Nat Comput (2012) 11:209–229
DOI 10.1007/s11047-011-9299-0

Efficient 3-*SAT* algorithms in the tile assembly model

Yuriy Brun

Published online: 26 January 2012

Abstract Self-assembly is a powerful process found in nature that guides simple objects assembling, on their own, into complex structures. Self-assembly is of interest to computer scientists because self-assembling systems can compute functions, assemble shapes, and guide distributed robotics systems. The tile assembly model is a formal mathematical model of self-assembly that allows the study of time and space complexities of self-assembling systems that lie at the heart of several molecular computer implementations and distributed computational software systems. These implementations and systems require efficient tile systems with small tilesets and fast execution times. The state of the art, however, requires vastly complex tile systems with large tilesets to implement fast algorithms. In this paper, I present $\mathbb{S}_{FS}$, a tile system that decides 3-*SAT* by creating $O^{\star}(1.8393^n)$ nondeterministic assemblies in parallel, improving on the previous best known solution that requires $\Theta(2^n)$ such assemblies. This solution directly improves the feasibility of building molecular 3-*SAT* solvers and efficiency of distributed software. I formally prove the correctness of the system, the number of required parallel assemblies, that the size of the system's tileset is $147 = \Theta(1)$, and that the assembly time is nondeterministic linear in the size of the input.

Keywords Self-assembly · 3-*SAT* · Efficient 3-*SAT* algorithms · Tile assembly model

Y. Brun (✉)
University of Washington,
Seattle, WA, USA
e-mail: brun@alum.mit.edu; brun@cs.washington.edu

1 Introduction

Self-assembly is a process that guides the creation of many of nature's systems, from nanoscale crystals to cosmos-scale galaxies. Specifically, self-assembly is the process of simple objects coming together and combining to form complex structures. Computer scientists and nanotechnologiests are interested in studying self-assembly because it is capable of computing functions (Adleman 2000; Adleman et al. 2002b; Berger 1966; Winfree 1998b), assembling complex shapes (Adleman et al. 2002a; Rothemund and Winfree 2000; Soloveichik and Winfree 2007), and guiding distributed robotics systems (Abelson et al. 2000; Brun and Reishus 2009; McLurkin et al. 2006).

1.1 Self-assembly and tiles

The tile assembly model (Winfree 1998a; Rothemund and Winfree 2000) is a formal mathematical model of self-assembly that allows studying the time and space complexities of self-assembling systems. Winfree showed that the tile assembly model is Turing universal (Winfree 1998b) by demonstrating that tile systems can emulate Wang tiles (Wang 1961). Wang tile systems with a seed can simulate Turing machines (Wang 1962). Even without seeds, Wang tiles are still Turing universal (Berger 1966; Robinson 1971). Adleman has identified two important measures of tile systems: assembly time and tileset size; in some ways, these measures are analogous to the time and space complexities of traditional computer programs (Adleman 2000).

Study of tile systems has led to two types of computational systems: Internet-sized distributed grids (Brun and Medvidovic 2008) and molecular computers (Adleman 2000;

Braich et al. 2002). Both types benefit from efficient tile systems with small tilesets: the speed of computational grids is proportional to the number of tile types (Brun and Medvidovic 2008) and the state of the art in DNA computation is systems with no more than 20 distinct tile types (Barish et al. 2005, 2009; Rothemund et al. 2004; Yin et al. 2008).

Winfree's universal tile systems are in some sense inefficient and require thousands of distinct tile types (Winfree 1998b). Lagoudakis et al. (Lagoudakis and LaBean 1999) presented a tile system that solves 3-*SAT*, though their best solution required $\Theta(n^2)$ distinct tile types and $\Theta(2^n)$ distinct nondeterministic assemblies to solve an n-variable problem, resulting in over 10^4 distinct tile types and 10^{15} distinct assemblies necessary to solve a 50-variable problem. I have begun the work of reducing the complexity by designing tile systems that solve complex computational problems using relatively small tilesets (e.g., adding using 8 distinct tile types (Brun 2007), multiplying using 28 (Brun 2007), factoring integers nondeterministically using 50 (Brun 2008a), and solving two NP-complete problems nondeterministically, 3-*SAT* using 64 (Brun 2008c) and *SubsetSum* using 49 (Brun 2008b)). While there has been some success in designing tile systems with small tilesets, thus far, tile systems implement only the most naïve, simple, and inefficient algorithms. For example, today's best known NP-complete problem-solving tile systems require $\Theta(2^n)$ distinct assemblies for an input of size n. While we do not know of polynomial-time algorithms to solve such problems, we do know of exponential-time algorithms with a base smaller than 2 (Woeginger 2003). Here, I present a tile system that implements a somewhat complex, known algorithm for solving 3-*SAT* using only $O^\star(1.8393^n)$ distinct assemblies (where the $O^\star$ notation hides constant and polynomial factors). This system uses 147 distinct tile types and demonstrates that complex algorithms can be implemented using tiles in a systematic manner, (1) directly improving the time and space complexities of tile-inspired computational-grid architectures (Brun and Medvidovic 2008) and (2) bridging theory and today's experimental limitations of DNA computing. While the tile assembly model's universality implies the existence of such a system, here, I construct the concrete system and demonstrate that it requires a small number of distinct tiles (147).

Chen and Ramachandran have previously attempted to improve the efficiency of 3-*SAT*-solving, molecular systems (Chen and Ramachandran 2001). They developed a DNA computer system that implements a randomized 3-*SAT* algorithm (Paturi et al. 1997). Just as my system, their work was purely theoretical and never implemented using DNA or other molecules. From the practical point of view, the randomized algorithm is similar to the nondeterministic algorithm I present here. Sakamoto et al. attempted to improve the efficiency of 3-*SAT*-solving, DNA computing systems by considering DNA encodings of clauses, rather than variables (Sakamoto et al. 2000). However, implementing DNA computation presents significant challenges (Braich et al. 2002), such as high error rates (Winfree and Bekbolatov 2003). One of the main reasons for the community's interest in self-assembly and tile systems is that, unlike DNA computing, self-assembly allows use of error-correction techniques (Adleman 2000; Winfree and Bekbolatov 2003). Clever tile designs can reduce or elinimate certain types of errors (Fujibayashi et al. 2009). Further, self-assembly systems allow for implementations that require a small, constant number of laboratory steps to execute, whereas DNA computing requires a larger number of steps. While some approaches have attempted to minimize this number of steps (Sakamoto et al. 2000), it still typically depends on the size of the input (Winfree et al. 1998).

1.2 Efficient algorithms for 3-*SAT*

3-*SAT* is a well known NP-complete problem of deciding whether a 3CNF Boolean formula is satisfiable. The naïve algorithms for solving 3-*SAT* explore the $\Theta(2^n)$ distinct assignments, for formulae with n distinct variables, checking if any one of the assignments satisfies the formula. While we are unaware of subexponential-time algorithms to solve NP-complete problems, there are algorithms that perform in exponential time but with a base smaller than 2. Woeginger (2003) provides a fairly complete survey of such deterministic algorithms, organized by the different techniques they employ. Here, I describe such an algorithm for 3-*SAT*. For my discussion, I define the $O^\star$ notation, which is similar to the O notation but ignores not only constant factors but also polynomial factors. Thus I will say $O^\star(m(x))$ for a complexity of the form $O(m(x) \cdot poly(x))$. The justification for this notation is that the exponential growth of $m(x)$ will dominate all polynomial factors for large x. For example, if f is a function such that $f(x) = O(1.4142^x x^4)$, then I write $f(x) = O^\star(1.4142^x)$. Note that the exponential term dominates and one could say $f(x) = O(1.4143^x)$ and forgo the $O^\star$ notation altogether; however, that would not most accurately describe the functions.

While the naïve algorithms explore each of the possible 2^n truth assignments to the n variables, a more intricate algorithm can explore a subset of those assignments by noting the following fact: if the Boolean formula contains the clause $(x_1 \vee \neg x_2 \vee x_3)$, then the algorithm need not explore any of the $2^{(n-3)}$ assignments with $x_1 = x_3 = FALSE$ and $x_2 = TRUE$ because this clause would not be satisfied by any of those assignments. Instead of trying each possible assignment for each variable, an algorithm could make the assignments based on the clauses. For example, if the first clause is of the form $(A \vee B \vee C)$, where A, B, and C are literals, then the assignments the system should explore are: 1. $A = TRUE$, 2. $A = FALSE$ and $B = TRUE$, and 3. $A = B = FALSE$ and $C = TRUE$.

Thus, deciding an n-variable m-clause Boolean formula can be done by recursively deciding three Boolean formulae: each with one fewer clause and with one, two, and three fewer variables, respectively. Thus if $T(n, m)$ denotes the time necessary to decide an n-variable m-clause Boolean formula, then $T(n, m) = O(1) + T(n-1, m-1) + T(n-2, m-1) + T(n-3, m-1)$. This recurrence has the closed form solution $T(n,m) = O^{\star}(1.8393^n)$ (Woeginger 2003). By examining the branching step, it is possible to improve the algorithm further to an $O^{\star}(1.6181^n)$ algorithm (Monien and Speckenmeyer 1985). Using quantitative analysis of the number of resulting 2-clauses from such branching improves the time complexity to $O^{\star}(1.5783^n)$ (Schiermeyer 1993). The champion algorithm using this technique achieves a time complexity of $O^{\star}(1.4963^n)$ (Kullmann 1997; Kullmann 1999), and other techniques result in even faster algorithms (Woeginger 2003). It is not my goal to explore the fastest such algorithm here, but rather to demonstrate that it is possible to implement one such complex algorithm using a tile system with a small tileset. I will thus concentrate on developing a tile system that follows the $O^{\star}(1.8393^n)$ algorithm, and argue that since the other algorithms are similar, it is possible to design tile systems for those algorithms as well.

The remainder of this paper is structured as follows: section 2 will define the tile assembly model and the concept of computation within that model. Section 3 will describe the tile system that implements the $O^{\star}(1.8393^n)$ algorithm for 3-*SAT* and prove the correctness and several other properties of that system. Finally, section 4 will summarize the contributions of this paper.

2 Tile assembly model

The tile assembly model (Winfree 1998a; Winfree 1998b; Rothemund and Winfree 2000) is a formal model of crystal growth. It was designed to model self-assembly of molecules such as DNA. It is an extension of a model proposed by Wang (Wang 1961). The model was fully defined by Rothemund and Winfree (Rothemund and Winfree 2000), and the definitions here are similar to those, and identical to the ones in (Brun 2007; Brun 2008a, b), but I restate them here for completeness and to assist the reader. Some of the ideas used in the definitions of nondeterministic computation originate from Winfree et al. (Winfree et al. 1998). I will first define the basics of the tile assembly model in section 2.1 and then the concept of computation within the model in section 2.2

While there exist a number of variations of the tile assembly model (Aggarwal et al. 2005; Demaine et al. 2008; Doty et al. 2010; Kao and Schweller 2006), some of which are more biologically accurate, I do not focus on them in this paper.

2.1 Tile assembly model definitions

Intuitively, the model has *tiles* or squares that stick or do not stick together based on various binding domains on their four sides. Each tile has a binding domain on its north, east, south, and west side, and may stick to another tile when the binding domains on the abutting sides of those tiles match and the total strength of all the binding domains on that tile exceeds the current temperature. The four binding domains define the type of the tile.

Formally, let Σ be a finite alphabet of binding domains such that $null \in \Sigma$. I will always assume $null \in \Sigma$ even when I do not specify so explicitly. A **tile** over a set of binding domains Σ is a 4-tuple $\langle \sigma_N, \sigma_E, \sigma_S, \sigma_W \rangle \in \Sigma^4$. A **position** is an element of $\mathbb{Z}^2$. The set of directions $D = \{N, E, S, W\}$ is a set of 4 functions from positions to positions, i.e., $\mathbb{Z}^2$ to $\mathbb{Z}^2$, such that for all positions (x, y), $N(x, y) = (x, y + 1)$, $E(x, y) = (x + 1, y)$, $S(x, y) = (x, y-1)$, $W(x, y) = (x-1, y)$. (Note that $N = S^{-1}$, $S = N^{-1}$, $E = W^{-1}$, and $W = E^{-1}$.) The positions (x, y) and (x', y') are neighbors iff $\exists d \in D$ such that $d(x, y) = (x', y')$. For a tile t, for $d \in D$, I will refer to $bd_d(t)$ as the binding domain of tile t on d's side. A special tile $empty = \langle null, null, null, null \rangle$ represents the absence of all other tiles.

A **strength function** $g\colon \Sigma \times \Sigma \to \mathbb{N}$, where g is commutative and $\forall \sigma \in \Sigma g(null, \sigma) = 0$, denotes the strength of the binding domains. It is common to assume that $g(\sigma, \sigma') = 0 \iff$ either $\sigma \neq \sigma'$ or $\sigma = \sigma' = null$. This simplification of the model implies that the abutting binding domains of two tiles have to match to bind. For the remainder of this paper, I will use $g = 1$ to mean $\forall \sigma \neq null, g(\sigma, \sigma) = 1$ and $\forall \sigma' \neq \sigma, g(\sigma, \sigma') = 0$.

Let T be a set of tiles containing the *empty* tile. A **configuration** of T is a function $A\colon \mathbb{Z} \times \mathbb{Z} \to T$. I write $(x, y) \in A$ iff $A(x, y) \neq empty$. A is finite iff there is only a finite number of distinct positions $(x, y) \in A$.

Finally, a **tile system** $\mathbb{S}$ is a triple $\langle T, g, \tau \rangle$, where T is a finite set of tiles containing *empty*, g is a strength function, and $\tau \in \mathbb{N}$ is the temperature.

If $\mathbb{S} = \langle T, g, \tau \rangle$ is a tile system and A is a configuration of some set of tiles $T' \subseteq \Sigma^4$ then a tile $t \in T$ can **attach** to A at position (x, y) and produce a new configuration A' iff:

- $(x, y) \notin A$, and
- $\sum_{d \in D} g(bd_d(t), bd_{d^{-1}}(A(d(x, y)))) \geq \tau$, and
- $\forall (u, v) \in \mathbb{Z}^2$, $(u, v) \neq (x, y) \Rightarrow A'(u, v) = A(u, v)$, and
- $A'(x, y) = t$.

That is, a tile can attach to a configuration only in *empty* positions and only if the total strength of the appropriate binding domains on the tiles in neighboring positions meets or exceeds the temperature τ. For example, if for all σ, $g(\sigma, \sigma) = 1$ and $\tau = 2$ then a tile t can attach only at positions with matching binding domains on the tiles in at least two adjacent positions.

Given a tile system $\mathbb{S} = \langle T, g, \tau \rangle$, a set of tiles Γ, and a **seed configuration** $S_0 \colon \mathbb{Z}^2 \to \Gamma$, if the above conditions are satisfied, one may attach tiles of T to S_0. Note that I allow the codomain of S_0 to be Γ, a set of tiles which may be different from T but should include the *empty* tile. Let $W_0 \subseteq \mathbb{Z}^2$ be the set of all positions where at least one tile from T can attach to S_0. For all $w \in W_0$ let U_w be the set of all tiles that can attach to S_0 at w. Let $\hat{S}_1$ be the set of all configurations S_1 such that for all positions $p \in S_0, S_1(p) = S_0(p)$ and for all positions $w \in W_0, S_1(w) \in U_w$ and for all positions $p \notin S_0 \cup W_0$, $S_1(p) = \mathit{empty}$. For all $S_1 \in \hat{S}_1$, I say that $\mathbb{S}$ **produces** S_1 on S_0 in one step. If $A_0, A_1, \ldots, A_n$ are configurations such that for all $i \in \{1, 2, \ldots, n\}$, $\mathbb{S}$ produces A_i on A_{i-1} in one step, then I say that $\mathbb{S}$ produces A_n on A_0 in n steps. When the number of steps taken to produce a configuration is not important, I will simply say $\mathbb{S}$ produces a configuration A on a configuration A' if there exists $k \in \mathbb{N}$ such that $\mathbb{S}$ produces A on A' in k steps. If the only configuration produced by $\mathbb{S}$ on A is A itself, then A is said to be a **final configuration**. If there is only one final configuration A produced by $\mathbb{S}$ on S_0, then $\mathbb{S}$ is said to produce a **unique** final configuration on S_0. Finally, if A is a final configuration produced by $\mathbb{S}$ on S_0 and n is the least integer such that A is produced by $\mathbb{S}$ on S_0 in n steps, then n is the **assembly time** of $\mathbb{S}$ on S_0 to produce A.

Note that a system may produce a unique final configuration, even though there exist non-unique sequences of attachments that continue growing at infinitum. Theoretically, such constructions pose no problem, though they may present problems to certain implementations of tile systems. In particular, the infinite configurations might consume all the tiles available for construction. It is possible to limit the definition of a unique final configuration to exclude systems that produce infinite configurations; however, such a restriction seems somewhat arbitrary and would only be helpful for some implementations of the tile assembly model. I choose not to restrict my definitions here, though I note that the systems presented in this paper do not suffer from this problem and produce no infinite configurations, and thus would satisfy the stricter definitions.

Winfree showed that the tile assembly model with $\tau = 2$ is Turing-universal (Winfree 1998b) by showing that a tile system can simulate Wang tiles (Wang 1961), which Berger showed to be universal (Berger 1966). Adleman et al. (Adleman et al. 2002b) showed that the tile assembly model with $\tau = 1$ is Turing-universal.

2.2 Computation in the tile assembly model

In (Brun 2007), I defined what it means to deterministically compute functions in the tile assembly model. In some implementations of tile assembly, many assemblies happen in parallel. In fact, it is often almost impossible to create only a single assembly. In (Brun 2008a), I extend the notion of computation in the tile assembly model to nondeterministic assemblies to leverage this inherent parallelism. For deterministic computation, I have defined a tile system to produce a unique final configuration on a seed if for all sequences of tile attachments, all possible final configurations are identical. In nondeterministic computation, different sequences of tile attachments attach different tiles in the same position. Intuitively, a system nondeterministically computes a function iff at least one of the possible sequences of tile attachments produces a final configuration which codes for the solution. Finally, in (Brun 2008b), I defined the notion of a tile system nondeterministically deciding a set.

Since a nondeterministic computation may have unsuccessful sequences of attachments, it is important to distinguish the successful ones. Further, in many implementations of the tile assembly model that would simulate all the nondeterministic executions at once, it is useful to be able to identify which executions succeeded and which failed in a way that allows selecting only the successful ones. For some problems, only an exponentially small fraction of the assemblies would represent a solution, and finding such an assembly would be difficult. For example, a DNA based crystal growing system would create millions of crystals, and only a few of them may represent the correct answer, while all others represent failed computations. Finding a successful computation by sampling the crystals at random would require time exponential in the input. Thus it would be useful to attach a special identifier tile to the crystals that succeed so that the crystals may be filtered to find the solution quickly. It may also be possible to attach the special identifier tile to solid support so that the crystals representing successful computations may be extracted from the solution. I thus specify one of the tiles of a system as an **identifier** tile that only attaches to a configuration that represents a successful sequence of attachments.

Often, computer scientists talk about deciding subsets of the natural numbers instead of computing functions. Deciding a subset of the natural numbers is synonymous with computing a function that has value 1 on arguments that are in the set, and value 0 on arguments that are not in the set. I adapt the definition of nondeterministically computing functions to nondeterministically deciding subsets of natural numbers. (There is also a direct analog of deciding sets deterministically, which I do not bother to formally specify here.) Let $\mathbb{N} = \mathbb{Z}_{\geq 0}$. Since for all constants $n \in \mathbb{N}$, the cardinalities of $\mathbb{N}^n$ and $\mathbb{N}$ are the same, one can encode an element of $\mathbb{N}^n$ as an element of $\mathbb{N}$. Thus it makes sense to talk about deciding subsets of $\mathbb{N}^n$. The below defined functions o_{s_m} can depend on the mapping of $\mathbb{N}^n \to \mathbb{N}$.

Let $v\colon \Gamma \cup T \to \{0, 1\}$ code each tile as a 1- or a 0-tile. Let $\hat{m} \in \mathbb{N}$ and let $\Omega \subseteq \mathbb{N}^{\hat{m}}$. For all $0 \leq m < \hat{m}$, let $o_{s_m}\colon \mathbb{N} \to \mathbb{Z}^2$ be injections. Let Δ be a set of (seed) configurations over Γ. Let the seed encoding functions $e_{s_m}\colon \Delta \to \mathbb{N}$ map a seed S to $\hat{m}$ numbers such that $e_{s_m}(S) = \sum_{i=0}^{\infty} 2^i v(S(o_{s_m}(i)))$ iff for no more than a constant number of (x, y) not in the union of the images of all o_{s_m}, $(x, y) \in S$. Let $\mathbb{S}$ be a tile system with T as its set of tiles, and let $r \in T$. Then I say that $\mathbb{S}$ **nondeterministically decides** a set Ω with identifier tile r iff for all $\vec{a} = \langle a_0, a_1, \ldots, a_{\hat{m}-1} \rangle \in \mathbb{N}^{\hat{m}}$ there exists a seed configuration S such that for all final configurations F that $\mathbb{S}$ produces on S, $r \in F(\mathbb{Z}^2)$ iff $\forall 0 \leq m < \hat{m}, e_{s_m}(S) = a_m$ and $\vec{a} \in \Omega$.

If for all $\hat{m} \in \mathbb{N}$, for all $0 \leq m < \hat{m}$, the o_{s_m} functions are allowed to be arbitrarily complex, the definition of computation in the tile assembly model is not very interesting because the computational intelligence of the system could simply be encoded in the o_{s_m} functions. For example, suppose h is the halting characteristic function (for all $a \in \mathbb{N}, h(a) = 1$ if the ath Turing machine halts on input a, and 0 otherwise) and o_{s_0} is such that the input a is encoded in some straight line if $h(a) = 1$ and in some jagged line otherwise. Then it would be trivial to design a tile system to solve the halting problem. Thus the complexities of the o_{s_m} functions need to be limited.

The problem of limiting the complexities is not a new one. When designing Turing machines, the input must be encoded on the tape and the theoreticians are faced with the exact same problem: an encoding that is too powerful could render the Turing machine capable of computing uncomputable functions. The common solution is to come up with a single straightforward encoding, e.g., for all $m \in \mathbb{N}$, converting the input element of $\mathbb{N}^m$ into an element of $\mathbb{N}$ via a mapping $\mathbb{N}^m \to \mathbb{N}$ and using the intuitive direct binary encoding of that element of $\mathbb{N}$ on the Turning machine tape for all computations (Sipser 1997). A similar approach is possible in the tile assembly model, requiring all systems to start with the input encoded the same way. In fact, it has been shown that such a definition conserves Turing universality of the tile systems (Winfree 1998b). However, the assembly time complexity of such systems may be adversely affected. In my definitions, I wish to give the system architect freedom in encoding the inputs for the sake of efficiency of computation; however, I restrict the o_{s_m} functions to be computable in linear time on a Turing machine. Thus these functions cannot add too much complexity-reducing power to the systems (the functions themselves cannot compute anything more complex than what linear-time algorithms can) while allowing the architects the freedom to place the inputs where they wish.

3 Solving 3-*SAT* efficiently with tiles

Implementing algorithms in the tile assembly model is not unlike implementing algorithms using Turing machines, or programming using a low-level language, such as assembly. The complexity of that process has led to only simple algorithms implemented into tile systems. Here, I propose the tile system $\mathbb{S}_{FS}$ (*FS* stands for "fast satisfiability") that implements the $O^{\star}(1.8393^n)$ algorithm for solving 3-*SAT*. The algorithm's running time implies that $\mathbb{S}_{FS}$ will create $O^{\star}(1.8393^n)$ distinct assemblies to decide an n-variable formula.

$\mathbb{S}_{FS}$ is a combination of several subsystems, each with a distinct job. Figure 1 shows the general placement of the distinct subsystems on a 2-D grid. The overall system will construct a right triangle, starting from region I, which encodes a Boolean formula ϕ. Region II will examine the eastmost clause of ϕ and determine which literals have not been assigned a value (at the start of the computation, there will always be 3 unassigned literals in each clause of a 3-*SAT* formula, but as the algorithm makes assignment decisions, clauses may have fewer such literals). Region III will make the nondeterministic decision on what assignments to make regarding the unassigned literals in the eastmost clause. Region IV will prepare the literals of the eastmost clause to be assigned by the decision made in region III, and region V will make those assignments. Region VI will simplify the rest of ϕ based on those assignments. At the top of region VI, the simplified ϕ, with one fewer clause, will emerge to serve as the input

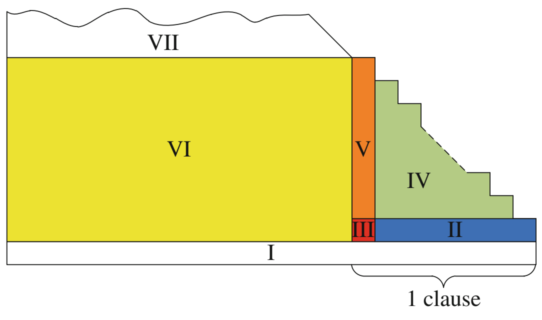


Fig. 1 A schematic of seven regions $\mathbb{S}_{FS}$ will use to decide 3-*SAT*

(like region I) for the remainder of the computation in region VII. That is, $\mathbb{S}_{FS}$ will operate recursively in Region VII on the simplified ϕ.

3.1 Notations and definitions

Let ϕ be a Boolean formula in conjunctive normal form. Let n be the number of distinct variables and m be the number of clauses in ϕ. For all $0 \leq \hat{m} < m$, I refer to the $\hat{m}$th clause as $c_{\hat{m}}$. For all $0 \leq \hat{n} < n$, I refer to the $\hat{n}$th variable as $x_{\hat{n}}$. A literal over a variable $x_{\hat{n}}$ is an element of $\{x_{\hat{n}}, \neg x_{\hat{n}}\}$. As is common, I assume that no clause of ϕ contains more than one literal over the same variable.

To assist the readability of this paper, I define the quantity v and a helper function $c: \mathbb{N} \to \mathbb{N}$. Let $v = \lceil \lg n \rceil + 1$. The intuition behind v is that each literal will require v tiles to encode it, $\lceil \lg n \rceil$ tiles to encode the index of the variable in binary, and 1 tile to encode whether or not the literal is the negation of that variable. For all $0 \leq \hat{m} < m$, let $c(\hat{m}) = \hat{m}(3v + 1) + 1$. Since each clause encoding includes the encodings of three literals and one extra clause-ending tile, the intuition behind c is that the $\hat{m}$th clause is encoded in columns $-c(\hat{m})$ through $-(c(\hat{m}+1) - 1)$. Note that for all $\hat{k} \in \{0, 1, 2\}$, the $\hat{k}$th literal of the $\hat{m}$th clause is encoded in columns $-(c(\hat{m}) + \hat{k}v)$ through $-(c(\hat{m}) + ((\hat{k}+1)v - 1))$.

Definition 1 (*General 3 CNF Boolean Formula*) For all m, for all n, for all n-variable, m-clause 3CNF Boolean formulae ϕ, ϕ is a *general* 3CNF Boolean formula iff each of the three literals of each clause either is identically a variable, is the negation of a variable, or is represented by *TRUE* or *FALSE* and no pair of literals within each clause are over the same variable.

Definition 2 (*General 3 CNF Boolean Formula Encoding*) For all general 3CNF Boolean formulae ϕ, an ordered sequence $s = \langle s_0, s_1, \ldots, s_z \rangle$ of elements of $\{\mathsf{x}, \neg\mathsf{x}, \mathsf{0}, \mathsf{0t}, \mathsf{0f}, 1, 1t, 1f, T, F, c\}$ is a *general encoding* of ϕ iff for all $0 \leq \hat{m} < m$:

- For all $0 \leq \hat{k} < 3$, either
  - the $\hat{k}$th literal of the $\hat{m}$th clause of ϕ is *FALSE*, and
    - $s_{c(\hat{m})+kv} \in \{\mathsf{x}, \neg\mathsf{x}\}$, and
    - for all $1 \leq i < v - 1$, $s_{c(\hat{m})+kv+i} \in \{\mathsf{0}, \mathsf{1}\}$, and
    - $s_{c(\hat{m})+kv+(v-1)} \in \{\mathsf{0f}, \mathsf{1f}, \mathsf{F}\}$, or
  - the $\hat{k}$th literal of the $\hat{m}$th clause of ϕ is *TRUE*, and
    - $s_{c(\hat{m})+kv} \in \{\mathsf{x}, \neg\mathsf{x}\}$, and
    - for all $1 \leq i < v - 1$, $s_{c(\hat{m})+kv+i} \in \{\mathsf{0}, \mathsf{1}\}$, and
    - $s_{c(\hat{m})+kv+(v-1)} \in \{\mathsf{0t}, \mathsf{1t}, \mathsf{T}\}$, or
  - the $\hat{k}$th literal of the $\hat{m}$th clause of ϕ is over some variable x_j, and
    - $s_{c(\hat{m})+kv} = \begin{cases} \mathsf{x} & \text{if the literal is } x_j \\ \neg\mathsf{x} & \text{if the literal is } \neg x_j, \text{ and} \end{cases}$
    - for all $1 \leq i < v$, $s_{c(\hat{m})+kv+i}$ is the ith bit of j, and
- $s_{c(\hat{m})+3v} = \mathsf{c}$.

The tile system $\mathbb{S}_{FS}$ will operate at temperature 2, and will use a fairly straightforward strength function g_{FS} over the set of binding domains $\Sigma_{FS} = \{null, \mathsf{t}, \mathsf{bt}, \mathsf{bbt}, \mathsf{ft}, \mathsf{fft}, fbt, bft, T, F, @, @^\star, 0, 1, 0t, 1t, 0f, 1f, x, \neg x, x^\star, \neg x^\star, c, \#, 0\#, 1\#, \#f, \#t, ::, 0:, 1:, 2:, 3:, 0:^\star, 1:^\star, 2:^\star, 1{:}1, 1{:}1^\star, 2{:}1, 2{:}1^\star, 3{:}1, 2{:}2, 2{:}2^\star, 3{:}2, 2{:}12, 2{:}12^\star, 3{:}12, \star\star, \star, |\}$. For the most part, g_{FS} will match two identical binding domains to 1, and two different binding domains to 0, with a few special wildcard binding domains that will map to 1 even with some unmatching domains. In other words, all attachments are either strength 0 or 1, and:

Intuitively, g_{FS} is such that:	Formally, $g_{FS}: \Sigma_{FS} \times \Sigma_{FS} \to \{0, 1\}$ such that:
null does not attach to anything,	For all $\sigma \in \Sigma_{FS}$, $g_{FS}(null, \sigma) = g_{FS}(\sigma, null) = 0$,
every binding domain attaches to itself,	For all $\sigma \in \Sigma_{FS} \setminus \{null\}$, $g_{FS}(\sigma, \sigma) = 1$,
# attaches to $\mathsf{0, 1, x, \neg x, 1f, 1t, 0f, 0t, T, F}, ^\star$, and @,	For all $\sigma \in \{\mathsf{0, 1, x, \neg x, 1f, 1t, 0f, 0t, T, F}, @^\star\}$, $g_{FS}(\#, \sigma) = g_{FS}(\sigma, \#) = 1$,
: : attaches to $\mathsf{0:, 1:, 2:, 1:1, 2:1, 2:2}$, and $\mathsf{2:12}$, and 2:12,	For all $\sigma \in \{\mathsf{0:, 1:, 2:, 1:1, 2:1, 2:2, 2:12}\}$, $g_{FS}(::, \sigma) = g_{FS}(\sigma, ::) = 1$,
#f attaches to $\mathsf{0f, 1f}$, and F,	For all $\sigma \in \{\mathsf{0f, 1f, F}\}$, $g_{FS}(\#\mathsf{f}, \sigma) = g_{FS}(\sigma, \#\mathsf{f}) = 1$,
#t attaches to $\mathsf{0t, 1t}$, and T,	For all $\sigma \in \{\mathsf{0t, 1t, T}\}$, $g_{FS}(\#\mathsf{t}, \sigma) = g_{FS}(\sigma, \#\mathsf{t}) = 1$,
0# attaches to $\mathsf{0, 0f}$, and $\mathsf{0t}$,	For all $\sigma \in \{\mathsf{0, 0f, 0t}\}$, $g_{FS}(\mathsf{0}\#, \sigma) = g_{FS}(\sigma, \mathsf{0}\#) = 1$,
1# attaches to $\mathsf{1, 1f}$, and $\mathsf{1t}$,	For all $\sigma \in \{\mathsf{1, 1f, 1t}\}$, $g_{FS}(\mathsf{1}\#, \sigma) = g_{FS}(\sigma, \mathsf{1}\#) = 1$,
$@^\star$ attaches to @ and *,	For all $\sigma \in \{@, ^\star\}$, $g_{FS}(@^\star, \sigma) = g_{FS}(\sigma, @^\star) = 1$,
and no other pairs of binding domains attach.	and for all other pairs $\sigma, \sigma' \in \Sigma_{FS}$, $g_{FS}(\sigma, \sigma') = g_{FS}(\sigma', \sigma) = 0$.

3.2 Clause examination (region II)

In this section, I define the tile system $\mathbb{S}_{EXAM}$, which will become the part of $\mathbb{S}_{FS}$ that will operate in region II, as denoted in Fig. 1. Since $\mathbb{S}_{FS}$ will operate on the first clause to fill in regions II through VI, and then recurse on the remaining simplified formula in region VII, for each clause there is a distinct copy of each region. Formally, for the $\hat{m}$th clause, Region II is part of a row, between positions $(-c(\hat{m}), c(\hat{m}))$ and $(-(c(\hat{m}) + 3v - 1), c(\hat{m}))$.

The goal of $\mathbb{S}_{EXAM}$ is to examine the first (eastmost) clause in the formula for the number of unassigned literals. Figure 2 shows the 37 tiles of T_{EXAM} that perform this examination. The binding domains on these tiles have some meaning. For example, the binding domain 2:1 means that out of the first two literals in a clause (indicated by the 2 before the :), only the 1st (indicated by the 1 after the :) is unassigned; the other literal must be *FALSE*, for reasons described below. I define the function p that maps clauses of general 3CNF Boolean formulae to binding domains. Let c be a clause of a general 3CNF Boolean formula. Then, if c contains the literal *TRUE*, then $p(c) = \mathsf{T}$; otherwise, the value of $p(c)$ is defined by Fig. 3.

$\mathbb{S}_{EXAM}$ will attach tiles just to the north of an encoding of clause c and will propagate that encoding one row north and make the west binding domain of the westmost tile be the value of $p(c)$. The clause examination lemma formally describes what $\mathbb{S}_{EXAM}$ does. Figure 4 shows an example execution of $\mathbb{S}_{EXAM}$ on the clause $(\neg x_2 \vee x_1 \vee x_0)$. Note that for this c, $p(c) = 2{:}12$, so the westmost tile's west binding domain is 2:12.

Lemma 1 (Clause Examination Lemma) For every clause c of a general 3CNF Boolean formula, for all $\alpha, \beta \in \mathbb{Z}$, let some seed S be such that $bd_W(S(\alpha, \beta + 1)) = \star\star$ and the sequence $s = \langle s_0, s_1, \ldots, s_{3v}, \mathsf{c}\rangle = \langle bd_N(S(\alpha - 1, \beta)), bd_N(S(\alpha - 2, \beta)), \ldots, bd_N(S(\alpha - 3v, \beta)), \mathsf{c}\rangle$ is a general Boolean encoding of c and $S(\alpha - 1, \beta + 1) = S(\alpha - 2, \beta + 1) = \cdots = S(\alpha - 3v, \beta + 1) = empty$. Let T_{EXAM} be the set of tiles described in Fig. 2. Then $\mathbb{S}_{EXAM} = \langle T_{EXAM}, g_{FS}, 2\rangle$ produces F on S such that:

c's literals			$p(c)$	$ac(c)$
first	second	third		
FALSE	*FALSE*	*FALSE*	3:	{F}
FALSE	*FALSE*	unassigned	3:1	{bbt}
FALSE	unassigned	*FALSE*	3:2	{bt}
FALSE	unassigned	unassigned	3:12	{bt, bft}
unassigned	*FALSE*	*FALSE*	2:	{t}
unassigned	*FALSE*	unassigned	2:1	{t, fbt}
unassigned	unassigned	*FALSE*	2:2	{t, ft}
unassigned	unassigned	unassigned	2:12	{t, ft, fft}

Fig. 3 For every clause c of a general 3CNF Boolean formula, $p(c) = \mathsf{T}$ and $ac(c) = \{@\}$ if c contains the literal *TRUE*, and otherwise, the values of $p(c)$ and $ac(c)$ are defined by this table. Sections 3.2 and 3.3 explain the meaning of the function $p(c)$ and $ac(c)$, respectively

– for all $0 \le i < 3v$, $bd_N(F(\alpha - 1 - i, \beta + 1)) =$

$$\begin{cases} @ & \textit{if } \exists j \le i \textit{ such that } s_j \in \{0t, 1t, \mathsf{T}\} \\ 0 & \textit{if } \nexists j \le i \textit{ such that } s_j \in \{0t, 1t, \mathsf{T}\} \textit{ and } s_i \in \{0, 0f, 1f\} \\ 1 & \textit{if } \nexists j \le i \textit{ such that } s_j \in \{0t, 1t, \mathsf{T}\} \textit{ and } s_i = 1 \\ \mathsf{x}^\star & \textit{if } \nexists j \le i \textit{ such that } s_j \in \{0t, 1t, \mathsf{T}\} \textit{ and } s_i = \mathsf{x} \\ \neg\mathsf{x}^\star & \textit{if } \nexists j \le i \textit{ such that } s_j \in \{0t, 1t, \mathsf{T}\} \textit{ and } s_i = \neg\mathsf{x}, \textit{ and} \end{cases}$$

– $bd_W(F(\alpha - 3v, \beta + 1)) = p(c)$.

Proof (Lemma 1) I will examine the operation of $\mathbb{S}_{EXAM}$ on the first, then second, and finally third clause of c.

1. I initially examine the first literal whose description is in columns $\alpha - 1$ through $\alpha - v$. Examine the tile that attaches in position $(\alpha - 1, \beta + 1)$. Because $bd_W(S(\alpha, \beta + 1)) = \star\star$, that tile must have the east binding domain $\star\star$. There are 2 such tiles. Note that (1) the north binding domains of these tiles are exactly the south binding domains with a $\star$ appended at the end, so the lemma holds for position $(\alpha - 1, \beta + 1)$, and (2) the west binding domain of both these tiles is 0:. (The binding domains that contain a number (v), followed by :, and then followed by a list of numbers (W) can be interpreted to mean that of the first v literals in this clause, the ones in W are unassigned. For example, 0: can be interpreted as the trivially true statement "of the first 0 variables, none are unassigned," and 2:12 can be interpreted as "of the first 2 variables, both the first and the second are unassigned.")

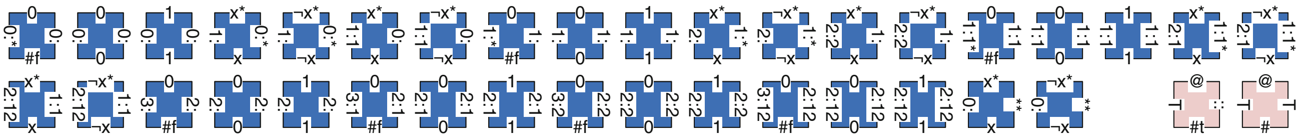


Fig. 2 The 37 tiles of T_{EXAM}

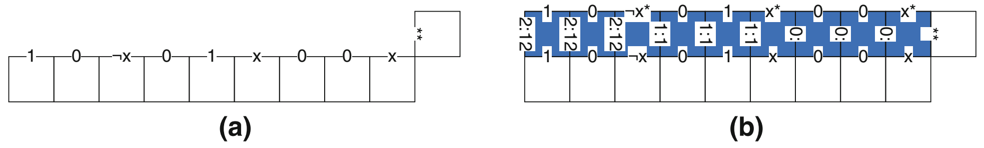


Fig. 4 An example execution of $\mathbb{S}_{EXAM}$. The seed (a) encodes the clause $(\neg x_2 \vee x_1 \vee x_0)$ and tiles from T_{EXAM} attach to examine the clause and find three unassigned literals (b)

Let $i = 1$. Because $bd_N(S(\alpha - 1 - i, \beta)) \in \{0, 1\}$, one of the tiles with the east binding domain 0: and the south binding domains 0 or 1 can attach there. Note that because the north binding domains are the same as the south binding domain on all these tiles, the lemma holds for position $(\alpha{-}1{-}i, \beta + 1)$. Further note that since the west binding domain of these tiles is 0; the same argument holds for $i = 2, 3, \ldots,$ as long as $bd_N(S(\alpha - 1 - i, \beta)) \in \{0, 1\}$. By the conditions on the lemma, if the first literal is unassigned, the argument holds through $i = v{-}1$. Otherwise, the argument holds through $i = v{-}2$.

If the first literal is *TRUE*, then $bd_N(S(\alpha - v, \beta)) \in \{\mathsf{T}, \mathsf{0t}, \mathsf{1t}\}$. Then the pink tile with the east binding domain : : must attach in position $(\alpha{-}v, \beta + 1)$. That tile's north binding domain is @, so the lemma holds for position $(\alpha{-}v, \beta + 1)$. Further, since the west binding domain of that tile is T, the other pink tile (the one with east and west binding domains T) will attach in positions $(\alpha{-}v{-}1, \beta + 1)$ through $(\alpha{-}3v, \beta + 1)$. Those tiles have the north binding domain @, so the lemma holds in all positions if the first literal is *TRUE*.

If the first literal is *FALSE*, then $bd_N(S(\alpha - v, \beta)) \in \{\mathsf{F}, \mathsf{0f}, \mathsf{1f}\}$. Then the tile with the #f south binding domain must attach in position $(\alpha{-}v, \beta + 1)$. That tile's north binding domain is 0, so the lemma holds for position $(\alpha{-}v, \beta + 1)$. Note that this tile's west binding domain is $0{:}^\star$.

Thus far, I have shown that the lemma holds in all cases up to position $(\alpha{-}v, \beta + 1)$.

2. Next, I examine the second literal whose description is in columns $\alpha{-}v{-}1$ through $\alpha{-}2v$. There are two possible cases:

   (a) If the first literal is *FALSE*, then the west binding domain of the tile that attached in position $(\alpha{-}v, \beta + 1)$ is $0{:}^\star$. Therefore one of the two tiles with that binding domain on their east sides and either the x or ¬x on their south sides must attach in position $(\alpha{-}v{-}1, \beta + 1)$. Note that (1) the north binding domains of these tiles are exactly the south binding domains with a * appended at the end, so the lemma holds for position $(\alpha{-}v{-}1, \beta + 1)$, and (2) the west binding domain of both these tiles is 1:. Let $i = 1$. Because $bd_N(S(\alpha - v - 1 - i, \beta)) \in \{0, 1\}$, one of the tiles with the east binding domain 1: and the south binding domains 0 or 1 can attach there. Note that because the north binding domains are the same as the south binding domain on all these tiles, the lemma holds for position $(\alpha{-}v{-}1{-}i, \beta + 1)$. Further note that since the west binding domain of these tiles is 1:, the same argument holds for $i = 2, 3, \ldots,$ as long as $bd_N(S(\alpha - v - 1 - i, \beta)) \in \{0, 1\}$. By the conditions on the lemma, if the second literal is unassigned, the argument holds through $i = v{-}1$. Otherwise, the argument holds through $i = v{-}2$.

   If the second literal is *TRUE*, then $bd_N(S(\alpha - 2v, \beta)) \in \{\mathsf{T}, \mathsf{0t}, \mathsf{1t}\}$. Then the pink tile with the east binding domain : : must attach in position $(\alpha{-}2v, \beta + 1)$. That tile's north binding domain is @, so the lemma holds for position $(\alpha{-}2v, \beta + 1)$. Further, since the west binding domain of that tile is T, the other pink tile (the one with east and west binding domains T) will attach in positions $(\alpha{-}2v{-}1, \beta + 1)$ through $(\alpha{-}3v, \beta + 1)$. Those tiles have the north binding domain @, so the lemma holds in all positions if the second literal is *TRUE*.

   If the second literal is *FALSE*, then $bd_N(S(\alpha - 2v, \beta)) \in \{\mathsf{F}, \mathsf{0f}, \mathsf{1f}\}$. Then the tile with the #f south binding domain must attach in position $(\alpha{-}2v, \beta + 1)$. That tile's binding domain is 0, so the lemma holds for position $(\alpha{-}2v, \beta + 1)$. Note that this tile's west binding domain is $1{:}^\star$.

   (b) If the first literal is unassigned, an argument holds that is very similar to the argument (2a). The only thing that has to change are some binding domains: Instead of $0{:}^\star$ substitute 0:, instead of 1: substitute 1:1, and instead of $1{:}^\star$ substitute $1{:}1^\star$.

□

Thus far, I have shown that the lemma holds in all cases up to position $(\alpha{-}2v, \beta + 1)$.

3. Finally, I examine the third literal whose description is in columns $\alpha{-}2v{-}1$ through $\alpha{-}3v$. There are four possible cases:

   (a) If the first and second literals are *FALSE*, then the west binding domain of the tile that attached in position $(\alpha{-}2v,\ \beta+1)$ is $\mathsf{1}{:}^\star$. Therefore one of the two tiles with that binding domain on their east sides and either the x or $\neg\mathsf{x}$ on their south sides must attach in position $(\alpha{-}2v{-}1,\ \beta+1)$. Note that (1) the north binding domains of these tiles are exactly the south binding domains with a * appended at the end, so the lemma holds for position $(\alpha{-}2v{-}1,\ \beta+1)$, and (2) the west binding domain of both these tiles is $\mathsf{2}{:}$.

Let $i=1$. Because $bd_N(S(\alpha-2v-1-i,\beta))\in\{\mathsf{0},\mathsf{1}\}$, one of the tiles with the east binding domain $\mathsf{2}{:}$ and the south binding domains $\mathsf{0}$ or $\mathsf{1}$ can attach there. Note that because the north binding domains are the same as the south binding domain on all these tiles, the lemma holds for position $(\alpha{-}2v{-}1{-}i,\ \beta+1)$. Further note that since the west binding domain of these tiles is $\mathsf{2}{;}$ the same argument holds for $i=2,3,\ldots,$ as long as $bd_N(S(\alpha-2v-1-i,\beta))\in\{\mathsf{0},\mathsf{1}\}$. By the conditions on the lemma, if the third literal is unassigned, the argument holds through $i=v{-}1$ and the west binding domain of the tile in position $(\alpha{-}3v,\ \beta+1)$ is $\mathsf{2}{:}$ so the lemma holds. Otherwise, the argument holds through $i=v{-}2$.

If the third literal is *TRUE*, then $bd_N(S(\alpha-3v,\beta))\in\{\mathsf{T},\mathsf{0t},\mathsf{1t}\}$. Then the pink tile with the east binding domain $::$ must attach in position $(\alpha{-}3v,\ \beta+1)$. That tile's binding domain is $\mathsf{@}$ and west binding domain is T, so the lemma holds.

If the third literal is *FALSE*, then $bd_N(S(\alpha-3v,\beta))\in\{\mathsf{F},\mathsf{0f},\mathsf{1f}\}$. Then the tile with the $\mathsf{\#f}$ south binding domain must attach in position $(\alpha{-}3v,\ \beta+1)$. That tile's binding domain is $\mathsf{0}$ and west binding domain is $\mathsf{3}{;}$ so the lemma holds.

(b) If the first literal is unassigned, but the second is *FALSE*, an argument holds that is very similar to the argument (1). The only thing that has to change are some binding domains: Instead of $\mathsf{1}{:}^\star$ substitute $\mathsf{1}{:}\,\mathsf{1}^\star$, instead of $\mathsf{2}{:}$ substitute $\mathsf{2}{:}\,\mathsf{1}$, and instead of $\mathsf{3}{:}$ substitute $\mathsf{3}{:}\,\mathsf{1}$.

(c) If the first literal is *FALSE*, but the second is unassigned, the substitutions are as follows: Instead of $\mathsf{1}{:}^\star$ substitute $\mathsf{1}{;}$ instead of $\mathsf{2}{:}$ substitute $\mathsf{2}{:}\,\mathsf{2}$, and instead of $\mathsf{3}{:}$ substitute $\mathsf{32}$.

(d) Finally, if both of the first two literals are unassigned, the substitutions are as follows: Instead of $\mathsf{1}{:}^\star$ substitute $\mathsf{1{:}1}$, instead of $\mathsf{2}{:}$ substitute $\mathsf{2{:}12}$, and instead of $\mathsf{3}{:}$substitute $\mathsf{3{:}12}$.

3.3 Assignment selection (region III)

In this section, I define the tile system $\mathbb{S}_{SELECT}$, which will become the part of $\mathbb{S}_{FS}$ that will operate in region III, as denoted in Fig. 1. Formally, for the $\hat{m}$th clause, Region III is the position $(-(c(\hat{m})+3v),c(\hat{m}))$.

The goal of $\mathbb{S}_{SELECT}$ is to nondeterministically select an assignment over the variables of the eastmost clause just as the $O^\star(1.8393^n)$ algorithm would, or, if there does not exist an assignment that can satisfy that clause, to halt the assembly. Thus, if $\mathbb{S}_{EXAM}$ finds that the clause has three unassigned literals, $\mathbb{S}_{SELECT}$ will pick either (1) the first literal to be true, or (2) the first literal to be false and the second to be true, or (3) the first two literals to be false and the third to be true. Alternatively, if $\mathbb{S}_{EXAM}$ finds that the clause has its first literal already assigned and the other two unassigned, $\mathbb{S}_{SELECT}$ will pick to ignore the first literal and either (1) the second literal to be true, or (2) the second literal to be false and the third to be true. The *clause assignment* defines the set of possible assignments for a clause c.

Definition 3 (Clause Assignment) For every clause c of a general 3CNF Boolean formula, a clause assignment $ac(c)$ is a proper subset of $\{\mathsf{t},\mathsf{bt},\mathsf{ft},\mathsf{bbt},\mathsf{bft},\mathsf{fbt},\mathsf{fft@},\mathsf{F}\}$, as defined by Fig. 3.

Figure 5 shows the 13 tiles of T_{SELECT} that perform the assignment selection. $\mathbb{S}_{SELECT}$ will attach a tile just to the west of the westmost tile attached by $\mathbb{S}_{EXAM}$ and nondeterministically select one of the possible assignments in $ac(c)$ as that tile's north binding domain. The assignment selection lemma formally describes what $\mathbb{S}_{SELECT}$ does. Figure 6 shows an example execution of $\mathbb{S}_{SELECT}$ on a seed that encodes that the clause in question has three unassigned literals. In this execution, $\mathbb{S}_{SELECT}$ chooses to assign the first literal to be *FALSE*, the second to be *TRUE*, and makes no assignment for the third literal at this time.

Lemma 2 (Literal Selection Lemma) *For every clause c of a general 3CNF Boolean formula, for all $\alpha,\beta\in\mathbb{Z}$, let some seed S be such that $bd_N(S(\alpha-1,\beta))=\mathsf{c}$ and $bd_W(S(\alpha,\ \beta+1))=p(c)$.*

Let T_{SELECT} be the set of tiles described in Fig. 5 and let $\mathbb{S}_{SELECT}=\langle T_{SELECT},g_{FS},2\rangle$. Then,

- *if c contains the TRUE literal, $\mathbb{S}_{SELECT}$ produces F on S such that $bd_W(F(\alpha-1,\beta+1))=\mathsf{@}$ and $bd_N(F(\alpha-1,\beta+1))=\mathsf{@}$,*
- *if all three of the literals in c are FALSE, then $\mathbb{S}_{SELECT}$ produces F on S such that $F=S$ (that is, no tiles attach to S), and*

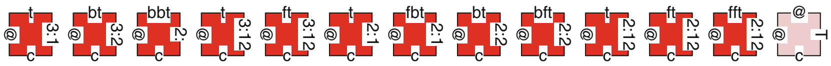


Fig. 5 The 13 tiles of T_{SELECT}

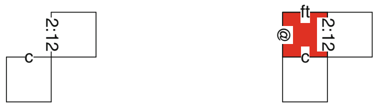


Fig. 6 An example execution of $\mathbb{S}_{SELECT}$. The seed (a) encodes the fact that a clause contains three unassigned literals and a tile from T_{SELECT} attaches to nondeterministically select the first literal to be *FALSE* and the second to be *TRUE*

– *otherwise, there exists j such that* $\mathbb{S}_{SELECT}$ *nondeterministically produces* $F_1, \ldots, F_j$ *on S such that for all* $0 \le i < j, bd_W(F_i(\alpha - 1, \beta + 1)) = @$ *and* $bd_N(F_i(\alpha - 1, \beta + 1)) \in ac(c)$, *and for each* $a \in ac(c)$, *there exists some* $0 \le i < j$ *such that* $bd_N(F_i(\alpha - 1, \beta + 1)) = a$.

Proof (Lemma 2) This lemma follows from direct analysis of the tiles in T_{SELECT}. Since region III consists of only a single position, examining the south and east binding domains of each tile reveals that every tile has the south binding domain c and

– if $bd_W(S(\alpha, \beta + 1)) = \mathsf{T}$ (which is only the case if c contains the *TRUE* literal), the tile that attaches in position $(\alpha - 1, \beta + 1)$ has the north and west binding domain @,
– if $bd_W(S(\alpha, \beta + 1)) = 3{:}$(which is only the case if all three of c's literals are *FALSE*), no tile in T_{SELECT} can attach in in position $(\alpha - 1, \beta + 1)$, and
– for all other possible c, there exist exactly as many tiles in T_{SELECT} with $p(c)$ as their east binding domains as there are elements of $ac(c)$, and for each element of $ac(c)$, exactly one such tile has that element as its north binding domain, and the west binding domains of all these tiles are @.

□

3.4 Clause rotation (region IV)

In this section, I define the tile system $\mathbb{S}_{ROTATE}$, which will become the part of $\mathbb{S}_{FS}$ that will operate in region IV, as denoted in Fig. 1. Formally, for the $\hat{m}$th clause, Region IV is right triangle with the right-angle corner on its southwest and the base in positions $(-c(\hat{m}), c(\hat{m}) + 1)$ through $(-(c(\hat{m}) + 3v - 1), c(\hat{m}) + 1)$.

The goal of $\mathbb{S}_{ROTATE}$ is to rotate a horizontally positioned clause encoding to be vertically positioned. This rotation later allows $\mathbb{S}_{SIMPLIFY}$ to simplify the formula. $\mathbb{S}_{ROTATE}$ will present the clause encoded in the north binding domains of part of its seed as the west binding domains of the completed right triangle. Figure 7 shows the 21 tiles of T_{ROTATE} that perform the rotation. The clause rotation lemma formally describes what $\mathbb{S}_{ROTATE}$ does. Figure 8 shows an example execution of $\mathbb{S}_{ROTATE}$ that rotates the clause $(\neg x_2 \vee x_1 \vee x_0)$.

Lemma 3 (Clause Rotation Lemma) *Let* T_{ROTATE} *be the set of tiles described in Fig.* 7 *and let* $\mathbb{S}_{ROTATE} = \langle T_{ROTATE}, g_{FS}, 2\rangle$. *Then for all* $\alpha, \beta \in \mathbb{Z}$, *for all* $\delta > 1$, *for every seed S such that:*

– *for all* $0 \le i < \delta - 1, bd_W(S(\alpha - i, \beta + 1 + i)) = \star$,
– *for all* $0 \le i < \delta, bd_N(S(\alpha - 1 - i, \beta)) \in \{0, 1, \mathsf{x}^\star, \neg x^\star, @\}$, *and*
– *for all other positions p in the square defined by the corners* (α, β) *and* $(\alpha - \delta, \beta + \delta)$, $S(p) = empty$,

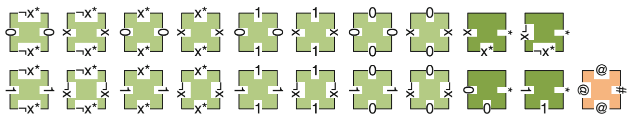
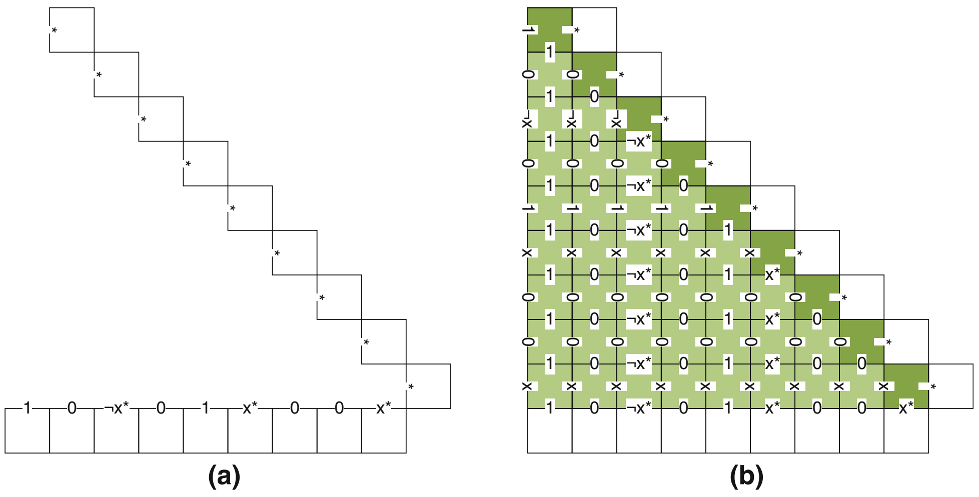


Fig. 7 The 21 tiles of T_{ROTATE}

Fig. 8 An example execution of $\mathbb{S}_{ROTATE}$. The seed (a) encodes the clause $(\neg x_2 \vee x_1 \vee x_0)$ and tiles from T_{ROTATE} attach to rotate that encoding (b)



$\mathbb{S}_{ROTATE}$ produces F on S such that:

- *if* $@ \in \{bd_N(S(\alpha-1,\beta)), bd_N(S(\alpha-2,\beta)), \ldots, bd_N(S(\alpha-\delta,\beta))\}$ *then for all* $0 \le i < \delta, bd_W(F(\alpha-\delta,\beta+1+i)) = @$, *and*
- *if* $@ \notin \{bd_N(S(\alpha-1,\beta)), bd_N(S(\alpha-2,\beta)), \ldots, bd_N(S(\alpha-\delta,\beta))\}$ *then for all* $0 \le i < \delta$,

$$bd_W(F(\alpha-\delta,\beta+1+i)) = \begin{cases} \mathsf{x} & \text{if } bd_N(S(\alpha-1-i,\beta)) = \mathsf{x}^\star \\ \neg\mathsf{x} & \text{if } bd_N(S(\alpha-1-i,\beta)) = \neg\mathsf{x}^\star \\ 0 & \text{if } bd_N(S(\alpha-1-i,\beta)) = 0 \\ 1 & \text{if } bd_N(S(\alpha-1-i,\beta)) = 1. \end{cases}$$

Proof (Lemma 3) First, note that T_{ROTATE} contains exactly one tile for every pair of possible east-south binding domain pairs that can occur as north and west binding domains of the seed, and of the tiles of T_{ROTATE}. Thus, the triangle formed by the seed will be filled in with tiles.

If there exists some $0 \le i < \delta$ such that $bd_N(S(\alpha-1-i,\beta)) = @$, then the orange tile with @ south and north binding domains will attach in column $\alpha-1-i$, as well as all columns to the west of that one, until column $\alpha-\delta$. Therefore the west binding domains of column $\alpha-\delta$ will be @.

Otherwise, I separate the green tiles of T_{ROTATE} (ones without @ binding domains) into two sets of tiles: $T_\star$ of tiles with the east * binding domain, and T_{REST} of tiles with other east binding domains. Because no tiles in T_{ROTATE} have * west binding domains, it is clear that the tiles of $T_\star$ can only attach in positions $(\alpha-i, \beta+i)$, for all $0 \le i < \delta$ (because the seed has * west binding domains in positions directly to the east of those). Note that for all tiles in $t \in T_\star$ have the property

$$bd_W(t) = \begin{cases} \mathsf{x} & \text{if } bd_S(t) = \mathsf{x}^\star \\ \neg\mathsf{x} & \text{if } bd_S(t) = \neg\mathsf{x}^\star \\ 0 & \text{if } bd_S(t) = 0 \\ 1 & \text{if } bd_S(t) = 1. \end{cases}$$

□

Now, notice that for each tile $t \in T_{REST}, bd_N(t) = bd_S(t)$ and $bd_W(t) = bd_E(t)$. Therefore, for all $0 \le i < \delta$, the value of $bd_N(S(\alpha-1-i, \beta+i))$ will be transferred northward by tiles from T_{REST}, until position $(\alpha-1-i, \beta+i+1)$, where a tile from $T_\star$ will transfer that value to its west binding domain, erasing the *, and then tiles from T_{REST} will transfer that value until position $(\alpha-\delta, \beta+i+1)$.

3.5 Assignment preparation (region V)

In this section, I define the tile system $\mathbb{S}_{PREP}$, which will become the part of $\mathbb{S}_{FS}$ that will operate in region V, as denoted in Fig. 1. Formally, for the $\hat{m}$th clause, Region V is part of a column between positions $(-(c(\hat{m})+3v), c(\hat{m})+1)$ and $(-(c(\hat{m})+3v), c(\hat{m})+3v)$.

The goal of $\mathbb{S}_{PREP}$ is to turn the assignment selected by $\mathbb{S}_{SELECT}$ into up to three literals that evaluate to *TRUE*. In other words, to apply the assignment to the clause. This application of the assignment is the final preparation before $\mathbb{S}_{SIMPLIFY}$ can simplify the formula. $\mathbb{S}_{PREP}$ will present the up to three literals as the west binding domains of the column in which it operates. Figure 9 shows the 36 tiles of T_{PREP} that perform the rotation. The assignment preparation lemma formally describes what $\mathbb{S}_{PREP}$ does. Figure 10 shows an example execution of $\mathbb{S}_{PREP}$ that prepares the clause $(\neg x_2 \vee x_1 \vee x_0)$ by using the decision that the first, from the right, literal (x_0) will be *FALSE*, the second literal (x_1) will be *TRUE*, and no assignment will be made for the third literal $(\neg x_2)$, at this time.

Lemma 4 (Assignment Preparation Lemma) *Let T_{PREP} be the set of tiles described in Fig. 9 and let $\mathbb{S}_{PREP} = \langle T_{PREP}, g_{FS}, 2\rangle$. Then for all $\alpha, \beta \in \mathbb{Z}$, for every clause c of a general 3CNF Boolean formula, for every $a \in ac(c)$, let $s = \langle s_0, s_1, \ldots, s_{3v-1}, \mathsf{c}\rangle$ be the encoding of c, and let the seed S be such that:*

- $bd_N(S(\alpha-1, \beta)) = a$,
- *for all* $0 \le i < 3v, bd_W(S(\alpha$
  - $bd_N(S(\alpha-1, \beta)) = a$,
  - *for all* $0 \le i < 3v, bd_W(S(\alpha, \beta+1+i)) = \begin{cases} @ & \text{if } a = @ \\ 0 & \text{if } s_i \in \{0, 0\mathsf{f}, 1\mathsf{f}\} \\ s_i & \text{otherwise,} \end{cases}$
  - *and for all other positions p from $(\alpha-1, \beta+1)$ through $(\alpha-1, \beta+3v)$, $S(p) = empty$.*

Then $\mathbb{S}_{PREP}$ produces F on S such that,

1. *if $a = @$, then for all $0 \le i < 3v, bd_W(F(\alpha-1, \beta+1+i)) = @$, and*
2. *if $a \notin \{@, \mathsf{F}\}$, for all $0 \le i < 3v$ $bd_W(F(\alpha-1, \beta+1+i)) = z_i$, where z_i is such that*

   (a) $z_0 = \begin{cases} s_0 & \text{if the 1st character of } a \text{ is } \mathsf{t} \\ \neg s_0 & \text{if the 1st character of } a \text{ is } \mathsf{f} \\ @ & \text{if the 1st character of } a \text{ is } \mathsf{b}\text{, and} \end{cases}$

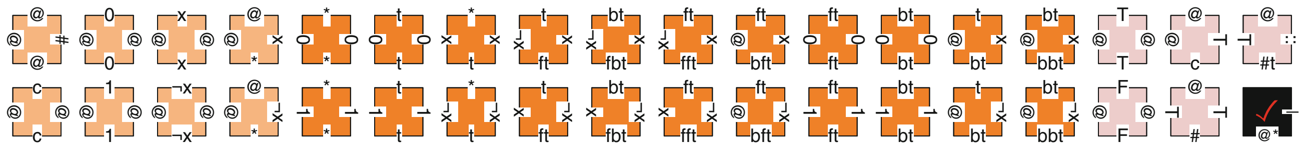


Fig. 9 The 36 tiles of T_{PREP}

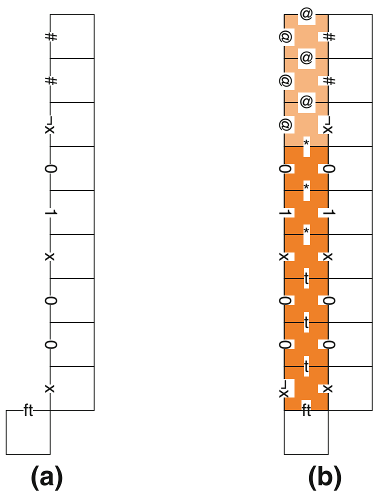


Fig. 10 An example execution of $\mathbb{S}_{PREP}$. The seed (a) encodes the clause $(\neg x_2 \vee x_1 \vee x_0)$ and the decision that the first, from the right, literal (x_0) will be *FALSE*, the second literal (x_1) will be *TRUE*, and no assignment will be made for the third literal $(\neg x_2)$, at this time. Tiles from T_{PREP} attach to make the appropriate assignment (b)

(b) *for all* $1 \le i < v$, $z_i = s_i$, *and*

(c) $$z_v = \begin{cases} s_{v+1} & \textit{if the 2nd character of } a \textit{ is } \mathsf{t} \\ \neg s_{v+1} & \textit{if the 2nd character of } a \textit{ is } \mathsf{f} \\ @ & \textit{if the 2nd character of } a \textit{ is} \\ & \mathsf{b} \textit{ or } a \textit{ has one character, and} \end{cases}$$

(d) *for all* $v+1 \le i < 2v, z_i =$
$$\begin{cases} @ & \textit{if } a \textit{ has one or two characters} \\ s_i & \textit{otherwise, and} \end{cases}$$

(e) $$z_{2v} = \begin{cases} s_{2v+1} & \textit{if } a \textit{ has a 3rd character} \\ & (\textit{which must be } \mathsf{t}) \\ @ & \textit{otherwise, and} \end{cases}$$

(f) *for all* $2v+1 \le i < 3v, z_i =$
$$\begin{cases} @ & \textit{if } a \textit{ has one or two characters} \\ s_i & \textit{otherwise.} \end{cases}$$

Proof (Lemma 4) The lemma has two cases, which I will prove separately.

1. If $a = @$, then $bd_N(S(\alpha-1,\beta)) = @$ and the orange tile with south and north binding domains @ will attach in positions $(\alpha-1,\beta+1),(\alpha-1,\beta+2),\ldots,(\alpha-1,\beta+1+3v)$, and therefore the west binding domains of tiles in F in all those positions will be @.
2. The second case contains six subcases, which I will prove sequentially, with each subcase depending on the previous one.

   (a) The only south binding domain of the tiles in T_{PREP} that has t as its first character is t. There are two tiles in T_{PREP} that have that binding domain on their south sides and either x or ¬x east binding domains. For all such tiles t, $bd_W(t) = bd_E(t)$, so $bd_W(F(\alpha-1, \beta+1)) = bd_W(S(\alpha, \beta+1)) = s_0$. Note that the north binding domain of these tiles is ★.
   The only south binding domains of the tiles in T_{PREP} that have f as their first characters are ft, fbt, and fft. There are six tiles in T_{PREP} that have those binding domains on their south sides and either x or ¬x east binding domains. For all such tiles $t, bd_W(t) = \neg bd_E(t)$, so $bd_W(F(\alpha-1, \beta+1)) = \neg bd_W(S(\alpha,\beta+1)) = \neg s_0$. Note that the north binding domain of these tiles are the same as their south binding domains, but with the first character dropped.
   The only south binding domains of the tiles in T_{PREP} that have b as their first characters are bt, bbt, and bft. There are six tiles in T_{PREP} that have those binding domains on their south sides and either x or ¬x east binding domains. For all such tiles $t, bd_W(t) = @$, so $bd_W(F(\alpha-1,\beta+1)) = @$. Note that the north binding domain of these tiles are the same as their south binding domains, but with the first character dropped.

   (b) Thus, the possible values of $bd_N(F(\alpha-1, \beta+1))$ are ★, t, ft, and bt. All tiles in $t \in T_{PREP}$ that have one of those binding domains on their south sides and either 0 or 1 east binding domains have $bd_W(t) = bd_E(t)$ and $bd_N(t) = bd_S(t)$. So for all $1 \le i < v$, $bd_W(F(\alpha-1, \beta+1+i)) = bd_W(S(\alpha, \beta+1+i)) = s_i$.

   (c) The first character of $bd_N(F(\alpha-1, \beta+v))$ is either ★ or the second character of a. If it is ★, the only tiles with ★ south binding domains and x or ¬x east binding domains have @ west and north binding domains, so $bd_W(F(\alpha-1,\beta+1+v)) = bd_N(F(\alpha-1,\beta+1+v)) = @$. Otherwise, if $bd_N(F(\alpha-1, \beta+v))$ is the second character of a, by the same argument as in (2a),
   $$bd_W(F(\alpha-1,\beta+1+v)) =$$
   $$\begin{cases} s_{v+1} & \text{if the 2nd character of a is } \mathsf{t} \\ \neg s_{v+1} & \text{if the 2nd character of a is } \mathsf{f} \\ @ & \text{if the 2nd character of a is } \mathsf{b}\text{, and} \end{cases}$$
   $$bd_N(F(\alpha-1,\beta+1+v)) =$$
   $$\begin{cases} ★ & \text{if the 2nd character of a is } \mathsf{t} \\ \mathsf{t} & \text{if the 2nd character of a is } \mathsf{f} \text{ or } \mathsf{b}. \end{cases}$$

   (d) If $bd_N(F(\alpha-1,\beta+1+v)) = @$, by the argument in (1), all north and west binding domains of tiles that attach above $(\alpha-1, \beta+1+v)$ will be @. Otherwise, by argument (2), for all $v+1 \le i < 2v$, $bd_W(F(\alpha-1, \beta+1+i)) = bd_W(S(\alpha, \beta+1+i)) = s_i$.

(e) The first character of $bd_N(F(\alpha-1,\ \beta+2v))$ is either $\star$ or the third character of a. If it is $\star$, the only tiles with $\star$ south binding domains and x or ¬x east binding domains have @ west and north binding domains, so $bd_W(F(\alpha-1,\beta+1+2v)) = bd_N(F(\alpha-1,\beta+1+2v)) =$ @. Otherwise, if $bd_N(F(\alpha-1,\ \beta+2v))$ is the third character of a (which can only be t), by the same argument as in (2a), $bd_W(F(\alpha-1,\ \beta+1+2v)) = s_{v+1}$, and $bd_N(F(\alpha-1,\beta+1+2v)) = \star$.

(f) If $bd_N(F(\alpha-1,\beta+1+2v)) =$ @, by the argument in (1), all north and west binding domains of tiles that attach above $(\alpha-1,\ \beta+1+2v)$ will be @. Otherwise, by argument (2), for all $2v+1 \le i < 3v$, $bd_W(F(\alpha-1,\ \beta+1+i)) = bd_W(S(\alpha,\ \beta+1+i)) = s_i$.

3.6 Formula simplification (region VI)

In this section, I define the tile system $\mathbb{S}_{SIMPLIFY}$, which will become the part of $\mathbb{S}_{FS}$ that will operate in region VI, as denoted in Fig. 1. Formally, for the $\hat{m}$th clause, Region VI is the rectangle defined by the corner positions $(-c(\hat{m}+1), c(\hat{m})+1)$ and $(-(c(\hat{m})-1), c(\hat{m})+3v)$.

The goal of $\mathbb{S}_{SIMPLIFY}$ is to simplify the formula by replacing instances of the up to three literals prepared by $\mathbb{S}_{PREP}$ with *TRUE* and negations of those literals with *FALSE*. $\mathbb{S}_{SIMPLIFY}$ will present an encoding of the simplified formula as the north binding domains of the rectangle in which it operates. Figure 11 shows the 63 tiles of $T_{SIMPLIFY}$ that perform the simplification. The formula simplification lemma formally describes what $\mathbb{S}_{SIMPLIFY}$ does. Figure 12 shows an example execution of $\mathbb{S}_{SIMPLIFY}$ that simplifies the formula $(\neg x_3 \vee \neg x_2 \vee x_0) \wedge (x_3 \vee x_2 \vee \neg x_1)$ by using the assignment $x_0 = FALSE$ and $x_1 = TRUE$.

Lemma 5 (Formula Simplification Lemma) *Let $T_{SIMPLIFY}$ be the set of tiles described in Fig. 11 and let $\mathbb{S}_{SIMPLIFY} = \langle T_{SIMPLIFY}, g_{FS}, 2\rangle$. Then for all $\alpha,\beta \in \mathbb{Z}$, for every literal ℓ, for every general 3CNF Boolean formula ϕ, for every encoding $s = \langle s_0, s_1, \ldots, s_z\rangle$ of ϕ, let the seed S be such that:*

- for all $0 \le i \le z$, $bd_N(S(\alpha-1-i,\ \beta)) = s_i$, and
- if $\ell \in \{TRUE, FALSE\}$,
  - $bd_W(S(\alpha,\beta+1)) =$ @, and
  - for all $1 \le j < v, bd_W(S(\alpha,\beta+1+j)) \in \{$0, 1, @$\}$, or
- if $\ell \notin \{TRUE, FALSE\}$,
  - $bd_W(S(\alpha,\beta+1)) = \begin{cases} \text{x} & \textit{if } \ell \textit{ is identically a variable} \\ \neg\text{x} & \textit{if } \ell \textit{ is the} \end{cases}$ *negation of a variable, and.*
  - for all $1 \le j < v$, $bd_W(S(\alpha,\ \beta+1+j))$ is the jth bit of ℓ's variable's index.

Then $\mathbb{S}_{SIMPLIFY}$ produces F on S such that, if ϕ' is identical to ϕ except that wherever ℓ appears in ϕ, it is replaced with *TRUE* in ϕ', and wherever $\neg\ell$ appears in ϕ, it is replaced with *FALSE* in ϕ', (if $\ell \in \{TRUE, FALSE\}$ then $\phi' = \phi$), then there exists an encoding $s' = \langle s'_0, s'_1, \ldots, s'_z\rangle$ of ϕ' such that for all $0 \le i \le z$, $bd_N(F(\alpha-1-i,\ \beta+v)) = s'_i$.

I now define two equality operators over binding domains: $\approx$ and =.

Definition 4 Two binding domains are $\approx$ if they are identical after removing the characters t and f from them. For example, 0 $\approx$ 0t $\approx$ 0f $\not\approx$ 1f.

Definition 5 Two binding domains are = if they are identical, or if they are both $\in \{$0f, 1f, F$\}$ or if they are both $\in \{$0t, 1t, T$\}$. For example, 1 = 1, 0f = F, but 0f $\neq$ 0.

Proof (Lemma 5) $\mathbb{S}_{SIMPLIFY}$ will operate within the $z \times v$ rectangle defined by S. That rectangle can be broken up into $v \times v$ squares. A set of three horizontally-adjacent $v \times v$ squares make up the representation of a clause of ϕ.

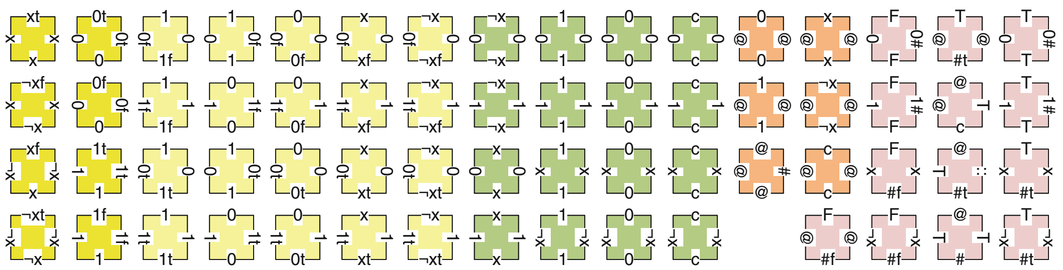


Fig. 11 The 63 tiles of $T_{SIMPLIFY}$

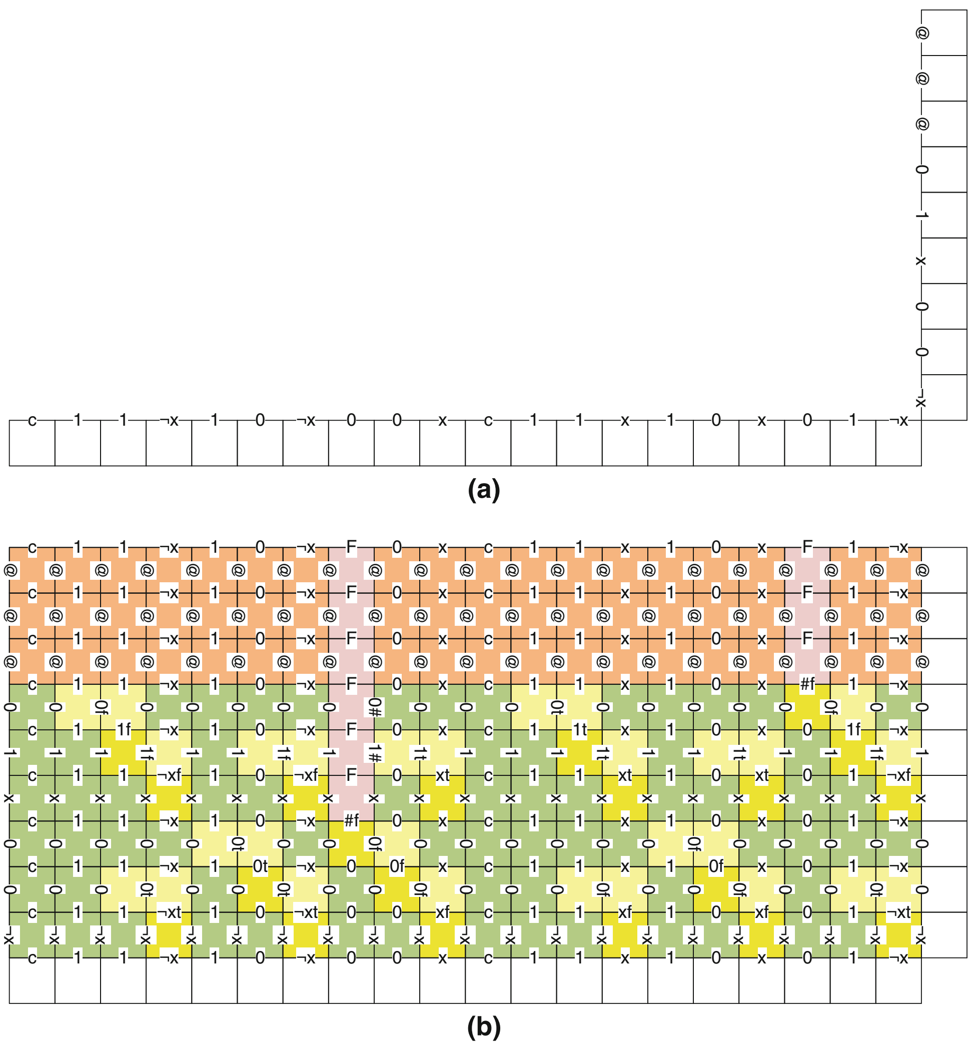


Fig. 12 An example execution of $\mathbb{S}_{SIMPLIFY}$. The seed (a) encodes the formula $(\neg x_3 \vee \neg x_2 \vee x_0) \wedge (x_3 \vee x_2 \vee \neg x_1)$ and an assignment of $x_0 = FALSE$ and $x_1 = TRUE$. Tiles from $T_{SIMPLIFY}$ attach to make the appropriate assignments and simplify the formula (b)

These $3v \times v$ rectangles are separated by $1 \times v$ rectangles (the columns with the c north binding domains). The tiles of $T_{SIMPLIFY}$ have the following special property: for each tile $t \in T_{SIMPLIFY}$, $bd_W(t) \approx bd_E(t)$. Therefore, as tiles attach to fill the $z \times v$ rectangle, for each row $\beta + j$, the east and west binding domains of all the tiles in that row will $\approx bd_W(S(\alpha, \beta + j))$. As I will show later, for each $v \times v$ square, the binding domains directly to the east of that square will be identical to those west binding domains of the seed. This allows me to argue that as long as the entire $z \times v$ rectangle is filled with tiles, each $v \times v$ square can attach tiles independently of what happens in other $v \times v$ squares. Therefore, to prove the lemma, it is sufficient to demonstrate the following fact for a single $v \times v$ square: Without loss of generality, let the square in question be the eastmost square, defined by corner positions $(\alpha{-}1, \beta + 1)$ and $(\alpha{-}v, \beta + v)$. Let ℓ' be the literal encoded by $\langle bd_N(S(\alpha - 1, \beta)), bd_N(S(\alpha - 2, \beta)), \ldots, bd_N(S(\alpha - v, \beta))\rangle$.

1. If $\ell \in \{TRUE, FALSE\}$, then
   - for all $0 \leq j < v$, $bd_N(F(\alpha{-}1{-}j, \beta + v)) = bd_N(S(\alpha{-}1{-}j, \beta))$.
2. If $\ell \notin \{TRUE, FALSE\}$, then
   - if $\ell = \ell'$ (respectively, $\ell = \neg\ell'$), then $\langle bd_N(F(\alpha - 1, \beta + v)), bd_N(F(\alpha - 2, \beta + v)), \ldots, bd_N(F(\alpha - v, \beta + v))\rangle$ encodes *TRUE* (respectively, *FALSE*), and
   - if $\ell \notin \{\ell', \neg\ell'\}$, for all $0 \leq j < v$, $bd_N(F(\alpha{-}1{-}j, \beta + v)) = bd_N(S(\alpha{-}1{-}j, \beta))$.

□

I now prove the above statement.

1. Suppose $\ell \in \{TRUE, FALSE\}$. In $T_{SIMPLIFY}$, every tile with an east binding domain @ has the west binding domain @, and its north binding domain = its south binding domain. For every north binding domain of the seed and the tiles in $T_{SIMPLIFY}$, there is exactly one tile with that south binding domain and @ east binding

domain. Therefore, these tiles will attach to the west of position $(\alpha, \beta + 1)$, all the way to $(\alpha - v, \beta + 1)$, and the north binding domains of that row will be = to the north binding domains of the seed row below it. Because $\ell \in \{TRUE, FALSE\}$, for all $1 \le j < v, bd_W(S(\alpha, \beta + 1 + j)) \in \{0, 1, @\}$. If that binding domain is @, the same argument as above holds. Otherwise, the green tiles with 0 and 1 east and west binding domains and the same properties as the above-mentioned @ tiles will attach to the west, propagating the seed's binding domains one row up at a time. Therefore, for all $0 \le j < v$, $bd_N(F(\alpha - 1 - j, \beta + v)) = bd_N(S(\alpha - 1 - j, \beta))$.

2. Suppose $\ell \notin \{TRUE, FALSE\}$.

   – Suppose also $\ell' \notin \{TRUE, FALSE\}$. In position $(\alpha - 1, \beta + 1)$, both the east and south binding domains contain x (because the west and north binding domains of the appropriate seed tiles contain x). Therefore, one of the bright yellow tiles will attach there. If the binding domains match identically, that binding domain is replicated on the north side of the tile, with an extra t tag. Otherwise, the binding domains are negations of each other, and the south binding domain in replicated on the north side of the tile, with an extra f tag. The light yellow tiles are the only ones with t and f tags on their south binding domains and east binding domains $\in \{0, 1\}$, so one of them must attach above the bright yellow tile. Note that these tiles move the t and f tag to their west binding domain. To the west of that tile, in position $(\alpha - 2, \beta + 2)$, if the corresponding row's west binding domain $\approx$ the corresponding column's north binding domain, then a bright yellow tile attaches and carries the t and f tag to the north. Otherwise, a light yellow tile attaches and removes all such tags. The result is that for all $1 \le j \le v$, as long as the first j bits of ℓ are equal to the first j bits of ℓ', the t and f tags are propagated northward and westward. Otherwise, these tags are removed. The rest of the square is filled with green tiles with no labels. Thus, if $\ell = \ell'$ then the tile in position $(\alpha - v, \beta + v)$ will have a north binding domain with the t tag, if $\ell = \neg\ell'$ then the tile in position $(\alpha - v, \beta + v)$ will have a north binding domain with the f tag, and ℓ and ℓ' are over different variables, the tile in position $(\alpha - v, \beta + v)$ will have a north binding domain with no tags. No other tiles with exposed north and west binding domains can have any tags. Therefore, if $\ell = \ell'$, the north binding domains of the tiles in row $\beta + v$ replace ℓ' with *TRUE*, if $\ell = \neg\ell'$, the north binding domains of the tiles in row $\beta + v$ replace ℓ' with *FALSE*, and otherwise ℓ is left unchanged. (Note that in the dth column, the west side's t and f tags can only occur in the $(d + 1)$st column, and since the $(v + 1)$st row does not get filled, there are now tags in the west binding domains of the westmost tiles of each row of the $v \times v$ square, confirming the earlier claim that for each $v \times v$ square, the binding domains directly to the east of that square will be identical to those west binding domains of the seed.)

   – Suppose $\ell' \in \{TRUE, FALSE\}$. The fact that $\ell' \in \{TRUE, FALSE\}$ does not impact the east $v - 1$ columns of the assembly because the encoding of *TRUE* and *FALSE* literals is only evident in the last element of the sequence being $= \mathsf{T}$ and $= \mathsf{F}$, respectively. Thus the exact same process will take place in the east $v - 1$ columns as described above; however, that process is almost irrelevant (other than the fact that those columns get filled with tiles) because what happens in column $(\alpha - v)$ "overrides" the actions of the columns to the east. If $\ell' = TRUE$ (respectively, *FALSE*), then $bd_N(S(\alpha - v, \beta)) = \mathsf{T}$ (respectively, F). Then the pink tiles with north binding domains T (respectively, F) will attach in positions $(\alpha - v, \beta + 1)$ through $(\alpha - v, \beta + v)$, ensuring that $bd_N(F(\alpha - v, \beta + v)) = \mathsf{T}$ (respectively, F), thus preserving the value of ℓ'.

□

3.7 Solving 3-*SAT*

Thus far, I have described five tile systems: $\mathbb{S}_{EXAM}$, $\mathbb{S}_{SELECT}$, $\mathbb{S}_{ROTATE}$, $\mathbb{S}_{PREP}$, and $\mathbb{S}_{SIMPLIFY}$ that I intend to use to solve 3-*SAT*. These systems will operate in regions II, III, IV, V, and VI in Fig. 1, respectively. The tile system $\mathbb{S}_{FS}$ combines these five systems to select a truth assignment for the first (eastmost) clause of a Boolean formula ϕ, simplify the rest of ϕ based on that assignment, and recurse (in region VII) on the simplified ϕ with one fewer clause. $\mathbb{S}_{FS}$ will nondeterministically create only $O^\star(1.8393^n)$ distinct assemblies to decide whether ϕ is satisfiable. Note that there are 147 distinct tiles that $\mathbb{S}_{FS}$ uses (the distinct tiles described so far and summarized in Fig. 15), and that each nondeterministic assembly assembles in time linear in the size of the input.

I will use the 8 tiles in Γ_{FS}, shown in Fig. 13, to encode the input ϕ. Informally, I will encode the formula's literals in row 0, such that the literals of each clause are together and place the special clause tile to the west of each clause. I will place the tiles with ⋆⋆ and ⋆ west binding domains on

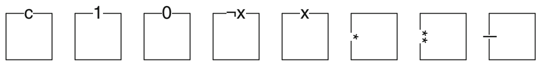


Fig. 13 The 8 tiles of Γ_{FS} used to encode inputs to $\mathbb{S}_{FS}$

a diagonal to the north and west of the ϕ, and I will place the tile with | west binding domain as the northmost and westmost tile in the diagonal. I define the seed more formally in the proof of Theorem 1. Figure 14 shows the seed $S_{FS\phi}$ that encodes the 3-variable 3-clause Boolean formula $(\neg x_3 \vee \neg x_2 \vee x_0) \wedge (x_3 \vee x_2 \vee \neg x_1) \wedge (\neg x_2 \vee x_1 \vee x_0)$.

Figure 15 shows the 147 tiles of T_{FS} that solve 3-*SAT*. Note that $T_{FS} = T_{EXAM} \cup T_{SELECT} \cup T_{ROTATE} \cup T_{PREP} \cup T_{SIMPLIFY}$. Figure 16 shows a sample execution of $\mathbb{S}_{FS}$ on the seed from Fig. 14. This execution assigns $x_0 = FALSE$ and $x_1 = TRUE$ in the first recursive step of the algorithm and then assigns $x_2 = FALSE$ and $x_3 = TRUE$ in the second step. In the third step, the algorithm finds that the third clause is already satisfied. Because this particular execution selected an assignment that satisfied the entire formula, the black ✓ tile is attached in the northwest corner.

The ✓ tile can only attach if the set of nondeterministic decisions made by the execution of $\mathbb{S}_{FS}$ results in a satisfying assignment. It is possible for the set of decisions to result in a case in which one or more clauses cannot be satisfied, halting the assembly and preventing the ✓ tile from attaching. Figure 17 shows an example execution that fails to attach the ✓ tile. In this execution, the decision made on the eastmost clause ($x_0 = x_1 = x_2 = FALSE$) prevents the westmost clause ($x_2 \vee x_1 \vee x_0$) from being satisfied, which prevents the ✓ tile from attaching.

Theorem 1 *Let $T_{FS} = T_{EXAM} \cup T_{SELECT} \cup T_{ROTATE} \cup T_{PREP} \cup T_{SIMPLIFY}$ (the set of tiles described in Fig. 15). Then $\mathbb{S}_{FS} = \langle T_{FS}, g_{FS}, 2\rangle$ nondeterministically decides 3-SAT with the black ✓ tile from T_{PREP} as the identifier tile.*

Proof (Theorem 1) Let Γ_{FS} be the set of tiles in Fig. 13. Let the seed S_{FS} be as follows:

- *For all $0 \le \hat{m} < m, S_{FS}(-(c(\hat{m}) - 1), c(\hat{m})) = \gamma_{\star\star}$, where $\gamma_{\star\star}$ is the tile in Γ_{FS} with the west binding domain $\star\star$.*
- *For all $0 \le \hat{m} < m$, for all $0 \le i < 3v-1$, $S_{FS}(-c(\hat{m}) - (i-1), c(\hat{m}) + (i+1)) = \gamma_{\star}$, where $\gamma_{\star}$ is the tile in Γ_{FS} with the west binding domain $\star$.*
- *$S_{FS}(-(c(m) - 2), c(m)) = \gamma_{|}$, where $\gamma_{|}$ is the tile in Γ_{FS} with the west binding domain |.*
- For all $0 \le \hat{m} < m$, for all $0 \le \hat{k} < 3$, $S_{FS}(-c(\hat{m}) - \hat{k}v, 0) = \gamma_{\neg}$, where if the $\hat{k}$th literal in the $\hat{m}$th clause is identically some variable, then $\gamma_{\neg}$ is the tile in Γ_{FS} with the north binding domain x and if the $\hat{k}$th literal in the $\hat{m}$th clause is the negation of some variable, then $\gamma_{\neg}$ is the tile in Γ_{FS} with the north binding domain ¬x.
- For all $0 \le \hat{m} < m$, for all $0 \le \hat{k} < 3$, for all $1 \le i < v, S_{FS}(-c(\hat{m}) - \hat{k}v - i, 0) = \gamma_z$, where if w is such that the $\hat{k}$th literal in the $\hat{m}$th clause is either x_w or $\neg x_w$, then z is the ith bit of w ($z = \lfloor \frac{w}{2^i} \rfloor \bmod 2$) and γ_z is the tile in Γ_{FS} with the north binding domain z.
- For all $0 \le \hat{m} < m, S_{FS}(-c(\hat{m}) - 3v, 0) = \gamma_c$, where γ_c is the tile in Γ_{FS} with the north binding domain c.
- And for all other positions (v, w), $S_{FS}(v, w) = empty$.

Observe that the sequence $\langle bd_N(S(-1,0)), bd_N(S(-2,0)), \ldots, bd_N(S(-(c(m)-1),0))\rangle$ is a general Boolean formula encoding of ϕ. Let $c = (\ell_1 \vee \ell_2 \vee \ell_3)$ be the first (eastmost) clause of ϕ. Observe that $bd_W(S(0,1)) = \star\star$. Therefore, by the clause examination lemma (Lemma 1), the tiles from T_{EXAM} will attach to produce $p(c)$ as the west binding domain of the tile in position ($-3v$, 1), and precisely the north binding domains of tiles in positions (-1, 1) through ($-3v$, 1) that are the preconditions of the clause rotation lemma (Lemma 1). By the literal selection lemma (Lemma 2), either

1. $\ell_1 = \ell_2 = \ell_3 = FALSE$ and no tiles will attach in position ($-3v-1$, 1), and therefore in no other position west and north of ($-3v-1$, 1), or
2. a tile from T_{SELECT} will attach in position ($-3v-1$, 1) to nondeterministically produce, for each $a \in ac(c)$, a configuration with a as the north binding domain and @ as the west binding domain of the tile in that position.

If a tile does attach in position ($-3v-1$, 1), the orange and pink tiles from $T_{SIMPLIFY}$ with east and west binding domains @ will attach to the west of position ($-3v-1$, 1), all the way to position ($-(c(m)-1)$, 1). Note that for each such tile t, $bd_N(t) = bd_S(t)$, so they will propagate the encoding of the rest of ϕ from row 0 up to row 1.

By the clause rotation lemma (Lemma 3), tiles from T_{ROTATE} will attach in the right triangle above positions (-1, 2) through ($-3v$, 2) to produce an encoding of c in the west binding domains of the tiles in positions ($-3v$, 2) through ($-3v$, $3v + 1$).

$$\text{Let } \ell'_1 = \begin{cases} \ell_1 & \text{if the 1st character of } a \text{ is } \mathsf{t} \\ \neg\ell_1 & \text{if the 1st character of } a \text{ is } \mathsf{f} \\ null & \text{if the 1st character of } a \text{ is } \mathsf{b}. \end{cases}$$

Let ℓ'_2 and ℓ'_3 be defined similarly, based on ℓ_2 and ℓ_3 and the 2nd and 3rd characters of a, respectively (where if a does not have a second or third character, that character can be assumed to be b). By the assignment preparation lemma (Lemma 5), the tiles of T_{PREP} will attach such that the west binding domains of the tiles in positions ($-3v-1$, 2) through ($-3v-1$, $v + 1$) encode ℓ'_1, the west binding domains of the tiles in positions ($-3v-1$, $v + 2$) through

$(-3v-1, 2v+1)$ encode ℓ'_2, and the west binding domains of the tiles in positions $(-3v-1, 2v+2)$ through $(-3v-1, 3v+1)$ encode ℓ'_3, (where *null* is encoded by @ followed by a series of elements of @, 0, 1).

Let ψ be identical to ϕ except without the first (east-most) clause. By the formula simplification lemma (Lemma 5) tiles from $\mathsf{T}_{SIMPLIFY}$ will attach in the rectangle defined by the corner positions $(-3v-2, 2)$ and $(-(c(m)-1), v+1)$. The sequence of north binding domains of the tiles attached in positions $(-3v-2, v+1)$ through $(-(c(m)-1), v+1)$ will encode ψ', where ψ' is identical to ψ, except every instance of ℓ'_1 is replaced with *TRUE* and every instance of $\neg\ell'_1$ is replaced with *FALSE*. By a second application of the formula simplification lemma (Lemma 5), tiles from $\mathsf{T}_{SIMPLIFY}$ will attach in the rectangle defined by the corner positions $(-3v-2, v+2)$ and

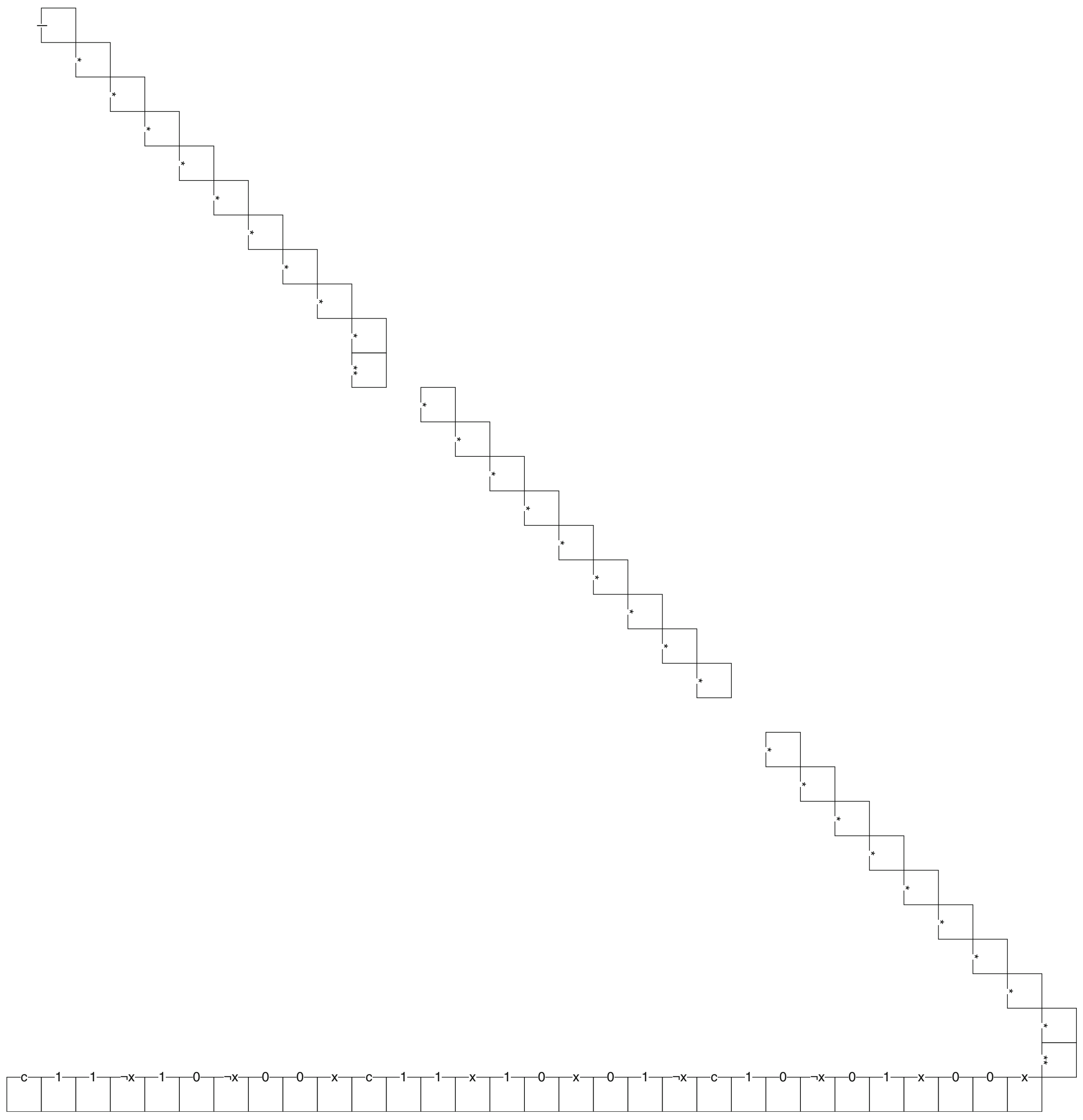


Fig. 14 An example seed used by $\mathbb{S}_{FS}$ uses the tiles from Γ_{FS} to encode the Boolean formula $(\neg x_3 \vee \neg x_2 \vee x_0) \wedge (x_3 \vee x_2 \vee \neg x_1) \wedge (\neg x_2 \vee x_1 \vee x_0)$

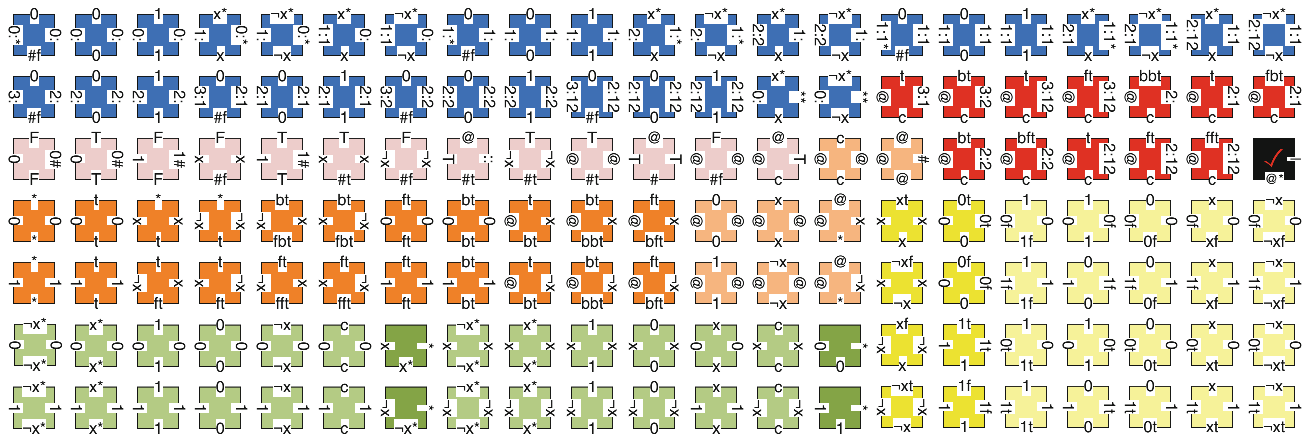


Fig. 15 The 147 tiles of T_{FS}

$(-(c(m)-1),\ 2v+1)$. The sequence of north binding domains of the tiles attached in positions $(-3v-2,\ 2v+1)$ through $(-(c(m)-1),\ 2v+1)$ will encode ψ'', where ψ'' is identical to ψ', except every instance of ℓ'_2 is replaced with *TRUE* and every instance of $\neg\ell'_2$ is replaced with *FALSE*. By a third application of the formula simplification lemma (Lemma 5), tiles from $\mathsf{T}_{SIMPLIFY}$ will attach in the rectangle defined by the corner positions $(-3v-2,\ 2v+2)$ and $(-(c(m)-1),\ 3v+1)$. The sequence of north binding domains of the tiles attached in positions $(-3v-2,\ 3v+1)$ through $(-(c(m)-1), 3v+1)$ will encode ψ''', where ψ''' is identical to ψ'', except every instance of ℓ'_3 is replaced with *TRUE* and every instance of $\neg\ell'_3$ is replaced with *FALSE*.

Observe that the sequence of north binding domains of tiles in positions $(-3v-2,\ 3v+1)$ through $(-(c(m)-1),\ 3v+1)$ encodes ψ''', a Boolean formula with one fewer clause than ϕ and every instance of the literals over the variables selected by the assignment a over the first clause replaced with *TRUE* and *FALSE* as appropriate. Further, notice that the seed tiles with **, *, and $|$ west binding domains are located in the same positions with respect to this new formula as they were with respect to the original ϕ in row 0. Thus the algorithm can recurse on ψ''', by selecting an assignment over the unassigned literals of the new eastmost clause, and applying that assignment to the formula. This process continues until either (1) all literals in some clause $\hat{m}$ are *FALSE* and no tile can attach to the north and west of position $(-(c(\hat{m})+3v), c(\hat{m}))$, as per the literal selection lemma (Lemma 2), or (2), the last (westmost) clause of ϕ selects a successful assignment, in which case, by the the assignment preparation lemma (Lemma 4), the north finding domain of the tile in position $(-(c(m)-1), c(m)-1) \in \{@, ^{\star}\}$. Note that the black ✓ tile can only attach in position $(-(c(m)-1),\ c(m))$ (because it has a $|$ east binding domain, and that domain only occurs as the west binding domain of the seed tile in position $(-(c(m)-2),\ c(m)))$. In case (1) above, since no tile can attach in position $(-(c(m)-1),\ c(m))$, the ✓ tile can never attach. In case (2) above, the ✓ tile does attach in position $(-(c(m)-1),\ c(m))$. Therefore, F will contain the ✓ tile iff the assignment selected by this particular set of nondeterministic decisions satisfies ϕ. Because whenever $\mathbb{S}_{FS}$ makes such a nondeterministic decision, it explores all possible assignments that satisfy a given clause, it nondeterministically explores all assignments that might satisfy ϕ. Therefore, $\mathbb{S}_{FS}$ nondeterministically decides 3-*SAT* with the ✓ tile as the identifier tile.

□

Theorem 2 *For all $n \in \mathbb{N}$, for all Boolean formula ϕ on n distinct variables, $\mathbb{S}_{FS}$ can nondeterministically form only $O^{\star}(1.8393^n)$ distinct assemblies.*

Proof (Theorem 2) The tiles of $\mathbb{S}_{SELECT}$ are the only ones that attach nondeterministically. For each clause, $\mathbb{S}_{SELECT}$ creates at most three distinct nondeterministic assemblies, each encoding a Boolean formula with one fewer clause and each of one, two, and three fewer variables, respectively. Thus, if the number of distinct nondeterministic assemblies created by $\mathbb{S}_{FS}$ on an n-variable m-clause Boolean formula is denoted $T(n, m)$, then $T(n, m) = T(n-1, m-1) + T(n-2, m-1) + T(n-3, m-1)$, and $T(3, 1) = 3$, $T(2, 1) = 2$, and $T(1, 1) = 1$. By Woeginger (2003), the solution to this recurrence is $O^{\star}(1.8393^n)$.

□

4 Contributions

I have defined $\mathbb{S}_{FS}$, a tile system that solves 3-*SAT* by nondeterministically creating $O^{\star}(1.8393^n)$ assemblies in

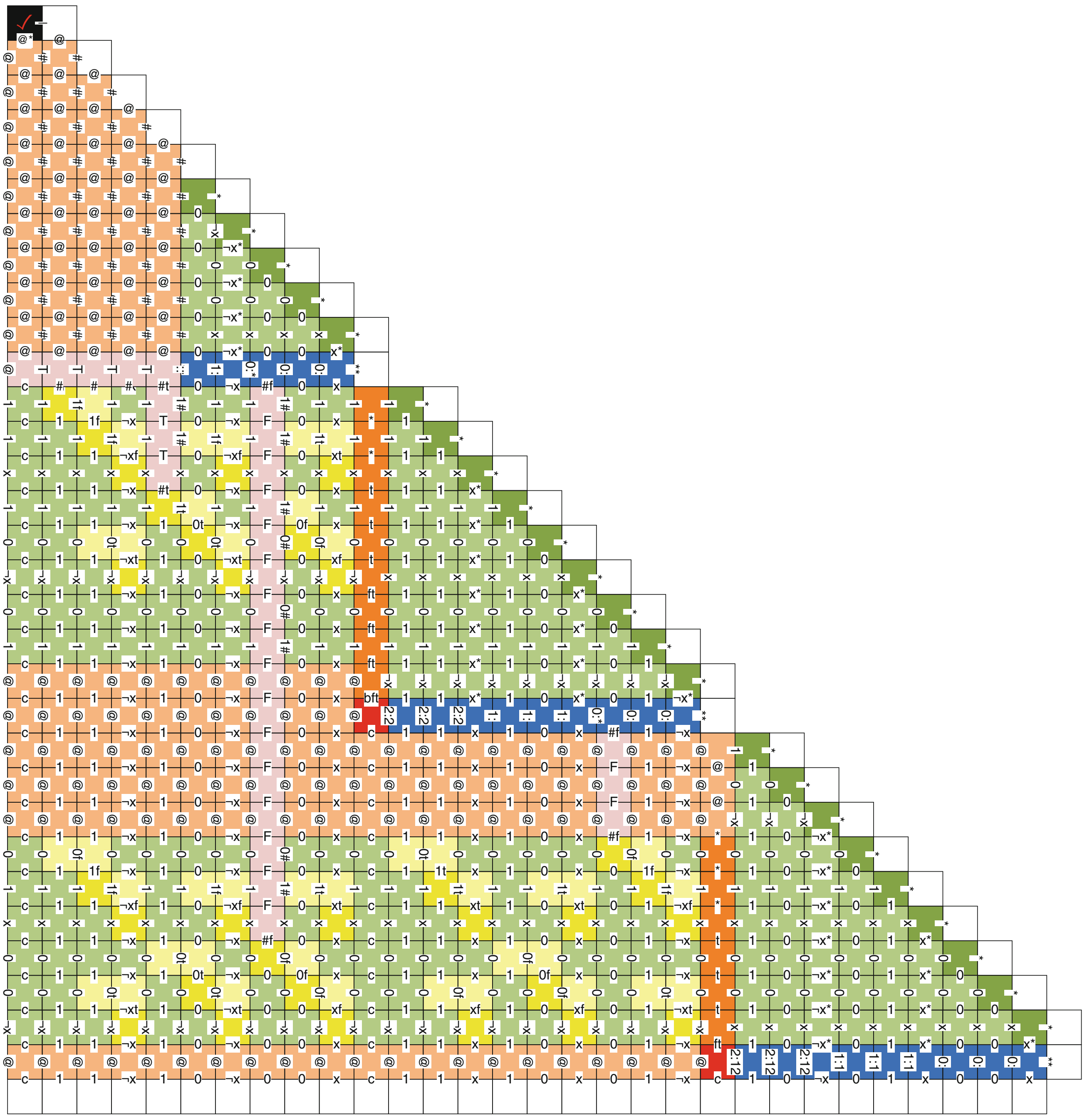


Fig. 16 An example execution of $\mathbb{S}_{FS}$ on the seed from Fig. 14. In this execution, tiles from T_{FS} attach to first assign $x_0 = FALSE$, $x_1 = TRUE$, $x_2 = FALSE$, and $x_3 = TRUE$. Because this assignment satisfies $(\neg x_3 \vee \neg x_2 \vee x_0) \wedge (x_3 \vee x_2 \vee \neg x_1) \wedge (\neg x_2 \vee x_1 \vee x_0)$, the black ✓ tile attaches in the northwest corner

parallel, for an n-variable Boolean formula. Each assembly assembles in time linear in the input size, explores some truth assignment, and attaches a special ✓ tile iff that assignment satisfies the formula. $\mathbb{S}_{FS}$ uses $147 = \Theta(1)$ distinct tile types.

$\mathbb{S}_{FS}$ helps bridge the gap between theoretical explorations of self-assembly, which require large tilesets to implement complex algorithms, and experimental endeavors, which have been able to combine no more than 20 distinct tiles in a single experiment. Further, $\mathbb{S}_{FS}$ improves the efficiency of existing tile-inspired distributed software systems used to leverage large public untrusted networks to solve NP-complete problems.

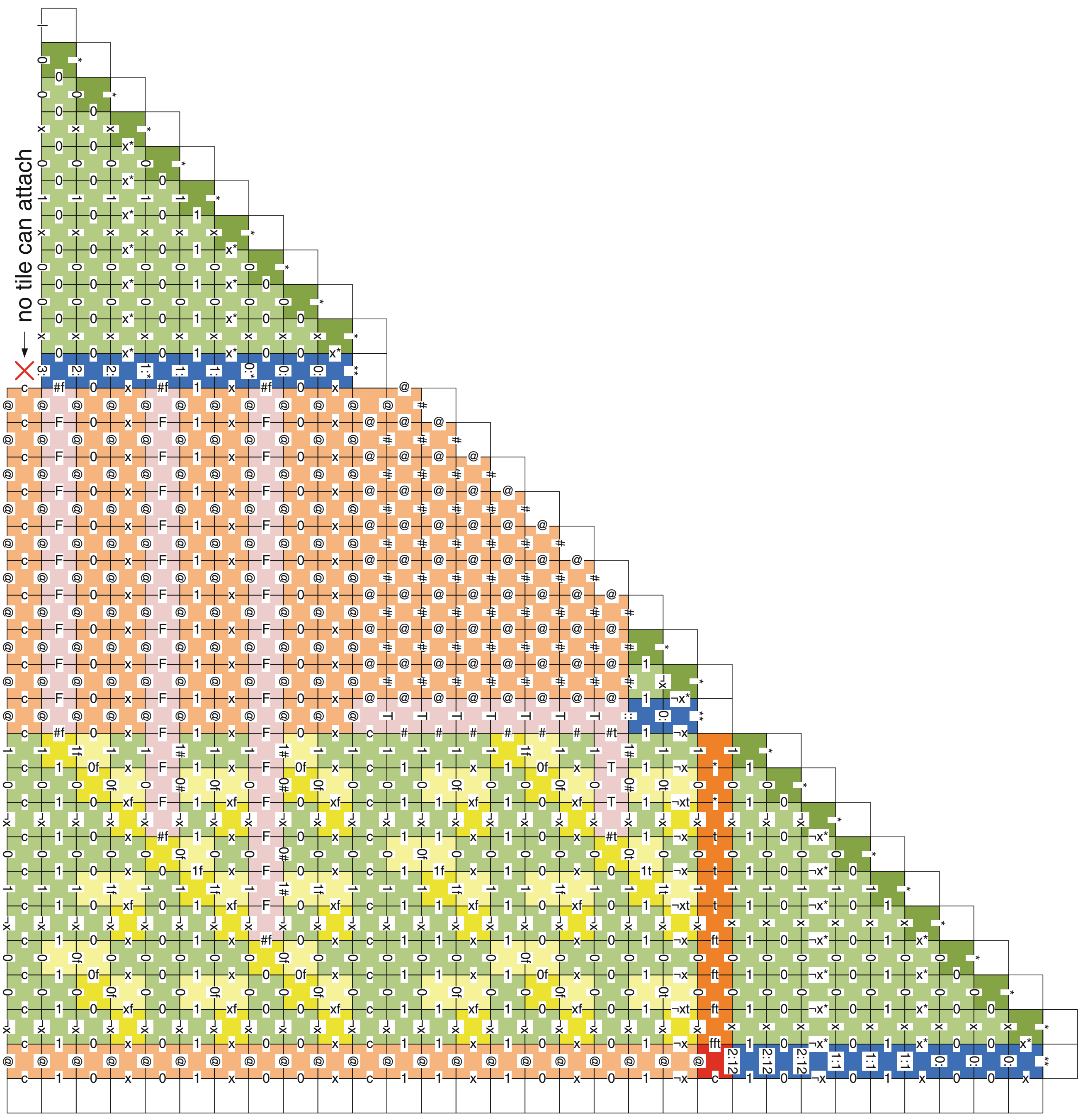


Fig. 17 An example execution of $\mathbb{S}_{FS}$ on the Boolean formula $(x_2 \vee x_1 \vee x_0) \wedge (x_3 \vee x_2 \vee \neg x_1) \wedge (\neg x_2 \vee x_1 \vee x_0)$. This execution chooses to assign *FALSE* to x_0, x_1, and x_2 after examining the eastmost clause. This decision prevents the system from being able to satisfy the westmost clause, thus halting the assembly and not allowing the ✓ tile to attach

References

Abelson H, Allen D, Coore D, Hanson C, Homsy G, Knight TF Jr, Nagpal R, Rauch E, Sussman GJ, Weiss R (2000) Amorphous computing. Commun ACM 43(5):74–82. ISSN 0001-0782. doi: 10.1145/332833.332842

Adleman L (2000) Towards a mathematical theory of self-assembly. Technical Report 00-722, Department of Computer Science, University of Southern California, Los Angeles, CA

Adleman L, Cheng Q, Goel A, Huang M-D, Kempe D, de Espanés PM, Rothemund PWK (May 2002a) Combinatorial optimization problems in self-assembly. In: Proceedings of the 34th annual ACM symposium on theory of computing (STOC02), Montreal, Quebec, Canada, pp 23–32. doi:10.1145/509907.509913

Adleman L, Kari J, Kari L, Reishus D (November 2002b) On the decidability of self-assembly of infinite ribbons. In: Proceedings of the 43rd annual IEEE symposium on foundations of computer science (FOCS02), Ottawa, Ontario, Canada, pp 530–537

Aggarwal G, Cheng Q, Goldwasser MH, Kao M-Y, de Espanés PM, Schweller RT (2005) Complexities for generalized models of self-assembly. SIAM J Comput 34(6):1493–1515. doi:10.1137/S0097539704445202

Barish R, Rothemund PWK, Winfree E (2005) Two computational primitives for algorithmic self-assembly: copying and counting. Nano Lett 5(12):2586–2592. doi:10.1021/nl052038l

Barish RD, Schulman R, Rothemund PWK, Winfree E (2009) An information-bearing seed for nucleating algorithmic self-assembly. Proc Natl Acad Sci USA. doi:10.1073/pnas.0808736106

Berger R (1966) The undecidability of the domino problem. Number 66 in Memoirs Series. American Mathematical Society, Providence, RI, USA

Braich R, Chelyapov N, Johnson CR, Rothemund PWK, Adleman L (2002) Solution of a 20-variable 3-SAT problem on a DNA computer. Science 296(5567):499–502. doi:10.1126/science.1069528

Brun Y (2007) Arithmetic computation in the tile assembly model: Addition and multiplication. Theor Comput Sci 378(1):17–31. ISSN 0304-3975. doi:10.1016/j.tcs.2006.10.025

Brun Y (2008a) Nondeterministic polynomial time factoring in the tile assembly model. Theor Comput Sci 395(1):3–23. ISSN 0304-3975. doi:10.1016/j.tcs.2007.07.051

Brun Y (2008b) Solving NP-complete problems in the tile assembly model. Theor Comput Sci 395(1):31–46. ISSN 0304-3975. doi:10.1016/j.tcs.2007.07.052

Brun Y (2008c) Solving satisfiability in the tile assembly model with a constant-size tileset. J Algorithm 63(4):151–166. ISSN 0196-6774. doi:10.1016/j.jalgor.2008.07.002

Brun Y, Medvidovic N (2008) Preserving privacy in distributed computation via self-assembly. Technical Report USC-CSSE-2008-819, University of Southern California, Center for Software Engineering

Brun Y, Reishus D (2009) Path finding in the tile assembly model. Theor Comput Sci 410(15):1461–1472. ISSN 0304-3975. doi:10.1016/j.tcs.2008.12.008

Chen K, Ramachandran V (2001) A space-efficient randomized DNA algorithm for k-SAT. DNA Comput LNCS 2054:199–208. doi:10.1007/3-540-44992-2_13

Demaine E, Demaine M, Fekete S, Ishaque M, Rafalin E, Schweller R (2008) Staged self-assembly: Nanomanufacture of arbitrary shapes with o(1) glues. In: Max G, Hao Y (eds) DNA computing, vol 4848. Berlin: Springer, pp 1–14. doi:10.1007/978-3-540-77962-9_1

Doty D, Patitz MJ, Reishus D, Schweller, RT, Summers SM (2010) Strong fault-tolerance for self-assembly with fuzzy temperature. In: Proceedings of the 51st annual IEEE symposium on foundations of computer science (FOCS10), pp 417–426. doi:10.1109/FOCS.2010.47

Fujibayashi K, Zhang DY, Winfree E, Murata S (2009) Error suppression mechanisms for DNA tile self-assembly and their simulation. Nat Comput 8:589–612. ISSN 1567-7818. doi:10.1007/s11047-008-9093-9

Kao M-Y, Schweller R (January 2006) Reducing tile complexity for self-assembly through temperature programming. In: Proceedings of the 17th annual ACM-SIAM symposium on discrete algorithms (SODA06), Miami, FL, USA, pp 571–580. doi:10.1145/1109557.1109620

Kullmann O (1997) Worst-case analysis, 3-SAT decision and lower bounds: approaches for improved SAT algorithms. DIMACS Ser Discret Math Theor Comput Sci 35:261–313

Kullmann O (1999) New methods for 3-SAT decisions and worst-case analysis. Theor Comput Sci 223:1–72. doi:10.1016/S0304-3975(98)00017-6

Lagoudakis MG, LaBean TH (1999) 2D DNA self-assembly for satisfiability. DIMACS Ser Discret Math Theor Comput Sci 54:141–154

McLurkin J, Smith J, Frankel J, Sotkowitz D, Blau D, Schmidt B (March 2006) Speaking swarmish: human-robot interface design for large swarms of autonomous mobile robots. In Proceedings of the AAAI spring symposium, Stanford, CA, USA

Monien B, Speckenmeyer E (1985) Solving satisfiability in less than 2^n steps. Discrete Appl Math 10(3):287–296. doi:10.1016/0166-218X(85)90050-2

Paturi R, Pudlák P, Zane F (1997) Satisfiability coding lemma. In: Proceedings of the 38th annual symposium on foundations of computer science (FOCS97), Miami Beach, FL, USA, pp 566–574. ISBN 0-8186-8197-7. doi:10.1109/SFCS.1997.646146

Robinson RM (1971) Undecidability and nonperiodicity for tilings of the plane. Invent Math 12(3):177–209

Rothemund PWK, Winfree E (May 2000) The program-size complexity of self-assembled squares. In: Proceedings of the 32nd annual ACM symposium on theory of computing (STOC00), Portland, OR, USA, pp 459–468. doi:10.1145/335305.335358

Rothemund PWK, Papadakis N, Winfree E (2004) Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biol 2(12):e424. doi:10.1371/journal.pbio.0020424

Sakamoto K, Gouzu H, Komiya K, Kiga D, Yokoyama S, Yokomori T (2000) Molecular computation by DNA hairpin formation. Science, 288(5469):1223–1226. doi:10.1126/science.288.5469.1223

Schiermeyer I (1993) Solving 3-satisfiability in less than 1.579^n steps. Comput Sci Logic 702:379–394

Sipser M (1997) Introduction to the theory of computation. PWS Publishing Company

Soloveichik D, Winfree E (2007) Complexity of self-assembled shapes. SIAM J Comput 36(6):1544–1569. doi:10.1137/S0097539704446712

Wang H (1961) Proving theorems by pattern recognition. II. Bell Syst Tech J 40:1–42

Wang H (1962) An unsolvable problem on dominoes. Technical Report BL30 (II-15), Harvard Computation Laboratory

Winfree E (1998a) Simulations of computing by self-assembly of DNA. Technical Report CS-TR:1998:22. California Institute of Technology, Pasadena, CA

Winfree E (1998b) Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology, Pasadena, CA, USA, June

Winfree E, Bekbolatov R (June 2003) Proofreading tile sets: error correction for algorithmic self-assembly. In: Proceedings of the 43rd annual IEEE symposium on foundations of computer science (FOCS02), vol 2943, pp 126–144, Madison, WI, USA. doi:10.1007/978-3-540-24628-2_13

Winfree E, Yang X, Seeman NC (1998) Universal computation via self-assembly of DNA: some theory and experiments. DNA Based Computers II, pp 191–213

Woeginger GJ (2003) Exact algorithms for NP-hard problems: a survey. Combinatorial Optimization - Eureka, You Shrink! 2570:185–207. doi:10.1007/3-540-36478-1_17

Yin P, Hariadi RF, Sahu S, Choi HMT, Park SH LaBean TH, Reif JH (2008) Programming DNA tube circumferences. Science 321(5890):824–826. doi:10.1126/science.1157312