The Theory of Finite-State Adventures

Wilfried Brauer[1], Markus Holzer[1], Barbara König[1], and Stefan Schwoon[2]

[1] Institut für Informatik, Technische Universität München, Germany
[2] Institutsverbund Informatik, Universität Stuttgart, Germany

{brauer,holzer,koenigb}@in.tum.de schwoosn@informatik.uni-stuttgart.de

Abstract. The study of finite-state adventures, abbreviated by FSA, is a topic that has, despite its origins that can be traced back to medieval times, been neglected in the relevant literature. We attempt to close this gap and give a full account of a five-level hierarchy of finite-state adventures.

1 Introduction

In this paper we treat the theory of finite-state adventures, abbreviated by FSA. This suite of excercises was designed for a second-year undergraduate course in theoretical computer science held by Wilfried Brauer at the Technische Universität München in the summer of 2001. The course covered, among other topics, the theories of automata, formal languages and computability.

In order to motivate the students and in order to encourage active learning [MAB93,GK93], it was our aim to devise examples that are both funny and illustrative for the structure of the Chomsky hierarchy. Furthermore we introduced the tool Grail (`http://www.csd.uwo.ca/research/grail/`) which can be used to manipulate finite automata and regular languages, and furthermore to solve puzzles similar to the ones given below.

In the following we will present these exercises as a five level adventure and shortly sketch the solutions. In the conclusion we will give a short summary of the reception of these exercises by students.

2 A fairy tale

In a university in the town there lived a professor. His students had loathed formal language theory for time out of mind, and you could tell their opinion on the subject without the bother of asking them. This is the story of how the students had an adventure, and found themselves doing and enjoying things altogether unexpected.

This paper is a tale of high adventure, undertaken by a company of lecturers in search of dragon-guarded gold. Their reluctant partners in this perilous quest were the students, comfort-loving unambitious creatures, who surprised

even themselves by their resourcefulness and skill. Encounters with finite automata, Petri nets, and flow diagrams, and a rather unwilling presence at the final examination, are just some of the adventures that befell them.

Our fairy tale takes place in a landscape through which the Great River meanders and where one can find enchanted doors, magic archways and useful keys aplenty. However, there also lurks a dangerous breed of dragons, which can be defeated only with a legendary sword.

Certainly, our brave adventurers do not set out without a map, an example of which can be seen in Figure 1, its symbols representing *dragons* (D), *swords* (S), *arches* (A), *rivers* (R), *gates* (G), *treasures* (T) or *keys* (K). We draw maps as finite automata where the possible paths correspond to words accepted by the automaton. The symbols form the alphabet Σ and we will either draw them or denote them by the letters given in brackets above. The aim is to find treasures, to pass any dragon unharmed and not to be hindered by a gate or arch.

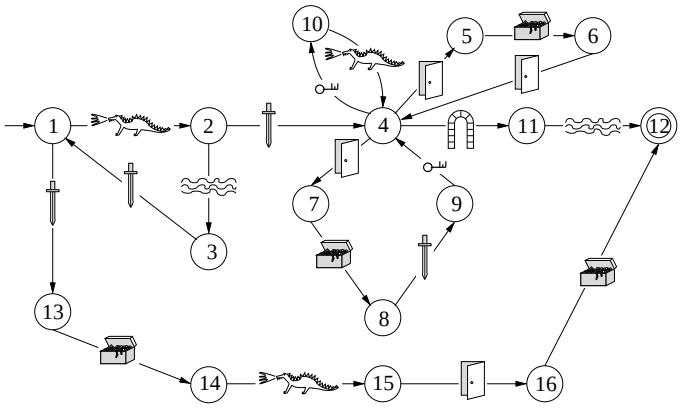

Fig. 1. An example FSA.

Adventures come in several variants which are now listed from the easiest task to the most difficult.

3 Finding a path (Level 0)

The zeroth level of the adventure consists of finding a path from an initial state to a final state. This is easy. Everybody can do this with breadth-first search or depth-first search.

4 Finding treasures (Level 1)

The main point of an adventure—of course—is to get treasures. But naturally it would be too easy without any challenges and dangers that our bold adventurer

has to face. So we demand that on a valid path from the initial state to a final state

(G) we can pass a gate only after we have found a key (a key can be used arbitrarily often and opens every gate);
(D) we have to jump into a river immediately after we have met a dragon, because the dragon will set us on fire; however, once we have found a sword, we can kill the dragon before it burns us and need not jump into the river;
(T) we must find at least two treasures.[3]

Looking at the adventure in Figure 1, it is not very difficult to figure out a solution. By following the state sequence $1, 2, 3, 1, 2, 4, 10, 4, 5, 6, 4, 5, 6, 4, 11, 12$ (length 16) one obtains a word of the language which is a solution. The question is of course: How can we solve problems of this kind in general?

The answer is to represent the conditions as finite automata G, D and T (see Figure 2) and to take the intersection (cross-product) with the automaton M describing the map. The adventure has a solution if and only if the language accepted by the resulting automaton is non-empty.

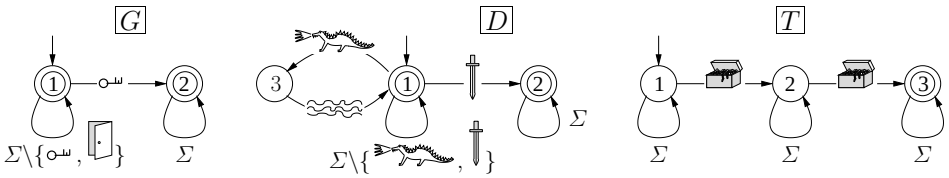

Fig. 2. Conditions represented by finite automata.

By using the finite-automaton tool Grail, students were able to compute the intersection of the languages, to test whether it was non-empty and to list possible solutions. For example, the Grail file `g.aut` representing the automaton G in Figure 2 looks as follows:

```
(START) |- 1
1 K 2
1 D 1
1 S 1
1 T 1
1 A 1
1 R 1
2 D 2
2 G 2
2 K 2
2 S 2
2 T 2
2 A 2
2 R 2
1 -| (FINAL)
2 -| (FINAL)
```

Using the Grail commands `fmcross` (for cross-product) and `fmenum` (for language enumeration) we obtain:

[3] In our universe, treasures will be replaced as soon as we leave the corresponding node, and the same is true for keys and killed dragons.

```
> fmcross m.aut < t.aut | fmcross g.aut | fmcross d.aut | fmenum
DRSDSKDGTGGTGAR
DRSDSKDGTGGTSKAR
DRSDSKDGTSKGTGAR
DRSDSKDGTGGTGKDAR
DRSDSKDGTGKDGTGAR
[...]
```

The first (and shortest) word accepted by the resulting automaton is the solution given above.

5 Counting keys (Level 2)

In the next level of difficulty we have magic keys which disappear as soon as we have opened a gate with them. Gates close again immediately after we have stepped through them. We can collect and possess as many keys as we like. This new condition is denoted by (G'). The other two conditions (D) and (T) remain unchanged.

By using the Pumping Lemma we first show that the language of all solutions of an adventure on this level is, in general, not regular. We consider the adventure depicted in Figure 3. Obviously the language of all possible solutions is $L = \{ K^m G^n T^2 \mid m \geq n \geq 1 \}$. It is easy to show with the Pumping Lemma for regular languages (uvw-Theorem) that L is not regular.

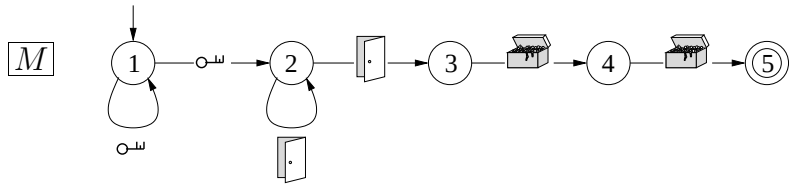

Fig. 3. An adventure generating a non-regular language in level 2.

It is not hard to argue that the language of the adventure is context-free. Obviously $L(M) \cap L(D) \cap L(T)$ is regular and can be represented by a finite automaton. The language $L(G')$ of all paths satisfying condition (G') is context-free since it is accepted by a pushdown automaton counting the available keys. By transforming this pushdown automaton we can also obtain a context-free grammar generating $L(G')$.

The intersection of the regular language $L(M) \cap L(D) \cap L(T)$ and the context-free language $L(G')$ is again context-free and can be computed and tested for emptiness. We supplied students with a Perl script which takes as input a finite automaton in Grail notation and a context-free grammar and tests whether the

intersection of their languages is empty. This algorithm was described in an earlier EATCS bulletin [ERS00].

```
> fmcross m.aut < t.aut | fmcross d.aut > mtd.aut
> isempty-fa-cfg mtd.aut g.gram
The intersection is not empty.
```

This procedure does not produce an explicit solution, but it is not hard to see that the shortest solution for our example adventure can be obtained by following the state sequence 1, 2, 3, 1, 2, 4, 10, 4, 7, 8, 9, 4, 7, 8, 9, 4, 11, 12 (length 18).

6 Counting keys and swords (Level 3)

For this level, we extend level 2 with the following rule: If we have a sword and meet a dragon, we can kill the dragon with the sword, but the sword will be spoilt from dragon blood and will be unusable thereafter. An alternative is of course to keep the sword and jump into a river in the next step.

It is not hard to show that the language of solutions is, in general, not context-free. We consider the automaton in Figure 4. The language of all possible solutions is $L = \{ K^k S^\ell G^m D^n T^2 \mid k \geq m \geq 1, \ell \geq n \geq 1 \}$. The Pumping Lemma for context-free languages ($uvwxy$-Theorem) can be conveniently used to show that L is not context-free.

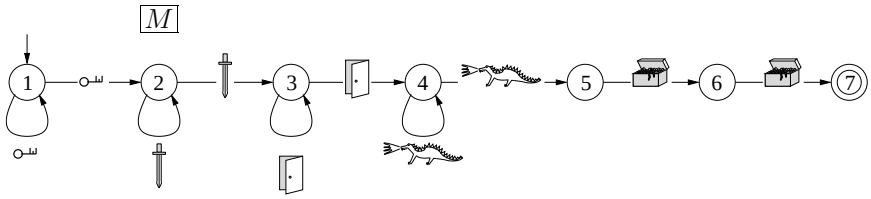

Fig. 4. An adventure generating a non-context-free language in level 3.

In order to show the decidability of this level, we can model the adventure with Petri nets. We first take the intersection of the map M with the finite automaton T representing the treasure condition; the resulting automaton is denoted by A_{MT}. We then construct a Petri net that corresponds to A_{MT}. Every state in this automaton is represented by a place and every edge by a Petri net transition, labelled by the same symbol. We add two more places to count the keys and swords we have collected. These are connected to the rest of the Petri net as depicted in Figure 5. Special care has to be taken with transitions representing encounters with dragons, since we still have the option of jumping into a river. So whenever a river follows right after a dragon, we also insert a

special dragon-river transition which does not consume a sword. Figure 5 shows schematically how a given adventure like the one in Figure 1 can transformed into a Petri net according to the rules of level 3. Note that all places apart from the ones counting keys and swords are 1-safe.

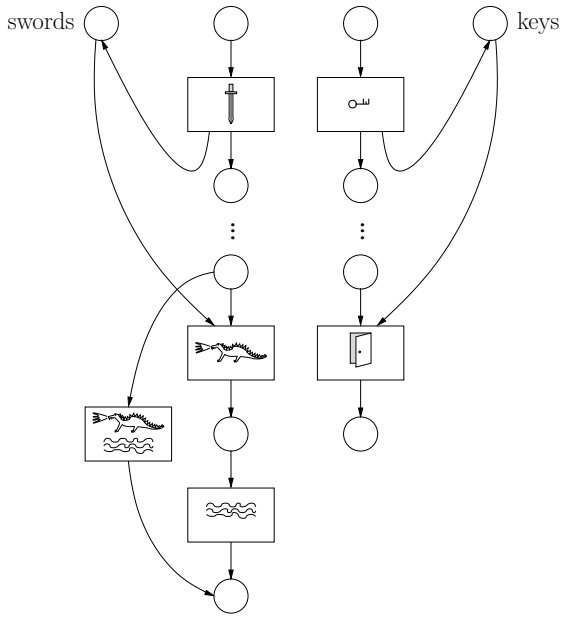

Fig. 5. Transforming a level 3 adventure into a Petri net.

Now, the adventure is solvable if and only if one of the places associated with a final state of the automaton can be covered by a marking. This problem is decidable and can be solved with the help of coverability graphs [Rei85].

The shortest solution to the adventure in Figure 1 is given by the state sequence 1, 2, 3, 1, 2, 4, 10, 4, 7, 8, 9, 4, 7, 8, 9, 4, 11, 12 (length 18).

7 Forbidden rivers and arches (Level 4)

In level 4 everything is as in level 3 with the added complication that we cannot cross a river if we possess a sword, because swords are too heavy and we would drown. Moreover, there is a magic arch, which does not allow us to pass whenever we possess a key. (No, sorry, throwing away of items is not allowed.)

Because of these two tests for emptiness, level 4 adventures can simulate a two-counter-machine [HU79]. In the course, students were not introduced to counter machines, but learned about LOOP, WHILE and GOTO programs [Wei87]

and their place in the computational hierarchy. GOTO programs with two variables x_1, x_2, which correspond to two-counter-machines, can be expressed in the following syntax:

$$P \ ::= \ L\colon S; P \mid L\colon \mathsf{HALT}$$

where L is a label and S is a statement of one of the following forms:

$$\begin{aligned} S ::= {} & x_i := 0 \mid x_i := x_i + 1 \mid x_i := x_i - 1 \mid \mathsf{GOTO}\ L \\ & \mid \ \mathsf{IF}\ (x_i = 0)\ \mathsf{THEN\ GOTO}\ L \mid \mathsf{HALT} \end{aligned}$$

Note that variables hold only non-negative integers and that $0 - 1$ is defined to be 0. Initially all variables are set to 0. One can easily see that GOTO programs can be represented by flow diagrams [Wei87] with the boxes as depicted on the left-hand side of Figure 6 and connected by directed arcs.

Figure 6 also shows how these boxes can be translated into adventures. Boxes can be connected by transitions with treasures as their labels. In order to satisfy condition (T), two treasures can be collected immediately before reaching a final state.

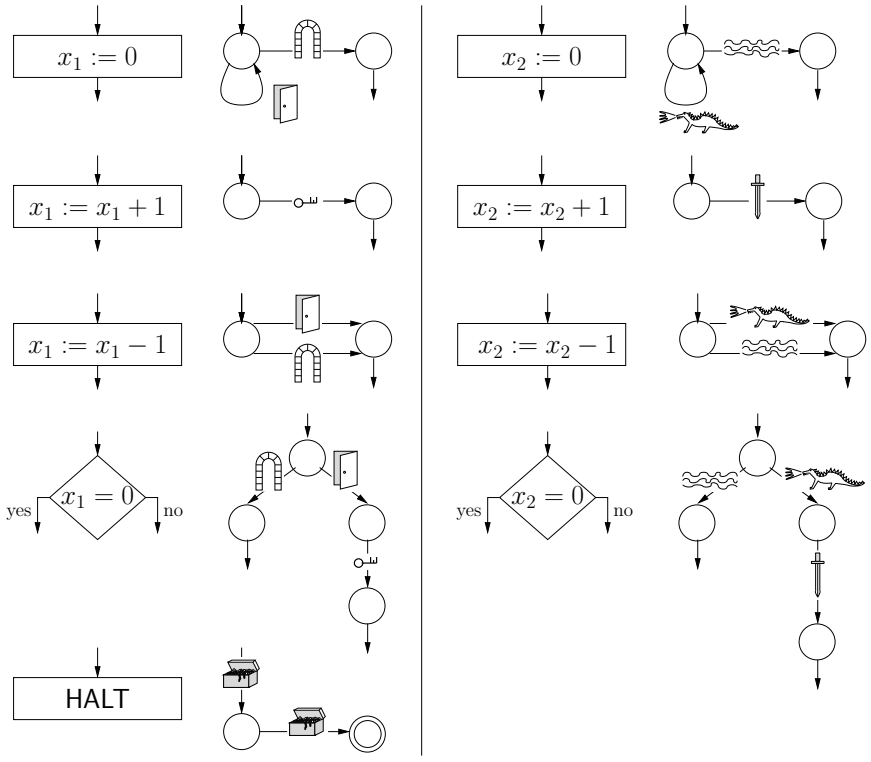

Fig. 6. Translation of flow diagrams into adventures.

It can be shown that a GOTO program with two variables terminates if and only if its translation into an adventure has a solution, and we have therefore reduced the halting problem of two-variable GOTO programs to the adventure problem (level 4). Since the halting problem is undecidable, so is the adventure problem.

Nonetheless, we have designed our example adventure in Figure 1 in such a way that it actually has a solution, we could, for example, follow the state sequence 1, 2, 3, 1, 2, 4, 10, 4, 7, 8, 9, 4, 10, 4, 5, 6, 4, 11, 12 (length 19).

8 Conclusion

The student reaction, given by personal feedback and in the lecture evaluation, was very positive. Students strongly approved of the use of Grail and the illustrative nature of the adventure example. Level 3 was not treated in the lecture, since Petri nets were not part of the course, but was included in this article since it adds a nice additional hierarchy level, even if it is not entirely compatible with the Chomsky hierarchy. All in all, the authors were also having quite a lot of fun in designing and solving these exercises.

Acknowledgements: We would like to express our thanks to the developers of Grail who provided us with a great tool. And we would like to acknowledge Javier Esparza who was the first person to solve level 3.

References

[ERS00] J. Esparza, P. Rossmanith, and S. Schwoon. A uniform framework for problems on context-free grammars. *EATCS Bulletin*, 72:169–177, October 2000.

[GK93] L. Gow and D. Kember. Conceptions of teaching and their relationship to student learning. *British journal of educational psychology*, 63:20–33, 1993.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, languages and computation.* Addison Wesley, Reading, Massachusetts, 1979.

[MAB93] F. Marton, G. Dall Alba, and E. Beaty. Conceptions of learning. *International Journal of Educational Research*, 19(3):277–300, 1993.

[Rei85] W. Reisig. *Petri Nets: An Introduction.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1985.

[Wei87] Klaus Weihrauch. *Computability.* Springer-Verlag, 1987.