E_x \otimes R_{\overline{x}}) \cup (R_x \otimes E_{\overline{x}})$, dropping both $R_x \otimes R_{\overline{x}}$ and $E_x \otimes E_{\overline{x}}$. Show that the clauses in $R_x \otimes R_{\overline{x}}$ can be derived from S by resolution. You can assume that E encodes a binary AND gate.

3 Blocked Clauses (3×3 Points)

If Blocked Clause Elimination (BCE) reduces a formula F to the empty formula then F is called a blocked set. Prove the following statements.

1. Any formula F can be partitioned into two blocked sets S and L such that $F = S \cup L$. Design a linear algorithm that produces L and S from F.
2. Blocked sets are not closed unter resolution. If F is a blocked set then $F \cup C_1 \otimes C_2$, where $C_1, C_2 \in F$ may not be a blocked set anymore.
3. Blocked sets are not closed unter partially assigning variables. If F is a blocked set then $F_{x=v}$ (the result of assigning v to x and subsequent simplification) may not be a blocked set anymore.

4 Covered Clauses (4 Points)

Let F be a formula and $C \in F$ a clause. A literal $l \notin C$ is called a *covered literal* of C if every non-tautological resolvent of C with a clause in F contains l. The clause C is called *covered* if it becomes blocked in F after adding any of its covered literals. Prove that adding a covered literal to a clause does not change the satisfiability of the formula.

5 Competition: SDVSTPP (10(+10) Points)

As an extension of the slightly unrealistic SDVSTP (Assignment 2), we introduce the much more precise *Stardew Valley Soil Tilling Planning Problem* (SDVSTPP). The difference between SDVSTP and SDVSTPP is that the latter now features discrete *time steps* as well as a *player*. At any given point in time, the player is positioned at exactly one of the grid cells. At each time step, the player performs one out of two possible kinds of actions:

1. *walk* to one of the up to four adjacent tiles,
2. perform some tilling action at one of the up to four adjacent tiles.

The tilling action's orientation is decided by the player position. Figure 1 shows all potentially valid operations for each of the five patterns, where the black square is the player's location.

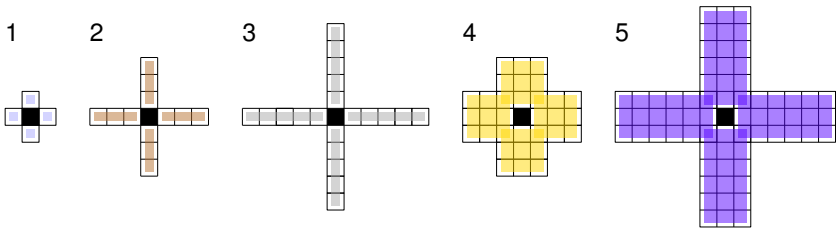
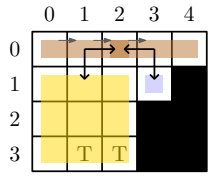


Figure 1: The five tilling patterns and their valid orientations.

Your task is to write an *optimal SAT-based SDVSTPP planner*. The planner application takes an `.sdvstp` file (just like the SDVSTP solver in Assignment 2) and assumes that the player is initially located at the *top left position*, i.e., at the first cell character read from the `.sdvstp` file. The planner should output the shortest possible plan in terms of time steps (i.e., player actions) that results in a grid state where tiles are tilled exactly according to the input specification (as in SDVSTP).

As shown in Figure 2, the plan should be output as a single line beginning with "`s `", where `U`/`D`/`L`/`R` corresponds to up/down/left/right movement and where `u`/`d`/`l`/`r` followed by `1`/`2`/`3`/`4`/`5` corresponds to tilling action 1–5 in up/down/left/right direction, respectively.



`s R d4 r2 R R l2 d1`

Figure 2: An example of an optimal SDVSTPP plan.

You receive 10 points for an optimal planner solving easy inputs and up to 10 bonus points for solving more difficult inputs.

Code skeleton: `https://github.com/satlecture/kit2025/blob/main/code/src/sdvstp/sdvstpp.cc`

Some benchmark instances: `https://github.com/satlecture/kit2025/tree/main/exercises/sdvstp`