A APPENDIX

A.1 Construction of PSST from RegEx

Case $e = \emptyset$ (see Figure 15). $\mathcal{T}_\emptyset = (\{q_{\emptyset,0}\}, \Sigma, \{x_\emptyset\}, \delta_\emptyset, \tau_\emptyset, E_\emptyset, q_{\emptyset,0}, (\emptyset, \emptyset))$, where there are no transitions out of $q_{\emptyset,0}$, namely, $\delta_\emptyset(q_{\emptyset,0}, a) = ()$ for every $a \in \Sigma$, $\tau_\emptyset(q_{\emptyset,0}) = ((); ())$, and $E_\emptyset$ is vacuous here.

Case $e = \varepsilon$ (see Figure 15). $\mathcal{T}_\varepsilon = (\{q_{\varepsilon,0}, f_{\varepsilon,0}\}, \Sigma, \{x_\varepsilon\}, \delta_\varepsilon, \tau_\varepsilon, E_\varepsilon, q_{\varepsilon,0}, (\{f_{\varepsilon,0}\}, \emptyset))$, where $\tau_\varepsilon(q_{\varepsilon,0}) = ((f_{\varepsilon,0}); ())$, and $E_\varepsilon(q_{\varepsilon,0}, \varepsilon, f_{\varepsilon,0})(x) = \varepsilon$. Note $F_2 = \emptyset$ here.

Case $e = a$ (see Figure 15). $\mathcal{T}_a = (\{q_{a,0}, q_{a,1}, f_{a,0}\}, \Sigma, \{x_a\}, \delta_a, \tau_a, E_a, q_{a,0}, (\emptyset, \{f_{a,0}\}))$, where $\tau_a(q_{a,0}) = ((q_{a,1}); ())$, $\delta_a(q_{a,1}, a) = (f_{a,0})$, $E_a(q_{a,0}, \varepsilon, q_{a,1})(x_a) = \varepsilon$, and $E_a(q_{a,1}, a, f_{a,0})(x_a) = x_a a$. Note $F_1 = \emptyset$ here.

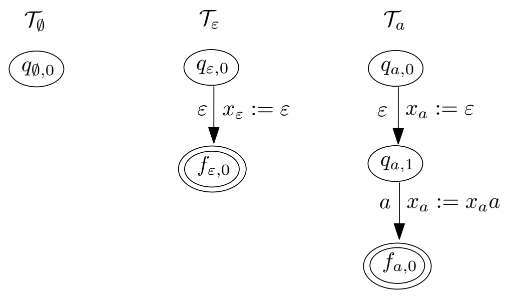


Fig. 15. The PSST $\mathcal{T}_\emptyset$, $\mathcal{T}_\varepsilon$, and $\mathcal{T}_a$

Case $e = [e_1^{+?}]$. Then $\mathcal{T}_e$ is constructed from $\mathcal{T}_{e_1}$ and $\mathcal{T}^-_{[e_1^{*?}]}$, similarly to the aforementioned construction of $\mathcal{T}_{[e_1^+]}$.

Case $e = [e_1^{\{m_1,m_2\}?}]$ for $1 \le m_1 < m_2$ (see Figure 16). Then $\mathcal{T}_e$ is constructed as the concatenation of $\mathcal{T}_{e_1}^{\{m_1\}}$ and $\mathcal{T}_{e_1}^{\{1,m_2-m_1\}?}$, where $\mathcal{T}_{e_1}^{\{1,m_2-m_1\}?}$ is illustrated in Figure 16, which is the same as $\mathcal{T}_{e_1}^{\{1,m_2-m_1\}}$ in Figure 9, except that the priorities of the ε-transition from $q^{(1)}_{e_1,0}$ to f'_0 has the highest priority and the priorities of the ε-transitions out of each $f^{(i)}_{e_1,2} \in F^{(i)}_{e_1,2}$ to f'_1 are swapped.

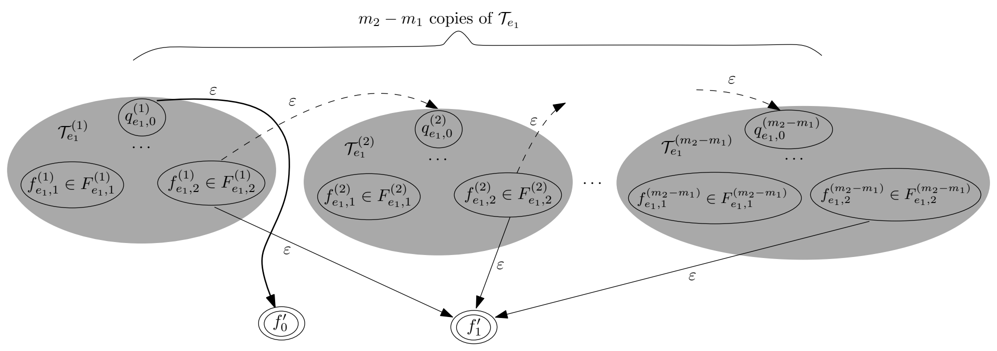

$m_2 - m_1$ copies of $\mathcal{T}_{e_1}$
$\mathcal{T}^{(1)}_{e_1}$
$q^{(1)}_{e_1,0}$
$f^{(1)}_{e_1,1} \in F^{(1)}_{e_1,1}$
$f^{(1)}_{e_1,2} \in F^{(1)}_{e_1,2}$
$\mathcal{T}^{(2)}_{e_1}$
$q^{(2)}_{e_1,0}$
$f^{(2)}_{e_1,1} \in F^{(2)}_{e_1,1}$
$f^{(2)}_{e_1,2} \in F^{(2)}_{e_1,2}$
$\mathcal{T}^{(m_2-m_1)}_{e_1}$
$q^{(m_2-m_1)}_{e_1,0}$
$f^{(m_2-m_1)}_{e_1,1} \in F^{(m_2-m_1)}_{e_1,1}$
$f^{(m_2-m_1)}_{e_1,2} \in F^{(m_2-m_1)}_{e_1,2}$
ε
f'_0
f'_1

Fig. 16. The PSST $\mathcal{T}_{e_1}^{\{1,m_2-m_1\}?}$

A.2 From extract, replace and replaceAll to PSSTs

Lemma 4.6. The satisfiability of STR reduces to the satisfiability of boolean combinations of formulas of the form $z = x \cdot y$, $y = \mathcal{T}(x)$, and $x \in \mathcal{A}$, where $\mathcal{T}$ is a PSST and $\mathcal{A}$ is an FA.

The proof is in two steps: first we remove $\$0$, $\$^{\leftarrow}$, and $\$^{\rightarrow}$, then we encode the remaining string functions with PSSTs.

A.2.1 Removing Special References. The first step in our proof is to remove the special references $\$0$, $\$^{\leftarrow}$, and $\$^{\rightarrow}$ from the replacement strings. These can be replaced in a series of steps, leaving only PSST transductions and replacement strings with only simple references ($\$i$). We will just consider replaceAll as replace is almost identical.

First, to remove $\$0$, suppose we have a statement $y := \mathsf{replaceAll}_{\mathsf{pat},\mathsf{rep}}(x)$ with $\$0$ in rep. We simply substitute $y := \mathsf{replaceAll}_{\mathsf{pat}',\mathsf{rep}'}(x)$ where $\mathsf{pat}' = (\mathsf{pat})$, and $\mathsf{rep}' = \mathsf{rep}[\$1/\$0, \$2/\$1, \ldots, \$(k+1)/\$k]$. That is, we make the complete match an explicit (first) capture, which shifts the indexes of the remaining capturing groups by 1.

Now suppose we have a statement $y := \mathsf{replaceAll}_{\mathsf{pat},\mathsf{rep}}(x)$ with $\$^{\leftarrow}$ or $\$^{\rightarrow}$ in rep. We replace it with the following statements, explained below, where $y_1, \ldots, y_5$ are fresh variables.

$$\begin{aligned}
y_1 &:= \mathsf{replaceAll}_{(\mathsf{pat}),\langle\$1\rangle}(x);\\
y_2 &:= \mathcal{T}_{\langle}(y_1);\\
y_3 &:= \mathcal{T}_{\mathsf{rev}}(y_2);\\
y_4 &:= \mathcal{T}_{\rangle}(y_3);\\
y_5 &:= \mathcal{T}_{\mathsf{rev}}(y_4);\\
y &:= \mathsf{replaceAll}_{\mathsf{pat}',\mathsf{rep}'}(y_5)
\end{aligned}$$

The first step is to mark the matched parts of the string with $\langle$ and $\rangle$ brackets (where $\langle$ and $\rangle$ are not part of the main alphabet). This is achieved by the first replaceAll.

Next, we use a PSST $\mathcal{T}_{\langle}$ that passes over the marked word. This is a copyful PSST that simply stores the word read so far into a variable X, except for the $\langle$ and $\rangle$ characters. It also has an output variable O, to which it also copies each character directly, except $\langle$. When it encounters $\langle$ it appends to O the string $\langle X \langle$. That is, it puts the entire string preceding each $\langle$ into the output, surrounded by $\langle$ at the start and end. This is copyful since X will be copied to both X and O in this step. For example, suppose the input string were $ab\langle c\rangle d\langle e\rangle f$, the output of $\mathcal{T}_{\langle}$ would be $ab\langle\underline{ab}\langle c\rangle d\langle\underline{abcd}\langle e\rangle f$. We have underlined the strings inserted for readability.

The next step is to do the same for $\rangle$. To achieve this we first reverse the string so that a PSST can read the end of the string first. A similar transduction to $\mathcal{T}_{\langle}$ is performed before the string is reversed again. In our example the resulting string is $ab\langle\underline{ab}\langle c\rangle\underline{def}\rangle d\langle\underline{abcd}\langle e\rangle\underline{f}\rangle f$.

Finally, we have $y := \mathsf{replaceAll}_{\mathsf{pat}',\mathsf{rep}'}(y_5)$ where $\mathsf{pat}' = \langle(\Sigma^{*?})\langle\mathsf{pat}\rangle(\Sigma^{*?})\rangle$, and

$$\mathsf{rep}' = \mathsf{rep}[\$1/\$^{\leftarrow}, \$2/\$1, \ldots \$(k+1)/\$k, \$(k+2)/\$^{\rightarrow}]\ .$$

That is, by inserting the preceding and succeeding text directly next to each match, we can use simple references $\$i$ instead of $\$^{\leftarrow}$ and $\$^{\rightarrow}$.

A.2.2 Encoding string functions as PSSTs. Once $\$0$, $\$^{\leftarrow}$, and $\$^{\rightarrow}$ are removed from the replacement strings, then the string functions can be replaced by PSSTs.

Lemma A.1. *For each string function $f = \mathsf{extract}_{i,e}$, $\mathsf{replace}_{\mathsf{pat},\mathsf{rep}}$, or $\mathsf{replaceAll}_{\mathsf{pat},\mathsf{rep}}$ without $\$^{\leftarrow}$ or $\$^{\rightarrow}$ in the replacements strings, a PSST $\mathcal{T}_f$ can be constructed such that*

$$\mathcal{R}_f = \{(w, w') \mid w' = f(w)\}.$$

Proof. The extract$_{i,e}$ can function be defined by a PSST $\mathcal{T}_{i,e}$ obtained from the PSST $\mathcal{T}_e$ (see Section 4.2) by removing all the string variables, except the string variable $x_{e'}$, where e' is the subexpression of e corresponding to the ith capturing group, and setting the output expression of the final states as $x_{e'}$.

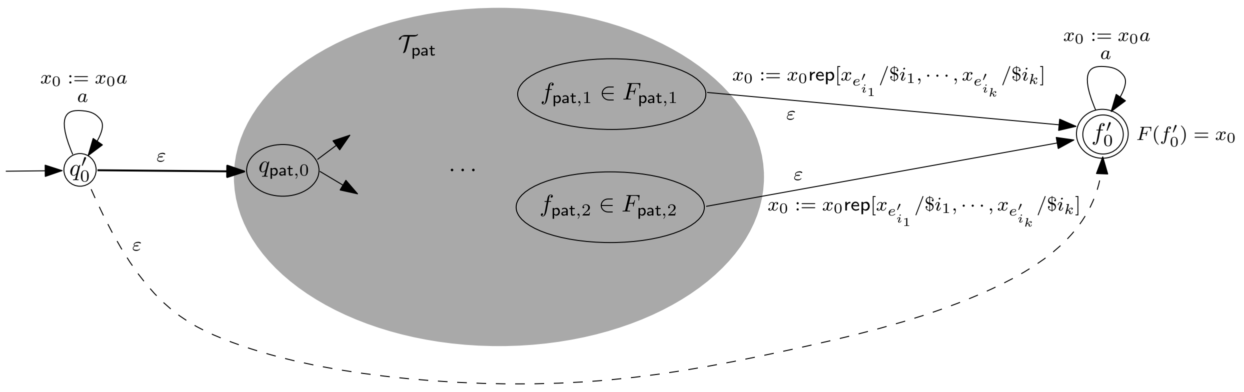
Next, we give the construction of the PSST for replaceAll$_{\mathsf{pat},\mathsf{rep}}$ where all the references in rep are of the form $\$i$.

Recall rep $= w_1\$i_1 w_2 \cdots w_k \$i_k w_{k+1}$. Let $e'_{i_1}, \ldots, e'_{i_k}$ denote the subexpressions of pat corresponding to the i_1th, …, i_kth capturing groups of pat. Then $\mathcal{T}_{\mathsf{replaceAll}_{\mathsf{pat},\mathsf{rep}}} = (Q_{\mathsf{pat}} \cup \{q'_0\}, \Sigma, X', \delta', \tau', E', q'_0, F')$ where

- $q'_0 \notin Q_{\mathsf{pat}}$,
- $X' = \{x_0\} \cup X_{\mathsf{pat}}$,
- $F'(q'_0) = x_0$, and $F'(q')$ is undefined for every $q' \in Q_{\mathsf{pat}}$,
- δ' comprises the transitions in δ_{pat}, and the transition $\delta'(q'_0, a) = (q'_0)$ for $a \in \Sigma$,
- τ' comprises the transitions in τ_{pat}, the transitions $\tau'(q'_0) = ((q_{\mathsf{pat},0}); ())$, $\tau'(f_{\mathsf{pat},1}) = ((q_{\mathsf{pat},0}); ())$ and $\tau'(f_{\mathsf{pat},2}) = ((q_{\mathsf{pat},0}); ())$ for $f_{\mathsf{pat},1} \in F_{\mathsf{pat},1}$ and $f_{\mathsf{pat},2} \in F_{\mathsf{pat},2}$,
- E' inherits E_{pat}, and includes the assignments $E'(q'_0, a, q'_0)(x_0) = x_0 a$ for $a \in \Sigma$, $E'(f, \varepsilon, q'_0)(x_0) = x_0 \mathsf{rep}[x_{e'_{i_1}}/\$i_1, \ldots, x_{e'_{i_k}}/i_k]$ and $E'(f, \varepsilon, q'_0)(x) = \mathsf{null}$ for every $f \in F_{\mathsf{pat},1} \cup F_{\mathsf{pat},2}$ and $x \in X_{\mathsf{pat}}$.

□

The construction of the PSST for replace$_{\mathsf{pat},\mathsf{rep}}$ is similar and illustrated in Fig. 17. The details are omitted.



$x_0 := x_0 a$
a
q'_0
ε
ε
$\mathcal{T}_{\mathsf{pat}}$
$q_{\mathsf{pat},0}$
$\cdots$
$f_{\mathsf{pat},1} \in F_{\mathsf{pat},1}$
$f_{\mathsf{pat},2} \in F_{\mathsf{pat},2}$
$x_0 := x_0\mathsf{rep}[x_{e'_{i_1}}/\$i_1, \cdots, x_{e'_{i_k}}/\$i_k]$
ε
ε
$x_0 := x_0\mathsf{rep}[x_{e'_{i_1}}/\$i_1, \cdots, x_{e'_{i_k}}/\$i_k]$
$x_0 := x_0 a$
a
f'_0
$F(f'_0) = x_0$

Fig. 17. The PSST $\mathcal{T}_{\mathsf{replace}_{\mathsf{pat},\mathsf{rep}}}$

A.3 Proof of Lemma 5.5

Lemma 5.5. *Given a PSST $\mathcal{T} = (Q_T, \Sigma, X, \delta_T, \tau_T, E_T, q_{0,T}, F_T)$ and an FA $\mathcal{A} = (Q_A, \Sigma, \delta_A, q_{0,A}, F_A)$, we can compute an FA $\mathcal{B} = (Q_B, \Sigma, \delta_B, q_{0,B}, F_B)$ in exponential time such that $\mathscr{L}(\mathcal{B}) = \mathcal{R}_{\mathcal{T}}^{-1}(\mathscr{L}(\mathcal{A}))$.*

We prove Lemma 5.5 in the sequel.

Let $\mathcal{T} = (Q_T, \Sigma, X, \delta_T, \tau_T, E_T, q_{0,T}, F_T)$ be a PSST and $\mathcal{A} = (Q_A, \Sigma, \delta_A, q_{0,A}, F_A)$ be an FA. Without loss of generality, we assume that $\mathcal{A}$ contains no ε-transitions. For convenience, we use $\mathcal{E}(\tau_T)$ to denote $\{(q, q') \mid q' \in \tau_T(q)\}$. For convenience, for $a \in \Sigma$, we use $\delta_A^{(a)}$ to denote the relation $\{(q, q') \mid (q, a, q') \in \delta_A\}$.

To illustrate the intuition of the proof of Lemma 5.5, let us start with the following natural idea of firstly constructing a PFA $\mathcal{B}$ for the pre-image: $\mathcal{B}$ simulates a run of $\mathcal{T}$ on w, and, for each $x \in X$, records an $\mathcal{A}$-abstraction of the string stored in x, that is, the set of state pairs $(p, q) \in Q_A \times Q_A$

such that starting from p, $\mathcal{A}$ can reach q after reading the string stored in x. Specifically, the states of $\mathcal{B}$ are of the form (q, ρ) with $q \in Q$ and $\rho \in (\mathcal{P}(Q_A \times Q_A))^X$. Moreover, the priorities of $\mathcal{B}$ inherit those of $\mathcal{T}$. The PFA $\mathcal{B}$ is then transformed to an equivalent FA by simply dropping all priorities. We refer to this FA as $\mathcal{B}'$.

Nevertheless, it turns out that this construction is flawed: A string w is in $\mathcal{R}_{\mathcal{T}}^{-1}(\mathscr{L}(\mathcal{A}))$ iff the (unique) accepting run of $\mathcal{T}$ on w produces an output w' that is accepted by $\mathcal{A}$. However, a string w is accepted by $\mathcal{B}'$ iff *there is a run of $\mathcal{T}$ on w, not necessarily of the highest priority*, producing an output w' that is accepted by $\mathcal{A}$.

While the aforementioned natural idea does not work, we choose to construct an FA $\mathcal{B}$ that simulates the *accepting* run of $\mathcal{T}$ on w, and, for each $x \in X$, records an $\mathcal{A}$-abstraction of the string stored in x, that is, the set of state pairs $(p, q) \in Q_A \times Q_A$ such that starting from p, $\mathcal{A}$ can reach q after reading the string stored in x. To simulate the accepting run of $\mathcal{T}$, it is necessary to record all the states accessible through the runs of higher priorities to ensure the current run is indeed the accepting run of $\mathcal{T}$ (of highest priority). Moreover, $\mathcal{B}$ also remembers the set of ε-transitions of $\mathcal{T}$ after the latest non-ε-transition to ensure that no transition occurs twice in a sequence of ε-transitions of $\mathcal{T}$.

Specifically, each state of $\mathcal{B}$ is of the form (q, ρ, Λ, S), where $q \in Q_T$, $\rho \in (\mathcal{P}(Q_A \times Q_A))^X$, $\Lambda \subseteq \mathcal{E}(\tau_T)$, and $S \subseteq Q_T$. For a state (q, ρ, Λ, S), our intention for S is that the states in it are those that can be reached in the runs of higher priorities than the current run, by reading the same sequence of letters and applying the ε-transitions as many as possible. Note that when recording in S all the states accessible through the runs of higher priorities, we do not take the non-repetition of ε-transitions into consideration since if a state is reachable by a sequence of ε-transitions where some ε-transitions are repeated, then there exists also a sequence of non-repeated ε-transitions reaching the state. Moreover, when simulating an a-transition of $\mathcal{T}$ (where $a \in \Sigma$) at a state (q, ρ, Λ, S), suppose $\delta_T(q, a) = (q_1, \cdots, q_m)$ and $\tau_T(q) = (P_1, P_2)$, then $\mathcal{B}$ nondeterministically chooses q_i and goes to the state $(q_i, \rho', \emptyset, S')$, where

- ρ' is obtained from ρ and $E_T(q, \sigma, q_i)$,
- Λ is reset to $\emptyset$,
- all the states obtained from S by applying an a transition should be *saturated by ε-transitions* and put into S', more precisely, all the states reachable from S by first applying an a-transition, then a sequence of ε-transitions, should be put into S',
- moreover, all the states obtained from $q_1, \cdots, q_{i-1}$ (which are of higher priorities than q_i) by saturating with ε-transitions should be put into S',
- finally, all the states obtained from those in $P_1' = \{q' \in P_1 \mid (q, q') \notin \Lambda\}$ (which are of higher priorities than q_i) by saturating with non-Λ ε-transitions first (i.e. the ε-transitions that do not belong to Λ), and applying an a-transition next, finally saturating with ε-transitions again, should be put into S', (note that according to the semantics of PSST, the ε-transitions in Λ should be avoided when defining P_1' and saturating the states in P_1' with ε-transitions).

The above construction does not utilize the so-called *copyless* property (i.e. for each transition t and each variable x, x appears at most once on the right-hand side of the assignment for t) [Alur and Cerný 2010; Alur and Deshmukh 2011], thus it works for general, or *copyful*, PSSTs [Filiot and Reynier 2017]. It can be noted that the powerset in $\rho \in (\mathcal{P}(Q_A \times Q_A))^X$ is required to handle copyful transductions as the contents of a variable may be used in many different situations, each requiring a different abstraction. If the PSST is copyless, we can instead use $\rho \in (Q_A \times Q_A)^X$. That is, each variable is used only once, and hence only one abstraction pair is needed. The powerset construction in the transitions can be replaced by a non-deterministic choice of the particular pair of states from Q_A that should be kept. This avoids the construction being exponential in the size of A, which in turn avoids the tower of exponential blow-up in the backwards reasoning.

For the formal construction of $\mathcal{B}$, we need some additional notations.

- For $S \subseteq Q_T$, $\delta_T^{(ip)}(S, a) = \{q_1' \mid \exists q_1 \in S, q_1' \in \delta_T(q_1, a)\}$.
- For $q \in Q_T$, if $\tau_T(q) = (P_1, P_2)$, then $\tau_T^{(ip)}(\{q\}) = S$ such that $S = P_1 \cup P_2$. Moreover, for $S \subseteq Q_T$, we define $\tau_T^{(ip)}(S) = \bigcup\limits_{q \in S} \tau_T^{(ip)}(\{q\})$. We also use $\left(\tau_T^{(ip)}\right)^*$ to denote the ε-closure of $\mathcal{T}$, namely, $\left(\tau_T^{(ip)}\right)^*(S) = \bigcup\limits_{n \in \mathbb{N}} \left(\tau_T^{(ip)}\right)^n(S)$, where $\left(\tau_T^{(ip)}\right)^0(S) = S$, and for $n \in \mathbb{N}$, $\left(\tau_T^{(ip)}\right)^{n+1}(S) = \tau_T^{(ip)}\left(\left(\tau_T^{(ip)}\right)^n(S)\right)$.
- For $S \subseteq Q_T$ and $\Lambda \subseteq \mathcal{E}(\tau_T)$, we use $\left(\tau_T^{(ip)} \setminus \Lambda\right)^*(S)$ to denote the set of states reachable from S by sequences of ε-transitions where *no* transitions (q, ε, q') such that $(q, q') \in \Lambda$ are used.
- For $\rho \in (\mathcal{P}(Q_A \times Q_A))^X$ and $s \in X \to (X \cup \Sigma)^*$, we use $s(\rho)$ to denote ρ' that is obtained from ρ as follows: For each $x \in X$, if $s(x) = \varepsilon$, then $\rho'(x) = \{(p, p) \mid p \in Q_A\}$, otherwise, let $s(x) = b_1 \cdots b_\ell$ with $b_i \in \Sigma \cup X$ for each $i \in [\ell]$, then $\rho'(x) = \theta_1 \circ \cdots \circ \theta_\ell$, where $\theta_i = \delta_A^{(b_i)}$ if $b_i \in \Sigma$, and $\theta_i = \rho(b_i)$ otherwise, and $\circ$ represents the composition of binary relations.

We are ready to present the formal construction of $\mathcal{B} = (Q_B, \Sigma, \delta_B, q_{0,B}, F_B)$.

- $Q_B = Q_T \times (\mathcal{P}(Q_A \times Q_A))^X \times \mathcal{P}(\mathcal{E}(\tau_T)) \times \mathcal{P}(Q_T)$,
- $q_{0,B} = (q_{0,T}, \rho_\varepsilon, \emptyset, \emptyset)$ where $\rho_\varepsilon(x) = \{(q, q) \mid q \in Q\}$ for each $x \in X$,
- δ_B comprises
  - the tuples $((q, \rho, \Lambda, S), a, (q_i, \rho', \Lambda', S'))$ such that
    * $a \in \Sigma$,
    * $\delta_T(q, a) = (q_1, \ldots, q_i, \ldots, q_m)$,
    * $s = E((q, a, q_i))$,
    * $\rho' = s(\rho)$,
    * $\Lambda' = \emptyset$, (Intuitively, Λ is reset.)
    * let $\tau_T(q) = (P_1, P_2)$, then $S' = \left(\tau_T^{(ip)}\right)^*(\{q_1, \ldots, q_{i-1}\} \cup \delta_T^{(ip)}\left(S \cup \left(\tau_T^{(ip)} \setminus \Lambda\right)^*(P_1'), a\right))$, where $P_1' = \{q' \in P_1 \mid (q, q') \notin \Lambda\}$;
  - the tuples $((q, \rho, \Lambda, S), \varepsilon, (q_i, \rho', \Lambda', S'))$ such that
    * $\tau_T(q) = ((q_1, \ldots, q_i, \ldots, q_m); \cdots)$,
    * $(q, q_i) \notin \Lambda$,
    * $s = E(q, \varepsilon, q_i)$,
    * $\rho' = s(\rho)$,
    * $\Lambda' = \Lambda \cup \{(q, q_i)\}$,
    * $S' = S \cup \left(\tau_T^{(ip)} \setminus \Lambda\right)^*(\{q_j \mid j \in [i-1], (q, q_j) \notin \Lambda\})$;
  - the tuples $((q, \rho, \Lambda, S), \varepsilon, (q_i, \rho', \Lambda', S'))$ such that
    * $\tau_T(q) = ((q_1', \ldots, q_n'); (q_1, \ldots, q_i, \ldots, q_m))$,
    * $(q, q_i) \notin \Lambda$,
    * $s = E(q, \varepsilon, q_i)$,
    * $\rho' = s(\rho)$,
    * $\Lambda' = \Lambda \cup \{(q, q_i)\}$,
    * $S' = S \cup \{q\} \cup \left(\tau_T^{(ip)} \setminus \Lambda\right)^*\left(\{q_j' \mid j \in [n], (q, q_j') \notin \Lambda\} \cup \{q_j \mid j \in [i-1], (q, q_j) \notin \Lambda\}\right)$. (Note that here we include q into S', since the non-ε-transitions out of q have higher priorities than the transition (q, ε, q_i).)
- Moreover, F_B is the set of states $(q, \rho, \Lambda, S) \in Q_B$ such that
  (1) $F_T(q)$ is defined,

(2) for every $q' \in S$, $F_T(q')$ is not defined,
(3) if $F_T(q) = \varepsilon$, then $q_{0,A} \in F_A$, otherwise, let $F_T(q) = b_1 \cdots b_\ell$ with $b_i \in \Sigma \cup X$ for each $i \in [\ell]$, then $(\theta_1 \circ \cdots \circ \theta_\ell) \cap (\{q_{0,A}\} \times F_A) \neq \emptyset$, where for each $i \in [\ell]$, if $b_i \in \Sigma$, then $\theta_i = \delta_A^{(b_i)}$, otherwise, $\theta_i = \rho(b_i)$.

A.4 Tower-Hardness of String Constraints with Streaming String Transductions

We show that the satisfiability problem for $\mathrm{STR}_{\mathrm{SL}}$ is Tower-hard.

Theorem A.2. *The satisfiability problem for* $\mathrm{STR}_{\mathrm{SL}}$ *is Tower-hard.*

Our proof will use tiling problems over extremely wide corridors. We first introduce these tiling problems, then how we will encode potential solutions as words. Finally, we will show how $\mathrm{STR}_{\mathrm{SL}}$ can verify solutions.

A.4.1 Tiling Problems. A *tiling problem* is a tuple (Θ, H, V, t^0, f) where Θ is a finite set of tiles, $H \subseteq \Theta \times \Theta$ is a horizontal matching relation, $V \subseteq \Theta \times \Theta$ is a vertical matching relation, and $t^0, f \in \Theta$ are initial and final tiles respectively.

A solution to a tiling problem over a n-width corridor is a sequence

$$\begin{array}{c} t_1^1 \dots t_n^1 \\ t_1^2 \dots t_n^2 \\ \dots \\ t_1^h \dots t_n^h \end{array}$$

where $t_1^1 = t^0$, $t_n^h = f$, and for all $1 \leq i < n$ and $1 \leq j \leq h$ we have $\left(t_i^j, t_{i+1}^j\right) \in H$ and for all $1 \leq i \leq n$ and $1 \leq j < h$ we have $\left(t_i^j, t_i^{j+1}\right) \in V$. Note, we will assume that t^0 and f can only appear at the beginning and end of the tiling respectively.

Tiling problems characterise many complexity classes [Börger et al. 1997]. In particular, we will use the following facts.

- For any n-space Turing machine, there exists a tiling problem of size polynomial in the size of the Turing machine, over a corridor of width n, that has a solution iff the n-space Turing machine has a terminating computation.
- There is a fixed (Θ, H, V, t^0, f) such that for any width n there is a unique solution

$$\begin{array}{c} t_1^1 \dots t_n^1 \\ t_1^2 \dots t_n^2 \\ \dots \\ t_1^h \dots t_n^h \end{array}$$

and moreover h is exponential in n. One such example is a Turing machine where the tape contents represent a binary number. The Turing machine starts from a tape containing only 0s and finishes with a tape containing only 1s by repeatedly incrementing the binary encoding on the tape. This Turing machine can be encoded as the required tiling problem.

A.4.2 Large Numbers. The crux of the proof is encoding large numbers that can take values between 1 and m-fold exponential.

A linear-length binary number could be encoded simply as a sequence of bits

$$b_0 \dots b_n \in \{0, 1\}^n .$$

To aid with later constructions we will take a more oblique approach. Let $(\Theta_1, H_1, V_1, t_1^0, f_1)$ be a copy of the fixed tiling problem from the previous section for which there is a unique solution, whose length must be exponential in the width. In the future, we will need several copies of this problem, hence the indexing here. Note, we assume each copy has disjoint tile sets. Fix a width n and let N_1 be the corresponding corridor length. A *level-1* number can encode values from 1 to N_1. In particular, for $1 \leq i \leq N_1$ we define

$$[i]_1 = t_1^i \dots t_n^i$$

where $t_1^i \dots t_n^i$ is the tiling of the ith row of the unique solution to the tiling problem.

A *level-2* number will be derived from tiling a corridor of width N_1, and thus the number of rows will be doubly-exponential. For this, we require another copy $(\Theta_2, H_2, V_2, t_2^0, f_2)$ of the above tiling problem. Moreover, let N_2 be the length of the solution for a corridor of width N_1. Then for any $1 \leq i \leq N_2$ we define

$$[i]_2 = [1]_1 t_1^i [2]_1 t_2^i \dots [N_1]_1 t_{N_1}^i$$

where $t_1^i \dots t_{N_1}^i$ is the tiling of the ith row of the unique solution to the tiling problem. That is, the encoding indexes each tile with it's column number, where the column number is represented as a level-1 number.

In general, a *level-m* number is of length $(m-1)$-fold exponential and can encode numbers m-fold exponential in size. We use a copy $(\Theta_m, H_m, V_m, t_m^0, f_m)$ of the above tiling problem and use a corridor of width N_{m-1}. We define N_m as the length of the unique solution to this problem. Then, for any $1 \leq i \leq N_m$ we have

$$[i]_m = [1]_{m-1} t_1^i [2]_{m-1} t_2^i \dots [N_{m-1}]_{m-1} t_{N_{m-1}}^i$$

where $t_1^i \dots t_{N_{m-1}}^i$ is the tiling of the ith row of the unique solution to the tiling problem.

Note that we can define regular languages to check that a string is a large number. In particular

$$R_m^n = \begin{cases} [\Theta_1]^n & m = 1 \\ \left[R_{m-1}^n \Theta_m\right]^* & m > 1 \,. \end{cases}$$

A.4.3 Hardness Proof. We show that the satisfiability problem for STR$_{\mathrm{SL}}$ is Tower-hard. We first introduce the basic framework of solving a hard tiling problem. Then we discuss the two phases of transductions required by the reduction. These are constructing a large boolean formula, and then evaluating the formula. This two phases are described in separate sections.

The Framework. The proof is by reduction from a tiling problem over an m-fold exponential width corridor. In general, solving such problems is hard for m-ExpSpace.

Let N_m be the width of the corridor. Fix a tiling problem

$$\left(\Theta, H, V, t^0, f\right) \,.$$

We will compose an STR$_{\mathrm{SL}}$ formula S with a free variable x. If S is satisfiable, x will contain a string encoding a solution to the tiling problem. In particular, the value of x will be of the form

$$\begin{gathered} [1]_m t_1^1 \dots [N_m]_m t_{N_m}^1 \# \\ [1]_m t_1^2 \dots [N_m]_m t_{N_m}^2 \# \\ \dots \\ [1]_m t_1^h \dots [N_m]_m t_{N_m}^h \# \,. \end{gathered}$$

That is, each row of the solution is separated by the # symbol. Between each tile of a row is it's index, encoded using the large number encoding described in the previous section.

The formula S will use a series of replacements and assertions to verify that the tiling encoded by x is a valid solution to the tiling problem. We will give the formula in three steps.

We will define the alphabet to be

$$\Sigma = \Theta \cup \overline{\Theta}$$

where Θ is the set of tiles, and $\overline{\Theta}$ is the set of characters required to encode large numbers, plus #.

The first part is

$$\begin{array}{l}
\text{assert}\left(x \in \left[\left[R^n_m \Theta\right]^* \#\right]^*\right); \\
\text{assert}\left(x \in R^n_m t^0\right); \\
\text{assert}\left(x \in \Sigma^* f \#\right); \\
\text{assert}\left(x \in \left[\left[\left[\sum_{(t_1,t_2)\in H} R^n_m t_1 R^n_m t_2\right]^* \left[R^n_m \Theta\right]^? \#\right]^*\right); \\
\text{assert}\left(x \in \left[\left[R^n_m \Theta\right]\left[\sum_{(t_1,t_2)\in H} R^n_m t_1 R^n_m t_2\right]^* \left[R^n_m \Theta\right]^? \#\right]^*\right);
\end{array}$$

The first asserts simply verify the format of the value of x is as expected and moreover, the first appearing element of Θ in the string is t^0, and the last element is f.

The final two assertions check the horizontal tiling relation. In particular, the first checks that even pairs of tiles are in H, while the second checks odd pairs are in H.

The main challenge is checking the vertical tiling relation. This is done by a series of transductions operating in two main phases. The first phase rewrites the encoding into a kind of large Boolean formula, which is then evaluated in the second phase.

Constructing the Large Boolean Formula. The next phase of the formula is shown below and explained afterwards. For convenience, we will describe the construction using transductions. After the explanation, we will describe how to achieve these transductions using replaceAll.

$$\begin{array}{l}
x^1_m = \mathcal{T}^1_m(x); \\
x^2_m = \mathcal{T}^2_m(x^1_m); \\
x^3_m = \mathcal{T}^3_m(x^2_m); \\
x^1_{m-1} = \mathcal{T}^1_{m-1}(x^3_m); \\
x^2_{m-1} = \mathcal{T}^2_{m-1}(x^1_{m-1}); \\
x^3_{m-1} = \mathcal{T}^3_{m-1}(x^2_{m-1}); \\
\ldots \\
x^1_1 = \mathcal{T}^1_1(x^3_2); \\
x^2_1 = \mathcal{T}^2_1(x^1_1); \\
x^3_1 = \mathcal{T}^3_1(x^2_1); \\
x_0 = \mathcal{T}_0(x^3_1) .
\end{array}$$

The Boolean formula is constructed by rewriting the encoding stored in x. We need to check the vertical tiling relation by comparing t^i_j with t^{i+1}_j. However, these are separated by a huge number of other tiles, which also need to be checked against their counterpart in the next row.

The goal of the transductions is to "rotate" the encoding so that instead of each tile being directly next to its horizontal counterpart, it is directly next to its vertical counterpart. Our transductions do not quite achieve this goal, but instead place the tiles in each row next to potential vertical counterparts. The Boolean formula contains large disjunctions over these possibilities and use the indexing by large numbers to pick out the correct pairs.

The idea is best illustrated by showing the first three transductions, $\mathcal{T}_m^1$, $\mathcal{T}_m^2$, and $\mathcal{T}_m^3$. We start with

$$\begin{gathered}[1]_m t_1^1 \dots [N_m]_m t_{N_m}^1 \# \\ [1]_m t_1^2 \dots [N_m]_m t_{N_m}^2 \# \\ \dots \\ [1]_m t_1^h \dots [N_m]_m t_{N_m}^h \# \,.\end{gathered}$$

The transducer $\mathcal{T}_m^1$ saves the row it is currently reading. Then, when reading the next row, it outputs each index and tile of the current row followed by a copy of the last row. The output is shown below. We use a disjunction symbol to indicate that, after the transduction, the tile should match one of the tiles copied after it. Between each pair of a tile and a copied row, we use the conjunction symbol to indicate that every disjunction should have one match. The result is shown below. To aid readability, we underline the copied rows. The parentheses $\langle\rangle$ are also inserted to aid future parsing.

$$\begin{gathered}\left\langle [1]_m t_1^2 \vee \underline{[1]_m t_1^1 \dots [N_m]_m t_{N_m}^1} \right\rangle \wedge \dots \wedge \left\langle [N_m]_m t_{N_m}^2 \vee \underline{[1]_m t_1^1 \dots [N_m]_m t_{N_m}^1} \right\rangle \\ \wedge \dots \wedge \\ \left\langle [1]_m t_1^h \vee \underline{[1]_m t_1^{h-1} \dots [N_m]_m t_{N_m}^{h-1}} \right\rangle \wedge \dots \wedge \left\langle [N_m]_m t_{N_m}^h \vee \underline{[1]_m t_1^{h-1} \dots [N_m]_m t_{N_m}^{h-1}} \right\rangle \,.\end{gathered}$$

After this transduction, we apply $\mathcal{T}_m^2$. This transduction forms pairs of a tile, with all tiles following it from the previous row (up to the next $\wedge$ symbol). This leaves us with a conjunction of disjunctions of pairs. Inside each disjunct, we need to verify that one pair has matching indices and tiles that satisfy the vertical tiling relation V. The result of the second transduction is shown below.

$$\begin{gathered}\left\langle [1]_m t_1^2 [1]_m t_1^1 \vee \dots \vee [1]_m t_1^2 [N_m]_m t_{N_m}^1 \right\rangle \wedge \dots \wedge \left\langle [N_m]_m t_{N_m}^2 [1]_m t_1^1 \vee \dots \vee [N_m]_m t_{N_m}^2 [N_m]_m t_{N_m}^1 \right\rangle \\ \wedge \dots \wedge \\ \left\langle [1]_m t_1^h [1]_m t_1^{h-1} \vee \dots \vee [1]_m t_1^h [N_m]_m t_{N_m}^{h-1} \right\rangle \wedge \dots \wedge \left\langle [N_m]_m t_{N_m}^h [1]_m t_1^{h-1} \vee \dots \vee [N_m]_m t_{N_m}^h [N_m]_m t_{N_m}^{h-1} \right\rangle \,.\end{gathered}$$

Notice that we now have each tile in a pair with its vertical neighbour, but also in a pair with every other tile in the row beneath. The indices can be used to pick out the right pairs, but we will need further transductions to analyse the encoding of large numbers.

To simplify matters, we apply $\mathcal{T}_m^3$. This transduction removes the tiles from the string, retaining each pair of indices where the tiles satisfy the vertical tiling relation. When the tiling relation is not satisfied, we insert $\perp_m$. We use $\langle$, #, and $\rangle$ to delimit the indices. We are left with a string of the form

$$\bigwedge \bigvee \langle [i]_m \# [j]_m \rangle \vee \perp_m \vee \dots \vee \perp_m \,.$$

We will often elide the $\perp_m$ disjuncts for clarity. They will remain untouched until the formula is evaluated in the next section.

We consider a pair $\langle [i]_m \# [j]_m \rangle$ to evaluate to true whenever $i = j$. The truth of the formula can be computed accordingly. However, it's not straightforward to check whether $i = j$ as they are large numbers. The key observation is that they are encoded as solutions to indexed tiling problems, which means we can go through a similar process to the transductions above.

First, recall that $[i]_m$ is of the form

$$[1]_{m-1} d_1^i [2]_{m-1} d_2^i \dots [N_{m-1}]_{m-1} d_{N_{m-1}}^i$$

where we use d to indicate tiles instead of t.

We apply three transductions $\mathcal{T}^1_{m-1}$, $\mathcal{T}^2_{m-1}$, and $\mathcal{T}^3_{m-1}$. The first copies the first index of each pair directly after the tiles of the second index. That is, each pair

$$\langle [i]_m \# [j]_m \rangle$$

is rewritten to

$$\left\langle [1]_{m-1} d^j_1 \vee [i]_m \right\rangle \wedge \ldots \wedge \left\langle [N_{m-1}]_{m-1} d^j_{N_{m-1}} \vee [i]_m \right\rangle \ .$$

Then we apply a similar second transduction: each disjunction is expanded into pairs of indices and tiles. The result is

$$\begin{gathered} \left\langle [1]_{m-1} d^j_1 [1]_{m-1} d^i_1 \vee \ldots \vee [1]_{m-1} d^j_1 [N_{m-1}]_{m-1} d^i_{N_{m-1}} \right\rangle \\ \wedge \ldots \wedge \\ \left\langle [N_{m-1}]_{m-1} d^j_{N_{m-1}} [1]_{m-1} d^i_1 \vee \ldots \vee [N_{m-1}]_{m-1} d^j_{N_{m-1}} [N_{m-1}]_{m-1} d^i_{N_{m-1}} \right\rangle \ . \end{gathered}$$

The third transduction replaces with $\perp_{m-1}$ all pairs where we don't have $d^j_k = d^i_{k'}$ (recall, we need to check that $i = j$ so the tiles at each position should be the same). As before, for a single pair, this leaves us with a string formula of the form

$$\bigwedge \bigvee \langle [i']_{m-1} \# [j']_{m-1} \rangle \vee \perp_{m-1} \vee \cdots \vee \perp_{m-1} \ .$$

Again, we will elide the $\perp_{m-1}$ disjuncts for clarity as they will be untouched until the formula is evaluated. Recalling that there are many pairs in the input string, the output of this series of transductions is a string formula of the form

$$\bigwedge \bigvee \bigwedge \bigvee \langle [i']_{m-1} \# [j']_{m-1} \rangle \ .$$

We repeat these steps using $\mathcal{T}^1_{m-2}$, $\mathcal{T}^2_{m-2}$, $\mathcal{T}^3_{m-2}$ all the way down to $\mathcal{T}^1_1$, $\mathcal{T}^2_1$, $\mathcal{T}^3_1$. We are left with a string formula of the form

$$\bigwedge \bigvee \cdots \bigwedge \bigvee \langle [i']_1 \# [j']_1 \rangle \ .$$

Recall each $[i']_1$ is of the form

$$d^{i'}_1 \ldots d^{i'}_n \ .$$

The final step interleaves the tiles of the two numbers. The result is a string formula of the form

$$\bigwedge \bigvee \cdots \bigwedge \bigvee \bigwedge dd' \ .$$

This is the formula that is evaluated in the next phase.

To complete this section we need to implement the above transductions using replaceAll.

First, consider $\mathcal{T}^1_m$. We start with

$$\begin{gathered} [1]_m t^1_1 \ldots [N_m]_m t^1_{N_m} \# \\ [1]_m t^2_1 \ldots [N_m]_m t^2_{N_m} \# \\ \ldots \\ [1]_m t^h_1 \ldots [N_m]_m t^h_{N_m} \# \ . \end{gathered}$$

We are aiming for

$$\begin{gathered} \left\langle [1]_m t^2_1 \vee [1]_m t^1_1 \ldots [N_m]_m t^1_{N_m} \right\rangle \wedge \ldots \wedge \left\langle [N_m]_m t^2_{N_m} \vee [1]_m t^1_1 \ldots [N_m]_m t^1_{N_m} \right\rangle \\ \wedge \ldots \wedge \\ \left\langle [1]_m t^h_1 \vee [1]_m t^{h-1}_1 \ldots [N_m]_m t^{h-1}_{N_m} \right\rangle \wedge \ldots \wedge \left\langle [N_m]_m t^h_{N_m} \vee [1]_m t^{h-1}_1 \ldots [N_m]_m t^{h-1}_{N_m} \right\rangle \ . \end{gathered}$$

We use two replaceAlls. The first uses $\$^{\leftarrow}$ to do the main work of copying the previous row into the current row a huge number of times. In fact, $\$^{\leftarrow}$ will copy too much, as it will copy everything that came before, not just the last row. The second replaceAll will cut down the contents of $\$^{\leftarrow}$ to only the last row. That is, we first apply $\text{replaceAll}_{\text{pat}_1,\text{rep}_1}$ and then $\text{replaceAll}_{\text{pat}_2,\text{rep}_2}$ where

$$\begin{array}{rcl}\text{pat}_1 & = & (t) \\ \text{rep}_1 & = & \$1 \triangleleft \$^{\leftarrow} \triangleright\end{array}$$

and $\triangleleft$ and $\triangleright$ are two characters not in Σ, and, letting $\Sigma_{\#} = \Sigma \setminus \{\#\}$,

$$\begin{array}{rcl}\text{pat}_2 & = & \triangleleft\Sigma_{\#}^{*}\#(\Sigma_{\#}^{*})\#\Sigma_{\#}^{*}\triangleright \\ \text{rep}_2 & = & \vee\$1\ .\end{array}$$

That is, the first replace adds after each tile the entire preceding string, delimited by $\triangleleft$ and $\triangleright$. The second replace picks out the final row of each string between $\triangleleft$ and $\triangleright$ and adds the $\vee$. Notice that the second replace does not match anything between $\triangleleft$ and $\triangleright$ on the first row. In fact, we need another replaceAll to delete the first row. That is $\text{replaceAll}_{\text{pat}_3,\text{rep}_3}$ where

$$\begin{array}{rcl}\text{pat}_3 & = & [\Sigma \cup \{\triangleleft, \triangleright\}]^{*} \triangleright \Sigma_{\#}^{*}\# \\ \text{rep}_3 & = & \varepsilon\ .\end{array}$$

Notice, the pattern above matches any row containing at least one $\triangleright$. This means only the first row will be deleted as delimiters have already been removed from the other rows. To complete the step, we replace all # with $\wedge$ and insert the parenthesis $\langle\rangle$ using another replaceAll (and a concatenation at the beginning and the end of the string).

The transduction $\mathcal{T}_m^2$ uses similar techniques to the above and we leave the details to the reader. The same is true of the other similar transductions $\mathcal{T}_i^1$ and $\mathcal{T}_i^2$.

Transduction $\mathcal{T}_m^3$ (and similarly the other $\mathcal{T}_i^2$) replaces all pairs

$$[i]_m t_i^2 [i+1]_m t_{i+1}^1$$

that do not satisfy the vertical tiling relation with $\perp_m$, and rewrites them to

$$\langle [i']_1 \# [j']_1 \rangle$$

if the vertical tiling relation is matched. This can be done in two steps: first replace the non-matches, then replace the matches. To replace the non-matches we use $\text{replaceAll}_{\text{pat}_1,\text{rep}_1}$ where

$$\begin{array}{rcl}\text{pat}_1 & = & \sum\limits_{(t_1,t_2)\notin V} R_m^n t_1 R_m^n t_2 \\ \text{rep}_1 & = & \perp_m\ .\end{array}$$

For the matches we use $\text{replaceAll}_{\text{pat}_2,\text{rep}_2}$ where

$$\begin{array}{rcl}\text{pat}_2 & = & \sum\limits_{(t_1,t_2)\in V} (R_m^n) t_1 (R_m^n) t_2 \\ \text{rep}_1 & = & \langle \$1\#\$2 \rangle\ .\end{array}$$

The final transduction takes a string of the form

$$\bigwedge \bigvee \cdots \bigwedge \bigvee \langle [i']_1 \# [j']_1 \rangle$$

where each $[i']_1$ is of the form

$$d_1^{i'} \ldots d_n^{i'}\ .$$

We need to interleave the tiles of the two numbers, giving a string of the form

$$\bigwedge \bigvee \cdots \bigwedge \bigvee \bigwedge dd'\ .$$

This can be done with a single $\mathsf{replaceAll}_{\mathsf{pat},\mathsf{rep}}$ where

$$\begin{aligned}\mathsf{pat} &= \langle(\Theta_1)\dots(\Theta_1)\#(\Theta_1)\dots(\Theta_1)\rangle \\ \mathsf{rep} &= \langle\$1\$(n+1) \wedge \cdots \wedge \$n\$(2n)\rangle\ .\end{aligned}$$

Evaluating the Large Boolean Formula. The final phase of S evaluates the Boolean formula and is shown below. Again we write the formula using transductions and explain how they can be done with replaceAll.

$$\begin{gathered}x_0^{\wedge} = \mathcal{T}_0(x_0); \\ x_1^{\vee} = \mathcal{T}_0^{\wedge}\left(x_0^{\wedge}\right); \\ x_1^{\wedge} = \mathcal{T}_1^{\vee}\left(x_1^{\vee}\right); \\ x_2^{\vee} = \mathcal{T}_1^{\wedge}\left(x_1^{\wedge}\right); \\ x_2^{\wedge} = \mathcal{T}_2^{\vee}\left(x_2^{\vee}\right); \\ x_3^{\wedge} = \mathcal{T}_2^{\wedge}\left(x_2^{\wedge}\right); \\ \dots \\ x_m^{\vee} = \mathcal{T}_{m-1}^{\wedge}\left(x_{m-1}^{\wedge}\right); \\ x_m^{\wedge} = \mathcal{T}_m^{\vee}\left(x_m^{\vee}\right); \\ x_f = \mathcal{T}_m^{\wedge}\left(x_m^{\vee}\right); \\ \mathsf{assert}\left(x_f \in \mathsf{pat}_f\right)\end{gathered}$$

The first transducer $\mathcal{T}_1$ reads the string formula

$$\bigwedge\bigvee\cdots\bigwedge\bigvee\bigwedge dd'\ .$$

copies it to its output, except replacing each pair dd' with $\top_1$ if $d = d'$ and with $\bot_1$ otherwise. This is requires two simple replaceAll calls.

The remaining transductions evaluate the innermost disjunction or conjunction as appropriate (the parenthesis $\langle\rangle$ are helpful here). For example $\mathcal{T}_1^{\vee}$ replaces the innermost $\bigvee v$ with $\top_1$ if $\top_1$ appears somewhere in the disjunction and $\bot_1$ otherwise. This can be done by greedily matching any sequence of characters from $\{\top_1, \bot_1, \vee\}$ that contains at least one $\top_1$ and replacing the sequence with $\top_1$, then greedily matching any remaining sequence of $\{\bot_1, \vee\}$ and replacing it with $\bot_1$. The evaluation of conjunctions works similarly, but inserts $\top_2$ and $\bot_2$ in the move to the next level of evaluation.

The final assert checks that x_f contains only the character $\top_{m+1}$ and fails otherwise.

This completes the reduction.