Emptiness of Stack Automata is NEXPTIME-complete: A Correction [*]

Christopher Broadbent[1], Arnaud Carayol[2], Matthew Hague[3][†]and Olivier Serre[4]

[1] Institut für Informatik (I7), Technische Universität München
[2] Laboratoire d'informatique de l'Institut Gaspard Monge, Université Paris-Est, and CNRS
[3] Royal Holloway, University of London
[4] IRIF, Université Paris Diderot - Paris 7, and CNRS

Abstract

A saturation algorithm for collapsible pushdown systems was published in ICALP 2012. This work introduced a class of *stack automata* used to recognised regular sets of collapsible pushdown configurations. It was shown that these automata form an effective boolean algebra, have a linear time membership problem, and are equivalent to an alternative automata representation appearing in LICS 2010. It was also claimed that the emptiness problem for stack automata is PSPACE-complete. Unfortunately, this claim is not true. We show that the problem is in fact NEXPTIME-complete when the stacks being accepted are collapsible pushdown stacks, rather than the annotated stacks used in ICALP 2012.

1 Outline

We begin with the preliminaries in Section 2. The complexity of emptiness checking is shown in Section 3.

2 Preliminaries

We give the definition of higher-order collapsible stacks before describing stack automata.

2.1 Higher-Order Collapsible Stacks

Higher-order collapsible stacks are a nested "stack-of-stacks" structure over a stack alphabet Σ. Each stack character contains a pointer — called a "link" — to a position lower down in the stack. The stack operations, defined below, create copies of sub-stacks. The link is intuitively a pointer to the context in which the stack character was first created. These links will be defined as tuples, the meaning of which is expanded upon after the following definition. Let the natural numbers $\mathbb{N}$ be $\{0, 1, 2, \ldots\}$.

Definition 2.1 (Order-n Collapsible Stacks)**.** *An* order-k link *is a tuple* (k, i) *where* $k \geq 1$ *and* i *are natural numbers. If* $k \in \{1, \ldots, n\}$ *we say the link is* up-to *order-*n. *Given a finite set of stack characters* Σ, *an* order-0 stack *with an up-to order-*n *link is* $a^{(k,i)}$ *where* $a \in \Sigma$ *and* (k, i) *is an up-to order-*n *link. An* order-k stack *with up-to order-*n *links is a sequence* $w = [w_1 \ldots w_\ell]_k$ *such that each* w_j *is an order-*$(k-1)$ *stack with up-to order-*n *links. Moreover, for each* w_j *and each order-*k *link* (k, i) *appearing on a character in* w_j, *we have* $i < j$. *Let* $Stacks_n$ *denote the set of order-*n *stacks with up-to order-*n *links.*

1

In the sequel we will refer to order-n stacks with up-to order-n links simply as order-n stacks. We will use order-k stack to mean an order-k stack with up-to order-n links, where n is clear from the context. We define the interpretation of the collapse links below. First, we give an example order-3 stack: $[[[a^{(3,1)}b^{(1,0)}]_1]_2[[c^{(2,1)}]_1[d^{(1,1)}e^{(1,0)}]_1]_2]_3$. Intuitively, the collapse links point to a position lower down in the stack. Hence, we can represent collapse links informally with arrows. Our example stack could be written

$$[\quad [\quad [\quad a \quad b \searrow]_1 \quad]_2 \searrow \quad [\quad [\quad c \quad]_1 \searrow \quad [\quad d \searrow e \searrow]_1 \quad]_2 \quad]_3 \ .$$

The collapse operation, defined below, will remove all parts of the stack above the destination of the topmost collapse link. Collapse on the stack above gives $[[[c^{(2,1)}]_1[d^{(1,1)}e^{(1,0)}]_1]_2]_3$. Note, we will often omit the collapse link annotations for readability.

Given an order-n stack $[w_1 \ldots w_\ell]_n$, we define

$$
\begin{array}{rcll}
top_n([w_1 \ldots w_\ell]_n) & = & w_1 & \text{when } \ell > 0 \\
top_k([w_1 \ldots w_\ell]_n) & = & top_k(w_1) & \text{when } k < n \text{ and } \ell > 0
\end{array}
$$

noting that $top_k(w)$ is undefined if $top_{k'}(w)$ is empty for any $k' > k$. For technical reasons, we also define $top_{n+1}(w) = [w]_{n+1}$ when w is an order-n stack. We remove the top portion of a top_k stack using, where $i > 0$,

$$
\begin{array}{rcll}
bot_n^i([w_1 \ldots w_\ell]_n) & = & [w_{\ell-i+1} \ldots w_\ell]_n & \text{when } i \leq \ell \\
bot_k^i([w_1 \ldots w_\ell]_n) & = & [bot_k^i(w_1)w_2 \ldots w_\ell]_n & \text{when } k < n \text{ and } \ell > 0 \ .
\end{array}
$$

For $top_1(w) = a$ where a has the link (k,i), the destination of the link is $bot_k^i(w)$.

When u is a $(k-1)$-stack and $v = [v_1 \ldots v_\ell]_n$ is an n-stack with $k \leq n$, we define $u :_k v$ as the stack obtained by adding u on top of the topmost k-stack of v. Formally, we let

$$
\begin{array}{rcll}
u :_k v & = & [uv_1 \ldots v_\ell]_n & \text{when } k = n \\
u :_k v & = & [(u :_k v_1)v_2 \ldots v_\ell]_n & \text{when } k < n
\end{array}
$$

2.2   Regularity of Collapsible Stacks

We are interested in regular representations of sets of collapsible pushdown stacks.   For this we use order-n stack automata, thus defining a notion of regular sets of stacks. These have a nested structure based on a similar automata model by Bouajjani and Meyer [1]. The handling of collapse links is similar to automata introduced by Broadbent *et al.* [3], except we read stacks top-down rather than bottom-up.

Definition 2.2 (Order-n Stack Automata)**.** *An order-n stack automaton*

$$A = (\mathbb{Q}_n, \ldots, \mathbb{Q}_1, \Sigma, \Delta_n, \ldots, \Delta_1, \mathcal{F}_n, \ldots, \mathcal{F}_1)$$

is a tuple where Σ is a finite stack alphabet, $\mathbb{Q}_n, \ldots, \mathbb{Q}_1$ are finite disjoint statessets, and

1. *for all $k \in \{2, \ldots, n\}$, we have that $\Delta_k \subseteq \mathbb{Q}_k \times \mathbb{Q}_{k-1} \times 2^{\mathbb{Q}_k}$ is a transition relation, and $\mathcal{F}_k \subseteq \mathbb{Q}_k$ is a set of accepting states, and*

2. *$\Delta_1 \subseteq \bigcup_{2 \leq k \leq n} \left(\mathbb{Q}_1 \times \Sigma \times 2^{\mathbb{Q}_k} \times 2^{\mathbb{Q}_1} \right)$ is a transition relation, and $\mathcal{F}_1 \subseteq \mathbb{Q}_1$ a set of accepting states.*

Stack automata are alternating automata that read the stack in a nested fashion. Order-k stacks are recognised from states in $\mathbb{Q}_k$. A transition $(q, q', Q) \in \Delta_k$ from q to Q for some $k > 1$ can be fired when the top_{k-1} stack is accepted from $q' \in \mathbb{Q}_{(k-1)}$. The remainder of the stack must be accepted from all states in Q. At order-1, a transition (q, a, Q_{col}, Q) is a standard alternating a-transition with the additional requirement that the stack pointed to by the collapse link of a is accepted from all states in Q_{col}. A stack is accepted if a subset of $\mathcal{F}_k$ is reached at the end of each order-k stack. In Section 2.2.1, we formally define the runs of a stack automaton. We write $w \in \mathcal{L}_q(A)$ whenever w is accepted from a state q. For ease of presentation, we write $q \xrightarrow{q'} Q \in \Delta_k$ instead of $(q, q', Q) \in \Delta_k$ and $q \xrightarrow[Q_{col}]{a} Q \in \Delta_1$ instead of $(q, a, Q_{col}, Q) \in \Delta_1$.

A (partial) run is informally pictured below, reading an order-3 stack using $q_3 \xrightarrow{q_2} Q_3 \in \Delta_3, q_2 \xrightarrow{q_1} Q_2 \in \Delta_2$ and $q_1 \xrightarrow[Q_{col}]{a} Q_1 \in \Delta_1$. Note, the transition $q_3 \xrightarrow{q_2} Q_3$ reads the topmost order-2 stack, with the remainder of the stack being read from Q_3. The node labelled Q_{col} begins a run on the stack pointed to by the collapse link of a. Note that the label of this node may contain other elements apart from Q_{col}. These additional elements come from the part of the run coming from the previous node (and other collapse links).

2.2.1   Formal Definition of a Run

We begin by defining the set of substacks of a stack, which are intuitively all suffixes of the stack.

Definition 2.3 (Subs(w))**.** *Given an order-n stack w, we denote by* Subs(w) *the smallest set of stacks such that $w \in$ Subs(w) and if $u :_k v \in$ Subs(w) for some $k \in \{1, \ldots, n\}$ then $v \in$ Subs(w).*

A stack automaton is essentially a stack- and collapse-aware alternating automaton, where collapse links are treated as special cases of the alternation. Fix a stack automaton

$$A = (\mathbb{Q}_n, \ldots, \mathbb{Q}_1, \Sigma, \Delta_n, \ldots, \Delta_1, \mathcal{F}_n, \ldots, \mathcal{F}_1) \ .$$

More formally, a run of a stack automaton over an order-n stack w associates to each stack $v \in$ Subs(w) at most one set of states Q_k per $k \in \{1, \ldots, n\}$. The run is accepting if the following conditions are met.

- There is an order-n stateset $Q_n \subseteq \mathbb{Q}_n$ associated with $[]_n \in$ Subs(w) and $Q_n \subseteq \mathcal{F}_n$.

- If $k \in \{1, \ldots, n-1\}$ and $\big([]_k :_{(k+1)} v\big) \in$ Subs(w) then this stack is associated with an order-k stateset $Q_k \subseteq \mathcal{F}_k$.

- Each $\big(a^{(k,i)} :_1 v\big) \in$ Subs(w) is associated with an order-1 stateset $Q_1 \subseteq \mathbb{Q}_1$ with v associated with an order-1 stateset $Q_1' \subseteq \mathbb{Q}_1$ and $bot_n^i\big(a^{(k,i)} :_1 v\big)$ associated with an order-k stateset $Q_k \subseteq \mathbb{Q}_k$ such that for each $q_1 \in Q_1$ there is a transition $q_1 \xrightarrow[Q_{col}]{a} Q \in \Delta_1$ such that $Q_{col} \subseteq Q_k$ and $Q \subseteq Q_1'$.

- Each $u :_k v \in$ Subs(w) with $k \in \{2, \ldots, n\}$ is associated with an order-k stateset $Q_k \subseteq \mathbb{Q}_k$ and an order-$(k-1)$ stateset $Q_{k-1} \subseteq \mathbb{Q}_{k-1}$ and v is associated with an order-k stateset

$Q'_k \subseteq \mathbb{Q}_{k-1}$ such that for each $q_k \in Q_k$ there is a transition $q_k \xrightarrow{q_{k-1}} Q \in \Delta_k$ such that $q_{k-1} \in Q_{k-1}$ and $Q \subseteq Q'_k$.

We write $w \in \mathcal{L}_q(A)$ to denote that the order-n stack w is accepted by A from $q \in \mathbb{Q}_k$ for some k. Similarly, for $Q \subseteq \mathbb{Q}_k$ for some k we write $w \in \mathcal{L}_Q(A)$ when w is accepted from each $q \in Q$. Note, if $Q = \emptyset$ then all stacks are accepted.

3 Emptiness of Stack Automata

In ICALP 2012 [2] we incorrectly stated that the emptiness problem for stack automata was PSPACE-complete. As pointed out by an anonymous reviewer, the algorithm given actually runs in PTIME and does not correctly implement the emptiness test. We show that the problem is, in fact, NEXPTIME-complete.

Theorem 3.1. *Let $n \geq w$ and A be an order-n stack automaton. Testing whether there exists an order-n collapsible pushdown stack w such that $w \in \mathcal{L}_q(A)$ for a given state q of A is NEXPTIME-complete.*

The above theorem is proved in Section 3.1 and Section 3.2 below.

3.1 Upper Bound

The upper bound can be obtained quite easily. We know from ICALP 2012 that stack automata are equivalent to the bottom-up automata introduced by Broadbent *et al.* [3]. More formally, we have the following proposition. The complexity is apparent from the proof presented in the paper.

Proposition 3.1 ([3], as Proposition 4). *For every order-n stack automaton A with initial state q, there is a bottom-up stack automaton B of size exponential in the size of A with initial state q' such that $\mathcal{L}_q(A) = \mathcal{L}_{q'}(B)$.*

Then, from Broadbent *et al.* [3], we know the emptiness problem for bottom-up stack automata is NP-complete. When applied to an exponentially large automaton, this gives us NEXPTIME as required.

Proposition 3.2 ([3], as Proposition 2). *Given fixed $n \geq 2$ and some automaton B, deciding whether there exists some order-n collapsible stack that it accepts is NP-complete.*

In conclusion, we have the following proposition.

Proposition 3.3. *Emptiness checking of order-n stack automata is in NEXPTIME.*

3.2 Lower Bound

The lower bound is by reduction from a tiling problem over a $2^\ell \times 2^\ell$ grid. It is known that, when ℓ is given in unary, there is a fixed tiling problem for which the problem in NEXPTIME-hard in the size of ℓ [4]. We begin by recalling the definition of a tiling problem before giving the reduction.

3.2.1 Tiling Problems

Definition 3.1 (Tiling Problem). *A tiling problem is a tuple* (Θ, H, V, t_I, t_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* $t_I, t_F \in \Theta$ *are initial and final tiles respectively.*

A solution to a tiling problem over a N-width and N-height corridor is a sequence

$$
\begin{array}{c}
t_1^1 \ldots t_N^1 \\
t_1^2 \ldots t_N^2 \\
\ldots \\
t_1^N \ldots t_N^N
\end{array}
$$

where $t_1^1 = t_I$, $t_N^N = t_F$, and for all $1 \leq i < N$ and $1 \leq j \leq N$ 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, the grid layout is for presentation purposes only, and the sequence should truly be written $t_1^1 \ldots t_N^1 t_1^2 \ldots t_N^2 \ldots t_1^N \ldots t_N^N$. We will assume that t_I and t_F can only appear at the beginning and end of the tiling respectively.

In the sequel we will fix a tiling problem (Θ, H, V, t_I, t_F) such that for any ℓ (in unary) finding a solution to the problem over a N-width and N-height corridor where $N = 2^\ell$ is NEXPTIME-hard [4].

3.2.2 Reduction to Stack Automata Emptiness

We first describe the shape of the stacks we wish to see, given a tiling problem as fixed above. Then we will show how to build a stack automaton which recognises such stacks which encode solutions to the tiling problem. For technical convenience, given ℓ in unary, we define $N = 2^\ell - 1$.

Encoding Solutions as Stacks We will define an order-2 stack automaton that will only accept stacks of the following form. The notation $\langle i \rangle$ for $0 \leq i \leq N$ is the ℓ-bit encoding of i, most significant bit first, using the stack characters $\hat{1}$ and $\hat{0}$. Collapse links are drawn as arrows and only appear if necessary. Characters without explicit links may have any valid link as they are not needed for the encoding. Note, the grid structure is for presentation only and the stack should be read as a sequence of order-1 stacks, from left-to-right and top-to-bottom.

That is, the stack begins with three stacks containing only a spacer character #. This is merely for technical reasons as it allows alternating transitions later to get started. On the next row in the diagram we have an order-1 stack for each tile position on the first row of the solution. The next row has the next row of the solution and so on. Each order-1 stack encodes a position as follows. The topmost character is another spacer # which contains a collapse

link. The collapse link is order-2 and points to the order-1 stack containing the tile vertically below. Note, this stack is not constructible using the standard stack operations, but the stack still meets the definition of a collapsible stack. After the # there are two numbers, giving the row index and column index respectively. These numbers are encoded as ℓ-bit binary numbers using the characters $\hat{1}$ and $\hat{0}$ and appearing with the most significant bit first. For example $\langle 0 \rangle \langle 1 \rangle$ appears on the stack as

$$\underbrace{\hat{0} \ldots \hat{0} \hat{0}}_{\ell \text{ bits}} \underbrace{\hat{0} \ldots \hat{0} \hat{1}}_{\ell \text{ bits}} .$$

Finally, each $t_{i,j} \in \Theta$ is a tile.

Recognising Tiling Solutions   We will define an order-2 stack automaton which only accepts valid encodings of solutions to the tiling problem. There are several key properties to assert.

1. The stack contains a sequence of order-1 stacks, where the topmost three stacks contains only a spacer and subsequent stacks are of the form

$$\{\#\} \left\{\hat{0}, \hat{1}\right\}^{2 \cdot \ell} \Theta .$$

2. The fourth topmost order-1 stack contains $\langle 0 \rangle \langle 0 \rangle$.

3. The bottommost order-1 stack contains $\langle N \rangle \langle N \rangle$.

4. For all but the first three and bottommost order-1 stacks, if the stack contains $\langle i \rangle \langle j \rangle$ then the order-1 stack beneath it contains $\langle i \rangle \langle j+1 \rangle$ if $j < N$ and $\langle i+1 \rangle \langle 0 \rangle$ if $j = N$.

5. For every order-1 stack containing $\langle i \rangle \langle j \rangle$ with $i < N$ the collapse link from # in the stack leads to an order-1 stack containing $\langle i+1 \rangle \langle j \rangle$.

6. We have $t_{0,0} = t_I$.

7. We have $t_{N,N} = t_F$.

8. For every $0 \le i \le N$ and $0 \le j < N$ we have $(t_{i,j}, t_{i,j+1}) \in H$.

9. For every $0 \le i < N$ and $0 \le j \le N$ we have $(t_{i,j}, t_{i+1,j}) \in V$.

There is a stack satisfying the above properties iff there is a solution to the tiling problem. At this point, the experienced reader may see how alternating automata can be used to enforce the above properties, as the manipulation of short binary numbers is fairly standard.

   Note, if we did not assert Property 4 we could have multiple stacks each with the same index $\langle i \rangle \langle j \rangle$. If this were the case then Property 5 would not ensure that the collapse links encoded a grid. Thus, our encoding will crucially rely on the width and height being fixed. That is, our encoding will not extend to EXPSPACE Turing machines which may have an unbounded corridor height.

   We define an order-2 stack automaton recognising only such stacks. In particular, we have

$$A = (\mathbb{Q}_2, \mathbb{Q}_1, \Sigma, \Delta_2, \Delta_1, \mathcal{F}_2, \mathcal{F}_1)$$

where the alphabet, states, and transitions are defined during the description below. We simultaneously argue for the correctness of the definition.

Proposition 3.4. *The tiling problem* (Θ, H, V, t_I, t_F) *has a solution over a* $2^\ell \times 2^\ell$ *corridor iff the order-2 stack automaton A is non-empty.*

The alphabet is $\Sigma = \{\#, \hat{0}, \hat{1}\} \cup \Theta$. The only accepting states are

$$\mathcal{F}_2 = \{q_f^2\} \subset \mathbb{Q}_2 \qquad \text{and} \qquad \mathcal{F}_1 = \{q_f^1\} \subset \mathbb{Q}_1 \ .$$

The initial state is $q_I \in \mathbb{Q}_2$ and there is a single transition from it which leads to a state for each property above:

$$q_I \xrightarrow{q_\#} \{p_1, \ldots, p_9\} \in \Delta_2$$

where $q_\# \in \mathbb{Q}_1$ and $p_1, \ldots, p_9 \in \mathbb{Q}_2$. From $q_\#$ we simply check the stack contains only a spacer:

$$q_\# \xrightarrow[\emptyset]{\#} \{q_f^1\} \ .$$

We also have a state $q_* \in \mathbb{Q}_1$ that accepts any order-1 stack. From it there is a transition

$$q_* \xrightarrow[\emptyset]{\#} \emptyset \in \Delta_1 \ .$$

The remaining properties are more involved.

1. To check that the stack is of the right basic shape, we have $p_1^1, p_1^2 \in \mathbb{Q}_2$ with

$$p_1 \xrightarrow{q_\#} \{p_1^1\}, p_1^1 \xrightarrow{q_\#} \{p_1^2\} \in \Delta_2$$

   to recognise the leading spacer stacks, and

$$p_1^2 \xrightarrow{q_S} \{p_1^2\}, p_1^2 \xrightarrow{q_S} \{q_f^2\} \in \Delta_2$$

   to recognise the remaining stacks, where q_S is a state for asserting the correct shape of order-1 stacks. Observe that the transition to q_f^2 can only be and must be used to read the bottommost order-1 stack in any accepting run.

   In particular, we have $q_S, q_B^0, \ldots, q_B^{2\cdot\ell} \in \mathbb{Q}_1$ where we first assert the leading spacer

$$q_S \xrightarrow[\emptyset]{\#} \{q_B^{2\cdot\ell}\} \in \Delta_1$$

   and then that we have $2 \cdot \ell$ bits with

$$q_B^i \xrightarrow[\emptyset]{\hat{0}} \{q_B^{i-1}\}, q_B^i \xrightarrow[\emptyset]{\hat{1}} \{q_B^{i-1}\} \in \Delta_1$$

   for all $1 \leq i \leq 2 \cdot \ell$ and finally a trailing tile with

$$q_B^0 \xrightarrow[\emptyset]{t} \{q_f^1\} \in \Delta_1$$

   for all $t \in \Theta$.

2. To check the topmost non-spacer stack contains $\langle 0 \rangle \langle 0 \rangle$ we have $p_2, p_2^1, p_2^2 \in \mathbb{Q}_2$ and

$$p_2 \xrightarrow{q_*} \{p_2^1\}, p_2^1 \xrightarrow{q_*} \{p_2^2\}, p_2^2 \xrightarrow{q_{\langle 0 \rangle \langle 0 \rangle}} \{q_f^2\} \in \Delta_2$$

   where, to check the order-1 stack we have $q_{\langle 0 \rangle \langle 0 \rangle}, q_{\langle 0 \rangle \langle 0 \rangle}^1, \ldots, q_{\langle 0 \rangle \langle 0 \rangle}^{2\cdot\ell} \in \mathbb{Q}_1$ and, for all $2 \leq i \leq 2 \cdot \ell$,

$$q_{\langle 0 \rangle \langle 0 \rangle} \xrightarrow[\emptyset]{\#} \{q_{\langle 0 \rangle \langle 0 \rangle}^{2\cdot\ell}\}, q_{\langle 0 \rangle \langle 0 \rangle}^i \xrightarrow[\emptyset]{\hat{0}} \{q_{\langle 0 \rangle \langle 0 \rangle}^{i-1}\}, q_{\langle 0 \rangle \langle 0 \rangle}^1 \xrightarrow[\emptyset]{\hat{0}} \emptyset \in \Delta_1 \ .$$

7

3. To check the bottommost order-1 stack contains $\langle N \rangle \langle N \rangle$ we have $p_3 \in \mathbb{Q}_2$ and

$$p_3 \xrightarrow{q_*} \{p_3\}, p_3 \xrightarrow{q_{\langle N \rangle \langle N \rangle}} \{q_f^2\} \in \Delta_2$$

where, to check the order-1 stack we have $q_{\langle N \rangle \langle N \rangle}, q^1_{\langle N \rangle \langle N \rangle}, \ldots, q^{2 \cdot \ell}_{\langle N \rangle \langle N \rangle} \in \mathbb{Q}_1$ and, for all $2 \leq i \leq 2 \cdot \ell$,

$$q_{\langle N \rangle \langle N \rangle} \xrightarrow[\emptyset]{\#} \left\{ q^{2 \cdot \ell}_{\langle N \rangle \langle N \rangle} \right\}, q^i_{\langle N \rangle \langle N \rangle} \xrightarrow[\emptyset]{\hat{1}} \left\{ q^{i-1}_{\langle N \rangle \langle N \rangle} \right\}, q^1_{\langle N \rangle \langle N \rangle} \xrightarrow[\emptyset]{\hat{1}} \emptyset \in \Delta_1 \ .$$

4. To check the sequence of binary numbers runs from $\langle 0 \rangle \langle 0 \rangle$ to $\langle N \rangle \langle N \rangle$ first notice that we can consider $\langle i \rangle \langle j \rangle$ as a $(2 \cdot \ell)$-bit binary number. In this case each stack contains the successor of the stack above it.

   We will make use of the following auxilliary states in $\mathbb{Q}_2$. That is, for all $1 \leq i \leq 2 \cdot \ell$ we have $q2_i^0, q2_i^1 \in \mathbb{Q}_2$. Note, the choice of notation here indicates that we are dealing with $(2 \cdot \ell)$-bit numbers instead of ℓ-bit numbers. We will need similar states for ℓ-bit numbers later.

   Intuitively, these states are used to check that binary encodings succeed each other. Take the two numbers 10011 and 10100. We will number bits from right to left, starting at 1. Hence, in the first number, the first and second bits are 1, the third is 0, the fourth is 0, and the fifth is 1. Note the latter binary number is the increment of the former. This can be identified by noticing that the rightmost 0 is bit 3. In the incremented number, bit 3 is now a 1 and all bits to the right are now 0. Thus, $q2_i^0$ will assert that the ith bit must be the rigthmost 0, and $q2_i^1$ will assert that the rightmost 1 must be the ith bit. Now, if one order-1 stack appears directly below the other, the upper stack will be accepted by some $q2_i^0$ and the lower stack will be accepted from the corresponding $q2_i^1$.

   We will also need to make use of states $q_=^1, \ldots, q_=^{2 \cdot \ell} \in \mathbb{Q}_2$ to assert the equality of bits at each position in the binary numbers between the topmost two stacks. From these we will guess what the value of the bits are by moving to states $q_=^{i,\hat{0}}, q_=^{i,\hat{1}} \in \mathbb{Q}_2$ for all $1 \leq i \leq 2 \cdot \ell$.

   The reader may notice here that there is some delay in setting up the right states. For example, to check the equality of bits 1 and 2, the automaton first needs to read a stack and move to $q_=^1$ and $q_=^2$, and then needs to guess the bit values (say $\hat{0}$) and then, while reading another stack, move to $q_=^{1,\hat{0}}$ and $q_=^{2,\hat{0}}$. This is why we have leading stacks only containing a spacer, to allow the look ahead to wind up. Thus, as well as checking the immediate stacks, there will be some states for checking the stacks ahead.

   Thus, for each $1 \leq i \leq 2 \cdot \ell$ we have from p_4 the transition

$$p_4 \xrightarrow{q_*} \left\{ p_4, q2_i^0, q_=^{i+1}, \ldots, q_=^{2 \cdot \ell} \right\} \in \Delta_2 \ .$$

   Notice, there are only polynomially many such transitions. The state p_4 appears to assert the sequence property of the next stack. Next, we make another set of transitions to guess the value of the bits to be tested by $q_=^i$. For this we have, for all $1 \leq i \leq 2 \cdot \ell$,

$$q2_i^0 \xrightarrow{q_*} \left\{ q'2_i^0 \right\}, q_=^i \xrightarrow{q_*} \left\{ q_=^{i,\hat{0}} \right\}, q_=^i \xrightarrow{q_*} \left\{ q_=^{i,\hat{1}} \right\} \in \Delta_2$$

   with $q'2_i^0 \in \mathbb{Q}_2$.

8

We are now ready to begin testing that the stacks are in sequence. That is, we check the current and succeeding stack all agree on the bits asserted by the states $q_=^{i,\hat{0}}$ and $q_=^{i,\hat{1}}$ and the increment of the rightmost $\hat{0}$ is done correctly. For this we have the transitions

$$q'2_i^0 \xrightarrow{z_i} \{q2_i^1\}, q_=^{i,\hat{0}} \xrightarrow{b_i^{\hat{0}}} \left\{q_{=,\$}^{i,\hat{0}}\right\}, q_=^{i,\hat{1}} \xrightarrow{b_i^{\hat{1}}} \left\{q_{=,\$}^{i,\hat{1}}\right\} \in \Delta_2$$

for all $1 \leq i \leq 2 \cdot \ell$ with $q_{=,\$}^{i,\hat{0}}, q_{=,\$}^{i,\hat{1}} \in \mathbb{Q}_2$. We delay the description of the order-1 states until we are finished at order-2 (see below). To complete the treatment at order-2 we have transitions that check the successor stack. These are, for all $1 \leq i \leq 2 \cdot \ell$,

$$q2_i^1 \xrightarrow{o_i} \emptyset, q_{=,\$}^{i,\hat{0}} \xrightarrow{b_i^{\hat{0}}} \emptyset, q_{=,\$}^{i,\hat{1}} \xrightarrow{b_i^{\hat{1}}} \emptyset \in \Delta_2 .$$

We also need transitions which allow the automaton to leave the bottommost stack unchecked (it is checked in the case above). For this we allow all states that should check a stack to simply move to the final state. That is,

$$p_4 \xrightarrow{q_*} \{q_f^2\}, q2_i^0 \xrightarrow{q_*} \{q_f^2\}, q'2_i^0 \xrightarrow{q_*} \{q_f^2\}, q_=^i \xrightarrow{q_*} \{q_f^2\}, q_=^{i,\hat{0}} \xrightarrow{q_*} \{q_f^2\}, q_=^{i,\hat{1}} \xrightarrow{q_*} \{q_f^2\} \in \Delta_2$$

for all $1 \leq i \leq 2 \cdot \ell$.

There are several properties the above transitions need to assert at order-1. First we have the states $z_i, z_{i,j} \in \mathbb{Q}_1$ for all $1 \leq i, j \leq 2 \cdot \ell$. These states check the rightmost $\hat{0}$ appears at position i. We first skip the leading spacer with

$$z_i \xrightarrow[\emptyset]{\#} \{z_{i,2\cdot\ell}\} \in \Delta_1 .$$

Then we have, for all $1 \leq i < j \leq 2 \cdot \ell$,

$$z_{i,j} \xrightarrow[\emptyset]{\hat{0}} \{z_{i,j-1}\}, z_{i,j} \xrightarrow[\emptyset]{\hat{1}} \{z_{i,j-1}\}, \in \Delta_1$$

and for all $1 < i \leq 2 \cdot \ell$

$$z_{i,i} \xrightarrow[\emptyset]{\hat{0}} \{z_{i,i-1}\}, z_{1,1} \xrightarrow[\emptyset]{\hat{0}} \emptyset \in \Delta_1$$

and for all $2 \leq j < i \leq 2 \cdot \ell$

$$z_{i,j} \xrightarrow[\emptyset]{\hat{1}} \{z_{i,j-1}\}, z_{i,1} \xrightarrow[\emptyset]{\hat{1}} \emptyset, z_{2,1} \xrightarrow[\emptyset]{\hat{1}} \emptyset \in \Delta_1 .$$

Similarly, we check the rightmost $\hat{1}$ bit with $o_i, o_{i,j} \in \mathbb{Q}_1$ for all $1 \leq i, j \leq 2 \cdot \ell$. We first skip the leading spacer with

$$o_i \xrightarrow[\emptyset]{\#} \{o_{i,2\cdot\ell}\} \in \Delta_1 .$$

Then we have, for all $1 \leq i < j \leq 2 \cdot \ell$,

$$o_{i,j} \xrightarrow[\emptyset]{\hat{0}} \{o_{i,j-1}\}, o_{i,j} \xrightarrow[\emptyset]{\hat{1}} \{o_{i,j-1}\}, \in \Delta_1$$

and for all $1 < i \leq 2 \cdot \ell$

$$o_{i,i} \xrightarrow[\emptyset]{\hat{1}} \{o_{i,i-1}\}, o_{1,1} \xrightarrow[\emptyset]{\hat{1}} \emptyset \in \Delta_1$$

and for all $2 \leq j < i \leq 2 \cdot \ell$

$$o_{i,j} \xrightarrow[\emptyset]{\hat{0}} \{o_{i,j-1}\}, o_{i,1} \xrightarrow[\emptyset]{\hat{0}} \emptyset, o_{2,1} \xrightarrow[\emptyset]{\hat{0}} \emptyset \in \Delta_1 .$$

Next, we check that the ith bit is a $\hat{0}$ with the states $b_i^{\hat{0}}, b_{i,j}^{\hat{0}} \in \mathbb{Q}_1$ for all $1 \leq i \leq j \leq 2 \cdot \ell$. We first skip the leading spacer with

$$b_i^{\hat{0}} \xrightarrow[\emptyset]{\#} \left\{ b_{i,2 \cdot \ell}^{\hat{0}} \right\} \in \Delta_1 .$$

Then we have, for all $1 \leq i < j \leq 2 \cdot \ell$,

$$b_{i,j}^{\hat{0}} \xrightarrow[\emptyset]{\hat{0}} \left\{ b_{i,j-1}^{\hat{0}} \right\}, b_{i,j}^{\hat{0}} \xrightarrow[\emptyset]{\hat{1}} \left\{ b_{i,j-1}^{\hat{0}} \right\}, \in \Delta_1$$

and for all $1 \leq i \leq 2 \cdot \ell$

$$b_{i,i}^{\hat{0}} \xrightarrow[\emptyset]{\hat{0}} \emptyset \in \Delta_1 .$$

Similarly for $\hat{1}$ and $b_i^{\hat{1}}$.

5. To check that the collapse links form a grid, we follow a similar strategy to the previous case, with some differences. Aside from the change in the handling of successors, the main change is that instead of checking adjacent order-1 stacks, the pairs of stacks we check are separated by collapse links. Thus we have to be able to send order-2 states through the collapse links from where they can assert properties of the linked-to order-1 stack. For this we introduce states of the form $[q]_\# \in \mathbb{Q}_1$ for all $q \in \mathbb{Q}_2$. From these states we have only the transition

$$[q]_\# \xrightarrow[\{q\}]{\#} \emptyset .$$

Recall, when following collapse links we need to move from a stack containing $\langle i \rangle \langle j \rangle$ to $\langle i+1 \rangle \langle j \rangle$. We use a slight modification of the previous strategy for testing the increment of i, except we restrict the checks to the leftmost ℓ bits. The equality checks are almost the same, except we have to move through the collapse links instead of down the stack.

Thus, similar to $q2_i^0$ and $q2_i^1$ we use for all $\ell < i \leq 2 \cdot \ell$ the states $q1_i^0, q1_i^1 \in \mathbb{Q}_2$ to check the rightmost $\hat{0}$ or $\hat{1}$ respectively in the leftmost ℓ bits appears at position i. We introduce as well $q1_{i,l}^1 \in \mathbb{Q}_2$ to pass $q1_i^1$ through the link. We also have $q_{=,l}^i \in \mathbb{Q}_2$ for all $1 \leq i \leq 2 \cdot \ell$ to verify the equality of bit positions through links.

As in the previous case, we wind up the alternation with the following transitions

$$p_5 \xrightarrow{q_*} \left\{ p_5, q_{=,l}^1, \ldots q_{=,l}^\ell, q1_i^0, q_{=,l}^{i+1}, \ldots, q_{=,l}^{2 \cdot \ell} \right\} \in \Delta_2$$

for all $\ell < i \leq 2 \cdot \ell$ Notice again there are only a polynomial number of such transitions. Here, p_5 appears on the right to verify the property at the next grid position (hence we verify all positions). We check the equality of the first ℓ bits since these encode the column index which has to be equal. The remaining states verify that the row position is incremented correctly through the links.

Next, we take one more step to guess the values of the bit positions being tested for equality. Thus we have transitions

$$q_{=,l}^i \xrightarrow{q_*} \left\{ q_{=,l}^{i,\hat{0}}, q_{=,l,l}^{i,\hat{0}} \right\}, q_{=,l}^i \xrightarrow{q_*} \left\{ q_{=,l}^{i,\hat{1}}, q_{=,l,l}^{i,\hat{1}} \right\} \in \Delta_2$$

for all $1 \leq i \leq 2 \cdot \ell$ where $q_{=,l}^{i,\hat{0}}, q_{=,l,l}^{i,\hat{0}}, q_{=,l}^{i,\hat{1}}, q_{=,l,l}^{i,\hat{1}} \in \mathbb{Q}_2$. Note, the states like $q_{=,l,l}^{i,\hat{0}}$ are used to pass the equals check through the link as will be seen below. At the same time we fire

$$q1_i^0 \xrightarrow{q_*} \left\{ q'1_i^0, q1_{i,l}^1 \right\} \in \Delta_2$$

where $q'1_i^0 \in \mathbb{Q}_2$.

Now we do the checking. To do the bit equality checks we have similar transitions to previously. That is for all $1 \leq i \leq 2 \cdot \ell$

$$q_{=,l}^{i,\hat{0}} \xrightarrow{b_i^{\hat{0}}} \emptyset, q_{=,l}^{i,\hat{1}} \xrightarrow{b_i^{\hat{1}}} \emptyset \in \Delta_2$$

and to pass the checks through to the linked stacks we have

$$q_{=,l,l}^{i,\hat{0}} \xrightarrow{\left[q_{=,l,\$}^{i,\hat{0}} \right]_\#} \emptyset, q_{=,l,l}^{i,\hat{1}} \xrightarrow{\left[q_{=,l,\$}^{i,\hat{1}} \right]_\#} \emptyset \in \Delta_2$$

and once the states have been passed through the links we have

$$q_{=,l,\$}^{i,\hat{0}} \xrightarrow{b_i^{\hat{0}}} \emptyset, q_{=,l,\$}^{i,\hat{1}} \xrightarrow{b_i^{\hat{1}}} \emptyset \in \Delta_2 \ .$$

Next, to check the increment we need for each $\ell < i \leq 2 \cdot \ell$

$$q'1_i^0 \xrightarrow{z_i'} \emptyset, q1_{i,l}^1 \xrightarrow{\left[q1_i^1 \right]_\#} \emptyset \in \Delta_2$$

and once the state has been passed through the link we have

$$q1_i^1 \xrightarrow{o_i'} \emptyset \in \Delta_2$$

where the order-1 states are described after we complete our description of the order-2 transitions.

To complete the order-2 description we need a way to handle the final row of the grid. For this we use a state $q_{\langle N \rangle \langle * \rangle} \in \mathbb{Q}_1$ which asserts the part of the binary number encoding the row consists of only $\hat{1}$ characters. We give its transitions below. Thus, the above states can avoid checking a stack by asserting it appears on the final row. That is

$$p_5 \xrightarrow{q_{\langle N \rangle \langle * \rangle}} \emptyset, q1_i^0 \xrightarrow{q_{\langle N \rangle \langle * \rangle}} \emptyset, q'1_i^0 \xrightarrow{q_{\langle N \rangle \langle * \rangle}} \emptyset,$$
$$q_{=,l}^{i} \xrightarrow{q_{\langle N \rangle \langle * \rangle}} \emptyset, q_{=,l}^{i,\hat{0}} \xrightarrow{q_{\langle N \rangle \langle * \rangle}} \emptyset, q_{=,l}^{i,\hat{1}} \xrightarrow{q_{\langle N \rangle \langle * \rangle}} \emptyset, q_{=,l,l}^{i,\hat{0}} \xrightarrow{q_{\langle N \rangle \langle * \rangle}} \emptyset, q_{=,l,l}^{i,\hat{1}} \xrightarrow{q_{\langle N \rangle \langle * \rangle}} \emptyset \in \Delta_2$$

for all $1 \leq i \leq 2 \cdot \ell$.

It remains to check several properties the above transitions need to assert at order-1. We have the states $z_i', z_{i,j}' \in \mathbb{Q}_1$ for all $\ell < i, j \leq 2 \cdot \ell$. These states check the rightmost $\hat{0}$ appears at position i in the encoding of the row number. We first skip the leading spacer with

$$z_i' \xrightarrow[\emptyset]{\#} \left\{ z_{i,2\cdot\ell}' \right\} \in \Delta_1 \ .$$

Then we have, for all $\ell < i < j \leq 2 \cdot \ell$,

$$z_{i,j}' \xrightarrow[\emptyset]{\hat{0}} \left\{ z_{i,j-1}' \right\}, z_{i,j}' \xrightarrow[\emptyset]{\hat{1}} \left\{ z_{i,j-1}' \right\}, \in \Delta_1$$

and for all $\ell + 1 < i \leq 2 \cdot \ell$

$$z'_{i,i} \xrightarrow[\emptyset]{\hat{0}} \{z'_{i,i-1}\}, z'_{\ell+1,\ell+1} \xrightarrow[\emptyset]{\hat{0}} \emptyset \in \Delta_1$$

and for all $\ell + 1 < j < i \leq 2 \cdot \ell$

$$z'_{i,j} \xrightarrow[\emptyset]{\hat{1}} \{z'_{i,j-1}\}, z'_{i,\ell+1} \xrightarrow[\emptyset]{\hat{1}} \emptyset, z_{\ell+2,\ell+1} \xrightarrow[\emptyset]{\hat{1}} \emptyset \in \Delta_1 .$$

Similarly, we check the rightmost $\hat{1}$ bit with $o'_i, o'_{i,j} \in \mathbb{Q}_1$ for all $\ell < i, j \leq 2 \cdot \ell$. We first skip the leading spacer with

$$o'_i \xrightarrow[\emptyset]{\#} \{o'_{i,2\cdot\ell}\} \in \Delta_1 .$$

Then we have, for all $\ell < i < j \leq 2 \cdot \ell$,

$$o'_{i,j} \xrightarrow[\emptyset]{\hat{0}} \{o'_{i,j-1}\}, o'_{i,j} \xrightarrow[\emptyset]{\hat{1}} \{o'_{i,j-1}\}, \in \Delta_1$$

and for all $\ell + 1 < i \leq 2 \cdot \ell$

$$o'_{i,i} \xrightarrow[\emptyset]{\hat{1}} \{o_{i,i-1}\}, o'_{\ell+1,\ell+1} \xrightarrow[\emptyset]{\hat{1}} \emptyset \in \Delta_1$$

and for all $\ell + 2 < j < i \leq 2 \cdot \ell$

$$o'_{i,j} \xrightarrow[\emptyset]{\hat{0}} \{o'_{i,j-1}\}, o'_{i,\ell+1} \xrightarrow[\emptyset]{\hat{0}} \emptyset, o_{\ell+2,\ell+1} \xrightarrow[\emptyset]{\hat{0}} \emptyset \in \Delta_1 .$$

It only remains to define the transitions from $q_{\langle N \rangle \langle * \rangle}$. For this we introduce several intermediate states $q_{\langle N \rangle \langle * \rangle}, q_{\langle N \rangle \langle * \rangle}^{\ell+1}, \ldots, q_{\langle N \rangle \langle * \rangle}^{2\cdot\ell} \in \mathbb{Q}_1$ and the transitions

$$q_{\langle N \rangle \langle * \rangle} \xrightarrow[\emptyset]{\#} \{q_{\langle N \rangle \langle * \rangle}^{2\cdot\ell}\}, q_{\langle N \rangle \langle * \rangle}^{2\cdot\ell} \xrightarrow[\emptyset]{\hat{1}} \{q_{\langle N \rangle \langle * \rangle}^{2\cdot\ell-1}\}, \ldots, q_{\langle N \rangle \langle * \rangle}^{\ell+2} \xrightarrow[\emptyset]{\hat{1}} \{q_{\langle N \rangle \langle * \rangle}^{\ell+1}\}, q_{\langle N \rangle \langle * \rangle}^{\ell+1} \xrightarrow[\emptyset]{\hat{1}} \emptyset \in \Delta_1 .$$

6. Next we check that the first tile is t_I. We will first introduce some order-1 states for checking the tile in a stack. That is, for each $t \in \Theta$ we have $q_t \in \mathbb{Q}_1$ from which we have the transitions

$$q_t \xrightarrow[\emptyset]{a} \{q_t\}, q_t \xrightarrow[\emptyset]{t} \emptyset \in \Delta_1$$

where a ranges over $\Sigma \setminus \Theta$.

Then to check the first tile we use

$$p_6 \xrightarrow{q_*} \{p_6^1\}, p_6 \xrightarrow{q_*} \{p_6^2\}, p_6 \xrightarrow{q_{t_I}} \emptyset \in \Delta_2$$

with the states $p_6, p_6^1, p_6^2 \in \mathbb{Q}_2$. These transitions simply skip the leading spacer stacks and check the first stack encoding a cell holds t_I.

7. To check the final tile is t_F we use

$$p_7 \xrightarrow{q_*} \{p_7\}, p_7 \xrightarrow{q_{t_F}} \{q_f^2\} \in \Delta_2$$

which simply iterate to the final order-1 stack and verify it contains t_F.

8. Next we need to define transitions that check the horizonal tiling relation. We will begin with a transition reading the first spacer

$$p_8 \xrightarrow{q_*} \{p'_8\} \in \Delta_2$$

where $p_8, p'_8 \in \mathbb{Q}_2$.

Next we will guess the pair $(t, t') \in H$ that the following two order-1 stacks will contain. For this we will need order-2 states $q^h_{(t,t')} \in \mathbb{Q}_2$ for each $(t, t') \in H$ from which we verify the following two order-1 stacks contain t and t' respectively. In other words, the horizontal tiling relation is satisfied. The transitions we have are

$$p'_8 \xrightarrow{q_*} \left\{ p'_8, q^h_{(t,t')} \right\} \in \Delta_2$$

where p'_8 appears on the right hand side to assert the horizontal tiling relation over subsequent pairs of order-1 stacks. From each $q^h_{(t,t')}$ we have

$$q^h_{(t,t')} \xrightarrow{q_t} \{q^h_{t'}\}, q^h_{t'} \xrightarrow{q_{t'}} \emptyset \in \Delta_2$$

where $q^h_{t'} \in \mathbb{Q}_2$ are intermediate states for each required t'.

Since the horizontal tiling does not apply to the final tile in each row, we have transitions allowing the condition to be relaxed here. That is, we have, for each $(t, t') \in H$,

$$p'_8 \xrightarrow{q_{\langle * \rangle \langle N \rangle}} \{p'_8\}, p'_8 \xrightarrow{q_{\langle * \rangle \langle N \rangle}} \{q^2_f\}, q^h_{(t,t')} \xrightarrow{q_{\langle * \rangle \langle N \rangle}} \emptyset \in \Delta_2 \ .$$

The transitions from p'_8 simply keep passing the state verifying the tiling relation along, or terminates if the bottommost stack is read. From $q^h_{(t,t')}$ we simply dismiss the requirement if the stack is the final tile in a row. Note, we do not have similar transitions from $q^h_{t'}$ since these states represent the relation between the next tile (stack) and the previous one, and thus need to be asserted at the end of a row too.

The state $q_{\langle * \rangle \langle N \rangle} \in \mathbb{Q}_1$ above is an order-1 state from which we verify the column index is N. To implement this state we need several intermediate states $q^1_{\langle * \rangle \langle N \rangle}, \ldots, q^{2 \cdot \ell}_{\langle * \rangle \langle N \rangle} \in \mathbb{Q}_1$. The first ℓ of these states allow any binary digit, while the final ℓ, which read the bits encoding the column index, only allow $\hat{1}$ digits to occur. That is,

$$q_{\langle * \rangle \langle N \rangle} \xrightarrow{\#}_{\emptyset} \left\{ q^1_{\langle * \rangle \langle N \rangle} \right\},$$
$$q^1_{\langle * \rangle \langle N \rangle} \xrightarrow{\hat{0}}_{\emptyset} \left\{ q^2_{\langle * \rangle \langle N \rangle} \right\}, q^1_{\langle * \rangle \langle N \rangle} \xrightarrow{\hat{1}}_{\emptyset} \left\{ q^2_{\langle * \rangle \langle N \rangle} \right\},$$
$$\ldots,$$
$$q^\ell_{\langle * \rangle \langle N \rangle} \xrightarrow{\hat{0}}_{\emptyset} \left\{ q^{\ell+1}_{\langle * \rangle \langle N \rangle} \right\}, q^\ell_{\langle * \rangle \langle N \rangle} \xrightarrow{\hat{1}}_{\emptyset} \left\{ q^{\ell+1}_{\langle * \rangle \langle N \rangle} \right\},$$
$$q^{\ell+1}_{\langle * \rangle \langle N \rangle} \xrightarrow{\hat{1}}_{\emptyset} \left\{ q^{\ell+2}_{\langle * \rangle \langle N \rangle} \right\}, \ldots, q^{2 \cdot \ell - 2}_{\langle * \rangle \langle N \rangle} \xrightarrow{\hat{1}}_{\emptyset} \left\{ q^{2 \cdot \ell - 1}_{\langle * \rangle \langle N \rangle} \right\}, q^{2 \cdot \ell - 1}_{\langle * \rangle \langle N \rangle} \xrightarrow{\hat{1}}_{\emptyset} \emptyset \in \Delta_1 \ .$$

9. Finally we need to define transitions that check the vertical tiling relation. As before, we will begin with a transition reading the first spacer

$$p_9 \xrightarrow{q_*} \{p'_9\} \in \Delta_2$$

where $p_9, p_9' \in \mathbb{Q}_2$.

Next we will guess the pair $(t, t') \in V$ that the next order-1 stack and its linked-to stack respectively will contain. For this we will need order-2 states $q_t^v, q_{t'}^\# \in \mathbb{Q}_2$ for each $(t, t') \in V$ from which we verify the next order-1 stack contain t and the order-1 stack linked to from the $\#$ in this stack contains t'. The transitions we have are

$$p_9' \xrightarrow{q_*} \left\{ p_9', q_t^v, q_{t'}^\# \right\} \in \Delta_2$$

where p_9' appears on the right hand side to assert the vertical tiling relation over subsequent pairs of order-1 stacks. From each q_t^v and $q_{t'}^\#$ we have

$$q_t^v \xrightarrow{q_t} \emptyset, q_{t'}^\# \xrightarrow{\left[q_{t'}^{v'} \right]_\#} \emptyset, q_{t'}^{v'} \xrightarrow{q_{t'}} \emptyset \in \Delta_2$$

where $q_{t'}^{v'} \in \mathbb{Q}_2$ are intermediate states for each required t'. Recall $\left[q_{t'}^{v'} \right]_\#$ will pass $q_{t'}^{v'}$ through the link on $\#$.

Since the vertical tiling does not apply to the final tile in each column, we have transitions allowing the condition to be relaxed here. That is, we have, for each $(t, t') \in V$,

$$p_9' \xrightarrow{q_{\langle N \rangle \langle * \rangle}} \emptyset, q_t^v \xrightarrow{q_{\langle N \rangle \langle * \rangle}} \emptyset \in \Delta_2 \ .$$

Recall $q_{\langle N \rangle \langle * \rangle}$ verifies the row index is N. Note, as soon as the automaton reaches the first order-1 stack containing a row index of N, then all subsequent stacks will contain the same row index, hence the transition to $\emptyset$ allowing the condition to be disbanded. From q_t^v we simply dismiss the requirement if the stack is the final tile in a column. Note, we do not have similar transitions from $q_{t'}^{v'}$ since these states represent the relation between the next tile (stack) and the previous one via the collapse link, and thus need to be asserted at the end of a row too.

4 Conclusion

We have shown that the problem of testing emptiness of stack automata is NEXPTIME-complete for collapsible pushdown stacks. In the case of *annotated stacks* where stack characters are augmented with further annotated stacks (instead of a link to a position elsewhere in the stack), our proof does not carry through. We conjecture that for annotated stacks the emptiness problem is EXPTIME-complete but leave a proof of this for future work.

References

[1] Ahmed Bouajjani and Antoine Meyer. Symbolic reachability analysis of higher-order context-free processes. In *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, pages 135–147, 2004.

[2] Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. A saturation method for collapsible pushdown systems. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, pages 165–176, 2012.

[3] Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. Recursion schemes and logical reflection. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 120–129, 2010.

[4] Daniel Gottesman and Sandy Irani. The quantum and classical complexity of translationally invariant tiling and hamiltonian problems. *CoRR*, abs/0905.2419, 2009.