Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Supporting system design with automaton learning algorithms

Bachelor's Thesis

Author
Áron Barcsa-Szabó

Advisor
Rebeka Farkas
Dr. András Vörös

November 26, 2019

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Barcsa-Szabó Áron*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2019. november 26.

Barcsa-Szabó Áron
hallgató

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomatervnek. A sablon használata opcionális. Ez a sablon LaTeX alapú, a *TeXLive* TeX-implementációval és a PDF-LaTeX fordítóval működőképes.

Abstract

This document is a LaTeX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* TeX implementation, and it requires the PDF-LaTeX compiler.

Chapter 1

Introduction

1.1 Overview

1.2 Problem Statement

1.3 Objective

1.4 Contribution

1.5 Outline

Chapter 2 provides an outline of the necessary technical background.

Chapter 2

Background

This chapter provides some theoretical background of the contributions presented in this thesis. First of all, the necessary basics of formal language and automaton theory are introduced, afterwards we discuss automaton learning algorithms.

2.1 Basics of automaton theory

First, we introduce the fundamentals of formal language theory, which are essential in order to understand automaton theory. Atomic elements of formal languages are alphabets, characters and words.

Definition 1 (Alphabet). Let Σ be a finite, non-empty set. Σ is an Alphabet, its elements are symbols or characters. ▪

Definition 2 (Word). If Σ is an alphabet, then any finite sequence comprised of the symbols of Σ are words (or Strings). Σ^n represents the set of every n length word in Σ. The set of every word under an alphabet, formally $\bigcup_{n>0} \Sigma^n$ is denoted by Σ^*. The empty word is denoted by ϵ. ▪

Words can be constructed using other words, the following definition defines these relations.

Definition 3 (Prefixes, Substrings and Suffixes). If $w = uvs$, where $w, u, v, s \in \Sigma^*$, u is the prefix, v is the substring, and s is the suffix of w. ▪

Using these atomic elements of formal language theory, the definition of formal language theory can be seen as follows.

Definition 4 (Formal Language). An arbitrary set of words under an Alphabet Σ is a Language. Formally: $L \subseteq \Sigma^*$. ▪

Definition 5 (Prefix-closure). Let $L \subseteq \Sigma^*$ and $L' = \{u \in \Sigma^*, v \in \Sigma^* : uv \in L\}$. In other words, L' is the set containing all the prefixes of every word of L. L is prefix-closed if $L = L'$. ▪

We will expand on formal languages more once we have a basic understanding of automatons.

Informally, Automatons, or automata are mathematical constructs which are able to determine if a sequence of inputs should be accepted or rejected. A bit more precisely, automata consist of states and is always in one of them. Starting from an initial state, based on the inputs received, the automaton changes, "moves" between states. Essentially, for every one of the inputs, based on the current state the automaton is in, it determines whether to keep, or change its current state. In order to determine if an input sequence should be accepted or not, some states are special, accepting states. If after processing a sequence of inputs, the final state of the automaton is accepting, the input sequence is accepted. If not, the input is rejected.

One of the most simple automaton is the Deterministic Finite Automaton.

Definition 6 (Deterministic Finite Automaton). A Deterministic Finite Automaton is a Tuple of $DFA = (S, s_0, \Sigma, \delta, F)$, where:

- S is a finite, non-epty set containing the states of the automaton,
- $s_0 \in S$ is the initial state,
- Σ is a finite Alphabet,
- $\delta : S \times \Sigma \to S$ is a transition function,
- $F \subseteq S$ is a set of the accepting states of the automaton. ▪

An example of a DFA (Deterministic Finite Automaton) from[7] can be seen in Example 1.

Example 1. *See Fig. 2.1. This example has four states, $S = \{q_0, q_1, q_2, q_3\}$ (hence $|S| = 4$). The initial state is marked by the start arrow, so $s_0 = q_0$. The alphabet can be inferred as $\Sigma = \{a, b\}$ because of the deterministic in Deterministic Finite Automaton, meaning every state must deterministically know what input causes what action. This means, that every state must have every member of the alphabet listed in its transitions. Transitions are visualized as $q_0 \xrightarrow{a} q_1$ given by the transition function (in this example) $\delta(q_0, a) = q_1$. The whole of the transition function in a table form can be seen in Table 2.1. Finally, the accepting states, or in this case, accepting state of the automaton is $F = \{q_3\}$.*

When talking about automata, it is important to discuss runs. A run of an automaton is to test for a certain input (word), if it is accepted or rejected. See Example 2.

Example 2. *A run of Fig. 2.1 with an input of $\{a, a, a\}$ would (following the transition function) end in state q_3 meaning the input is accepted. A rejected input could be $\{a, b, b\}$, which would stop at state q_1, a non-accepting state. On deeper examination, one can see, that this automaton only accepts runs with inputs containing $4i + 3a$.*

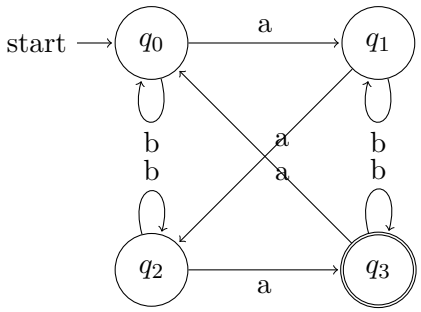

Figure 2.1: A simple DFA

δ	q_0	q_1	q_2	q_3
a	q_1	q_2	q_3	q_0
b	q_0	q_1	q_2	q_3

Table 2.1: The transition function of the automaton seen in Fig. 2.1

DFAs are great to model system behavior based on inputs, but in order to work with reactive systems, we also need to handle outputs. Mealy machines are automata designed to do just this, while becoming more complicated with regard to accepting inputs.

Definition 7 (Mealy machine). A Mealy machine or Mealy automaton is a Tuple of $M = (S, s_0, \Sigma, \Omega, \delta, \lambda)$, where:

- S is a finite, non-empty set containing the states of the automaton,
- $s_0 \in S$ is the initial state,
- Σ is the input alphabet of the automaton,
- Ω is the output alphabet of the automaton,
- $\delta : Q \times \Sigma \to Q$ is the transition function and
- $\lambda : Q \times \Sigma \to \Omega$ is the output function. ▪

Mealy machines can be regarded as deterministic finite automata over the union of the input alphabet and an output alphabet with just one rejection state, which is a sink, or more elegantly, with a partially defined transition relation.

An example of a deterministic Mealy machine can be seen in Example 3.

Example 3. *An example of a deterministic Mealy machine can be seen in Fig. 2.2. The formal definition of the automaton can be seen below.*

- $S = \{a, b, c, d, d', e, f\}$
- $s_0 = a$
- $\Sigma = \{water, pod, button, clean\}$
- $\Omega = \{\checkmark, ☕, \star\}$

The transitions, as seen in Fig. 2.2 are visualized as $s_0 \xrightarrow{input/output} s_1$*, which denotes the machine moving from state* s_0 *to state* s_1 *on the specified input, while causing the specified output. Also, some simplifications are done, e.g. in this transition:* $d \xrightarrow{\{water,pod\}/\checkmark} d$ *we see a visual simplification of having both transitions merged to one arrow, this is only for visual convenience. Fig. 2.2 is also a great example of sinks, as seen in state f, the machine accepts anything, and never changes. This is a variation of the accepting state seen in DFAs.*

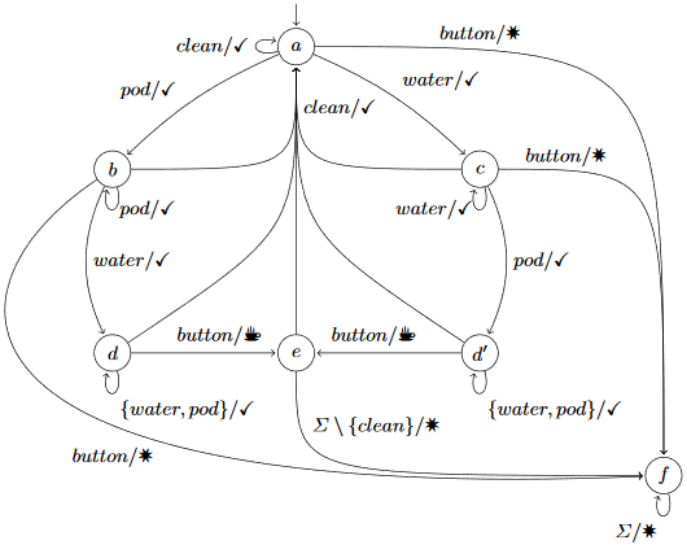

Figure 2.2: Mealy machine representing the functionality of a coffee machine.[7]

Since automatons deal with alphabets, formal language theory is essential not only to work with them, but to build them efficiently. Often automata are used to model and test real-life systems. Naturally, questions of efficiency, correctness arise, which is why we will next expand on the relations of automatons and formal languages.

Definition 8 (Recognized language of automata). The language $L \subseteq \Sigma$ containing all the accepted words by an automaton M is called the recognized language of the automaton. It is denoted by L(M) = L. ▪

Definition 9 (Regular language). A formal language L is regular, iff there exists a Deterministic Finite Automaton M, for which L(M) = L, in other words, iff there exists a DFA with the recognized language of L. ▪

We will now introduce a semantic helper δ^* for both DFAs and Mealy machines. δ^* is an extension of the δ transition function, as $\delta^* : S \times \Sigma^* \to S$ defined by $\delta^*(s, \epsilon) = s$ and $\delta^*(s, \alpha w) = \delta^*(\delta(s, \alpha), w)$, essentially giving us the state of the automaton after running an input sequence from a specified state.

Definition 10 (Myhill-Nerode relation). A DFA $M = (S, s_0, \Sigma, \delta, F)$ induces the following equivalence relation $\equiv_M$ on Σ^* (when L(M) = Σ):

$$x \equiv_M y \iff \delta^*(s, x) = \delta^*(s, y)$$

where $x, y \in \Sigma^*$. This means, that x and y are equivalent with respect to $\equiv_M$.[4]. ▪

The Myhill-Nerode relation is an equivalence relation, meaning $x \equiv_M y$ is reflexive, symmetric and transitive. Also, the following properties apply to it[4].

- Right congruence: $\forall x, y \in \Sigma^* : (x \equiv_M y \implies \forall a \in \Sigma : xa \equiv_M ya)$
  also, by induction, this can be extended to:
  $\forall x, y \in \Sigma^* : (x \equiv_M y \implies \forall w \in \Sigma^* : xw \equiv_M yw)$.
- It respects membership wrt. R:
  $\forall x, y \in \Sigma^* : x \equiv_M y \implies (x \in R \iff y \in R)$

- $\equiv_M$ is of finite index, has finitely many equivalency classes. Since for every state $s \in S$, the sequences which end up in s are in the same equivalence class, the number of these classes is exactly $|S|$, which is a finite set.

Using this relation, we can introduce the Myhill-Nerode theorem, which neatly ties together the previous definitions.

Theorem 1 (Myhill-Nerode theorem[4][6]). Let $L \subseteq \Sigma^*$. The following three statements are equivalent:

- L is regular.
- there exists a Myhill-Nerode relation for L.
- the relation $\equiv_L$ is of finite index.

For proof, see [4][6]. ▪

Next, we will introduce the same topics to Mealy machines, which are a bit more complex in this regard. In order to do this, we need to introduce a semantic helper similar to δ^*, but considering the output function of Mealy machines. $\lambda^* : S \times \Sigma^* \to \Omega$, defined by $\lambda^*(s, \epsilon) = \varnothing$ and $\lambda^*(s, w\alpha) = \lambda(\delta^*(s, w), \alpha)$.

When monitoring the behavior of Mealy machines, one of the most important metrics given an input is not whether it is accepted or rejected (as it would be with a DFA), but rather what specific output was caused by an input. The behavior of a Mealy machine, a specific run of it, has a pattern of $i_1, o_1, i_2, o_2, .., i_n, o_n$, where i are inputs and o are outputs. But in order to characterize these runs, we actually do not need every output from this pattern, we only need the final one. This means, that a $[\![M]\!]$: $\Sigma^* \to \Omega$ semantic functional fully captures the behavioral semantics of Mealy machines. We define $[\![M]\!]$: $\Sigma^* \to \Omega$ as $[\![M]\!](\mathrm{w}) = \lambda^*(s_0, w)$. Informally,the $[\![M]\!]$ functional accepts a set of inputs, and returns the last output given after running them from the initial state of the automaton.

Example 4. *Given the Mealy machine $M_{coffeemachine}$ in Fig. 2.2, the runs:*
<clean, ✓>,
<pod water button, ☕>
are in $[\![M_{coffeemachine}]\!]$, since these input words cause these outputs, while the runs
<clean, ☕> and
<water button button, ✓>
are not, since these input sequences do not produce those outputs.

Similarly to the Myhill-Nerod relations in DFAs, we can introduce an equivalence relation over the $P : \Sigma^* \to \Omega$ functional. P being an abstraction of $[\![M]\!]$, since the latter only maps from initial states.

Definition 11 (Equivalence of words wrt. $\equiv_P$[7]). Given a Mealy machine $M = (S, s_0, \Sigma, \Omega, \delta, \lambda)$, two words, $u, u' \in \Sigma^*$ are equivalent with respect to $\equiv_P$:
$u \equiv_P u' \iff (\forall v \in \Sigma^* : P(s, uv) = P(s, u'v))$.
We write [u] to denote the equivalence class of u wrt. $\equiv_P$. ▪

This definition is more along the lines of the right congruence property observed in the Myhill-Nerode relations, the original ($u \equiv_P u' \iff P(s, u) = P(s, u')$) format would of course still hold, in this case, $v = \epsilon$.

Example 5. *Taking Fig. 2.2 as an example, the following words are equivalent wrt.* $\equiv_{[\![M]\!]}$*:*

$$\begin{aligned} & \textit{water, pod} \\ \equiv_{[\![M]\!]} \quad & \textit{water, water, pod} \\ \equiv_{[\![M]\!]} \quad & \textit{pod pod water.} \end{aligned}$$

The first two of these are straightforward, since both lead to the same state, d′, more interesting is the pod pod water input, which ends in state d. Observably, state d and d′ wrt. outputs operate exactly the same regardless of continuation, this is why the equivalence holds.

Theorem 2 (Characterization theorem[7]). Iff mapping $P : \Sigma^* \to \Omega$ $\equiv_P$ has finitely many equivalence classes, there exists a Mealy machine M, for which P is a semantic functional.

Proof($\Longleftarrow$): As seen in the case of the Myhill-Nerode finite index property for DFAs, same states in Mealy machines will obviously be in same equivalence classes. This implies, that the number of classes in (or in other words, the index of) $\equiv_P$ is at most the number of states the Mealy machine contains, which is finite by definition.
Proof($\Longrightarrow$): Consider the following Mealy machine: $M_P = (S, s_0, \Sigma, \Omega, \delta, \lambda)$:

-S is given by the equivalence classes of $\equiv_P$.
-s_0 is given by $[\epsilon]$.
-δ is defined by $\delta([u], \alpha) = [u\alpha]$.
-λ is defined by $\lambda([u], \alpha) = o$, where $P(u\alpha) = o$.

A Mealy machine constructed this way fulfills what the theorem states, P is a semantic functional of it, in other words, $[\![M]\!] = \mathrm{P}$. ▪

With this theorem, we can define regularity for mappings $P : \Sigma^* \to \Omega$. We call a $P : \Sigma^* \to \Omega$ mapping regular, iff there exists a corresponding Mealy machine for which $[\![M]\!] = \mathrm{P}$, or equivalently, if P has a finite number of equivalence classes, analogously to the previously seen "classical" regularity.

The introduction of regularity helps us in the construction of automata, specifically, the construction of canonical automata.

Definition 12 (Minimal automata). An automata M recognizing the language L is minimal iff every $M' \neq M$ automata that also recognize L have at least as many states as M. ▪

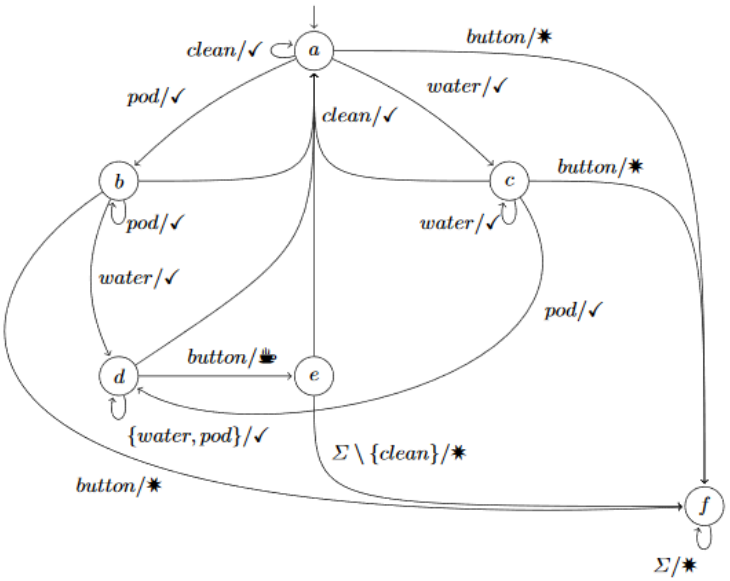

Figure 2.3: Minimal version of the Mealy machine seen in 2.2

Definition 13 (Canonical automaton). An automaton accepting the language L is canonical iff it is minimal, and contains every other automaton that accepts L. ▪

Constructing automata to be canonical, especially in the case of Mealy machines is important with regards to efficiency and is the backbone of automaton learning. We will steer to algorithm theory that does this, before actually covering the topic of automaton learning itself. The next proposition comes straightforward from the previously presented characterization theorem.

Proposition (Bounded reachability[7]): Every state of a minimal Mealy machine with n states has an access sequence, i.e., a path from the initial state to this state, of length at most n−1. Every transition of this Mealy machine can be covered by a sequence of length at most n from the initial state.

The process of constructing automata will use the concept of partition refinement. It works based on distinguishing suffixes, suffixes of words which mark, witness the difference between two access sequences. We'll introduce the following notion to formalize this.

Definition 14 (k-distinguishability[7]). Two states, $s, s' \in S$ are k-distinguishable iff there is a word $w \in \Sigma^*$ of length k or shorter, for which $\lambda^*(s, w) \neq \lambda^*(s', w)$. ▪

Definition 15 (exact k-distinguishability). Two states, $s, s' \in S$ are exact k-distinguishable, denoted by $k^=$ iff s and s' are k-distinguishable, but not (k-1)-distinguishable ▪

Essentially, if two states, s and s' are k-distinguishable, then when processing the same input sequence, from some suffix of the word w with length at most k, they will produce differing outputs. Using this, we can observe, that whenever two states, $s_1, s_2 \in S$ are (k+1)-distinguishable, then they each have an α-successor, s_1' and s_2' for some $\alpha \in \Sigma$, such that s_1' and s_2' are k-distinguishable. This suggests, that:

- no states are 0-distinguishable and

- two states s_1 and s2 are (k+1)-distinguishable iff there exists an input symbol $\alpha \in \Sigma$, such that $\lambda(s_1, \alpha) \neq \lambda(s_2, \alpha)$ or $\delta(s_1, \alpha)$ and $\delta(s_2, \alpha)$ are k-distinguishable.[7]

This way, if we have an automaton M, we can construct its minimal version, by iteratively computing k-distinguishability for increasing k, until stability, that is until the set of exactly k-distinguishable states is empty.

Example 6. *Given the Mealy machine seen in Fig.2.2, we can use k-distinguishability to refine its partitions. The initial state, the initial partition would be:*
$P_1 = \{a, b, c\}, \{d, d'\}, \{e\}, \{f\}$
since when k=1, a, b and c are not 1-distinguishable, but d and d' separate on the behavior of the button input, while e and f are separated by the suffix clean. Let's see the k=2 scenario.
$P_2 = \{a\}, \{b\}, \{c\}, \{d, d'\}, \{e\}, \{f\}$
Here, water and pod separate a, b and c, while d and d' can still no longer be separated. If observed, even if k is increased, d and d' can not be refined. This means, that they are indistinguishable, they can be merged together without altering behavior. This shows the process of acquiring the minimal machine seen in Fig. 2.3.

The process explained in the above example is partition refinement, the exact algorithm and proof of its validity can be seen in [7], it is a version of the minimization algorithm for DFAs proposed by Hopcroft[2].

We will define one last relation which will help us in the next section to compare automata minimization and automata learning.

Let $M = (S, s_0, \Sigma, \Omega, \delta, \lambda)$ and $M = (S', s'_0, \Sigma, \Omega, \delta', \lambda')$ be two Mealy machines with shared alphabets. We call a surjective function $f_k : S \to S'$ existential k-epimorphism between M and M', if for all $s' \in S', s \in S$ where $f_k(s) = s'$ and with any $\alpha \in \Sigma$, we have: $f_k(\delta(s, \alpha)) = \delta'(s', \alpha)$, and all states, that are mapped by f_k to the same state of M' are not k-distinguishable. It is straightforward to establish that all intermediate models arising during the partition refinement process are images of the considered Mealy machine under a k-epimorphism, where k is the number of times all transitions have been investigated.[7] Essentially this establishes P_1 and P_2 from Example 6 as images of the Mealy machine seen in Fig. 3 under k-epimorphisms where k=1 and k=2 respectively.

Active automaton learning algorithms operate in a very similar way, but they are not the same as minimization algorithms, since they do not have access to the automatons they are learning. We will extend upon this in the next section.

2.2 Automaton learning

Automaton learning is modeling a system without having specific knowledge of its the internal behavior. To accomplish this, a model needs to be inferred by observing the external behavior of the system. This learned model is, as the name suggests, an automaton.
Formally: Active automata learning is concerned with the problem of inferring an automaton model for an unknown formal language L over some alphabet Σ[3].

In order to monitor a system, a way of access to its behavioral information is required. There are two approaches, which separate the two types of automaton learning as well.

Passive automaton learning In case of passive automaton learning, the gathering of information is not part of the learning process, but rather a prerequisite to it. The learning is performed on a pre-gathered positive an/or negative example set of the systems behavior. In passive automaton learning, the success of the process is determined not only by the efficiency of the algorithm, but the methodology and time used to gather the data.

Active automaton learning In case of active automaton learning, the behavioral infromation is gathered by the learning algorithm in an "active" way via queries. In order to accomplish this, learning is separated to two components: the learner, which learns, and the teacher, which can answer questions about the system under learning.

Active automaton learning follows the MAT, or the Minimally Adequate Teacher model proposed by Dana Angluin[1]. It separates the algorithm to a learner and a teacher, where the teacher can only answer the minimally adequate questions needed to learn the system. These two questions, or queries are are follows:

Membership query Given a $w \in \Sigma^*$ word, the query return the $o \in \Omega$ output corresponding to it, treating the word as a string of inputs. We write $mq(w) = o$ to denote that executing the query w on the system under learning (SUL) leads to the output o: $[\![SUL]\!](w) = o$ or $\lambda^*(s_0, w) = o$.

Equivalence query Given a hypothesis automaton M, the query tries to determine if the hypothesis is behaviorally equivalent to the SUL, and if not, finding the diverging behavior, and return with the example of it. We write $eq(H) = c$, where $c \in \Sigma^*$, to denote an equivalence query on hypothesis H, returning a counterexample c. The counter example provided is the sequence of inputs for which the output of system under learning and the output of the hypothesis differ: $[\![H]\!](c) \neq mq(c)$.

The learner component uses membership queries to construct a hypothesis automaton, then refines this hypothesis by the counterexamples provided by equivalence queries. Once counterexamples can not be found this way, the learners hypothesis is behaviorally equivalent to the SUL, the learning can terminate and the output of the learning is the current hypothesis.

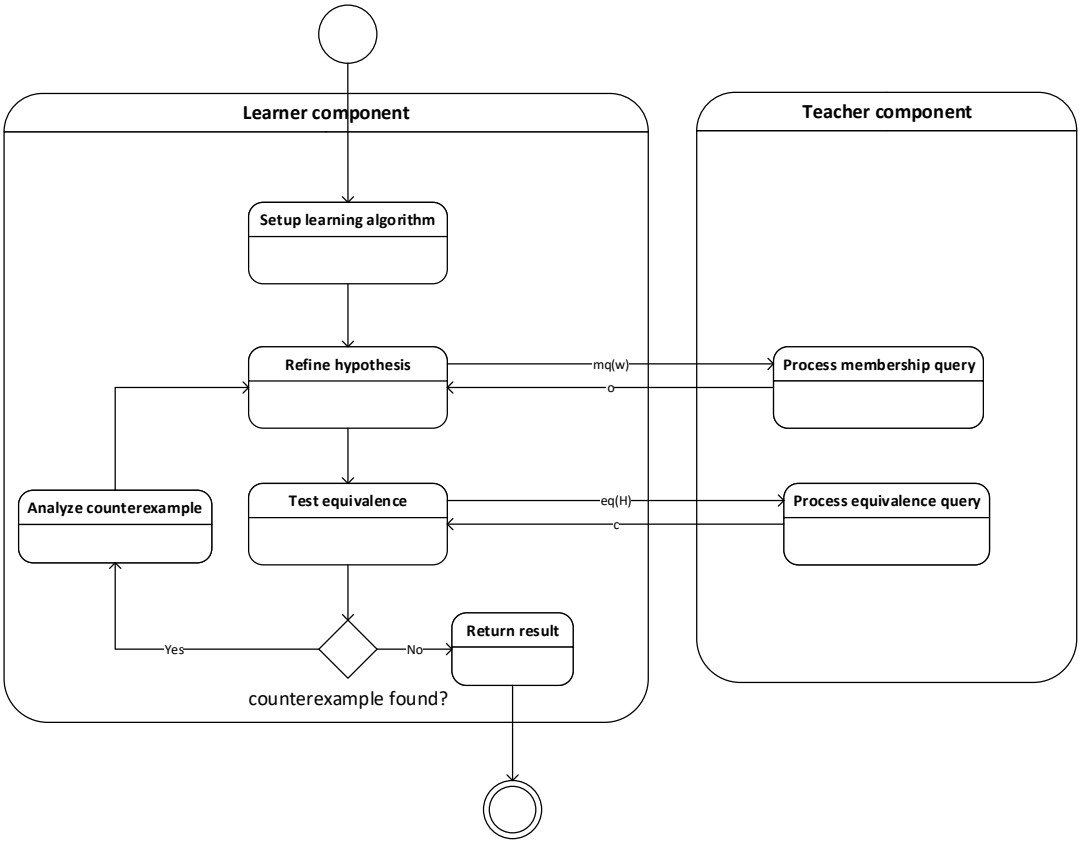

Figure 2.4: Abstract model of active automata learning algorithms

As seen on Fig. 2.4, the learning proceeds in rounds, generating and refining hypothesis models by exploring the SUL via membership queries. As the equivalence checks produce counterexamples, the next round of this hypothesis refinement is steered by the counterexamples produced.

Using an analogous strategy to the minimization of automatons seen in the previous section, starting only with a one state hypothesis automaton, we explore over all words in the alphabet in order to refine and extend this hypothesis. Here, there is a dual way of characterizing (and distinguishing) between states[7]:

- By words reaching them. A prefix-closed set of S_p words, reaching each state exactly once, defines a spanning tree of the automaton. This characterization aims at providing exactly one representative element from each class of $\equiv_P$ on the SUL. Active learning algorithms incrementally construct such a set S_p.
  This prefix-closedness is necessary for S_p to be a "spanning tree" of the Mealy machine. Extending S_p with all the one-letter continuations of words in S_p will result in the tree covering all the transitions of the Mealy machine. L_p will denote all the one-letter continuations that are not already contained in S_p.

- By their future behavior with respect to an increasing vector of words of Σ^*. This vector $< d_1, d_2, ..., d_k >$ will be denoted by D, and contains the "distinguishing suffixes". The corresponding future behavior of a state, here given in terms of its access

sequence $u \in S_p$, is the output vector$< mq(u * d_1), ..., mq(u * d_k) > \in \Omega^k$, which leads to an upper approximation of the classes of $\equiv_{[\![SUL]\!]}$. Active learning incrementally refines this approximation by extending the vector until the approximation is precise.

While the second characterization defines the states of the automaton, where each output vector corresponds to one state, the spanning tree on L_p is used to determine the transitions of these states. In order to characterize the relation between the SUL $M = (S, s_0, \Sigma, \Omega, \delta, \lambda)$ and the hypothesis model $M' = (S', s'_0, \Sigma, \Omega, \delta', \lambda')$ (note, that M and M' only share alphabets) let $D \subseteq \Sigma^*$. We call a surjective function $f_D : S \to S'$ existential D-epimorphism (surjective homomorphism) between M and M' if, for all $s' \in S'$ there exists an $s \in S$ with $f_D(s) = s'$ such that for all $\alpha \in \Sigma$ and all $d \in D$: $f_D(\delta(s, \alpha)) = \delta'(s', \alpha)$, and $\lambda^*(s, d) = \lambda^*(s', d)$.
Note, that active learning deals with canonical Mealy machines, in other words, the canonical form of SUL, and not, the perhaps much larger Mealy machine of SUL itself.

Since active learning algorithms maintain an incrementally growing extended spanning tree for $H = (S_H, h_0, \Sigma, \Omega, \delta_H, \lambda_H)$, i.e., a prefix-closed set of words reaching all its states and covering all transitions, it is straightforward to establish that these hypothesis models are images of the canonical version of SUL under a canonical existential D-epimorphism, where D is the set of distinctive futures underlying the hypothesis construction[7]

- define $f_D : S_{SUL} \to S_H$ by $f_D(s) = h$ as following: if $\exists w \in S_p \cup L_p$, where $\delta(s_0, w) = s$, then $h = \delta_H(h_0, w)$. Otherwise h may be chosen arbitrarily.
- It suffices to consider the states reached by words in the spanning tree to establish the defining properties of f_D. This straightforwardly yields:
  - $f_D(\delta(s, \alpha)) = \delta_H(h, \alpha)$ for all $\alpha \in \Sigma$, which reflects the characterization from below.
  - $\lambda^*(s, d) = \lambda^*_H(h, d)$ for all $d \in D$, which follows from the maintained characterization from above.[7]

In basic logic, D-epimorphisms and k-epimorphisms do not differ, they both deal with establishing constructed models being images of the model they are based on. D-epimorphisms could replace k-epimorphisms where $D = \Sigma^k$, it can be suggested, that there is no need to differentiate. However, there is in important difference of complexity between the two. While k-distinguishability supports polynomial time, black-box systems do not. Also, the "existential" in existential D-epimorphism is important: f_D must deal with unknown states, ones that haven't been encountered yet. This implies that characterization can only be valid for already encountered states.

Active learning algorithms can be proven correct using the following three-step pattern:

- Invariance: The number of states of each hypothesis has an upper bound of $\equiv_{[\![SUL]\!]}$.
- Progress: Before the final partition is reached, an equivalence query will provide a counterexample, where an input word leads to a different output on the SUL and on the hypothesis. This difference can only be resolved by splitting at least one state, which increases the state count.
- Termination: The refinement terminates after at most the index of $\equiv_{[\![SUL]\!]}$ many steps, caused directly by the described invariance and progress properties.

The following subsection introduces the first active automaton learning algorithm this thesis covers.

2.2.1 Direct Hypothesis Construction

The Direct Hypothesis Construction algorithm, which hypothesis construction can be seen in Algorithm 1 follows the idea of the breath-first search of graph theory, it constructs the hypothesis using a queue of states, which is initialized with the states of the spanning tree to be maintained. Explored states are removed from this queue, while the discovered successors are enqueued, if they are provably new states. The algorithm starts with a one-state hypothesis, including only the initial state, reached by ϵ and D = Σ. It then tries to complete the hypothesis, for every state we determine the behavior of the state under D, we will call this behavior the extended signature of said state. States with a new extended signatures are provably new states, so to guarantee further investigation, we enqueue all their successors. Initially, $D = \Sigma$, so only the $1^{=}$-distinguishable states are revealed during the first iteration. This is extended straightforwardly to comprise a prefix closed set of access sequences. [7][5]

Algorithm 1: Hypothesis construction of the Direct Hypothesis Construction algorithm as seen in [7].

Input: S_p: a set of access sequences, D: a set of suffixes, an input alphabet Σ
Output: A Mealy machine $H = (S, s_0, \Sigma, \Omega, \delta, \lambda)$

```
initialize hypothesis H, create a state for all elements of S_p
initialize a queue Q with the states of H
while Q is not empty do
    s = dequeue state from Q
    u = access sequence from s_0 to s
    for d ∈ D do
        o = mq(ud)
        set λ(s, d) = o
    end
    if exists an s' ∈ S, where the output signature of s' is the same as s then
        reroute transitions of s to s' in H
        remove s from H
    else
        create and enqueue successors of s for every input in Σ into Q, if not
          already in S_p
    end
end
Remove entries of D \ Σ from λ
return H
```

After the run of the Hypothesis construction seen in Algorithm 1, the output automaton H is used in an equivalence query eq(H) = c, to find if a counterexample c exists. If no counteraxamle can be found, the learning terminates, H is the learned automaton. Else, if a counterexample c is found, for which $\lambda_H(s_0, c) \neq mq(c)$, c is used to enlarge the suffixes in D and a new iteration of Algorithm 1 begins, using the now extended set D and all the access sequences found in the previous iteration, the current spanning tree S_p.

To formalize the treatment of counterexamples, we introduce the following notation: let $[u]_H$, where $u \in \Sigma^*$, be the word in S_p which reaches $\delta^*(s_0, u)$ in hypothesis H. Using notation, the following theorem can be established.

Theorem 3 (Counterexample decomposition[7]). For every counterexample c, there is a decomposition $c = u\alpha w$, where u is its prefix, α is its string, or action, and w is its suffix, such that $mq([u]_H av) \neq mq([u\alpha]_H w)$. ▪

Chapter 3

LaTeX-eszközök

3.1 A szerkesztéshez használatos eszközök

Ez a sablon TeXstudio 2.8.8 szerkesztővel készült. A TeXstudio egy platformfüggetlen, Windows, Linux és Mac OS alatt is elérhető LaTeX-szerkesztőprogram számtalan hasznos szolgáltatással (figure 3.1). A szoftver ingyenesen letölthető[1].

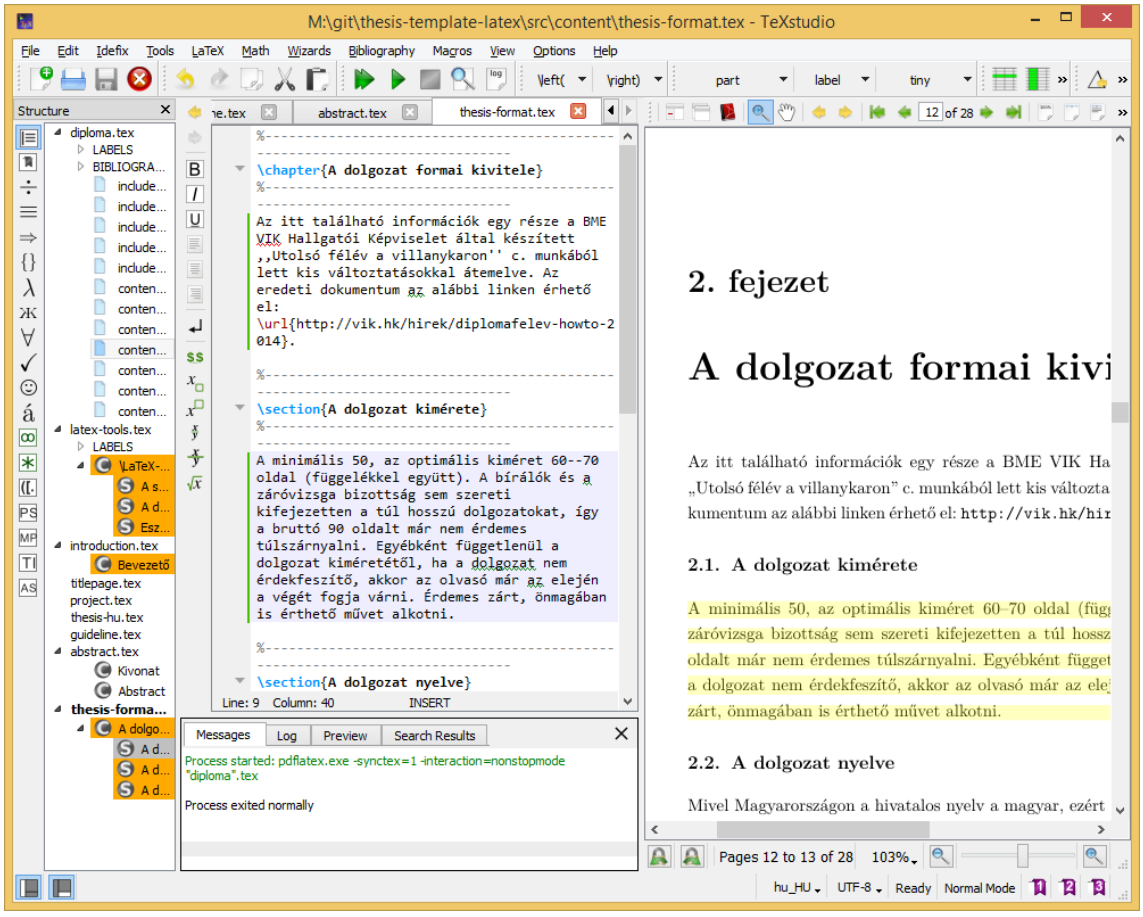

Figure 3.1: A TeXstudio LaTeX-szerkesztő.

[1] A TeXstudio hivatalos oldala: `http://texstudio.sourceforge.net/`

A TeXstudio telepítése után érdemes még letölteni a magyar nyelvű helyesírásellenőrző-szótárakat hozzá. A TeXstudio az OpenOffice-hoz használatos formátumot tudja kezelni. A TeXstudio beállításainál a `General` fülön a `Dictionaries` résznél tudjuk megadni, hogy melyik szótárat használja.

Egy másik használható Windows alapú szerkesztőprogram a LEd[2] (LaTeX Editor), a TeXstudio azonban stabilabb, gyorsabb, és jobban használható.

3.2 A dokumentum lefordítása Windows alatt

A TeXstudio és a LEd kizárólag szerkesztőprogram (bár az utóbbiban DVI-nézegető is van), így a dokumentum fordításához szükséges eszközöket nem tartalmazza. Windows alatt alapvetően két lehetőség közül érdemes választani: MiKTeX (`http://miktex.org/`) és TeX Live (`http://www.tug.org/texlive/`) programcsomag. Az utóbbi működik Mac OS X, GNU/Linux alatt és Unix-származékokon is. A MiKTeX egy alapcsomag telepítése után mindig letölti a használt funkciókhoz szükséges, de lokálisan hiányzó TEX-csomagokat, míg a TeX Live DVD ISO verzóban férhető hozzá. Ez a dokumentum TeX Live 2008 programcsomag segítségével fordult, amelynek DVD ISO verziója a megadott oldalról letölthető. A sablon lefordításához a disztribúcióban szereplő `magyar.ldf` fájlt a `http://www.math.bme.hu/latex/` változatra kell cserélni, vagy az utóbbi változatot be kell másolni a projekt-könyvtárba (ahogy ezt meg is tettük a sablonban) különben anomáliák tapasztalhatók a dokumentumban (pl. az ábra- és táblázat-aláírások formátuma nem a beállított lesz, vagy bizonyos oldalakon megjelenik alapértelmezésben egy fejléc). A TeX Live 2008-at még nem kell külön telepíteni a gépre, elegendő DVD-ről (vagy az ISO fájlból közvetlenül, pl. DaemonTools-szal) használni.

Ha a MiKTeX csomagot használjuk, akkor parancssorból a következő módon tudjuk újrafordítani a teljes dokumentumot:

```
$ texify -p thesis.tex
```

A `texify` parancs a MiKTex programcsomag `miktex/bin` alkönyvtárában található. A parancs gondoskodik arról, hogy a szükséges lépéseket (fordítás, hivatkozások generálása stb.) a megfelelő sorrendben elvégezze. A `-p` kapcsoló hatására PDF-et generál. A fordítást és az ideiglenes fájlok törlését elvégezhetjük a sablonhoz mellékelt `manual_build.bat` szkript segítségével is.

A TEX-eszközöket tartalmazó programcsomag binárisainak elérési útját gyakran be kell állítani a szerkesztőprogramban, például TeXstudio esetén legegyszerűbben az `Options / Configure TeXstudio... / Commands` menüponttal előhívott dialógusablakban tehetjük ezt meg.

A PDF-LATEX használata esetén a generált dokumentum közvetlenül PDF-formátumban áll rendelkezésre. Amennyiben a PDF-fájl egy PDF-nézőben (pl. Adobe Acrobat Reader vagy Foxit PDF Reader) meg van nyitva, akkor a fájlleírót a PDF-néző program tipikusan lefoglalja. Ilyen esetben a dokumentum újrafordítása hibaüzenettel kilép. Ha bezárjuk és újra megnyitjuk a PDF dokumentumot, akkor pedig a PDF-nézők többsége az első oldalon nyitja meg a dokumentumot, nem a legutóbb olvasott oldalon. Ezzel szemben például az egyszerű és ingyenes Sumatra PDF nevű program képes arra, hogy a megnyitott dokumentum megváltozását detektálja, és frissítse a nézetet az aktuális oldal megtartásával.

[2] A LEd hivatalos oldala: `http://www.latexeditor.org/`

3.3 Eszközök Linuxhoz

Linux operációs rendszer alatt is rengeteg szerkesztőprogram van, pl. a KDE alapú Kile jól használható. Ez ingyenesen letölthető, vagy éppenséggel az adott Linux-disztribúció eleve tartalmazza, ahogyan a dokumentum fordításához szükséges csomagokat is. Az Ubuntu Linux disztribúciók alatt például legtöbbször a `texlive-*` csomagok telepítésével használhatók a LaTeX-eszközök. A jelen sablon fordításához szükséges csomagok (kb. 0,5 GB) az alábbi paranccsal telepíthetők:

```
$ sudo apt-get install texlive-latex-extra texlive-fonts-extra texlive-fonts-recommended
    texlive-lang-european texlive-xetex texlive-science
```

Amennyiben egy újabb csomag hozzáadása után hiányzó fájlra utaló hibát kapunk a fordítótól, telepítenünk kell az azt tartalmazó TeX Live csomagot. Ha pl. a `bibentry` csomagot szeretnénk használni, futtassuk az alábbi parancsot:

```
$ apt-cache search bibentry
texlive-luatex - TeX Live: LuaTeX packages
```

Majd telepítsük fel a megfelelő TeX Live csomagot, jelen esetben a ‘texlive-lualatex‘-et. (Egy LaTeX csomag több TeX Live csomagban is szerepelhet.)

Ha gyakran szerkesztünk más LaTeXdokumentumokat is, kényelmes és biztos megoldás a teljes TeX Live disztribúció telepítése, ez azonban kb. 4 GB helyet igényel.

```
sudo apt-get install texlive-full
```

Chapter 4

A dolgozat formai kivitele

Az itt található információk egy része a BME VIK Hallgatói Képviselet által készített „Utolsó félév a villanykaron” c. munkából lett kis változtatásokkal átemelve. Az eredeti dokumentum az alábbi linken érhető el: `http://vik.hk/hirek/diplomafelev-howto-2015`.

4.1 A dolgozat kimérete

Szakdolgozat esetében minimum 30, 45 körüli ajánlott oldalszám lehet az iránymutató. De mindenképp érdemes rákérdezni a konzulensnél is az elvárásokra, mert tanszékenként változóak lehetnek az elvárások.

Mesterképzésen a Diplomatervezés 1 esetében a beszámoló még inkább az Önálló laboratóriumi beszámolókhoz hasonlít, tanszékenként eltérő formai követelményekkel, – egy legalább 30 oldal körüli dolgozat az elvárt – és az elmúlt fél éves munkáról szól. De egyben célszerű, ha ez a végleges diplomaterv alapja is. (A végleges 60-90 oldal körülbelül a hasznos részre nézve)

4.2 A dolgozat nyelve

Mivel Magyarországon a hivatalos nyelv a magyar, ezért alapértelmezésben magyarul kell megírni a dolgozatot. Aki külföldi posztgraduális képzésben akar részt venni, nemzetközi szintű tudományos kutatást szeretne végezni, vagy multinacionális cégnél akar elhelyezkedni, annak célszerű angolul megírnia diplomadolgozatát. Mielőtt a hallgató az angol nyelvű verzió mellett dönt, erősen ajánlott mérlegelni, hogy ez mennyi többletmunkát fog a hallgatónak jelenteni fogalmazás és nyelvhelyesség terén, valamint – nem utolsó sorban – hogy ez mennyi többletmunkát fog jelenteni a konzulens illetve bíráló számára. Egy nehezen olvasható, netalán érthetetlen szöveg teher minden játékos számára.

4.3 A dokumentum nyomdatechnikai kivitele

A dolgozatot A4-es fehér lapra nyomtatva, 2,5 centiméteres margóval (+1 cm kötésbeni), 11–12 pontos betűmérettel, talpas betűtípussal és másfeles sorközzel célszerű elkészíteni.

Annak érdekében, hogy a dolgozat külsőleg is igényes munka benyomását keltse, érdemes figyelni az alapvető tipográfiai szabályok betartására [**?**].

Chapter 5

A LaTeX-sablon használata

Ebben a fejezetben röviden, implicit módon bemutatjuk a sablon használatának módját, ami azt jelenti, hogy sablon használata ennek a dokumentumnak a forráskódját tanulmányozva válik teljesen világossá. Amennyiben a szoftver-keretrendszer telepítve van, a sablon alkalmazása és a dolgozat szerkesztése LaTeX-ben a sablon segítségével tapasztalataink szerint jóval hatékonyabb, mint egy WYSWYG (*What You See is What You Get*) típusú szövegszerkesztő esetén (pl. Microsoft Word, OpenOffice).

5.1 Címkék és hivatkozások

A LaTeX dokumentumban címkéket (`\label`) rendelhetünk ábrákhoz, táblázatokhoz, fejezetekhez, listákhoz, képletekhez stb. Ezekre a dokumentum bármely részében hivatkozhatunk, a hivatkozások automatikusan feloldásra kerülnek.

A sablonban makrókat definiáltunk a hivatkozások megkönnyítéséhez. Ennek megfelelően minden ábra (*figure*) címkéje `fig:` kulcsszóval kezdődik, míg minden táblázat (*table*), képlet (*equation*), fejezet (*section*) és lista (*listing*) rendre a `tab:`, `eq:`, `sec:` és `lst:` kulcsszóval kezdődik, és a kulcsszavak után tetszőlegesen választott címke használható. Ha ezt a konvenciót betartjuk, akkor az előbbi objektumok számára rendre a `\figref`, `\tabref`, `\eqref`, `\sectref` és `\listref` makrókkal hivatkozhatunk. A makrók paramétere a címke, amelyre hivatkozunk (a kulcsszó nélkül). Az összes említett hivatkozástípus, beleértve az `\url` kulcsszóval bevezetett web-hivatkozásokat is a `hyperref`[1] csomagnak köszönhetően aktívak a legtöbb PDF-nézegetőben, rájuk kattintva a dokumentum megfelelő oldalára ugrik a PDF-néző vagy a megfelelő linket megnyitja az alapértelmezett böngészővel. A `hyperref` csomag a kimeneti PDF-dokumentumba könyvjelzőket is készít a tartalomjegyzékből. Ez egy szintén aktív tartalomjegyzék, amelynek elemeire kattintva a nézegető behozza a kiválasztott fejezetet.

5.2 Ábrák és táblázatok

Használjunk vektorgrafikus ábrákat, ha van rá módunk. PDFLaTeX használata esetén PDF formátumú ábrákat lehet beilleszteni könnyen, az EPS (PostScript) vektorgrafikus képformátum beillesztését a PDFLaTeX közvetlenül nem támogatja (de lehet konvertálni,

[1] Segítségével a dokumentumban megjelenő hivatkozások nem csak dinamikussá válnak, de színezhetők is, bővebbet erről a csomag dokumentációjában találunk. Ez egyúttal egy példa lábjegyzet írására.

lásd később). Ha vektorgrafikus formában nem áll rendelkezésünkre az ábra, akkor a veszteségmentes PNG, valamint a veszteséges JPEG formátumban érdemes elmenteni. Figyeljünk arra, hogy ilyenkor a képek felbontása elég nagy legyen ahhoz, hogy nyomtatásban is megfelelő minőséget nyújtson (legalább 300 dpi javasolt). A dokumentumban felhasznált képfájlokat a dokumentum forrása mellett érdemes tartani, archiválni, mivel ezek hiányában a dokumentum nem fordul újra. Ha lehet, a vektorgrafikus képeket vektorgrafikus formátumban is érdemes elmenteni az újrafelhasználhatóság (az átszerkeszthetőség) érdekében.

Kapcsolási rajzok legtöbbször kimásolhatók egy vektorgrafikus programba (pl. CorelDraw) és onnan nagyobb felbontással raszterizálva kimenthatők PNG formátumban. Ugyanakkor kiváló ábrák készíthetők Microsoft Visio vagy hasonló program használatával is: Visio-ból az ábrák közvetlenül PDF-be is menthetők.

Lehetőségeink Matlab ábrák esetén:

- Képernyőlopás (*screenshot*) is elfogadható minőségű lehet a dokumentumban, de általában jobb felbontást is el lehet érni más módszerrel.
- A Matlab ábrát a `File/Save As` opcióval lementhetjük PNG formátumban (ugyanaz itt is érvényes, mint korábban, ezért nem javasoljuk).
- A Matlab ábrát az `Edit/Copy figure` opcióval kimásolhatjuk egy vektorgrafikus programba is és onnan nagyobb felbontással raszterizálva kimenthatjük PNG formátumban (nem javasolt).
- Javasolt megoldás: az ábrát a `File/Save As` opcióval EPS *vektorgrafikus* formátumban elmentjük, PDF-be konvertálva beillesztjük a dolgozatba.

Az EPS kép az `epstopdf` programmal[2] konvertálható PDF formátumba. Célszerű egy batch-fájlt készíteni az összes EPS ábra lefordítására az alábbi módon (ez Windows alatt működik).

```
@echo off
for %%j in (*.eps) do (
  echo converting file "%%j"
  epstopdf "%%j"
)
echo done .
```

Egy ilyen parancsfájlt (`convert.cmd`) elhelyeztük a sablon `figures\eps` könyvtárába, így a felhasználónak csak annyi a dolga, hogy a `figures\eps` könyvtárba kimenti az EPS formátumú vektorgrafikus képet, majd lefuttatja a `convert.cmd` parancsfájlt, ami PDF-be konvertálja az EPS fájlt.

Ezek után a PDF-ábrát ugyanúgy lehet a dokumentumba beilleszteni, mint a PNG-t vagy a JPEG-et. A megoldás előnye, hogy a lefordított dokumentumban is vektorgrafikusan tárolódik az ábra, így a mérete jóval kisebb, mintha raszterizáltuk volna beillesztés előtt. Ez a módszer minden – az EPS formátumot ismerő – vektorgrafikus program (pl. CorelDraw) esetén is használható.

A képek beillesztésére a chapter 3ben mutattunk be példát (figure 3.1). Az előző mondatban egyúttal az automatikusan feloldódó ábrahivatkozásra is láthatunk példát. Több képfájlt is beilleszthetünk egyetlen ábrába. Az egyes képek közötti horizontális és vertikális margót metrikusan szabályozhatjuk (figure 5.1). Az ábrák elhelyezését számtalan

[2] a korábban említett LaTeX-disztribúciókban megtalálható

tipográfiai szabály egyidejű teljesítésével a fordító maga végzi, a dokumentum írója csak preferenciáit jelezheti a fordító felé (olykor ez bosszúságot is okozhat, ilyenkor pl. a kép méretével lehet játszani).

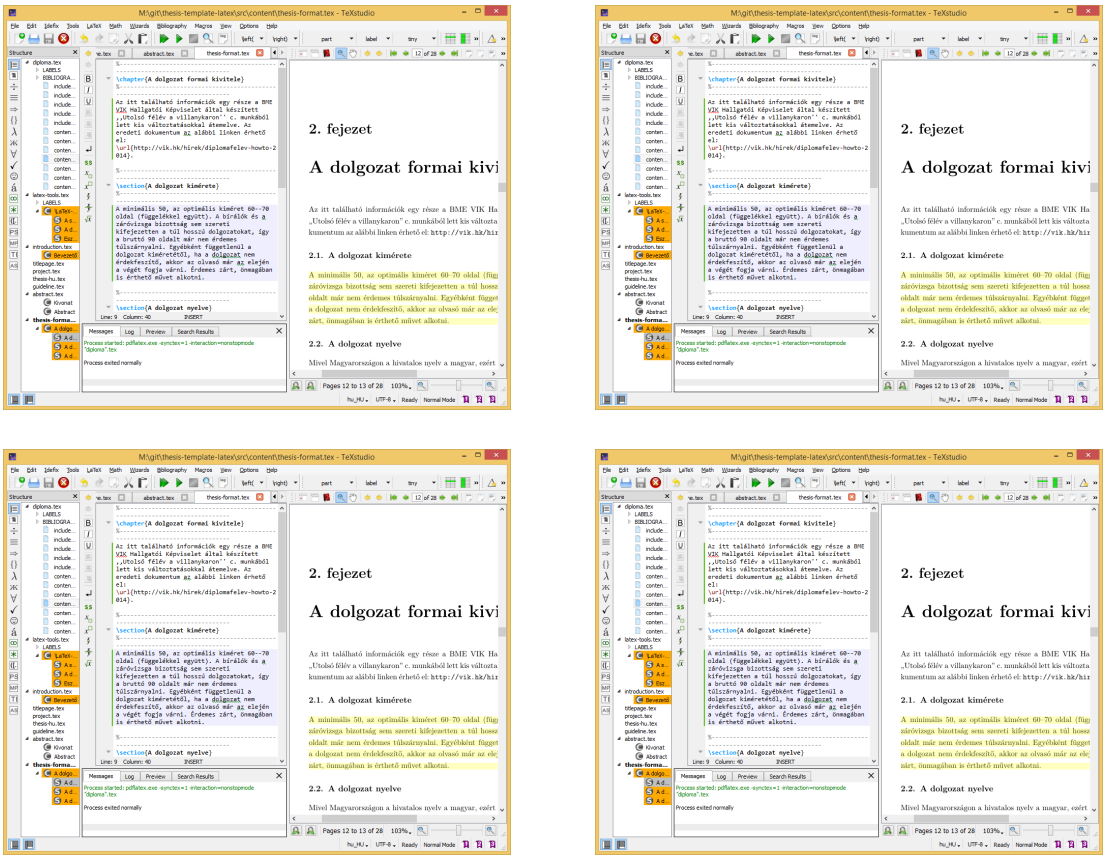

Figure 5.1: Több képfájl beillesztése esetén térközöket is érdemes használni.

A táblázatok használatára az 5.1 táblázat mutat példát. A táblázatok formázásához hasznos tanácsokat találunk a `booktabs` csomag dokumentációjában.

Órajel	Frekvencia	Cél pin
CLKA	100 MHz	FPGA CLK0
CLKB	48 MHz	FPGA CLK1
CLKC	20 MHz	Processzor
CLKD	25 MHz	Ethernet chip
CLKE	72 MHz	FPGA CLK2
XBUF	20 MHz	FPGA CLK3

Table 5.1: Az órajel-generátor chip órajel-kimenetei.

5.3 Felsorolások és listák

Számozatlan felsorolásra mutat példát a jelenlegi bekezdés:

- *első bajusz:* ide lehetne írni az első elem kifejését,
- *második bajusz:* ide lehetne írni a második elem kifejését,

- *ez meg egy szakáll:* ide lehetne írni a harmadik elem kifejését.

Számozott felsorolást is készíthetünk az alábbi módon:

1. *első bajusz:* ide lehetne írni az első elem kifejését, és ez a kifejtés így néz ki, ha több sorosra sikeredik,
2. *második bajusz:* ide lehetne írni a második elem kifejését,
3. *ez meg egy szakáll:* ide lehetne írni a harmadik elem kifejését.

A felsorolásokban sorok végén vessző, az utolsó sor végén pedig pont a szokásos írásjel. Ez alól kivételt képezhet, ha az egyes elemek több teljes mondatot tartalmaznak.

Listákban a dolgozat szövegétől elkülönítendő kódrészleteket, programsorokat, pszeudokódokat jeleníthetünk meg (5.1. kódrészlet).

```
\begin{enumerate}
  \item \emph{első bajusz:} ide lehetne írni az első elem kifejését,
  és ez a kifejtés így néz ki, ha több sorosra sikeredik,
  \item \emph{második bajusz:} ide lehetne írni a második elem kifejését,
  \item \emph{ez meg egy szakáll:} ide lehetne írni a harmadik elem kifejését.
\end{enumerate}
```

Listing 5.1: A fenti számozott felsorolás LaTeX-forráskódja

A lista keretét, háttérszínét, egész stílusát megválaszthatjuk. Ráadásul különféle programnyelveket és a nyelveken belül kulcsszavakat is definiálhatunk, ha szükséges. Erről bővebbet a `listings` csomag hivatalos leírásában találhatunk.

5.4 Képletek

Ha egy formula nem túlságosan hosszú, és nem akarjuk hivatkozni a szövegből, mint például a $e^{i\pi} + 1 = 0$ képlet, *szövegközi képletként* szokás leírni. Csak, hogy másik példát is lássunk, az $U_i = -d\Phi/dt$ Faraday-törvény a $\operatorname{rot} E = -\frac{dB}{dt}$ differenciális alakban adott Maxwell-egyenlet felületre vett integráljából vezethető le. Látható, hogy a LaTeX-fordító a sorközöket betartja, így a szöveg szedése esztétikus marad szövegközi képletek használata esetén is.

Képletek esetén az általános konvenció, hogy a kisbetűk skalárt, a kis félkövér betűk ($\mathbf{v}$) oszlopvektort – és ennek megfelelően $\mathbf{v}^T$ sorvektort – a kapitális félkövér betűk ($\mathbf{V}$) mátrixot jelölnek. Ha ettől el szeretnénk térni, akkor az alkalmazni kívánt jelölésmódot célszerű külön alfejezetben definiálni. Ennek megfelelően, amennyiben $\mathbf{y}$ jelöli a mérések vektorát, ϑ a paraméterek vektorát és $\hat{\mathbf{y}} = \mathbf{X}\vartheta$ a paraméterekben lineáris modellt, akkor a *Least-Squares* értelemben optimális paraméterbecslő $\hat{\vartheta}_{LS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ lesz.

Emellett kiemelt, sorszámozott képleteket is megadhatunk, ennél az `equation` és a `eqnarray` környezetek helyett a korszerűbb `align` környezet alkalmazását javasoljuk (több okból, különféle problémák elkerülése végett, amelyekre most nem térünk ki). Tehát

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \tag{5.1}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x}, \tag{5.2}$$

ahol $\mathbf{x}$ az állapotvektor, $\mathbf{y}$ a mérések vektora és $\mathbf{A}$, $\mathbf{B}$ és $\mathbf{C}$ a rendszert leíró paramétermátrixok. Figyeljük meg, hogy a két egyenletben az egyenlőségjelek egymáshoz igazítva

jelennek meg, mivel a mindkettőt az & karakter előzi meg a kódban. Lehetőség van számozatlan kiemelt képlet használatára is, például

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u},$$
$$\mathbf{y} = \mathbf{C}\mathbf{x}.$$

Mátrixok felírására az $\mathbf{A}\mathbf{x} = \mathbf{b}$ inhomogén lineáris egyenlet részletes kifejtésével mutatunk példát:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}. \tag{5.3}$$

A `\frac` utasítás hatékonyságát egy általános másodfokú tag átviteli függvényén keresztül mutatjuk be, azaz

$$W(s) = \frac{A}{1 + 2T\xi s + s^2T^2}. \tag{5.4}$$

A matematikai mód minden szimbólumának és képességének a bemutatására természetesen itt nincs lehetőség, de gyors referenciaként hatékonyan használhatók a következő linkek:
`http://www.artofproblemsolving.com/LaTeX/AoPS_L_GuideSym.php`,
`http://www.ctan.org/tex-archive/info/symbols/comprehensive/symbols-a4.pdf`,
`ftp://ftp.ams.org/pub/tex/doc/amsmath/short-math-guide.pdf`.
Ez pedig itt egy magyarázat, hogy miért érdemes `align` környezetet használni:
`http://texblog.net/latex-archive/maths/eqnarray-align-environment/`.

5.5 Irodalmi hivatkozások

Egy LaTeX dokumentumban az irodalmi hivatkozások definíciójának két módja van. Az egyik a `\thebibliograhy` környezet használata a dokumentum végén, az `\end{document}` lezárás előtt.

```
\begin{thebibliography}{9}

\bibitem{Lamport94} Leslie Lamport, \emph{\LaTeX: A Document Preparation System}.
Addison Wesley, Massachusetts, 2nd Edition, 1994.

\end{thebibliography}
```

Ezek után a dokumentumban a `\cite{Lamport94}` utasítással hivatkozhatunk a forrásra. A fenti megadás viszonylag kötetlen, a szerző maga formázza az irodalomjegyzéket (ami gyakran inkonzisztens eredményhez vezet).

Egy sokkal professzionálisabb módszer a BiBTeX használata, ezért ez a sablon is ezt támogatja. Ebben az esetben egy külön szöveges adatbázisban definiáljuk a forrásmunkákat, és egy külön stílusfájl határozza meg az irodalomjegyzék kinézetét. Ez, összhangban azzal, hogy külön formátumkonvenció határozza meg a folyóirat-, a könyv-, a konferenciacikk- stb. hivatkozások kinézetét az irodalomjegyzékben (a sablon használata esetén ezzel nem is kell foglalkoznia a hallgatónak, de az eredményt célszerű ellenőrizni). felhasznált hivatkozások adatbázisa egy `.bib` kiterjesztésű szöveges fájl, amelynek szerkezetét a Az 5.2 kódrészlet demonstrálja. A forrásmunkák bevitelekor a sor végi vesszők külön figyelmet igényelnek, mert hiányuk a BiBTeX-fordító hibaüzenetét eredményezi. A forrásmunkákat

típus szerinti kulcsszó vezeti be (`@book` könyv, `@inproceedings` konferenciakiadványban megjelent cikk, `@article` folyóiratban megjelent cikk, `@techreport` valamelyik egyetem gondozásában megjelent műszaki tanulmány, `@manual` műszaki dokumentáció esetén stb.). Nemcsak a megjelenés stílusa, de a kötelezően megadandó mezők is típusról-típusra változnak. Egy jól használható referencia a `http://en.wikipedia.org/wiki/BibTeX` oldalon található.

```
@book{Wettl04,
  author    = {Ferenc Wettl and Gyula Mayer and Péter Szabó},
  publisher = {Panem Könyvkiadó},
  title     = {\LaTeX~kézikönyv},
  year      = {2004},
}

@article{Candy86,
  author       = {James C. Candy},
  journaltitle = {{IEEE} Trans.\ on Communications},
  month        = {01},
  note         = {\doi{10.1109/TCOM.1986.1096432}},
  number       = {1},
  pages        = {72--76},
  title        = {Decimation for Sigma Delta Modulation},
  volume       = {34},
  year         = {1986},
}

@inproceedings{Lee87,
  author    = {Wai L. Lee and Charles G. Sodini},
  booktitle = {Proc.\ of the IEEE International Symposium on Circuits and Systems},
  location  = {Philadelphia, PA, USA},
  month     = {05~4--7},
  pages     = {459--462},
  title     = {A Topology for Higher Order Interpolative Coders},
  vol       = {2},
  year      = {1987},
}

@thesis{KissPhD,
  author      = {Peter Kiss},
  institution = {Technical University of Timi\c{s}oara, Romania},
  month       = {04},
  title       = {Adaptive Digital Compensation of Analog Circuit Imperfections for Cascaded Delta-
    Sigma Analog-to-Digital Converters},
  type        = {phdthesis},
  year        = {2000},
}

@manual{Schreier00,
  author       = {Richard Schreier},
  month        = {01},
  note         = {\url{http://www.mathworks.com/matlabcentral/fileexchange/}},
  organization = {Oregon State University},
  title        = {The Delta-Sigma Toolbox v5.2},
  year         = {2000},
}

@misc{DipPortal,
  author       = {{Budapesti űMszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai
    Kar}},
  howpublished = {\url{http://diplomaterv.vik.bme.hu/}},
  title        = {Diplomaterv portál (2011. február 26.)},
}

@incollection{Mkrtychev:1997,
  author    = {Mkrtychev, Alexey},
  booktitle = {Logical Foundations of Computer Science},
  doi       = {10.1007/3-540-63045-7_27},
  editor    = {Adian, Sergei and Nerode, Anil},
  isbn      = {978-3-540-63045-6},
  pages     = {266-275},
  publisher = {Springer Berlin Heidelberg},
```

```
  series    = {Lecture Notes in Computer Science},
  title     = {Models for the logic of proofs},
  url       = {http://dx.doi.org/10.1007/3-540-63045-7_27},
  volume    = {1234},
  year      = {1997},
}
```

Listing 5.2: Példa szöveges irodalomjegyzék-adatbázisra BibTEX használata esetén.

A stílusfájl egy `.sty` kiterjesztésű fájl, de ezzel lényegében nem kell foglalkozni, mert vannak beépített stílusok, amelyek jól használhatók. Ez a sablon a BiBTEX-et használja, a hozzá tartozó adatbázisfájl a `mybib.bib` fájl. Megfigyelhető, hogy az irodalomjegyzéket a dokumentum végére (a `\end{document}` utasítás elé) beillesztett `\bibliography{mybib}` utasítással hozhatjuk létre, a stílusát pedig ugyanitt a `\bibliographystyle{plain}` utasítással adhatjuk meg. Ebben az esetben a `plain` előre definiált stílust használjuk (a sablonban is ezt állítottuk be). A `plain` stíluson kívül természetesen számtalan más előre definiált stílus is létezik. Mivel a `.bib` adatbázisban ezeket megadtuk, a BiBTEX-fordító is meg tudja különböztetni a szerzőt a címtől és a kiadótól, és ez alapján automatikusan generálódik az irodalomjegyzék a stílusfájl által meghatározott stílusban.

Az egyes forrásmunkákra a szövegből továbbra is a `\cite` paranccsal tudunk hivatkozni, így az 5.2. kódrészlet esetén a hivatkozások rendre `\cite{Wettl04}`, `\cite{Candy86}`, `\cite{Lee87}`, `\cite{KissPhD}`, `\cite{Schreirer00}`, `\cite{Mkrtychev:1997}` és `\cite{DipPortal}`. Az egyes forrásmunkák sorszáma az irodalomjegyzék bővítésekor változhat. Amennyiben az aktuális számhoz illeszkedő névelőt szeretnénk használni, használjuk az `\acite{}` parancsot.

Az irodalomjegyzékben alapértelmezésben csak azok a forrásmunkák jelennek meg, amelyekre található hivatkozás a szövegben, és ez így alapvetően helyes is, hiszen olyan forrásmunkákat nem illik az irodalomjegyzékbe írni, amelyekre nincs hivatkozás.

Mivel a fordítási folyamat során több lépésben oldódnak fel a szimbólumok, ezért gyakran többször is le kell fordítani a dokumentumot. Ilyenkor ez első 1-2 fordítás esetleg szimbólum-feloldásra vonatkozó figyelmeztető üzenettel zárul. Ha hibaüzenettel zárul bármelyik fordítás, akkor nincs értelme megismételni, hanem a hibát kell megkeresni. A `.bib` fájl megváltoztatáskor sokszor nincs hatása a változtatásnak azonnal, mivel nem mindig fut újra a BibTeX fordító. Ezért célszerű a változtatás után azt manuálisan is lefuttatni (TeXstudio esetén `Tools/Bibliography`).

Hogy a szövegbe ágyazott hivatkozások kinézetét demonstráljuk, itt most sorban meghivatkozzuk a [**?**], [**?**], [**?**], [**?**], [**?**] és a [**?**][3] forrásmunkát, valamint a [**?**] weboldalt.

Megjegyzendő, hogy az ékezetes magyar betűket is tartalmazó `.bib` fájl az `inputenc` csomaggal betöltött `latin2` betűkészlet miatt fordítható. Ugyanez a `.bib` fájl hibaüzenettel fordul egy olyan dokumentumban, ami nem tartalmazza a `\usepackage[latin2]{inputenc}` sort. Speciális igény esetén az irodalmi adatbázis általánosabb érvényűvé tehető, ha az ékezetes betűket speciális latex karakterekkel helyettesítjük a `.bib` fájlban, pl. á helyett `\'{a}`-t vagy ő helyett `\H{o}`-t írunk.

[3] Informatikai témában gyakran hivatkozunk cikkeket a Springer LNCS valamely kötetéből, ez a hivatkozás erre mutat egy helyes példát.

Irodalomhivatkozásokat célszerű először olyan szolgáltatásokban keresni, ahol jó minőségű bejegyzések találhatók (pl. ACM Digital Library,[4] DBLP,[5] IEEE Xplore,[6] SpringerLink[7]) és csak ezek után használni kevésbé válogatott forrásokat (pl. Google Scholar[8]). A jó minőségű bejegyzéseket is érdemes megfelelően tisztítani.[9] A sablon angol nyelvű változatában használt `plainnat` beállítás egyik sajátossága, hogy a cikkhez generált hivatkozás a cikk DOI-ját és URL-jét is tartalmazza, ami gyakran duplikátumhoz vezet – érdemes tehát a DOI-kat tartalmazó URL mezőket törölni.

5.6 A dolgozat szerkezete és a forrásfájlok

A diplomatervsablonban a TeX fájlok két alkönyvtárban helyezkednek el. Az `include` könyvtárban azok szerepelnek, amiket tipikusan nem kell szerkesztenünk, ezek a sablon részei (pl. címoldal). A `content` alkönyvtárban pedig a saját munkánkat helyezhetjük el. Itt érdemes az egyes fejezeteket külön TEX állományokba rakni.

A diplomatervsablon (a kari irányelvek szerint) az alábbi fő fejezetekből áll:

1. 1 oldalas *tájékoztató* a szakdolgozat/diplomaterv szerkezetéről (`include/guideline.tex`), ami a végső dolgozatból törlendő,
2. *feladatkiírás* (`include/project.tex`), a dolgozat nyomtatott verzójában ennek a helyére kerül a tanszék által kiadott, a tanszékvezető által aláírt feladatkiírás, a dolgozat elektronikus verziójába pedig a feladatkiírás egyáltalán ne kerüljön bele, azt külön tölti fel a tanszék a diplomaterv-honlapra,
3. *címoldal* (`include/titlepage.tex`),
4. *tartalomjegyzék* (`thesis.tex`),
5. a diplomatervező *nyilatkozat*a az önálló munkáról (`include/declaration.tex`),
6. 1-2 oldalas tartalmi *összefoglaló* magyarul és angolul, illetve elkészíthető még további nyelveken is (`content/abstract.tex`),
7. *bevezetés*: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása (`content/introduction.tex`),
8. sorszámmal ellátott *fejezetek*: a feladatkiírás pontosítása és részletes elemzése, előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések, a tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása, a megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek,
9. esetleges *köszönetnyilvánítás*ok (`content/acknowledgement.tex`),
10. részletes és pontos *irodalomjegyzék* (ez a sablon esetében automatikusan generálódik a `thesis.tex` fájlban elhelyezett `\bibliography` utasítás hatására, az section 5.5ban leírtak szerint),

[4] `https://dl.acm.org/`
[5] `http://dblp.uni-trier.de/`
[6] `http://ieeexplore.ieee.org/`
[7] `https://link.springer.com/`
[8] `http://scholar.google.com/`
[9] `https://github.com/FTSRG/cheat-sheets/wiki/BibTeX-Fixing-entries-from-common-sources`

11. *függelékek* (`content/appendices.tex`).

A sablonban a fejezetek a `thesis.tex` fájlba vannak beillesztve `\include` utasítások segítségével. Lehetőség van arra, hogy csak az éppen szerkesztés alatt álló `.tex` fájlt fordítsuk le, ezzel lerövidítve a fordítási folyamatot. Ezt a lehetőséget az alábbi kódrészlet biztosítja a `thesis.tex` fájlban.

```
\includeonly{
  guideline,%
  project,%
  titlepage,%
  declaration,%
  abstract,%
  introduction,%
  chapter1,%
  chapter2,%
  chapter3,%
  acknowledgement,%
  appendices,%
}
```

Ha az alábbi kódrészletben az egyes sorokat a `%` szimbólummal kikommentezzük, akkor a megfelelő `.tex` fájl nem fordul le. Az oldalszámok és a tartalomjegyék természetesen csak akkor billennek helyre, ha a teljes dokumentumot lefordítjuk.

5.7 Alapadatok megadása

A diplomaterv alapadatait (cím, szerző, konzulens, konzulens titulusa) a `thesis.tex` fájlban lehet megadni.

5.8 Új fejezet írása

A főfejezetek külön `content` könyvtárban foglalnak helyet. A sablonhoz 3 fejezet készült. További főfejezeteket úgy hozhatunk létre, ha új TEX fájlt készítünk a fejezet számára, és a `thesis.tex` fájlban, a `\include` és `\includeonly` utasítások argumentumába felvesszük az új `.tex` fájl nevét.

5.9 Definíciók, tételek, példák

Definition 16 (Fluxuskondenzátor térerőssége). Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. ▪

Example 7. *Példa egy példára. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.*

Theorem 4 (Kovács tétele). Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. ▪

Acknowledgements

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Bibliography

[1] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87 – 106, 1987. ISSN 0890-5401. URL https://doi.org/10.1016/0890-5401(87)90052-6.

[2] John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations*, pages 189 – 196. Academic Press, 1971. ISBN 978-0-12-417750-5. DOI: https://doi.org/10.1016/B978-0-12-417750-5.50022-1. URL http://www.sciencedirect.com/science/article/pii/B9780124177505500221.

[3] Falk Howar and Bernhard Steffen. *Active Automata Learning in Practice*, pages 123–148. Springer International Publishing, Cham, 2018. ISBN 978-3-319-96562-8. DOI: 10.1007/978-3-319-96562-8_5. URL https://doi.org/10.1007/978-3-319-96562-8_5.

[4] Dexter C. Kozen. *Myhill—Nerode Relations*, pages 89–94. Springer Berlin Heidelberg, Berlin, Heidelberg, 1977. ISBN 978-3-642-85706-5. DOI: 10.1007/978-3-642-85706-5_16. URL https://doi.org/10.1007/978-3-642-85706-5_16.

[5] Maik Merten, Falk Howar, Bernhard Steffen, and Tiziana Margaria. Automata learning with on-the-fly direct hypothesis construction. In Reiner Hähnle, Jens Knoop, Tiziana Margaria, Dietmar Schreiner, and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation*, pages 248–260, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-34781-8.

[6] A. Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958. ISSN 00029939, 10886826. URL http://www.jstor.org/stable/2033204.

[7] Bernhard Steffen, Falk Howar, and Maik Merten. *Introduction to Active Automata Learning from a Practical Perspective*, pages 256–296. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-21455-4. DOI: 10.1007/978-3-642-21455-4_8. URL https://doi.org/10.1007/978-3-642-21455-4_8.

Appendix

A.1 A TeXstudio felülete

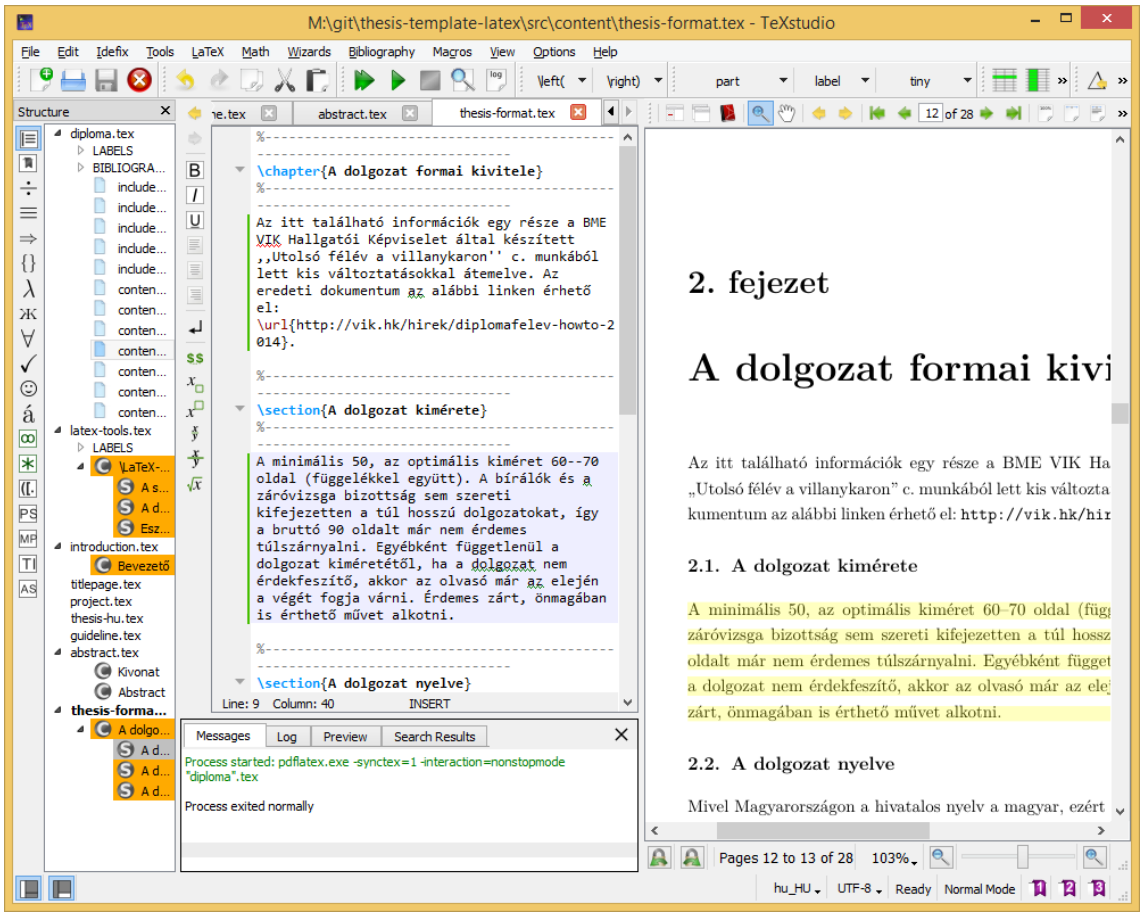

Figure A.2: A TeXstudio LaTeX-szerkesztő.

A.2 Válasz az „Élet, a világmindenség, meg minden” kérdésére

A Pitagorasz-tételből levezetve

$$c^2 = a^2 + b^2 = 42. \tag{A.5}$$

A Faraday-indukciós törvényből levezetve

$$\operatorname{rot} E = -\frac{dB}{dt} \qquad \longrightarrow \qquad U_i = \oint_{\mathbf{L}} \mathbf{Edl} = -\frac{d}{dt} \int_A \mathbf{Bda} = 42. \tag{A.6}$$