Formal Models for Data Languages

Jan Vašák, 2024
Supervisor: Ing. Ondřej Lengál Ph.D

BRNO UNIVERSITY OF TECHNOLOGY — FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2024

Automata Models

Data Words. This work is mostly focused on automata models over *data words*. Data words are sequences of pairs of *symbols* and *data values*. Symbols are elements of a finite *alphabet* Σ, and data values are elements of a countably infinite *data domain* $\mathbb{D}$. For example,

$$(a, 1)(b, 2)(a, 42)$$

is a data word over $\Sigma = \{a, b\}$, and $\mathbb{D} = \mathbb{N}$ ($\mathbb{N}$ being the set of natural numbers). Data words are commonly used in formal models working with infinite sets (e.g. integers), or large finite sets (e.g. Unicode symbols).

Register Automata. A register automaton (RA) extends a finite automaton with a finite set of *registers*. Each register can store up to one data value. When running a word, the RA can check the (non-)equality of the current input data value (*in*) to all its registers.

Register Set Automata. A register set automaton (RsA) is the same as an RA, except its registers (*set-registers*) store a set of data values instead of just one and have (non-)membership tests instead of (non-)equality tests. RsA^{rm} is an extension of RsA allowing the removal of *in*.

History Register Automata. A history register automaton (HRA), like an RsA, also has set-registers storing sets of data values. However, an HRA updates its registers in a different way to an RsA. For an HRA transition, two sets of registers a specified. The first set species which set-registers exactly the input value must be stored in to enable the transition. The second specifies which set-registers exactly it will be stored after the transition is taken. HRAs also allow transitions that reset registers, but do not read any input.

Streaming Data-String Transducers. A (deterministic) streaming data-string transducer (SDST) is a transducer model with a set of *data-variables* (RA-style registers), and a set of *data-string variables*, each storing a data word. Additionally, SDSTs operate on a totally ordered data domain, and thus allow to test inequality of data-variables to the current input data value.

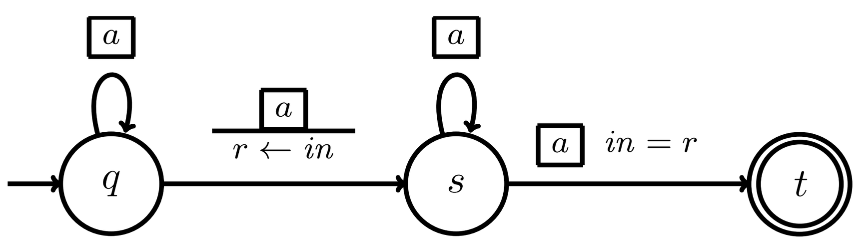

Figure 1: An RA accepting the language of words, whose last data value is not unique

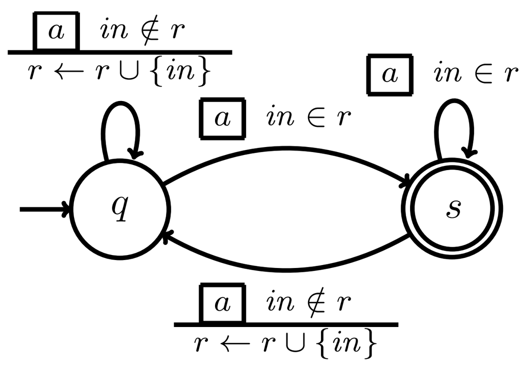

Figure 2: An RsA accepting the language from Figure 1

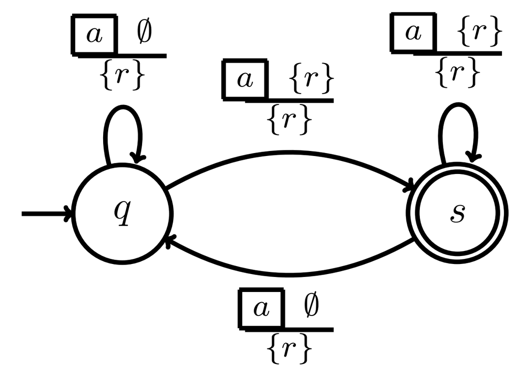

Figure 3: An HRA accepting the language from Figure 1

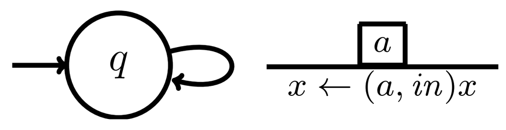

Figure 4: An SDST reversing the input string (its output is defined as *x* in *q*)

Extending SDSTs

Programs as SDSTs. In their original form, SDSTs can represent (imperative and functional) single-pass list-processing programs. Thus, they can be used for formal analysis and verification of such programs. Examples of such programs include a program reversing the input list (see Figure 4), or a program checking whether the input list is sorted.

SDST Extension. We extend SDSTs by adding a set of set-registers, that can be tested for (non-)membership of the current data-value and updated by adding or removing a data value. However, data-variables are restricted from the original model in that they cannot be tested for inequality, only (non)-equality.

The main result for our extension is the following.

Theorem 1. *The functional equivalence problem for SDSTs with set-registers is decidable.*

Which is an important result for formal analysis and verification use. Our extension could be used to represent single-pass list-processing programs with a set type data-structure available for them to use.

```
def remove_duplicates(input_list):
    result = list()
    set = set()
    For i in input_list:
        if i not in set:
            result.append(i)
            set.add(i)
    return result
```

Figure 5: A program representable by an SDST with set-registers (and not by an SDST)

Relating RsAs and HRAs

We compare the expressive powers of the two models. First, we state that all HRAs can be converted to RsA^{rm}s.

Proposition 2. *HRA $\subseteq$ RsA^{rm}.*

The same proof can be used for their deterministic variants as well.

Corollary 3. *DHRA $\subseteq$ DRsA^{rm}.*

The other direction of Proposition 2 is left as an open problem. However, we do have a result for the deterministic variants.

Proposition 4. *DHRA $\subsetneq$ DRsA^{rm}.*

I.e., deterministic RsA^{rm}s are more expressive than deterministic HRAs.

RsA Emptiness Parametrization

The *emptiness problem* is the problem of whether an RsA accepts any string at all. We parametrize its complexity based on the number of registers for both the normal variant and the removal extension.

Proposition 5. *The emptiness problem for RsA$_1$ is NL-complete.*

Proposition 6. *The emptiness problem for RsA$_1^{rm}$ is NL-complete.*

Proposition 7. *The emptiness problem for RsA$_n^{rm}$ is in $\mathbf{F}_{(2^n+1)}$.*

Corollary 8. *The emptiness problem for RsA$_n$ is in $\mathbf{F}_{(2^n+1)}$.*

RsA-based Regex Matching

Regex Matcher. A regex matcher was built using RsAs as a model for matching. It uses the regex parser from Python's re module [5] and constructs an RA from the acquired syntax tree. The RA is then determinised using an existing algorithm [3] that can determinise a class of RAs into DRsA. The desired input is then run on the constructed DRsA.

Experiments. Using the ReDOS attack generators rescue [6] and rxxr2 [1], vulnerable regexes with back-references were extracted from a set of regexes used in practice. The regexes with their generated attack strings were then run on the RsA matcher, Python's re module, the pcre2 library [4], and GNU grep [2].

Though the generators were unable to defeat grep, we show that it is possible with a hand-crafted regex and input in Figure 6. Figures 7 and 8 show the results of the pcre2 and re for the determinised regexes.

Table 1: Numbers of timeouts (10 s) for each matcher on the determinised regexes only.

	Total	deter-minised	DRsA timeouts	pcre2 timeouts	re timeouts	grep timeouts
rxxr2	97	47	1	36	43	0
rescue	60	22	1	18	21	0

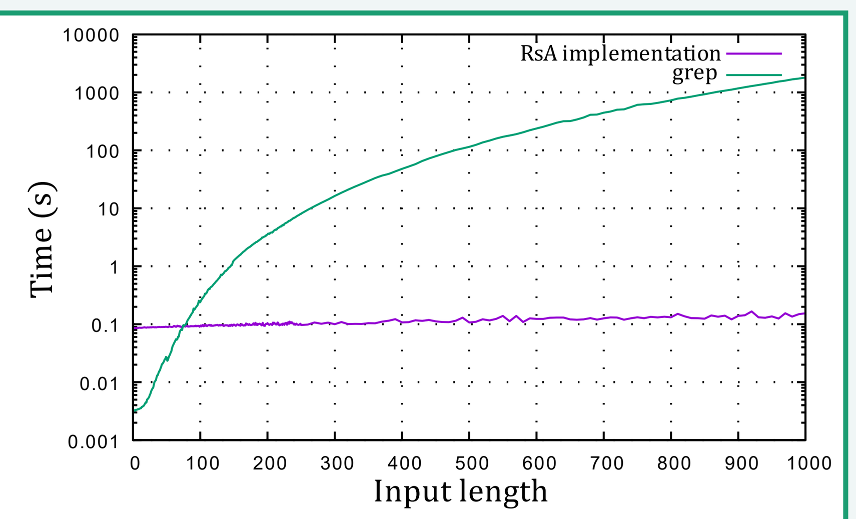

Figure 6 (top right): RsA and grep comparison for a hand-crafted regex and input

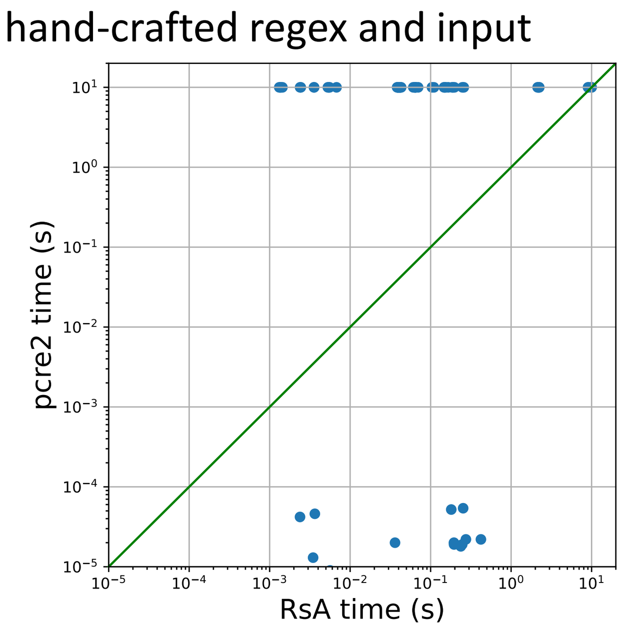

Figure 7: A scatterplot comparing pcre2 and RsA for merged rxxr2 and rescue data

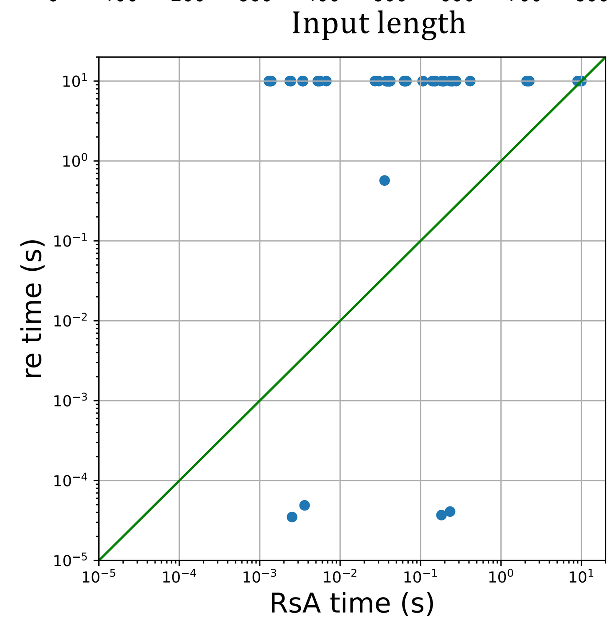

Figure 8: A scatterplot comparing re and RsA for merged rxxr2 and rescue data

[1] Asiri Rathnayake and Hayo Thielecke. Static analysis for regular expression exponential runtime via substructural logics. CoRR, abs/1405.7058, 2014
[2] Free Software Foundation, Inc. GNU grep 3.6. 2021 [online]. [cit. 2022-02-02]. Available at: https://git.savannah.gnu.org/cgit/grep.git.
[3] Gulčíková, S. and Lengál, O. Register Set Automata (Technical Report). arXiv, 2022 [online]. DOI: 10.48550/ARXIV.2205.12114.]. Available at: https://arxiv.org/abs/2205.12114.
[4] Hazel, P. Perl-compatible Regular Expressions. Version 10.42. 2022 [online]. [cit. 2024-04-12]. Available at: https://www.pcre.org.
[5] Python Software Foundation. Python Standard Library - re Module. Version 3.10.12. 2023 [online]. [cit. 2024-04-20]. Available at: https://docs.python.org/3/library/re.html
[6] Yuju Shen, Yanyan Jiang, Chang Xu, Ping Yu, Xiaoxing Ma, and Jian Lu. Rescue: crafting regular expression DoS attacks. In ASE'18, pages 225–235. ACM, 2018.