Reading Guide

More concise representation of regular languages by automata and regular expressions

Minerva University

CS142: Theory of Computation

Professor Rena Levitt

December 16, 2022

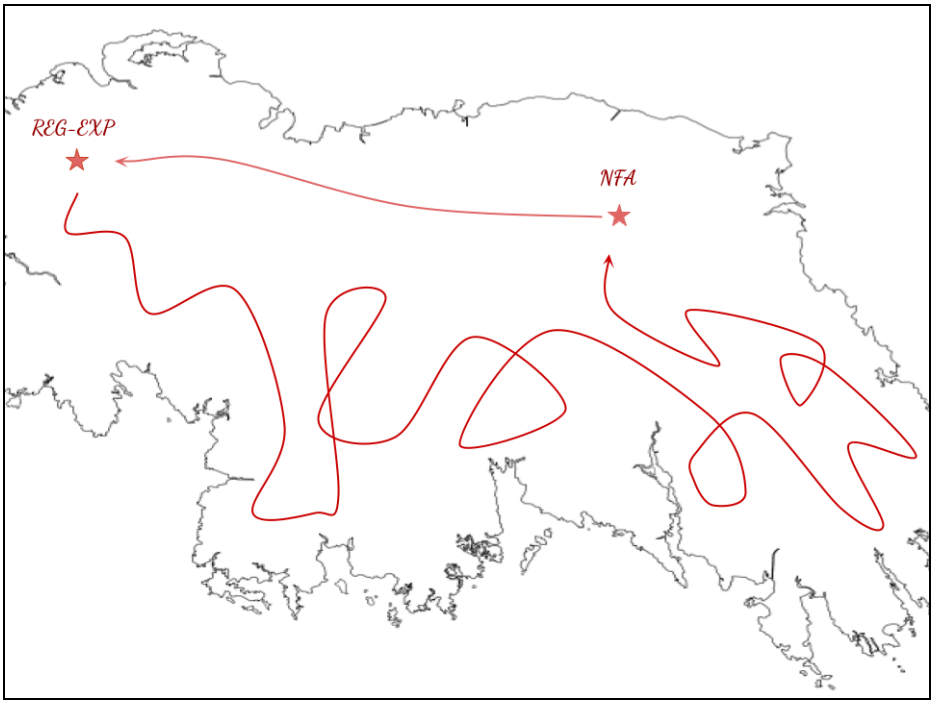
Table of Content

Introduction

Abbreviations

1. Regex (for narrative) or reg-ex (for formulas) = regular expressions
2. NPDA/DPDA = nondeterministic/deterministic pushdown automata
3. h-NPDA/h-DPDA = constant height NPDA/DPDA
4. SLP = straight line programs
5. DAG = directed acyclic graphs
6. NFA/DFA/FA = nondeterministic/deterministic finite automata

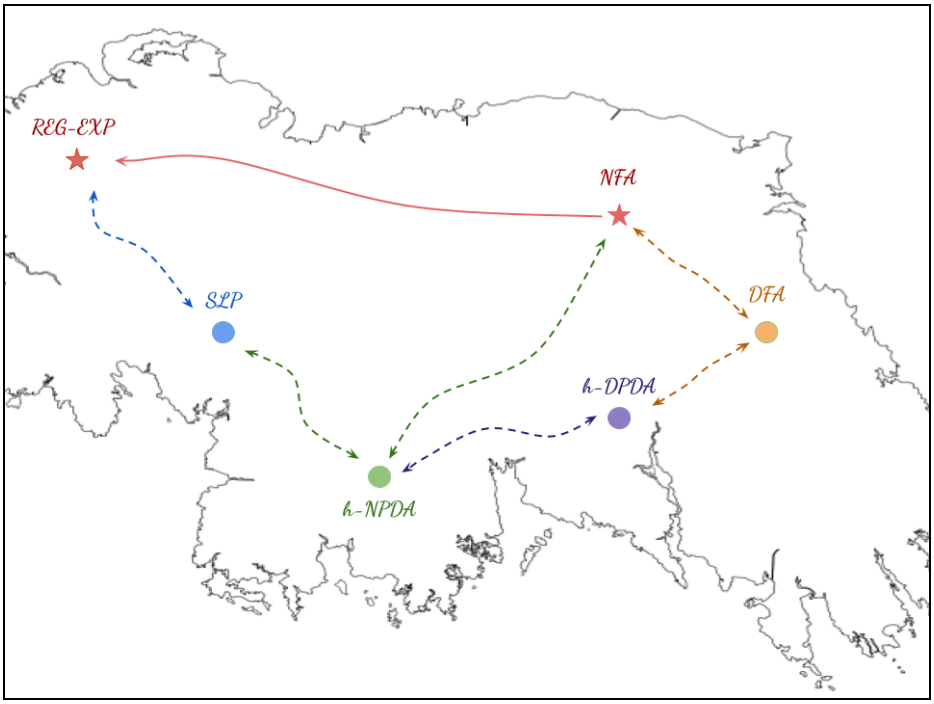
Welcome aboard our journey around the magic land of languages! Let us take a look at the map of our land. Today, we will go from the NFA Town to the Regex City, which we can do by following a nice straight road. This corresponds to the linearly-sized formalization of NFA → Regex.



But oh no, turns out the road only goes one way! To make our way back from Regex city to NFA town, we end up having to take a convoluted road that goes over the same places many times. This corresponds to the exponentially-sized formalization of Regex→ NFA. We have seen the conversion procedure for this in the textbook (Chapter 1.3, section Equivalence with Finite Automata) but have not explored the space complexity. Please reference this resource for more information.

Our initial state does not look promising. Surely there is a shorter way!

Fortunately, there might just be a path back to NFA town that goes through towns of SLPs and h-NPDAs!



This is a shorter road that also might get us to some cool other places like h-DPDA and DFA town more efficiently. Making sure we can use this new path is our goal now since this path might allow all humans and computers from Regex city to operate much more efficiently.

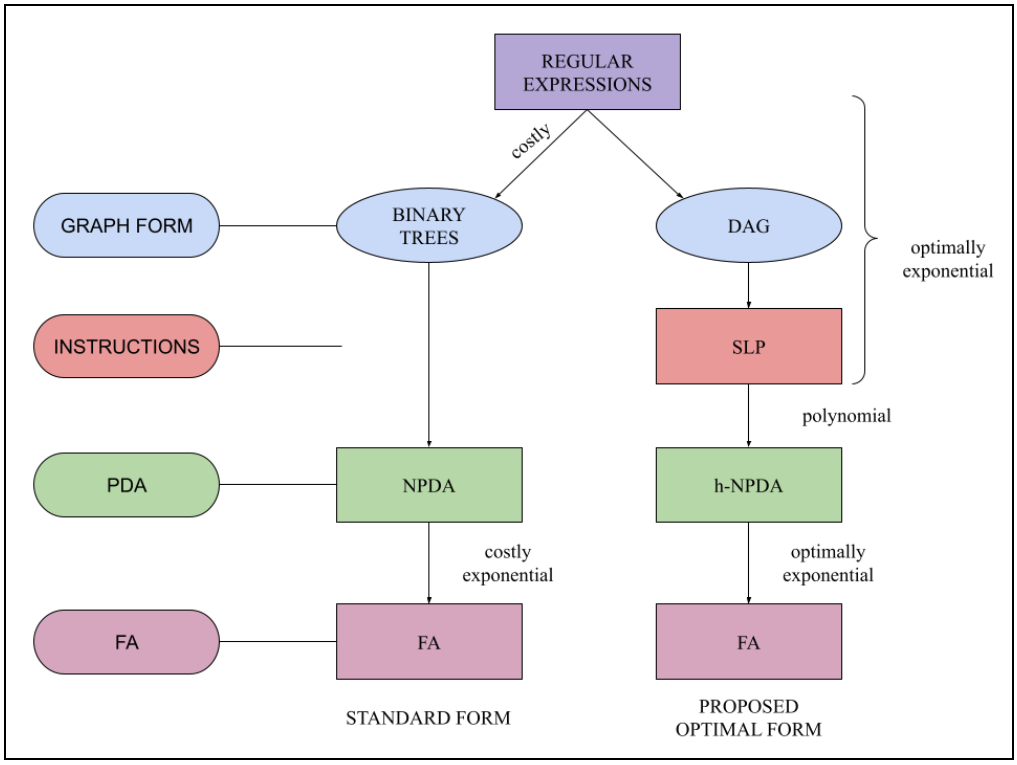
This corresponds to regular expressions serving as a foundational language in theory of computation, so finding ways to formalize this language in more concise ways can help us be more efficient in problem solving.

The initial path from Regex to NFA has many obstacles that make it uneconomical:

(1) Binary trees for Regex representation cost a lot due to redundancy - subtrees are usually used more than once.

(2) Going from NPDA to NFA is costly because of seemingly unbounded memories of NPDAs. NFAs then have to "pay up" in exponentially many states to represent the unbounded pushdown structure.

Obstacle (1) can be overcome by using SLPs which have an optimal exponential gap with Regex. Obstacle (2) can be overcome by using bounded height h-NPDAs that have optimal exponential and optimal double exponential simulation costs by FAs. All that's missing for the best path from Regex to NFA is a link between SLPs and h-NPDAs that is polynomial for both directions. Combining these notions, we might have a more economical solution for Regex → NFA.



Preliminaries

The size of a regular expression r refers to the number of occurrences of symbols plus the number of occurrences of operators inside r.

For $r = a \cdot (a + b)^* + (a + b)^* \cdot b \cdot a^*$, we have size$(r) = 16$.

What is a constant height push-down automata?

NPDA bounded by h is formally defined as $Q, \Sigma, \Gamma, H, qo, F, h$ where h $\in$ N is a constant denoting the pushdown height. Language for constant height NPDA exists for any word in L(A) if there exists an accepting computation along which the pushdown store never contains more than h symbols.

What is an SLP?

An SLP is a more economical way to represent regular expressions in the form of instructions. Given variables $X = \{x_1.,.,.,.... x_l\}$ SLP on a finite sequence of instructions:

$$P \equiv \text{instr}_1 ; \ldots \text{instr}_i ; \ldots \text{instr}_\ell,$$

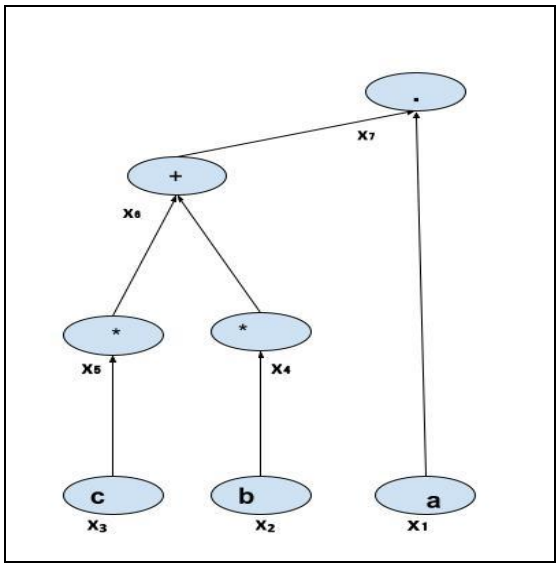
where the ith instruction instr_i has one of the following forms:

(i) $x_i := \emptyset$, $x_i := \varepsilon$, or $x_i := a$ for any symbol $a \in \Sigma$,
(ii) $x_i := x_j + x_k$, $x_i := x_j \cdot x_k$, or $x_i := x_j^*$, for $1 \leqslant j, k < i$.

SLP can also be expressed as DAG, whose vertices are labeled with symbols and operators. For example, SLP P:

$P \equiv x_1 := a;$
$x_2 := b;$
$x_3 := c;$
$x_4 := x_2^*;$
$x_5 := x_3^*;$
$x_6 := x_4 + x_5 ;$
$x_7 := x_6 \cdot x_1 ;$

The DAG is:



Fan-out refers to the frequency of a variable that has been reused several times in the right parts. From the diagram on the fan-out variable is 1 each of the repeated variables have been used once.

The size of a straight line program P is the ordered pair size(P) = (length(P), fan-out(P)), where length(P) denotes the number of instructions in P, and fan-out(P) is the maximum fan-out of its variables. The length of our given P in this case is the size(P)=(7,1).

When we expand we begin with the last term and expand upwards.Starting with $x_7 := x_6 \,.\, x_1$:

1. Substitute the values x_6 and x_1 : $x_7 := (x_4 + x_5) \circ a$;
2. Substitute the values $(x_4 + x_5)$: $x_7 := (x_2 * + x_3 *) \circ a$;
3. Substitute the values (x_3 and x_3): $x_7 := (b * + c *) \circ a$;

We get our regular expression once we have expanded our SLP and remain with symbols only. Our regular expression is therefore reg-exp(P) = $(b\ *\ +\ c\ *\ )\ \circ\ a$ The size of our regular expression will be 7 as we have 3 symbols and 4 regular operators in P.

Sizes of SLPs and classical regular expressions.

For each SLP P, length(P) = size(reg-exp(P)) if and only if fan-out(P) = 1. For fan-out greater than 1 we can get an exponential gap between the size of SLPs and regular expressions which shows SLPs offer a shorter way to express regular expressions.

The following recursive example shows that, even only with fan-out 2, we get an exponential gap.

$$
\begin{aligned}
P_\ell \equiv\ & x_1 := a; \\
& x_2 := x_1 \cdot x_1; \\
& x_3 := x_2 \cdot x_2; \\
& \quad \vdots \\
& x_\ell := x_{\ell-1} \cdot x_{\ell-1}.
\end{aligned}
$$

By expanding the SLP to create a regular expression the size of it would be reg-exp(P) = $a^{2^{l-1}}$ which is exponential in size.The size is exponential as expansion of our example SLP to a classic regular expressions leads to an exponential size growth factor of symbols.For example in our example 2 above,if l =4 we would expect 7 symbols of a .Thus, for any $l \geq 1$, we can obtain size(reg-exp(P)) =$2^{length(P_l)-1}$ which is exponential. The size is equal to the number of symbols we would have in our regular expression.In contrast for fan-out equal to 1 like in example 1, we would simply have at most one substitution for any variable in our SLP which

would be equal to the number of instruction (SLP length) when we expand the SLP to a regular expression.

From a constant height NPDA to an SLP

Let $A = \langle Q, H, q_0, \{q_f\}, h\rangle$ be a constant height NPDA. Then there exists an SLP P_A such that reg-exp(P_A) denotes L(A), with

$$length(P_A) \leq O(h \cdot |Q|^4 \cdot |\Gamma| + |Q|^2 \cdot |\Sigma|) \text{ and fan-out}(P_A) \leq |Q|^2 + 1.$$

That is, for regular language over a fixed alphabet, the size of P_A is polynomial in the size of A.

You may remember a similar procedure for translating an NFA into a regular expression in the first unit of this course. If not, reference the Sipser textbook Chapter 1.3 Regular Expressions, section Equivalence with Finite Automata, Example 1.68.

Before the first phase of the translation process, we input variables $\chi_\alpha := \alpha$ of form (i) which are maximum one per each $\alpha \in \Sigma \cup \{\varepsilon, \emptyset\}$. Then, we go through the translation algorithm:

Phase I:

for each $1 \leqslant i, j \leqslant k$ **do**
  $[q_i, 0, 0, q_j] = \sum_{\alpha \in \Psi_0(q_i, q_j)} \alpha$;
for $s = 0$ **to** $k-1$ **do**
 for each $1 \leqslant i, j \leqslant k$ **do**
  $[q_i, s+1, 0, q_j] = [q_i, s, 0, q_j] + [q_i, s, 0, q_{s+1}] \cdot [q_{s+1}, s, 0, q_{s+1}]^* \cdot [q_{s+1}, s, 0, q_j]$;

Phase II:

for $t = 1$ **to** h **do begin**
  for each $1 \leqslant i, j \leqslant k$ **do**
$(\diamond)[q_i, 0, t, q_j] = [q_i, 0, t-1, q_j] + \sum_{X \in \Gamma} \sum_{p \in \Psi_+(q_i, X)} \sum_{q \in \Psi_-(X, q_j)} [p, k, t-1, q]$;
  for $s = 0$ **to** $k-1$ **do**
   for each $1 \leqslant i, j \leqslant k$ **do**
    $[q_i, s+1, t, q_j] = [q_i, s, t, q_j] + [q_i, s, t, q_{s+1}] \cdot [q_{s+1}, s, t, q_{s+1}]^* \cdot [q_{s+1}, s, t, q_j]$;
end;
return $[q_1, k, h, q_k]$

The output of this algorithm is a finite sequence of instructions with an output variable $[q_1,\ k,\ h, q_k]$. We need the output instructions to be in form (ii) so the authors introduce new variables whose purpose is just that.

What does the notation $[q, s, t, q_j]$ *mean?*

As defined in the article, for each i, j $\in$ {1, ... , k}, s $\in$ {0, ... , k}, and t $\in$ {0, ... , h}, $[q, s, t, q_j]$ is the set of strings x $\in \Sigma^*$ such that, for each of them, there exists *at least one* computation

with the following properties.

- Computation starts in state q_i, pushdown is empty.
- Computation ends in state q_j, pushdown empty again.
- During the competition whenever the pushdown is empty, the current finite control state $\in \{q_1, \dots, q_s\}$ except for q_i and q_j themselves. When s = 0, the pushdown is never empty.
- During the computation, the pushdown height does not exceed t.

Why is the size of P_A *polynomial in the size of A for regular language over a fixed alphabet?*

Intuitively, from the algorithm we can see that there is a finite number of instructions that exist so the time should scale polynomially. A more thorough breakdown of the number of steps is the following: there are no more than $|\Sigma| + 2$ instructions. In phase I the instructions are $\leq k^2 \cdot |\Sigma| + k^3$. For Phase II the maximum number of instructions is $h \cdot k^3 \cdot (k \cdot |\Gamma| + 4)$. Overall we have $length(P_A) \leq O(h \cdot k^4 \cdot |\Gamma| + k^2)$.

Why are the instructions in Phase I and II limited in such a way?

Check out the structure of the pseudocode. For Phase I, the first for loop makes all combinations of the characters in the alphabet. It is actually two different loops for i and j and leads to a maximum of $k^2 \cdot |\Sigma|$. The second for loop first goes once from 0 to k-1, O(k). Then, the nested for loop is similar to the previous one but here we are summing the expressions so we have k^3. This leads to a total of $\leq k^2 \cdot |\Sigma| + k^3$.

Extension: Follow a similar process to deduce the answer for Phase II.

Extension Problem: Take the PDA in the diagram below. Turn it into an NPDA, Assume a height constraint of $h = 4$ and turn it into an SLP.

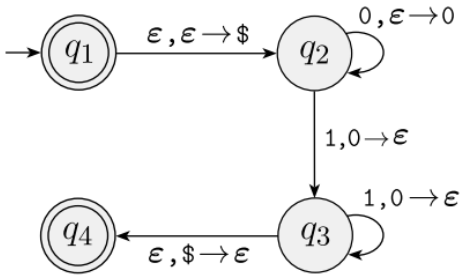


FIGURE 2.15
State diagram for the PDA M_1 that recognizes $\{0^n 1^n | n \geq 0\}$

From an SLP to a constant height NPDA

h-NPDA A_p constructed from SLP P is linear in relation to size of P. To formalize the resulting h-NPDA, we convert DAG D_p describing SLP P into A_p.

Seven-tuple NPDA $A_i = <Q_i, \Sigma, \Gamma_i, H_i, q_{o,i}, \{q_{f,i}\}>$ is derived from each instruction α_i of the SLP. There are two parts of regular expressions we must convert into an SLP:

FIRST, source nodes D_p that describe variable instructions $\alpha \in \Sigma \cup \{\varepsilon, \emptyset\}$ of SLP will turn elementary NPDA A_p (NFAs since the stack is not used):

1. If $\alpha = \emptyset$, A_i pushes symbol α onto the stack and moves to the next state.

   $A_i = <\{q_{o,i}, q_{f,i}\}, \Sigma, \emptyset, \{q_{o,i}, \alpha, q_{f,i}\}, q_{o,i}, \{q_{f,i}\}>$.

2. If $\alpha \neq \emptyset$, A_i stays in the same state, without pushing anything onto the stack

   $A_i = <\{q_{o,i}, q_{f,i}\}, \Sigma, \emptyset, \emptyset, q_{o,i}, \{q_{f,i}\}>$ (the notation is like this for consistency, but $q_{f,i}$ is unreachable).

For next part, assume $1 \leq a, b \leq i$ for all a, b and NPDAs for a, b are given. Internal nodes of D_p that describe operations $+, \circ, *$ will become the following h-NPDAs:

3. For x_i instruction $+$ in $v_a + v_b$ such that $x_i = x_a + x_b$ the h-NPDA is

   $A_i = <Q_a \cup Q_b \cup \{q_{o,i}, q_{f,i}\}, \Sigma, \Gamma_a \cup \Gamma_b \cup \{X_i\}, H_i, q_{o,i}, \{q_{f,i}\}>$, where

   $H_i = H_a \cup H_b \cup \{q_{o,i}, +X, q_{o,a}\}, \{q_{f,a}, -X, q_{f,i}\}, \{q_{o,i}, +X, q_{o,b}\}, \{q_{f,b}, -X, q_{f,i}\}$.

   This means that A_i nondeterministically chooses to simulate A_a or A_b and for each, pushes a symbol onto the stack at the start of simulation and pops that symbol at the end.

4. For x_i instruction $\circ$ in $v_a \circ v_b$ such that $x_i = x_a \circ x_b$ the h-NPDA is

   $A_i = <Q_a \cup Q_b \cup \{q_{o,i}, q_{m,i}, q_{f,i}\}, \Sigma, \Gamma_a \cup \Gamma_b \cup \{L_i, R_i\}, H_i, q_{o,i}, \{q_{f,i}\}>$, where

   $H_i = H_a \cup H_b \cup \{q_{o,i}, +L_i, q_{o,a}\}, \{q_{f,a}, -L_i, q_{m,i}\}, \{q_{m,i}, +R_i, q_{o,b}\}, \{q_{f,b}, -R_i, q_{f,i}\}$

This means that A_i sequentially to run A_a, for which it pushes $+ L_i$ at the start of simulation, then A_b, for which it pushes $+ R_i$ at the start and pops these symbols at the end of each.

5. For x_i instruction * in v_a * such that $x_i = x_a$ * the h-NPDA is

   $A_i = <Q_a \cup \{q_{o,i}, q_{f,i}\}, \Sigma, \Gamma_a \cup \{X_i\}, H_i, q_{o,i}, \{q_{f,i}\}>$, where

   $H_i = H_a, \{q_{o,i}, \varepsilon, q_{o,a}\}, \{q_{o,i}, +X_i, q_{o,a}\}, \{q_{f,a}, -X_i, q_{o,i}\}$

   This means that A_i nondeterministically chooses how many times to simulate A_a, for which it pushes $+ X_i$ at the start of simulation, then pops it. The transition with the empty string signifies that A_a can go to the next state immediately with no symbol (* signifies 0 repeats too).

What does this mean for the size of formalization from SLP to h-NPDA?

Each A_i has at most 3 new states and 2 new pushdown symbols when describing inner nodes, so $|Q_l| < 3l$ and $|\Gamma_l| < 2l$. The height of NPDA is equal to the depth of the graph describing P, so $h = D(p) < l$. As we can see from the construction, this *formalization from SLP to h-NPDA* is linear.

Extension problem. Take the SLP from example 2 in the paper and convert it into a constant height NPDA.

Constant height PDAs vs. Finite State Automata

Why are we comparing the size of constant height PDAs to the standard finite state automata?

The point of making these comparisons is to show to the reader that the sizes are optimal, i.e, in the smallest size that it could be in.

How do we show that the sizes are optimal?

- h-NPDA is equivalent to an NFA with $|Q'| \leq |Q| \cdot |\Gamma^{\leq h}|$ states.
- h-DPDA is equivalent to a DFA with no more than $|Q| \cdot |\Gamma^{\leq h}|$ states.

Based on this, NPDA $A = \langle Q, \Sigma, \Gamma, H, q0, \{q_f\}, h\rangle$ can be formalized with an equivalent DFA $B$with no more than $2^{|Q| \cdot |\Gamma \leq h|}$ states. Let's assume the B can have fewer states. Then, if we have two words that vary by at least one element, $x \in L_{\Gamma, h}$ and $y \notin L_{\Gamma, h}$ then B ends its computations in the same state for both due to the limited number of states. It implies that $x \in L_{\Gamma, h}$ iff $y \in L_{\Gamma, h}$ but we already know $y \notin L_{\Gamma, h}$ and obtain a contradiction.

Conclusion

Although not yet known to be optimal, the bounds of formalizations in this paper are:

- h-NPDA → SLP: polynomial and linear upper bounds
- h-NPDA→ h-DPDA: exponential lower bound.

If proven to be optimal, these formalizations will provide more concise representations of regular languages.

WORD COUNT: 2458 words

Appendix

Appendix A: LO/HC Applications by the Authors

Among the CS142 LOs, #automata and #complexity are contingent on how well someone explains, justifies, and analyzes a given theory based on the context. This is the key assumption we used for our grading. Although the paper is written for an audience with working technical knowledge of theories of computation, we are grading the paper with the assumption that the reader is a student of a CS142 course.

#automata

This LO was applied throughout the whole paper. Automata theory is the branch that is concerned with studying abstract machines and the problems that they can solve.

Introduction

The authors refer to several types of finite automata - mainly, NFAs and PDAs. The author establishes an important definition in the introduction - constraint height PDAs that accept regular languages. The authors make a claim that “it is not possible to bound the pushdown height by a constant”. This is an important point that describes the memory structure of pushdown automata and their difference from finite automata, yet the author does not explain why it is impossible to bound the height and how this limitation would influence the conclusions of the paper. We can infer that since PDAs have higher descriptive power than FAs, which is why they can recognize regular languages, even when the height is bounded (since regular languages are processed by memoryless finite automata). However, this is one of the central assumptions that is not elaborated.

Preliminaries

An especially strong application of the LO is seen in the transition function definitions of the NPDAs since the authors explained all possible types of transition, then justified the form used for transitions by highlighting how it helps us understand the pushdown store better. Although the authors skip over the details of how standard NPDAs and DPDAs can be turned into the proposed form, the explanation of the form is robust enough that even student-readers can understand it quickly. This section is a great example of #automata use since the definitions of the different types of automata are robust enough that many details can be skipped over.

Translations

The authors use the definition of h-NPDAs they established in the preliminaries. Especially when describing how each h-NPDA is constructed from SLP instructions, the author provides a detailed, clear explanation of how each type of node is turned into h-NPDA and clearly establishes all transitions, such that we can see the pushdown store contents for each transition,

The authors provide extreme detail in showing translations between different automata. One especially strong application is the use of 2 language examples to show that the given size of DFAs is indeed optimal. The author shows that any length less than what is established would mean that one state in the DFA accepts both a string within and a string outside the language.

Since most sections apply the LO well, we give the authors a 4.

#complexity

This LO was centered on the idea of descriptive complexity – a way of comparing how much space it takes to formalize a regular language based on the size of the formalization.

Introduction

In the introduction they never explain why going from NFAs to REs is exponential but we have put this critique under #audience rather than here.

Preliminaries

The authors give a clear operational definition of complexity through size and do a great job of providing examples of how formalization sizes are calculated.

Translations

From NPDA to SLP, authors demonstrate that cost is polynomial by summing the complexities of each phase of the algorithm. However, they do not go into detail about why the bounds are the ones that they have given. We tried clarifying this in our guide with a question and an answer.

Authors establish all other translations in great detail, which makes it easy to see how the sizes are calculated. One especially strong example of #complexity application is when the authors describe the cost of SLP → NPDA. The authors not only establish linearity, but show how the translation can be further optimized by renaming pushdown symbols. Then, the pushdown alphabet becomes equal to the fan-out of SLP. Since the authors gave detailed descriptions of formalization sizes and proved optimality when relevant, the paper can be given a 4.

#logic

The authors effectively describe a detailed algorithm for the translation process. They clearly break it up into several phases and define their roles. The algorithm is also formulated as pseudocode with clear notation, inputs, outputs, and finite number of steps.

One extremely strong use of this LO is in their proofs of optimality by contradictions, where they use 2 languages to obtain contradictions for smaller size DFAs.

#deduction: This HC goes hand in hand with the #logic LO.

#machines

Overall, we would give the authors a 4. In section 2, Preliminaries, the authors clearly build on the definitions of machines from a clearly defined NFA, to NPDA, to a constant height NPDA. They utilize formal language well by clearly articulating the formal definitions (e.g. NFA as a 5-tuple), using appropriate notation. Their approach is clear because as they build on from one machine to another they easily compare and contrast them so the reader easily understands what the new differences are. For example, they contrast NFAs and NPDAs since the latter have a stack with well defined roles. Then it is clear what the new constraint is on the stack with the constant height for the constant height pushdown automaton.

#algorithms

The authors effectively describe a detailed algorithm for the translation process. They clearly break it up into several phases and define their roles. The algorithm is also formulated as pseudocode with clear notation, inputs, outputs, and finite number of steps. To strengthen the application they could have also backed it up with an example to demonstrate how the whole process works.

#scienceoflearning

The authors use dual coding. When explaining the concept of SLPs the authors use both natural language as well as two graphical representations. Then they show the standard binary tree representation of a regular expression. Then they also show a new DAG representation of an

SLP. They do this in support as they solve example 3. This hints at the idea of dual coding since they have used two different stimuli to help the readers learn the information as they encode the information more effectively in their brains. This also helps the reader to retrieve the information easier later on throughout the paper. Another place they use dual coding is in Section 3.

The procedure to convert constant height NPDA to an SLP is expressed in natural language but also in a pseudocode. The authors also remind the reader of connections between the new material and the reader's previous knowledge (Keene's recursive algorithm) which is the Active recall strategy.

The application can be further improved by better establishing the goal of the paper and potential obstacles, so that students know what to look out for while reading the paper.

#deduction

This HC goes hand in hand with the LO #logic.

#composition

We would give them a 4. They follow the stylistic conventions of the field throughout the paper. Having read the Sipser textbook, for example, the style feels familiar. It uses nuances of the language, appropriate terminology, but not too heavy jargon which effectively communicates their point across.

#organization

We would give them a 3. The paper follows the traditional layout with abstract, introduction, preliminaries, followed by the main analysis, and then conclusion, acknowledgement, and references. It makes it easy to follow and understand with appropriate titles, subtitles, and plot labels.

#audience

Overall the application is strong and we would give them a 3. Section 2, Preliminaries, is very clear and easy to follow. The authors have highlighted important points by having subtitles for each proof, lemma, theorem, solution that are bold or italicized. However, some parts still remain unclear for a college-level reader. The authors also make many comments like “routine exercise” and “obvious” too freely, which makes it discouraging to read the paper and hard to follow their thought processes.

WORD COUNT: 1345 words.

Appendix B: HCs Applied by us

#strategize

The main stakeholders were the four people on our team. Before we began writing the assignment, we had a meeting to discuss our strengths and weaknesses. We did that by discussing our previous feedback and grades on each LO so that teammates who are strong at #logic would help out another teammate with that part of the table and also contribute to the LO analysis more. We also discussed our expectations for the assignment in terms of time commitment, availability, and quality. We set up different milestones that kept us on track. We created a timeline:

- Wednesday: read the paper thoroughly before the second session, by
- Monday: finalize individual components
- Tuesday: set up a meeting to discuss and help each other out
- Wednesday: Appendices and finalize anything else

By creating this strategic approach on how to most effectively complete the work we were able to finalize the assignment in good quality before the deadline.

#rightproblem

When we were first starting off on the assignment it was not clear to us what the purpose of adding a height constraint and why we needed SLPs. To clarify this to the reader, we clearly defined what the current initial state is and what the goal state of the paper is to show why this solution is needed at all. The initial state was defined as the disproportionate exponential gap from Regex→NFA, with the goal defined as reaching a more economical way to represent this. Then, we identify the scale of the problem, noting how concise representations have positive implications on the whole field of theory of computation. We then defined obstacles as two clear tension points, which are resolved by SLPs and h-NPDAs, thus establishing the basis for the further proofs in the paper. The application of this LO helps with readability, structuring, and strengthens use of #science of learning by providing a visual aid for the problem definition.

#responsibility

We are overall very proud of the level of responsibility our team exhibited while working on the assignment together. All of our group members were very proactive. We had a groupchat where we would often send updates on our progress as well as any questions we had and others would try to help by answering them. We also sent out calendar reminders and meeting invitations that motivated us to have the work finalized before every time we met. We missed some of the milestones we had set for ourselves (from the timeline in #strategize). Still, when that happened the teammates gave good reasons long before the deadline, developed future steps to get back on track and met those commitments. Everyone tried to do their best work and collaborate which led to a strong assignment. Since we established this norm of being proactive

and responsible early on, and the last two sessions of class helped with that, the atmosphere and communication in the team was great.

References

Geffert, V., Mereghetti, C., & Palano, B. (2010). More concise representation of regular languages by automata and regular expressions. Information and Computation, 208(4), 385–394. doi:10.1016/j.ic.2010.01.002.