Supplementary Material - Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples

Gail Weiss [1] **Yoav Goldberg** [2] **Eran Yahav** [1]

This supplementary material contains a description of the L* algorithm (Section 1), and additional experimental results and details (Section 2).

1. Angluin's L* Algorithm

Algorithm 1 L* Algorithm with explicit membership and equivalence queries.

```
$S \leftarrow \{\epsilon\}, E \leftarrow \{\epsilon\}$
for $(s \in S)$, $(a \in \Sigma)$, and $(e \in E)$ do
    $T[s, e] \leftarrow$ Member$(s \cdot e)$
    $T[s \cdot a, e] \leftarrow$ Member$(s \cdot a \cdot e)$
end for
while True do
    while $(s_{new} \leftarrow Closed(S, E, T) \neq \perp)$ do
        $Add(S, s_{new})$
        for $(a \in \Sigma, e \in E)$ do
            $T[s_{new} \cdot a, e] \leftarrow$ Member$(s_{new} \cdot a \cdot e)$
        end for
    end while
    $\mathcal{A} \leftarrow MakeHypothesis(S, E, T)$
    $cex \leftarrow$ Equivalence$(\mathcal{A})$
    if $cex = \perp$ then
        return $\mathcal{A}$
    else
        $e_{new} \leftarrow FindSuffix(cex)$
        $Add(E, e_{new})$
        for $(s \in S, a \in \Sigma)$ do
            $T[s, e_{new}] \leftarrow$ Member$(s \cdot e_{new})$
            $T[s \cdot a, e_{new}] \leftarrow$ Member$(s \cdot a \cdot e_{new})$
        end for
    end if
end while
```

Angluin's L* algorithm (1987) is an exact learning algorithm for regular languages. The algorithm learns an unknown regular language U over an alphabet Σ, generating a DFA that accepts U as output. We only provide a brief and informal description of the algorithm; for further details see (Angluin, 1987; Berg et al., 2005).

Algorithm 1 shows the L* algorithm. This version is adapted from Alur et al. (2005), where the membership and equivalence queries have been made more explicit than they appear in Angluin (1987).

The algorithm maintains an *observation table* (S, E, T) that records whether strings belong to U. In Algorithm 1, this table is represented by the two-dimensional array T, with dimensions $|S| \times |E|$, where, informally, we can view S as a set of words that lead from the initial state to states of the hypothesized automaton, and E as a set of words serving as experiments to separate states. The table T itself maps a word $w \in (S \cup S \cdot \Sigma) \cdot E$ to `True` if $w \in U$ and `False` otherwise.

The table is updated by invoking membership queries to the teacher. When the algorithm reaches a consistent and closed observation table (meaning that all states have outgoing transitions for all letters, without contradictions), the algorithm constructs a hypothesized automaton $\mathcal{A}$, and invokes an *equivalence query* to check whether $\mathcal{A}$ is equivalent to the automaton known to the teacher. If the hypothesized automaton accepts exactly U, then the algorithm terminates. If it is not equivalent, then the teacher produces a counterexample showing a difference between U and the language accepted by $\mathcal{A}$.

A simplified run through of the algorithm is as follows: the learner starts with an automaton with one state—the initial state—which is accepting or rejecting according to the classification of the empty word. Then, for every state in the automaton, for every letter in the alphabet, the learner verifies by way of membership queries that for every shortest sequence reaching that state, the continuation from that prefix with that letter is correctly classified. As long as an inconsistency exists, the automaton is refined. Every time the automaton reaches a consistent state (a complete transition function, with no inconsistencies by single-letter

[1]Technion, Haifa, Israel [2]Bar Ilan University, Ramat Gan, Israel. Correspondence to: Gail Weiss <sgailw@cs.technion.ac.il>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

extensions), that automaton is presented to the teacher as an equivalence query. If it is accepted, the algorithm completes; otherwise, it uses the teacher-provided counterexample to further refine the automaton.

2. Additional Results

2.1. Random Regular Languages

We show results for extraction from GRU and LSTM networks with varying hidden sizes, trained on small regular languages of varying alphabet size and DFA size. Each extraction was limited to 30 seconds, and had initial refinement depth 10. For each of the combinations of state size and target language complexity, 3 networks of each type were trained, each on a different (randomly generated) language. The full results of these experiments are shown in Table 1. Note that each row in each of the tables represents 3 experiments, i.e. in total $9 \times 2 \times 3 = 54$ random DFAs were generated, trained on, and re-extracted.

We note with satisfaction that in 36 of the 54 experiments, the extraction process reached equivalence on a regular language identical to the target language the network had been trained on. We also note that on one occasion in the LSTM experiments the extraction reached equivalence too easily, accepting an automaton of size 2 that ultimately was not a great match for the network. Such a problem could be countered by increasing the initial split depth, for instance when sampling shows that a too-simple automaton has been extracted.

We also ran extraction with Omlin and Giles' a-priori quantization on each of these networks, with quantization level 2 and a time limit of 50 seconds. The extractions did not complete in this time. For completeness, we present the size, coverage, and accuracies of these partially extracted DFAs in Table 2. The smaller size of the extracted DFAs for networks with larger hidden size is a result of the transition function computation taking longer: this slows the BFS exploration and thus the extraction visits less states in the allotted time.

2.2. Balanced Parentheses

We trained a GRU and an LSTM network on the irregular language of balanced parentheses, and then attempted to extract automata from these networks. For both, L* at first proposed a series of automata each representing the language of balanced parentheses to increasing nesting depth. Some of these are shown in Fig. 1. For the GRU network, after a certain point in the extraction, we even found a counterexample which showed the network had not generalized correctly to the language of balanced parentheses, and the next automaton returned resembled—but was not quite—an automaton for balanced parentheses nested to some depth.

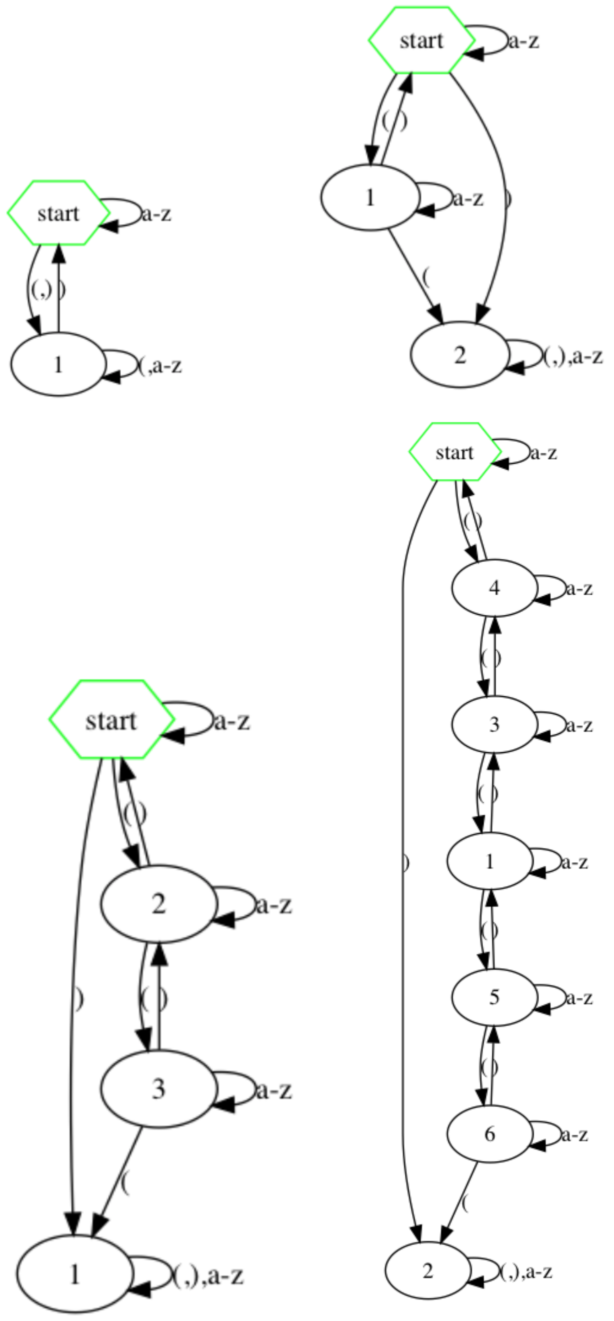


Figure 1. Select automata of increasing size for recognizing balanced parentheses over the 28 letter alphabet `a-z,(,)`, up to nesting depths 1 (flawed), 1 (correct), 2, and 5, respectively.

Table 1. Results for DFA extracted using our method from 2-layer GRU and LSTM networks with various state sizes, trained on random regular languages of varying sizes and alphabets. Each row in each table represents 3 experiments with the same parameters (network hidden-state size, alphabet size, and minimal target DFA size). Single values represent the average of the 3 experiments, multiple values list the result for each experiment. An extraction time of 30 seconds signals a timed out extraction (for which the last automaton proposed by L* is taken as the extracted automaton).

Extraction from LSTM Networks — Our Method

Hidden	Alphabet	Language /	Extraction	Extracted	Average Extracted DFA Accuracy				
Size	Size	Target DFA Size	Time (s)	DFA Size	l=10	l=50	l=100	l=1000	Training
50	3	5	2.9	5,5,6	100.0	99.96	99.86	99.90	100.0
100	3	5	2.9	5,5,2	92.96	92.96	93.73	93.46	91.06
500	3	5	11.8	5,5,5	100.0	100.0	100.0	99.96	100.0
50	5	5	30, 30, 30	68, 59, 115	99.96	99.93	99.76	99.93	99.99
100	5	5	30, 7.7, 30	57, 5, 38	99.96	99.96	99.96	99.90	100.0
500	5	5	30, 20.7, 19.0	5, 5, 5	100.0	100.0	99.93	99.90	100.0
50	3	10	30, 30, 11.1	10, 10, 10	99.96	99.96	99.90	99.90	100.0
100	3	10	7.6, 30, 7.7	10, 10, 11	99.96	99.93	99.96	99.96	100.0
500	3	10	30, 30, 30	10, 9, 10	92.30	92.80	93.70	93.43	92.30

Extraction from GRU Networks — Our Method

Hidden	Alphabet	Language /	Extraction	Extracted	Average Extracted DFA Accuracy				
Size	Size	Target DFA Size	Time (s)	DFA Size	l=10	l=50	l=100	l=1000	Training
50	3	5	1.7	5,5,6	100.0	100.0	99.86	99.96	100.0
100	3	5	4.1	5,5,5	100.0	100.0	100.0	99.96	100.0
500	3	5	7.0	5,5,5	100.0	100.0	100.0	100.0	100.0
50	5	5	30, 30, 8.2	150,93,5	100.0	99.90	99.93	99.86	100.0
100	5	5	9.0, 8.0, 30	5,5,16	100.0	100.0	99.96	99.96	99.99
500	5	5	15.5, 30, 25.6	5,5,5	100.0	100.0	99.96	100.0	100.0
50	3	10	30, 30, 30	11,11,155	99.96	99.83	99.93	99.93	99.99
100	3	10	11.0	11,10,11	100.0	99.93	99.96	99.93	100.0
500	3	10	30, 30, 30	10,10,10	100.0	99.93	100.0	99.90	100.0

Table 2. Results for automata extracted using Omlin & Giles' a-priori quantization as described in their 1996 paper, with quantization level 2, from the same networks used in Table 1 (3 networks for each set of parameters and network type).

Extraction from LSTM Networks — O&G Quantization

Hidden	Alphabet	Language/	Extracted			Coverage / Accuracy (%)											
Size	Size	Target DFA Size	DFA Sizes			l=1		l=5		l=10		l=15		l=50		Training	
50	3	5	3109	3107	3107	100	100	100	100	30.66	83.65	3.87	81.53	0.0	NA	27.44	88.0
100	3	5	2225	2252	2275	100	100	100	100	7.57	80.50	0.07	50.0	0.0	NA	19.31	84.57
500	3	5	585	601	584	100	100	100	100	0.0	NA	0.0	NA	0.0	NA	8.80	71.71
50	5	5	1956	1973	1962	100	100	100	73.93	0.03	100	0.0	NA	0.0	NA	12.39	78.34
100	5	5	1392	1400	1400	100	100	100	64.3	0.0	NA	0.0	NA	0.0	NA	11.19	74.80
500	5	5	359	366	366	100	100	33.43	70.60	0.0	NA	0.0	NA	0.0	NA	6.24	73.92
50	3	10	3135	3238	3228	100	100	100	100	29.43	83.72	4.57	94.19	0.0	NA	27.70	88.80
100	3	10	2294	2282	2272	100	100	100	100	0.90	91.30	0.0	NA	0.0	NA	16.83	81.07
500	3	10	586	589	589	100	100	100	100	0.0	NA	0.0	NA	0.0	NA	8.48	74.77

Extraction from GRU Networks — O&G Quantization

Hidden	Alphabet	Language/	Extracted			Coverage / Accuracy (%)											
Size	Size	Target DFA Size	DFA Sizes			l=1		l=5		l=10		l=15		l=50		Training	
50	3	5	4497	4558	4485	100	100	100	100	50.73	90.52	25.26	91.95	2.66	96.25	42.28	91.08
100	3	5	3188	3184	3197	100	100	100	100	3.50	66.66	0.07	50.0	0.0	NA	19.14	83.63
500	3	5	1200	1221	1225	100	100	100	100	0.0	NA	0.0	NA	0.0	NA	8.98	74.45
50	5	5	2810	2796	2802	100	100	100	87.97	0.10	100	0.0	NA	0.0	NA	14.56	80.61
100	5	5	1935	1941	1936	100	100	100	73.17	0.0	NA	0.0	NA	0.0	NA	12.05	76.39
500	5	5	721	706	749	100	100	91.03	55.94	0.0	NA	0.0	NA	0.0	NA	9.52	71.53
50	3	10	4598	4582	4586	100	100	100	100	15.73	79.76	1.23	70.71	0.0	NA	24.32	86.93
100	3	10	3203	3192	3194	100	100	100	100	0.3	81.25	0.0	NA	0.0	NA	19.18	83.84
500	3	10	1226	1209	1209	100	100	100	100	0.0	NA	0.0	NA	0.0	NA	13.39	76.64

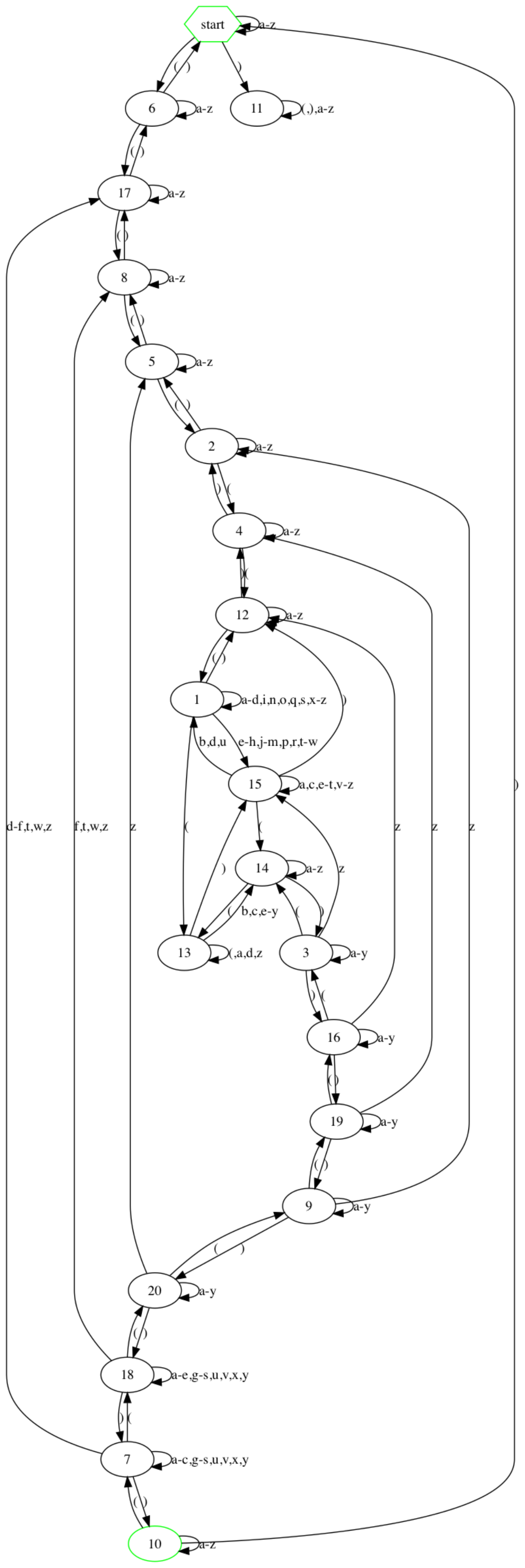


Figure 2. Automaton no longer representing a language of balanced parentheses up to a certain depth. (Showing how a trained network may be overfitted past a certain sample complexity.)

Table 3. Extraction of automata from an LSTM network trained to 100% accuracy on the training set for the language of balanced parentheses over the 28-letter alphabet `a-z,(,)` . The table shows the counterexamples and the counterexample generation times for each of the successive equivalence queries posed by L* during extraction, for both our method and a brute force approach. Each successive equivalence query from L* was an automaton classifying the language of all words with balanced parentheses up to nesting depth n, with increasing n.

Refinement Based		Brute Force	
Counterexample	Time (seconds)	Counterexample	Time (seconds)
))	1.4	))	1.5
(())	1.6	tg(gu()uh)	57.5
((()))	3.1	((wviw(iac)r)mrsnqqb)iew	231.5
(((())))	3.1		
((((()))))	3.4		
(((((())))))	4.7		
((((((()))))))	6.3		
(((((((())))))))	9.2		
((((((((()))))))))	14.0		

We show this automaton in Fig. 2.

In our main submission, we show the counterexamples returned during a 400-second extraction from the GRU network, as generated either by random sampling or by our method. For completeness, we present now in Table 3 the counterexamples for the LSTM extraction.

2.3. Other Interesting Examples

2.3.1. Counting

We trained an LSTM network with 2 layers and hidden size 100 (giving overall state size $d_s = 2 \times 2 \times 100 = 400$) on the regular language

$$[a-z]*1[a-z1]*2[a-z2]*3[a-z3]*4[a-z4]*5[a-z5]*\$$$

over the 31-letter alphabet {`a`,`b`, ...,`z`,`1`,`2`, ...,`5`}, i.e. the regular language of all sequences `1+2+3+4+5+` with lowercase letters `a-z` scattered inside them. We trained this network on a train set of size 20000 and tested it on a test set of size 2000 (both evenly split on positive and negative examples), and saw that it reached 100% accuracy on both.

We extracted from this network using our method. Within 2 counterexamples (the provided counterexample `12345`, and another generated by our method), and after a total of 9.5 seconds, L* proposed the automaton representative of the network's target language, shown in Fig. 3. However, our method did not accept this DFA as the correct DFA for the network. Instead, after a further 85.4 seconds of exploration and refinement, the counterexample `aca11` was found and returned to L*, meaning: our method found that the network accepted the word `aca11` — despite this word not being in the target language of the network and the network having 100% accuracy on both its train and test set.

Ultimately, after 400 seconds our method extracted from the network (but did not reach equivalence on) a DFA with 118 states, returning the counterexamples listed in Table 4 and achieving 100% accuracy against the network on its train set, and 99.9+% accuracy on all sampled sequence lengths. We note that by the nature of our method, the complexity of this DFA is necessarily an indicator of the inherent complexity of the concept to which the trained network has generalized.

2.3.2. Tokenized JSON Lists

We trained a GRU network with 2 layers and hidden size 100 on the regular language representing a simple tokenized JSON list with no nesting,

$$(\backslash[\backslash])|(\backslash[[S0NTF](,[S0NTF])*\backslash])\$$$

over the 8-letter alphabet {`[`,`]`,`S`,`0`,`N`,`T`,`F`,`,`}, to accuracy 100% on a training set of size 20000 and a test set of size 2000, both evenly split between positive and negative examples. As before, we extracted from this network using our method.

Within 2 counterexamples (1 provided and 1 generated) and a total of 3.8 seconds, our method extracted the automaton shown in Fig. 4a, which is almost but not quite representative of the target language. 7.12 seconds later it returned a counterexample to this DFA which pushed L* to refine further and return the DFA shown in Fig. 4b, which is also almost but not quite representative of zero-nesting tokenized JSON lists.

Ultimately, after 400 seconds, our method extracted (but did not reach equivalence on) an automaton of size 441, returning the counterexamples listed in Table 5 and achieving 100% accuracy against the network on both its train set and all sampled sequence lengths. As before, we note that each

Table 4. Counterexamples returned to the equivalence queries made by L* during extraction of a DFA from a network trained to 100% accuracy on both train and test sets on the regular language [a-z]*1[a-z1]*2[a-z2]*3[a-z3]*4[a-z4]*5[a-z5]*$ over the 31-letter alphabet {`a`,`b`, ...,`d`,`1`,`2`, ...,`5`}. Counterexamples highlighting the discrepancies between the network behavior and the target behavior are shown in bold.

Counterexample Generation for the Counting Language

Counterexample	Generation Time (seconds)	Network Classification	Target Classification
12345	provided	True	True
512345	8.18	False	False
aca11	85.41	True	False
blw11	0.50	True	False
dnm11	0.96	False	False
bzm11	0.90	False	False
drxr11	0.911	True	False
brdb11	0.90	False	False
elrs11	1.16	True	False
hu11	1.93	False	False
ku11	2.59	False	False
ebj11	2.77	False	False
pgl11	3.77	True	False
reeg11	4.16	False	False
eipn11	5.66	False	False

Table 5. Counterexamples returned to the equivalence queries made by L* during extraction of a DFA from a network trained to 100% accuracy on both train and test sets on the regular language (\[\])|(\[[S0NTF](,[S0NTF])*\])$ over the 8-letter alphabet { `[`,`]`,`S`,`0`,`N`,`T`,`F`,`,` }. Counterexamples highlighting the discrepancies between the network behaviour and the target behaviour are shown in bold.

Counterexample Generation for the Non-Nested Tokenized JSON-lists Language

Counterexample	Generation Time (seconds)	Network Classification	Target Classification
[]	provided	True	True
[SS]	3.49	False	False
[[,]	7.12	True	False
[S,,	8.61	True	False
[0,F	8.38	True	False
[N,0,	8.07	False	False
[S,N,0,	9.43	True	False
[T,S,	9.56	False	False
[S,S,T,[]	15.15	False	False
[F,T,[	3.23	False	False
[N,F,S,0	10.04	True	False
[S,N,[,,,,	27.79	True	False
[T,0,T,	28.06	True	False
[S,T,0,],	26.63	True	False

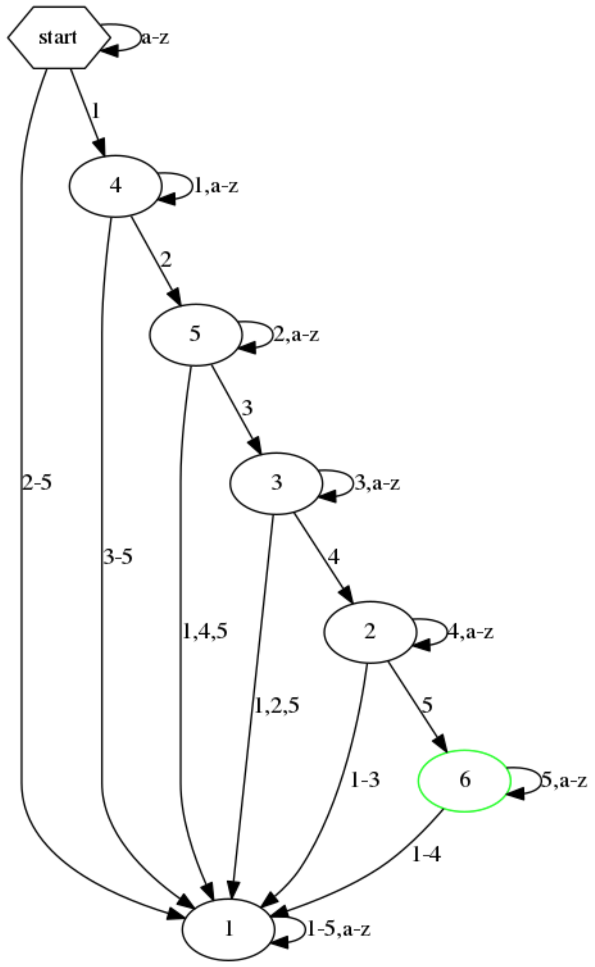


Figure 3. DFA representing the regular language [a-z]*1[a-z1]*2[a-z2]*3[a-z3]*4[a-z4]*5[a-z5]*$ over the alphabet {a,b,...,z,1,2,...,5}

.

state split by the method is justified by concrete inputs to the network, and so the extraction of a large DFA is a sign of the inherent complexity of the learned network behavior.

2.4. Train Set Details

The sizes, sample lengths, and positive to negative ratios of the samples in the train sets are listed here (Table 6) for each of the networks used in our main experiments, as well as for the JSON and Counting languages. Note that for some languages (such as the first Tomita grammar, `1*`), there are very few positive/negative samples. For these languages, the test sets are less balanced between positive and negative samples.

We reiterate that all networks used in this work were trained to 100% train set accuracy and reached at least 99.9% on a set of 1000 samples from each of the lengths $4, 7, 10, ..., 28$.

An explicit description of the Tomita grammars can be found in (Tomita, 1982).

References

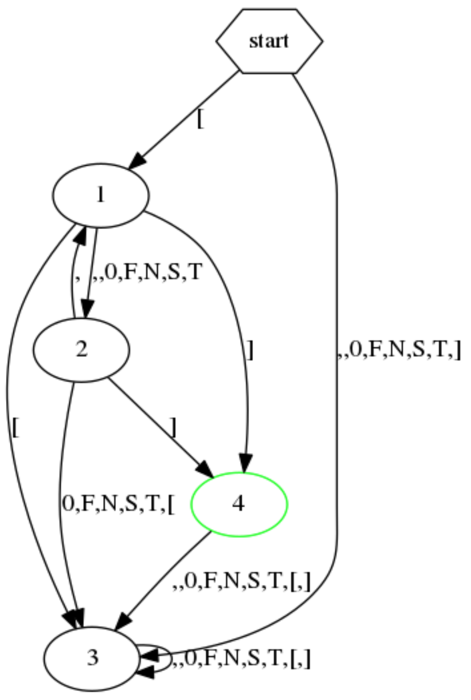
Alur, R., Černý, P., Madhusudan, P., and Nam, W. Synthesis of interface specifications for java classes. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '05, pp. 98–109, New York, NY, USA, 2005. ACM. ISBN 1-58113-830-X. doi: 10.1145/1040305.1040314. URL `http://doi.acm.org/10.1145/1040305.1040314`.

Angluin, D. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987. doi: 10.1016/0890-5401(87)90052-6. URL `https://doi.org/10.1016/0890-5401(87)90052-6`.

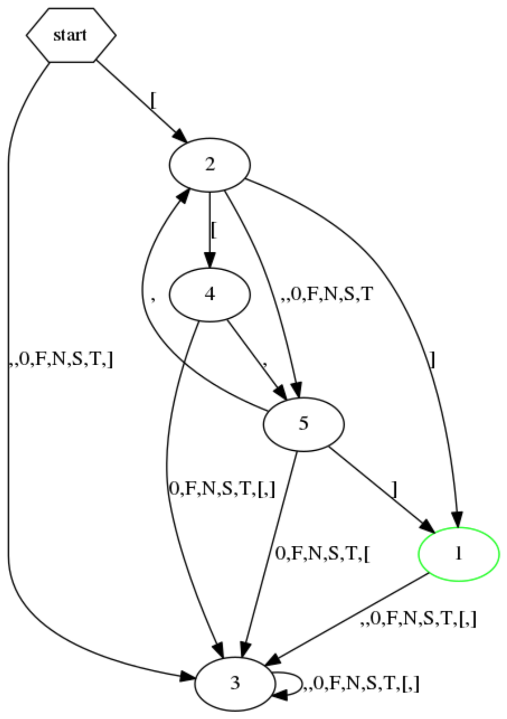
Berg, T., Jonsson, B., Leucker, M., and Saksena, M. Insights to angluin's learning. *Electr. Notes Theor. Comput. Sci.*, 118:3–18, 2005. doi: 10.1016/j.entcs.2004.12.015. URL `https://doi.org/10.1016/j.entcs.2004.12.015`.

Omlin, C. W. and Giles, C. L. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52, 1996. doi: 10.1016/0893-6080(95)00086-0. URL `https://doi.org/10.1016/0893-6080(95)00086-0`.

Tomita, M. Dynamic construction of finite automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, pp. 105–108, Ann Arbor, Michigan, 1982.



(a)



(b)

Figure 4. Two DFAs resembling, but not perfectly, the correct DFA for the regular language of tokenized JSON lists, (\[\])|(\[[S0NTF](,[S0NTF])*\])$. DFA 4a is almost correct, but accepts also list-like sequences in which the last item is missing, i.e., there is a comma followed by a closing bracket. DFA 4b is returned by L^* after the teacher (network) rejects 4a, but is also not a correct representation of the target language — treating the sequence `[,` as a legitimate list item equivalent to the characters `S,0,N,T,F`
.

Table 6. Train set statistics for networks used in this work. The random regular language networks used in the main work were based on minimal DFAs of size 10 over alphabets of size 3, with 3 languages per hidden state size. We list statistics for their train sets in grouped by the hidden size. The train set sizes and lengths were the same for each of these random languages, but the number of positive/negative samples found each time varied slightly.

Train Set Stats

Language	Architecture	Hidden Size	Train Set Size	Of Which Positive Samples	Lengths in Train Set
Tomita 1	GRU	100	613	14	0-13,16,19,22
Tomita 2	GRU	100	613	8	0-13,16,19,22
Tomita 3	GRU	100	2911	1418	0-13,16,19,22
Tomita 4	GRU	100	2911	1525	0-13,16,19,22
Tomita 5	GRU	100	1833	771	0-13,16,19,22
Tomita 6	GRU	100	3511	1671	0-13,15-20
Tomita 7	GRU	100	2583	1176	0-13,16,19,22
Random 1-3	GRU	50	16092	8038, 7768, 8050	1-15,16,18,...,26
Random 4-6	GRU	100	16092	7783, 7842, 8167	1-15,16,18,...,26
Random 7-9	GRU	500	16092	8080, 8143, 7943	1-15,16,18,...,26
Balanced Parentheses	GRU	50	44697	25243	0-73
Balanced Parentheses	LSTM	50	44816	28781	0-81
emails	LSTM	100	40000	20000	0-34
JSON Lists	GRU	100	20000	10000	0-74
Counting	LSTM	100	20000	10000	0-43