Annales Mathematicae et Informaticae
57 (2023) pp. 49–64
DOI: https://doi.org/10.33039/ami.2023.03.002
URL: https://ami.uni-eszterhazy.hu

P4Query: Static analyser framework for P4*

Dániel Lukács, Gabriella Tóth, Máté Tejfel

Faculty of Informatics, ELTE, Eötvös Loránd University,
Budapest, Hungary
{dlukacs,kistoth,matej}@inf.elte.hu

Abstract. There are many important tasks in a conventional software development process which can be supported by different analysis techniques. P4 is a high level domain-specific language for describing the data plane layer of packet processing algorithms. It has several uncommon language elements and concepts that often make the analysis of P4 programs a laborious task. The paper presents P4Query, an analysis framework for the P4 language that enables the specification of different P4-related analysis methods in a generic and data-centric way. The framework uses an internal graph representation which contains the results of applied analysis methods too. In this way, the framework supports the rapid implementation of new analysis methods in a way where the results will be also easily reusable by other methods.

Keywords: P4 language, static analysis, analysis framework

AMS Subject Classification: 68N20 (Theory of compilers and interpreters)

1. Introduction

Optimization, verification and refactoring are important tasks of a software development process. All of them can be effectively supported by static functional and

*The research has been supported by the project “Application Domain Specific Highly Reliable IT Solutions” implemented with the support of the NRDI Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Sub programme) funding scheme.
This research is in part supported by the project no. FK_21 138949, provided by the National Research, Development and Innovation Fund of Hungary.
Supported by the ÚNKP-21-4 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund.

Submitted: May 31, 2022
Accepted: March 3, 2023
Published online: March 19, 2023

non-functional (e.g. execution time estimation) analysis. This analysis can be especially interesting in the case of languages having uncommon language constructs or language structures e.g. in the case of some domain-specific languages.

P4 [2] is a high level, domain-specific programming language. It is developed mainly for describing the data plane layer of packet processing algorithms of different network devices (e.g. switches, routers) in a protocol and target-independent way. Listing 1 illustrates a P4 program. The program first defines the applied header structure (in rows 1–23), then the parser part (in rows 24–35) describes how the fields of the defined headers will be set from the input bit stream (input packet). Controller parts (see rows 39–62) can modify values of fields of headers and metadata by applying lookup tables. During an application of a lookup table the program finds the appropriate row based on the keys in the table. The keys can be specific fields of the packets or some metadata. After the program finds the right row it will execute the action (usually some modifications on the packet) described by the row. It is important to note that the data plane program only defines the possible actions and describes the structure of the lookup tables, namely the keys of the table and the possible results of the lookups. However concrete data of the tables (which actions will be executed with which parameters for which key values) are defined by the control plane program, therefore it will not appear in P4. Finally, the deparse part (see rows 64–70) defines how the output bit stream (output packet) will be created from the headers.

Listing 1. P4 example.

```
1   // Definitions
2   typedef bit<9>  egSpec_t;
3   typedef bit<48> macAddr_t;
4   typedef bit<32> ip4Addr_t;
5
6   // Headers
7   header ethernet_t {
8     macAddr_t dstAddr;
9     macAddr_t srcAddr;
10    bit<16>   etherType;
11  }
12
13  header ipv4_t {
14    bit<8>    ttl;
15    ip4Addr_t srcAddr;
16    ip4Addr_t dstAddr;...
17  }
18
19  struct headers {
20    ethernet_t   ethernet;
21    ipv4_t       ipv4;
22  }
23
24  // Parser
25  parser MyParser(...) {
26    state start { transition parse_ethernet; }
27    state parse_ethernet {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType) {
      TYPE_IPV4: parse_ipv4;
      default: accept; } }
  state parse_ipv4 {
    packet.extract(hdr.ipv4);
    transition accept; }
}

// Control
control MyIngress(in headers hdr, ...) {
  apply {
    if (hdr.ipv4.isValid()) {
           ipv4_lpm.apply();
    }
  }
  action drop() {
    mark_to_drop(standard_metadata);
  }

  action ipv4_forward(macAddr_t dstAddr,
                      egSpec_t port) {
    standard_metadata.egress_spec = port;
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
  }
  table ipv4_lpm {
    key = {  hdr.ipv4.dstAddr: lpm;  }
    actions = {
      ipv4_forward;
      drop;
      NoAction;}
    ... }
}

//Deparser
control MyDeparser(packet_out packet,
                   in headers hdr) {
  apply {
    packet.emit(hdr.ethernet);
    packet.emit(hdr.ipv4);}
}
```

The paper describes an analysis framework for P4[1]. The framework makes possible development of different P4-related analysis methods in a generic and modular way. It uses an internal graph representation where the results of the methods are also represented as part of the graph (mainly by adding new edges to the graph). In this way, the methods can use each other's results as well. The proposed tool also allows rapid prototyping of different analytical concepts using a common toolset.

[1] The code is available from https://github.com/P4ELTE/P4Query.

2. Related work

Considering related work there are much research applying specific analysis techniques for P4. Most of them concentrate on error checking of P4 programs. For example, Assert-P4 [6] and Vera [15] can check the correctness of predefined properties using annotated P4 source code and Vera can also detect some common errors using custom source code without annotations. They use symbolic execution for the analysis. P4V [7] creates a formula, which describes the proper behavior of the program and checks the satisfiability of it with SMT solver. These three tools are created for an earlier version of P4, namely $P4_{14}$. There are also verification tools, which can manage the newer version ($P4_{16}$) too. BF4 [4] is created as a P4C backend, which can not only detect error possibilities, but it is able to repair them by adding new keys to the lookup tables of the program and modify the table contents. Another tool p4-data-flow [1] uses data flow analysis to detect potential bugs in P4 switch codes.

Some other tools use analytical methods for different purposes. For example, p4pktgen [11] uses symbolic execution for automatically generating test cases. Flightplan [16] can split a P4 program into a set of cooperating P4 programs and maps them to run as a distributed system formed of several, possibly heterogeneous, data planes. During this process they use several analysis techniques, to collect variables whose values need to be transferred between different data planes. SafeP4 [5] is a language which has precise semantics and a static type system that can be used to obtain guarantees about the validity of all headers which are used or modified by the program. The type checker of the language (P4Check) can also check P4 programs executing some static analysis on them.

Comparing these approaches this paper presents a generic framework that allows the implementation of several analyses methods which can use each other's results as well.

A major inspiration for this work was RefactorErl [3], a static analysis tool for Erlang. RefactorErl stores program information in a graph called the Semantic Program Graph using relation databases, and provides its own query language for exploring the stored information. Many features in our work – such as using graph databases and their built-in query languages instead of in-house solutions – can be considered to be the streamlining of best practices found in RefactorErl.

As Section 4.1 introduces, P4Query uses a Gremlin graph database as a storage backend. Recently, other works also leveraged graph databases for static analysis purposes. ProgQuery [14] is a similar static analysis tool for Java, built on Neo4j and its Cypher query language. The authors emphasize that using Neo4j yielded substantial improvements in analysis time and memory usage.

The expressive power of Gremlin is proved in another recent work [18]: the authors store C code information in Neo4j, and define recurring vulnerability patterns as Gremlin queries. Using this approach, the authors discovered 18 previously unknown vulnerabilities in the Linux kernel.

3. Motivation

P4 is a relatively new, domain-specific language having some uncommon language elements (for example match-action tables). The language makes possible the description of the data plane layer of packet processing algorithms. For a real implementation, however, in addition to the program part described in P4, a suitable control plane layer is needed. This part is more or less a black hole while we consider only the P4 source code. For these reasons, testing, verification, and generally functional and non-functional analysis of P4 programs are non-trivial tasks that sometimes require language-specific techniques.

Section 2 introduces several applications using different analysis methods for P4. The authors also have some earlier results for determining potentially erroneous code parts [17] and for predicting execution cost [9] of P4 programs. These methods usually require different analysis techniques, however these techniques often can have very similar subtasks (for example creating an abstract syntax tree or a control flow graph).

This paper presents a framework that allows the implementation of different analysis methods using a common basis. The framework applies an extensible integral graph representation where the results of the different analysis methods are represented also as part of the graph. This makes possible execution of different methods in a hierarchical order where methods can use the results (or some part of the results) of previously applied methods.

The framework supports the rapid implementation of new analysis methods applicable for P4 language in a way where results of the methods will be easily reusable.

4. Analysis framework

Traditional compilers are designed around passes: the frontend passes parse source code into an intermediate representation (IR), midend passes transform and add new information to the IR, and the backend passes create target-specific object code from the IR. Modular compilers like the P4 reference compiler P4C [12] improve this design by structuring the passes into a library: backends assemble their own frontends and midends from a catalogue of passes provided by the compiler. Moreover, P4C allows getting information from older states of the IR, as each transformation pass returns an immutable IR instance. Even here, the three-fold separation of frontend-midend-backend have to remain: in order satisfy midend-dependencies, and subsequently, backend-dependencies, the backend must sequentially execute the frontend, the midend, and finally its own passes.

One goal of the experiment we present here is to relax the three-fold structure and allow passes to reuse (depend on) each other's functionally arbitrarily, and without burdening the compiler programmer with manually finding the right sequence in which to execute the different passes.

4.1. Tool architecture

4.1.1. Description

Figure 1 depicts the four-fold design we propose as a solution for relaxing traditional compiler architecture. In principle, the components called end-user *applications* constitute the backend, i.e. the part that provides useful services to users. (See Section 5 for a few examples of applications built on top of the P4Query.) Superficially, applications provide their services by using the services provided by the *infrastructure* (see Section 4.2), also known as the frontend. In reality, both the infrastructure and applications operate on a large shared *graph* that collects all our knowledge about the program code.

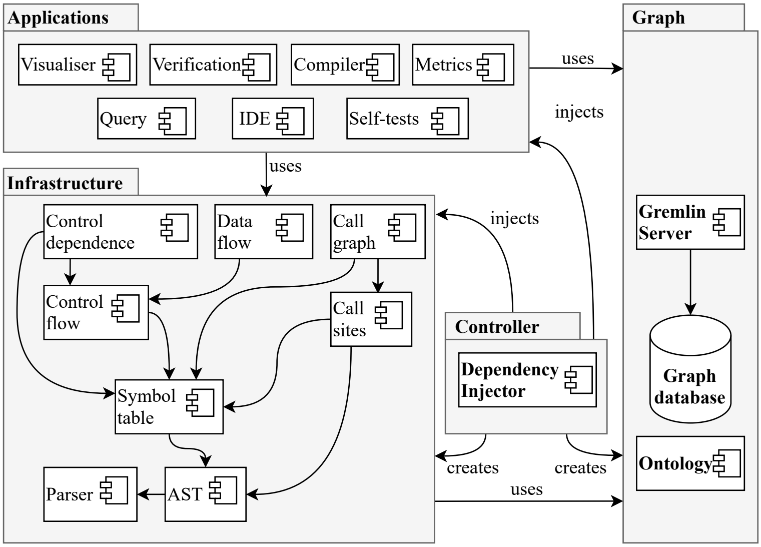


Figure 1. Architectural design of P4Query.

The infrastructure consists of a set of interdependent *analyser* components (or passes): higher-level analysers can only be executed when lower-level analysers already inserted the necessary knowledge into the graph. Similarly to analysers, applications also depend on a subset of the analysers in the infrastructure. But unlike analysers, applications are expected to only read (never modify) the graph, and consequently there are no application-dependent parts in the infrastructure. Applications must also provide a user interface (e.g. command line interface) through which their services can be accessed.

In the middle, the *controller* component ensures (via dependency injection) that all dependencies are satisfied without collisions. To achieve this, components register their provided services and their requirements to the control component, and the controller figures out in which topological order to start the analysers. The controller also guarantees that when the user executes a work-intensive application, only the minimally necessary components will be performed.

The graph is also exposed to the user (not depicted) to enable custom features (e.g. to attach external loggers, visualisers, and validators).

4.1.2. Design goals

Besides proposing a more relaxed structure of analysis passes, we had three additional goals in sight: support for different applications through uniform APIs, ease of extensibility, and data-driven programming.

In systems with *uniform APIs*, programmers have to learn only one paradigm to maintain, extend or otherwise alter the system (e.g. in the case of P4C, visitors and passes are the main concepts of such a uniform API). By supporting different applications (or backends), we mean providing a comfortable way to implement different end-user services, by reusing the same static code analysis operations. P4Query realizes uniform APIs by relying on a graph database. The information in the knowledge graph is accessed using graph queries written in the Gremlin (see 4.1.3) query language. The implication is that users, application developers, and infrastructure developers are using one, uniform data structure (the graph), and are accessing it using the same mechanism (graph queries).

Our second design goal, *ease of extensibility* is also illustrated by Figure 1. The four-fold arrangement was inspired by declarative build systems and the blackboard pattern used in distributed MI. When developers introduce new features, this arrangement enables them to think declaratively: instead of thinking about where to insert their feature inside a sequence of operations, they only have to think about their dependencies, i.e. what kind of analysers could help them.

Finally, our third design goal is *data-driven programming*. Thanks to the uniform graph API and the controller-managed dependency resolution, programmers are forced to think in terms of data instead of code: they have to look at what code knowledge is in the graph already, figure out what data they want to insert, and possibly find existing analysers that make writing queries easier for them. The information in the graph is regulated by a well-defined graph schema, and the graph topology is regulated by the well-defined requirements and services of the analysers. Moreover, since the graph instance is detached from the code analysis framework, the programmers can access it by external tools for visualising, monitoring and validating purposes. Like this, programmers can almost completely avoid understanding the existing code base, and only have to look at and interact with the data in the graph.

4.1.3. Gremlin API

In the tool architecture described in this section, we use a Gremlin graph database as a storage backend (knowledge graph). Gremlin is a compositional query language and API that is implemented by many large-scale graph databases. This makes it possible to change graph implementations with almost no modification to the P4Query code base. In earlier work [8], we also profiled a few graph backends for control flow traversals, and verified that – apart from the initial overhead – in-memory graph databases have comparable performance to built-in memory manipulation.

Another consequence is that analysers have to be implemented as graph query workflows. Since Gremlin is Turing-complete [13], theoretically all of the work can be delegated to the database, and with this, the choice of the workflow language (e.g. Java) can become negligible. Still, in our experience, coarsely granularised queries can hinder code maintainability, as these are often more difficult to read and modify (due to their of lack of common convenience features, e.g. exception handling). For this reason, we still decided to split the workload between the controller and the database.

4.2. Infrastructure

As we see earlier in Figure 1, the heavy-lifting in P4Query is done by various code analyser components, each adding new information about the P4 code to the knowledge graph using what is already there. We now introduce a few analyser modules using an example: Figure 2 depicts a small subset of the knowledge graph taken after we executed control flow analysis, call analysis, and call sites analysis on the P4 code in Listing 1 (specifically the `MyIngress` control). First, the controller finds the topological order of their dependencies, and then starts executing them in order. In this case, the first dependency executed is the parser, parsing the P4 code and filling the knowledge graph with the syntax tree nodes and edges.

Each analyser components adds new edges as an overlay graph. These overlays (domains) are separated by the `dom` edge-attributes: for example `CFG` is the domain introduced by the control flow analysis, and `CALL` is the domain of the call analysis, `SITES` is that of call site analysis. The `role` edge-attribute describes edge semantics inside their domain. For example, `calls` in `CALL` links a procedure to those procedures that it calls, such as `MyIngress` control (node 1) calling the `ipv4_lpm` table lookup (node 8). On the other hand, `calls` in `SITES` links call statements to the called procedure, such as the *direct application* of table `ipv4_lpm` (node 7) calling table `ipv4_lpm` (node 8).

At the same time, the `CFG` domain contains `flow`, `entry`, and `return` edges (among others) to denote the flow of control between various nodes of the syntax tree, and to identify entry and exit nodes. For example, by following these edges, you can see how control flows from `MyIngress` entry point (node 1) through the conditional (node 4), terminating on the call of `ipv4_lpm` (node 8). The figure also partially includes domains of other analysers, such as `SYMBOL`. This analyser creates the graph-equivalent of a symbol table by identifying which declaration declares which name, and links usages of this name in the scope of the declaration to the declaration.

We should note that topological order is, in general, not unique: the controller is free to start independent analysers in any order (even in parallel). This is not a concern as long as analysers can correctly declare their exact dependencies. Still, since all analysers work on the same shared graph, it may happen that – due to

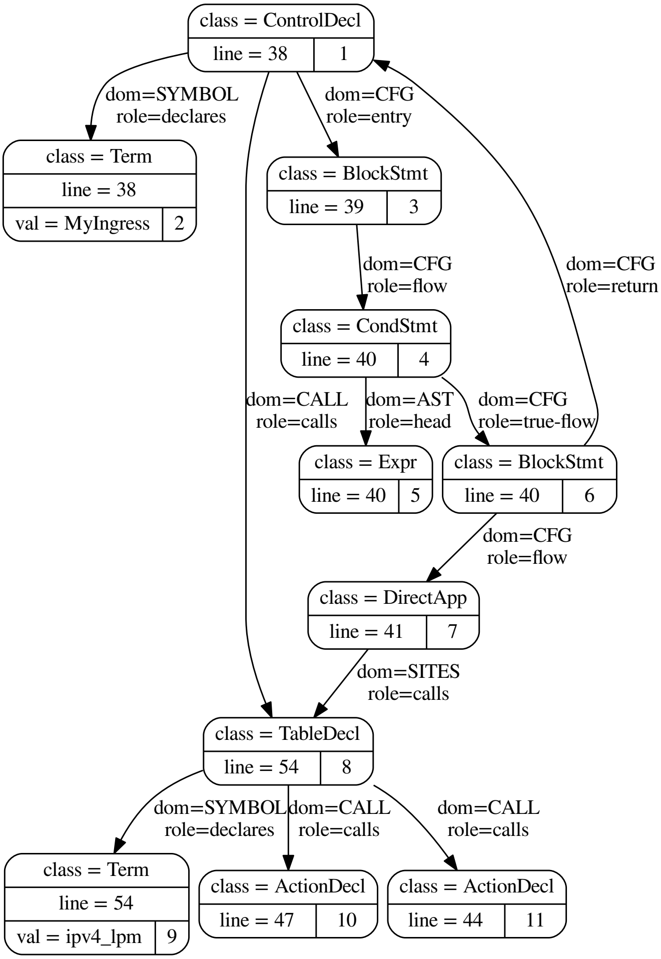


Figure 2. Knowledge graph excerpt of the Listing 1 code.

faulty implementation – an analyser have a “hidden” (unclaimed) dependency. In our experience with the aforementioned analysers, these occurrences are uncommon. Still, to avoid such bugs, we emphasize proper testing (Section 4.3) and recommend implementors to avoid writing general queries (such as selecting *all* elements) and always specify completely the elements to be selected.

4.3. Testing

Testing framework of the tool aims to achieve two main objectives: to provide the correct behaviour of the analysers and to detect possible spoils of the analysers during the development phase. To achieve these goals the tool applies unit tests and integration tests.

Unit tests need to be fast, so they work with the smallest part of the analysers, their functions. One function usually defines one query of the graph, which insert new edges into it, therefore in these cases, the tests check if the right edges are added to the graph. Using an actual P4 source to test these functions would be too costly, therefore we define the most simple graphs to check the function.

While unit tests need to be fast, integration tests can be slower, so we can use P4 files as the inputs to test the analysers. When one analyser needs to be tested, it uses the P4 file and executes every analyser, that it depends on and the tests will check the result of this running.

These tests are important for the P4Query developers, who would like to modify the predefined analysers or supplement the tool with new analysers. After the development of an analyser, the developer can insert the unit tests of the new functions and the correctness of them can be checked by these tests. For unit tests, the developer needs to define the smallest graph, which can cover most of the behaviours of the functions. If the functions are well tested, the developer can continue with integration tests and checks the correct behaviour with real P4 files.

The architecture gives the opportunity to insert this test framework as an application, which depends on all of the tested analysers. As an application, it fits into the tool as a component, which can be easily executed.

5. Case studies

In this section, we illustrate the viability of the platform by showcasing a few applications we are currently building on top of P4Query in related research.

5.1. Visualisation

Since it is the easiest to understand, the first application we introduce is graph visualisation. This application expects a list of analyser component names, executes them, and then, prints a subgraph of the knowledge graph containing only the domains of the analysers in the list. For example, to print the full version of the graph in Figure 2, we should execute P4Query with the following arguments:

```
p4query draw example.p4 –A CFG SYMBOL CALL SITES AST
```

The subcommand `draw` tells P4Query to run the visualiser on the file `example.p4`, while `-A` is a flag (defined by the visualiser UI) expects the analyser names that will be passed to the visualiser application.

A possibly interesting implementation detail here is that the visualiser technically depends on *all* the analysers defined in P4Query, since it must be able to visualise anything the user may pass. Yet, we still managed to avoid executing those that are not requested by the user (and not dependencies of the requested ones): we implemented dependency resolution in the controller using Java dependency injection (DI), and DI offers lazy initialization of the dependencies. This way we can filter the analysers and only initialize those that were requested by the user.

5.2. Verification

Verification is a possible extension of the tool, which is added to it as an application. The main focus is to detect errors and suspicious cases, which can be caused by the use of invalid header or uninitialized fields. The goal of this detection is to report these uses for the developer to avoid undefined behaviour in the programs.

The approach of the checking is defined in our previous paper [17], but in short, it calculates the pre-and post-conditions of the different blocks (i.e the control apply functions, the tables and actions), the parser and the deparser of the program, and based on these condition pairs it can detect improper use of the fields and headers. Three cases can be detected: when there are some errors in a block; when there is any inconsistency between the blocks; and when the post-/precondition of the parser/deparser is inconsistent with the pre-/postcondition of the control function.

Listing 2. Conditions of `MyIngress`.

```
MyIngress:
[
// true condition and ipv4_forward
  (Pre:
     valid:   [ipv4, ipv4.dstAddr, ipv4.ttl,
               ethernet, ethernet.dstAddr],
     invalid: [drop],
   Post:
     valid:   [ipv4, ipv4.dstAddr, ipv4.ttl,
               ethernet, ethernet.dstAddr],
     invalid: [drop]),
// true condition and drop
  (Pre:
     valid:   [ipv4, ipv4.dstAddr],
     invalid: [drop],
   Post:
     valid:   [drop, ipv4, ipv4.dstAddr],
     invalid: []),
// true condition and NoAction
  (Pre:
     valid:   [ipv4, ipv4.dstAddr],
     invalid: [ipv4, ipv4.dstAddr, drop],
   Post:
     valid:   [ipv4, ipv4.dstAddr],
     invalid: [ipv4, ipv4.dstAddr, drop]),
// false condition
  (Pre:
     valid:   [ipv4, ipv4.dstAddr],
     invalid: [ipv4, ipv4.dstAddr, drop],
   Post:
     valid:   [ipv4, ipv4.dstAddr],
     invalid: [ipv4, ipv4.dstAddr, drop]),
]
```

Listing 2 illustrates conditions calculated for control MyIngress in Listing 1. We can see 4 pairs of conditions because it has 4 possible execution paths – there are three where the condition of the branch is true, and the table executes one of the possible actions i.e. *ipv4_forward*, *drop* or *NoAction*, and one where the condition of the branch is false.

This calculation is built into the tool as an application. It uses two experts: the call graph and the control-flow graph. While traversing backwards in the call graph it can reach the applied ("called") actions and tables. Whenever it reaches a vertex like these, it starts to traverse through the proper subgraph of the control-flow graph and calculates the conditions of the actual block. Every condition is stored in the graph as a property of the called vertex of the call graph, therefore when the method reaches the actual call in the control-flow graph – for example a table is called in a control function – it can use the conditions of the called block, which have already been calculated.

5.3. Compiler

In related research [9, 10], we work on a static cost analysis tool for P4: the tool expects as input a P4 program source code together with execution environment parameters, and outputs various metrics (e.g. execution time, energy efficiency) without actually running the P4 program.

In the current paper, we will not go into details on how the cost analysis tool calculates these metrics, but the principle is that we decompose the P4 program into primitive instructions whose expected cost is constant and already known. Implementations of P4 externals such as extern calls (e.g. `packet.extract` in Listing 1) and lookup tables (e.g. `ipv4_lpm` in Listing 1) can also be passed to the tool in the form of these primitive instructions with known costs.

Listing 3. Stack machine code of `MyIngress`.

```
data:
  ...
  headers = 149
  headers.ethernet = 149   // size 114
  ...
  headers.ipv4 = 263
  headers.ipv4.valid = 263
  headers.ipv4.size = 264
  headers.ipv4.srcAddr = 265
  headers.ipv4.dstAddr = 297
  ...
code:
  ...
  // call isValid(hdr.ipv4) on line 144
  214:  load 0         // 0: local address of hdr
  215:  const 114      // 114: size of hdr.ethernet
  216:  add            // address of hdr.ipv4
  217:  invoke 144 1
  // test isValid return value
  218:  ifeq 224
```

```
// call ipv4_lpm(hdr) on line 38
219:  load 0         // 0: local address of hdr
222:  invoke 38 1
223:  pop
// terminate with status OK
224:  const 0
225:  return
```

From this, it follows that part of the static cost analysis problem reduces to a compilation-and-linking problem. As an experiment, we implemented a compiler to solve this problem as an application in P4Query. The main reason we chose P4Query, instead of the much more mature P4C compiler framework was that at first we did not know what kind of representation or code we will need to output: the control and extensibility provided by P4Query and Gremlin queries gave us tools to experiment and create quick, recyclable prototypes to help us arrive at a final vision. While P4C's safety mechanisms (e.g. C++ static type system) support developing stable software, in the case of prototyping and experimentation these same mechanisms are unused, or possibly even slowing down development.

Our current target representation for cost analysis is a sequential stack machine with an instruction set similar to JVM bytecode. Listing 3 depicts the compiler output of `MyIngress` in Listing 1. In the figure all values (bits and sizes) are represented as integers (this is a requirement by our cost analysis approach). Both `isValid` and `ipv4_lpm` have external implementation that had to be linked with the calls. While most P4 targets will not support stack machines, we chose this representation as it is relatively easy to generate, and relatively straightforward to implement. We also believe that as long as we do not count the cost of maintaining the stack, we can still make good cost estimations.

The compiler is built on top of the control flow analyser in P4Query: we traverse the CFG, and process each node by traversing the syntax tree under the node. We also use the call graph to find which label to jump to when a function is called. Thus, much of the compiler state can be delegated to the persistent knowledge graph, and only very specific data (e.g. instruction labels) and linking requires program state outside the graph.

6. Evaluation

Scalability is a very important aspect in the case of analyser tools. For investigating scalability of P4Query we have created dummy P4 programs in which the complexity of the program structure and program logic are increased continuously. In the basic case, two header type were used with one header instances each. The program first parses the two headers, then applies a table which can modify some fields of the headers, and finally it deparses them. In the second program the same structure is applied twice. The four headers are parsed (and finally deparsed) one after the other and two tables are applied sequentially. The first table uses the first header pair, and the second one the second header pair. And so on if the *complexity* of one test program said to be x then there are x header instances of both header types and x tables in the program. As a result if we increase the *complexity* of a test program, its syntax tree will be more complicated and time-consuming to process during different analysis.

Figure 3 illustrates the runtime of P4Query if we execute the CFG analyser (and its dependencies, including the syntax tree and other analysers). We highlighted the results, where the *complexity* of the program is 1, 2, 4, 8 and 16, with a fitted linear regression curve. The diagram shows that the runtime increases in linear time, so we expect P4Query to easily handle even more complex programs. Additionally, we can also inspect the runtime of individual analyses: looking at the corresponding components in each column, we see they are increasing linearly as well, which implies that it is possible to give efficient implementations of the static analysis algorithms in Gremlin.

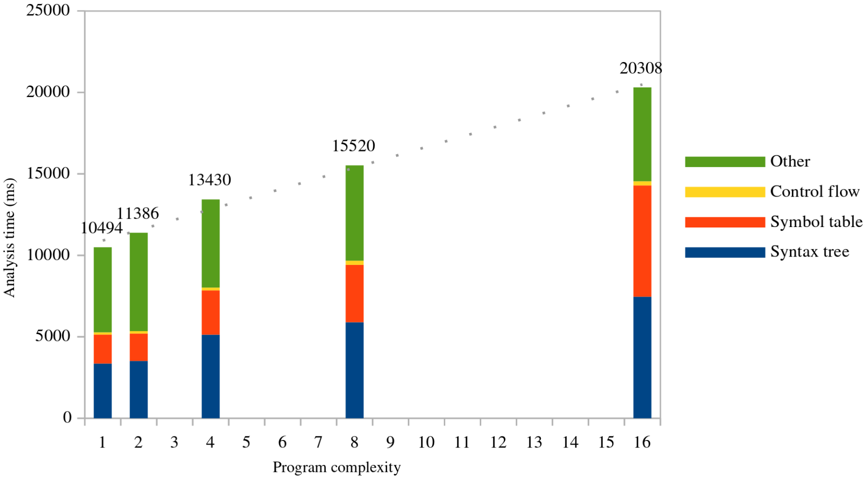


Figure 3. P4Query execution time for different program sizes.

7. Conclusion and future work

Our major purpose was to create a tool, which can facilitate and support the work of P4 developers while making the possibility to experiment with these programs. Its modular structure gives the opportunity for the user to avoid the usage of several tools for different analyses, although it makes the possibility to have all information in one place.

The framework uses a graph representation of the investigated program. All of the analysers are based on the syntax tree of the examined P4 source and they extend it with new edges while creating new subgraphs – like control-flow or call graph – or new labels to store the calculated information.

In the future, we would like to extend the tool with new analyses to give some other useful information for the developers about their P4 source. Our nearest idea is to supplement it with the dependency graph and def-use graph with which we will be able to give report, which are based on the connection between the statements.

References

[1] K. Birnfeld, D. C. da Silva, W. Cordeiro, B. B. N. de França: *P4 Switch Code Data Flow Analysis: Towards Stronger Verification of Forwarding Plane Software*, in: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, 2020, pp. 1–8, doi: https://doi.org/10.1109/NOMS47738.2020.9110307.

[2] P. Bosshart, et. al: *P4: Programming Protocol-independent Packet Processors*, SIGCOMM Comput. Commun. Rev. 44.3 (2014), pp. 87–95, issn: 0146-4833, doi: http://doi.acm.org/10.1145/2656877.2656890.

[3] I. Bozó, D. Horpácsi, Z. Horváth, R. Kitlei, J. Köszegi, T. M., M. Tóth: *RefactorErl - Source Code Analysis and Refactoring in Erlang*, in: Proceedings of the 12th Symposium on Programming Languages and Software Tools, ISBN 978-9949-23-178-2, Tallin, Estonia, Oct. 2011, pp. 138–148.

[4] D. Dumitrescu, R. Stoenescu, L. Negreanu, C. Raiciu: *Bf4: Towards Bug-Free P4 Programs*, in: Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '20, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 571–585, isbn: 9781450379557, doi: https://doi.org/10.1145/3387514.3405888.

[5] M. Eichholz, E. Campbell, N. Foster, G. Salvaneschi, M. Mezini: *How to Avoid Making a Billion-Dollar Mistake: Type-Safe Data Plane Programming with SafeP4*, in: 33rd European Conference on Object-Oriented Programming, ECOOP 2019, July 15-19, 2019, London, United Kingdom, ed. by A. F. Donaldson, vol. 134, LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 12:1–12:28, doi: https://doi.org/10.4230/LIPIcs.ECOOP.2019.12.

[6] L. Freire, M. Neves, L. Leal, K. Levchenko, A. Schaeffer-Filho, M. Barcellos: *Uncovering Bugs in P4 Programs with Assertion-Based Verification*, in: Proceedings of the Symposium on SDN Research, SOSR '18, Los Angeles, CA, USA: Association for Computing Machinery, 2018, isbn: 9781450356640, doi: https://doi.org/10.1145/3185467.3185499.

[7] J. Liu, W. Hallahan, C. Schlesinger, M. Sharif, J. Lee, R. Soulé, H. Wang, C. Caşcaval, N. McKeown, N. Foster: *P4V: Practical Verification for Programmable Data Planes*, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '18, Budapest, Hungary: ACM, 2018, pp. 490–503, isbn: 978-1-4503-5567-4, doi: http://dx.doi.org/10.1145/3230543.3230582.

[8] D. Lukács, G. Pongrácz, M. Tejfel: *Are Graph Databases Fast Enough for Static P4 Code Analysis?*, in: Proceedings of the 11th International Conference on Applied Informatics 2020, CEUR Workshop Proceedings, 2020, pp. 213–223, url: http://ceur-ws.org/Vol-2650/#paper22.

[9] D. Lukács, G. Pongrácz, M. Tejfel: *Control flow based cost analysis for P4*, Open Computer Science 11.1 (2021), pp. 70–79, doi: https://doi.org/10.1515/comp-2020-0131.

[10] D. Lukács, G. Pongrácz, M. Tejfel: *Model Checking-Based Performance Prediction for P4*, Electronics 11.14 (2022), issn: 2079-9292, doi: https://doi.org/10.3390/electronics11142117.

[11] A. Nötzli, J. Khan, A. Fingerhut, C. Barrett, P. Athanas: *P4pktgen: Automated Test Case Generation for P4 Programs*, in: Proceedings of the Symposium on SDN Research, SOSR '18, Los Angeles, CA, USA: Association for Computing Machinery, 2018, isbn: 9781450356640, doi: https://doi.org/10.1145/3185467.3185497.

[12] P4 Language Consortium: *P4C reference compiler for the P4$_{16}$ programming language*, https://github.com/p4lang/p4c, [Online; accessed 06-June-2021], 2017.

[13] M. A. Rodriguez: *The Gremlin graph traversal machine and language (invited talk)*, Proceedings of the 15th Symposium on Database Programming Languages - DBPL 2015 (2015), doi: http://dx.doi.org/10.1145/2815072.2815073.

[14] O. Rodriguez-Prieto, A. Mycroft, F. Ortin: *An Efficient and Scalable Platform for Java Source Code Analysis Using Overlaid Graph Representations*, IEEE Access 8 (2020), pp. 72239–72260, doi: https://doi.org/10.1109/ACCESS.2020.2987631.

[15] R. Stoenescu, D. Dumitrescu, M. Popovici, L. Negreanu, C. Raiciu: *Debugging P4 Programs with Vera*, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '18, Budapest, Hungary: ACM, 2018, pp. 518–532, isbn: 978-1-4503-5567-4, doi: http://dx.doi.org/10.1145/3230543.3230548.

[16] N. Sultana, et. al: *Flightplan: Dataplane Disaggregation and Placement for P4 Programs*, in: 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21), USENIX Association, 2021, isbn: 978-1-939133-21-2, url: https://www.usenix.org/conference/nsdi21/presentation/sultana.

[17] G. Tóth, M. Tejfel: *Component-based error detection of P4 programs*, Acta Cybernetica (2021), to appear.

[18] F. Yamaguchi, N. Golde, D. Arp, K. Rieck: *Modeling and Discovering Vulnerabilities with Code Property Graphs*, in: 2014 IEEE Symposium on Security and Privacy, 2014, pp. 590–604, doi: https://doi.org/10.1109/SP.2014.44.

Annales Mathematicae et Informaticae
57 (2023) pp. 65–77
DOI: https://doi.org/10.33039/ami.2023.04.002
URL: https://ami.uni-eszterhazy.hu

Logical conditions in programming languages: review, discussion and generalization*

Benedek Nagy[ab*], **Khaled Abuhmaidan**[c], **Monther Aldwairi**[d]

[a]Department of Mathematics, Faculty of Arts and Sciences,
Eastern Mediterranean University, Famagusta,
North Cyprus, via Mersin-10, Turkey
corresponding author: nbenedek.inf@gmail.com

[b]Department of Computer Science,
Institute of Mathematics and Informatics,
Eszterházy Károly Catholic University,
Eger, Hungary

[c]Faculty of Computing and Information Technology,
Sohar University, Oman
khmaidan@su.edu.om

[d]College of Technological Innovation, Zayed University,
144534 Abu Dhabi, United Arab Emirates
monther.aldwairi@zu.ac.ae

Abstract. Boolean logic is widely used in almost every discipline including linguistics, philosophy, mathematics, computer science and engineering. Boolean logic is characterized by the two possible truth values, and various logical connectives/operations allow us to make compound statements, conditions. Most of the programming languages, if not all, have some of the logic operations: conjunction, disjunction and negation. Actually, since the set of these three operations form a basis, any logical statement can be formed by them. However, on the one hand, there are smaller bases as well, i.e., one of the conjunction or disjunction is already superfluous. Moreover, there are bases with only one operation, e.g., by NAND. On the other hand, one may allow other operations helping the programmer/user to define the conditions of conditional statements and loops in a simpler manner. In this paper we

*This research was supported by Zayed University, Research Office, Research Incentive Fund Award #R20089.

Submitted: April 8, 2022
Accepted: April 20, 2023
Published online: April 28, 2023

discuss these issues, including some practical points, implementation issues and short cut evaluations for various operations.

Keywords: High level programming languages, conditional statements, loop conditions, logical connectives, short cut evaluation, formal logic

AMS Subject Classification: 03B70, 03B05, 68N15, 68N20, 97P40

1. Introduction

Classical Boolean logic is a well-known and widely used basis of various mathematical disciplines and also of computer sciences, including hardware and software related fields. In this paper, we are, first, analysing how it occurs in high-level programming languages, e.g., conditional statements (see Section 2). Then overviewing classical logic we give reasons why the logical operations conjunction, disjunction and negation are used, and not a smaller basis of the operations. However, there are other widely used logical operations which may help our lives to be easier, e.g., implication and equivalence, if they are also allowed to be used in our formulations. But then, one may ask the question, why the programmers should translate all their conditions to a form that may include only that three connectives that are built in the programming languages, and why the others are not like that. This is the motivation of our paper. We show how the other usual logical operations can be used in our programs. We give some thoughts also on possible implementations, e.g., based on preprocessing or a functional representation that is close to the so-called Polish notation introduced by Łukasiewicz [12].

2. Conditions in programs

In high-level programming languages both the branching and the loop structures occur frequently. The branching is usually done by conditional statements. The usual form of a conditional statement (although it may be slightly varied in various programming languages) is either

`if` (condition) `then` statement

or

`if` (condition) `then` statement(1) `else` statement(2)

For details about the syntax used in various languages that are not explained here, the readers are referred to textbooks [8–11, 22, 25, 28]. Note that some of the languages are case sensitive and some of them are not.

In this paper we are interested and concentrate on that part which includes logic: The (condition) in the above structures means logical condition, i.e., a logical formula that is evaluated by the computer, and if it evaluates to true then the statement (or statement(1) in the second case) is executed, otherwise they are not executed at this time (but statement(2) is executed in the second case) and the program then continues by the next statement.

The logical condition can be a simple condition, e.g., $(x < 15)$ or a compound statement. In most of the programming languages the statement can also be compound, i.e., a block built up by a sequence of statements, for syntax see the mentioned textbooks.

The loops usually have heads with conditions and bodies with statements. The condition in the head is also a logical condition that is evaluated to be either true or false and based on that either the statements of the body of the loop are executed (once more) or the program continues by the statement after the body of the loop. There are various types of loops that can be used (depending also on the used language). Most of the languages have `for` loops, however, in some of the languages that type of loop does not have a formal logical condition, but the body should be executed for some specific values of the loop variable (e.g., in Pascal and Python). In some other languages, including C, C++, Java and also Javascript, the for loop is very similar to the `while` loops that we are explaining shortly in the sequel. In `while` loops, after the word `while` a condition is written (for syntactic details the author is referred to the textbooks mentioned before), then the body with its statement(s) is written. In the programming languages in which the for loop is similar, after the word `for` in brackets, first an initialisation statement (that is executed only once before checking the loop condition at the first time), then in the middle, the loop condition itself that is very similar to the loop condition of `while` loops, finally, the third part in the bracket is the increment statement, that is a statement which is executed after the body each time when the body is executed, right before the loop condition is (re)checked. Finally, there is also another type of loop, where the condition is after the body. In Pascal it is written as follows: between `repeat` and `until` the statement(s) and right after `until` the logical condition (to decide if the statement(s) in the body are executed once more). In other languages, e.g., in C, C++, Java and Javascript, these are written as `do - while` loops having the body between these words and the logical condition after the second. For more details about its syntax in various languages we recommend to check the mentioned textbooks. We note that any algorithms can be implemented without this type of loop.

Here, we are concentrating on the logical conditions. Thus, let us see how one can build complex conditions. The usual classical connectives to make compound statement are the conjunction, disjunction and negation, see Table 1. In some high level languages there is a simple datatype for Boolean values and there are also built in constant values for both the truth-values.

Although in the programming language C, there is no specific type for Boolean values, integers and pointers can be used also for this purpose in such a way that the value 0 or $NULL$ is understood as false, while any other values are understood as true. In various other languages the values 0 and 1 also play the role of false and true, for details the reader is referred to the above listed (text)books.

Table 1. The logical operations and constants in various high-level programming languages.

Programming Language	conjunction	disjunction	negation	Boolean/logical type	true,	false
Pascal	`and`	`or`	`not`	boolean	`true`	`false`
C	`&&`	`||`	`!`	-		
C++	`&&`	`||`	`!`	bool	`true`	`false`
Java	`&&`	`||`	`!`	boolean	`true`	`false`
JavaScript	`&&`	`||`	`!`	boolean	`true`	`false`
Python	`and`	`or`	`not`	bool	`true`	`false`

3. Bases of Boolean logic

As we have already seen, in Boolean logic there are two truth-values: true and false. Usually they are represented by 1 and 0, respectively. Boolean variables, denoted by, e.g., A, may take one of these values at interpretation and evaluation. As usual in almost every part of mathematics, the operators are unary and binary ones. The only unary operator, the negation is usually (syntactically) written as $\sim A$ or $\neg A$ in formal logic (with a formula A). In Boolean algebra, the notation $\bar{A}$ is used, while various programming languages use various notations as we have already shown them in Table 1. The (semantical) result of this operation is that the compound statement having this operation as the main operation has the opposite truth value than the subformula without this main operation (i.e., the truth value of formula A).

Table 2 shows all possible Boolean operations. Note that the first and last rows are not functions of any of the variables A and B, but always false and true (they are in fact the Boolean constants and can be seen as functions with zero arguments). As we have already mentioned in many programming languages the programmer can use these constants as well, e.g., to have a (theoretically) infinite loop with a condition that is always true. There are two other rows, which are in fact the copies of the values of A and B, respectively. These rows do not define logical connectives, neither.

Further, two of the rows are showing the negation of the variables A and B, respectively, thus the unary operation negation occurs twice in the table.

The remaining ten rows define the ten binary operations, and there are no more [7, 18]. On the one hand, obviously, the conjunction (logical and) and disjunction (logical or) are among them. On the other hand, most of the other operations have also their own names, as they play important roles, either in natural languages (implication, equivalence, exclusive or), in logical deductions (implication) or in the hardware industry (NAND, NOR).

As we have seen there are eleven Boolean operations. How it can happen then, to use only three of them in programming languages? The answer is in the properties of the Boolean algebra [1, 15], i.e., the concept of base set of operations.

Table 2. The Boolean operations (based on all possible binary Boolean functions).

formula	name	values			
A		1	1	0	0
B		1	0	1	0
0	-	0	0	0	0
$A \nmid B$	NAND	0	0	0	1
$A \not\subset B$		0	0	1	0
$\neg A$	negation	0	0	1	1
$A \not\supset B$		0	1	0	0
$\neg B$	negation	0	1	0	1
$A \bigoplus B$	eXclusive OR	0	1	1	0
$A \vert B$	NOR, Sheffer stroke	0	1	1	1
$A \wedge B$	and	1	0	0	0
$A \equiv B$	EQUivalence	1	0	0	1
B	-	1	0	1	0
$A \supset B$	IMPlication	1	0	1	1
A	-	1	1	0	0
$A \subset B$	reverse implication	1	1	0	1
$A \vee B$	or	1	1	1	0
1	-	1	1	1	1

We say that a subset S of the operations listed in Table 2 is a base, if for any number of Boolean variables, one can write equivalent formula L' using the variables and the operations of S to any logical formula (or Boolean function) L of the same variables. A base is also called a functionally complete set of operations, and by Post, it is known that the set must contain at least one operation without each of the following five properties:

- monotonic (by "increasing" the input, i.e., by changing the value of any of the arguments from 0 to 1, the value of the result cannot decrease, i.e., cannot switch from 1 to 0, e.g., $\wedge$ and $\vee$ are monotonic),
- linear/counting (those rows of Table 2 are referred here that have even number of 1's, and thus, also even number of 0's),
- self-dual (those operations are counted here for which by flipping – i.e., changing from 0 to 1 or vice versa – the values of all the arguments, the result must also change to the opposite),
- truth-preserving (if all the arguments have a value truth, then also the result is true) and
- false-preserving (if all the arguments have a value false, then also the result is false),

see more details, e.g., in [19].

It is well-known (see, e.g., [13, 18]) that $S = \{\neg, \wedge, \vee\}$ is a base. On one hand, this is the most used base, since in Boolean algebraic description usually, exactly these operations are used. Further, these operations are used to define the conjunctive and disjunctive normal form expressions, and it is well-known that for any Boolean formula there is an equivalent one in conjunctive/disjunctive normal form. On the other hand this is not a minimal base, as we recall in the next subsection.

3.1. Why not smaller bases?

Although, $S = \{\neg, \wedge, \vee\}$ is a base, it is not minimal: by the use of the De-Morgan identities (and the law of double negation), i.e., $(A \wedge B)$ is equivalent to $\neg(\neg A \vee \neg B)$ and $(A \vee B)$ is equivalent to $\neg(\neg A \wedge \neg B)$, thus one may exclude either $\vee$ or $\wedge$.

Then, in a minimalist language (in terms of defining as less things as possible to make a high level programming language), one may use one of the sets $S_\wedge = \{\neg, \wedge\}$ and $S_\vee = \{\neg, \vee\}$ and keep only negation and one of the binary connectives of S.

Moreover, there are even smaller bases: Sheffer has already shown that one operation is enough to express any Boolean formulae/function [24], he has used the operation named after him, as Sheffer stroke (and it is known also as NOR, i.e., Negated OR, in Boolean algebra and logic gates). On the other hand, the operation NAND also has this property: any logical formulae can be expressed also by using the sole operator NAND. These operations do not have any of the five properties described by Post.

Now, we may ask the question that is given in the head of the subsection: if we have smaller bases, why do we use three connectives and not less in the programming languages.

The answer is obvious: these three connectives are so natural and it is very easy to connect them to natural languages:

- the negation refers for the negative statements, when (usually) the verb part is negated, e.g., "John does not go to the school today." Sometimes another verb meaning the negation of the other exists in the language, e.g., 'remember' could play similar role as 'do not forget'.
- the conjunction refers to connect two (independent) statements by the connective 'and', e.g., "John goes to the school and Bob plays football today." In the language we may use various connectives, e.g., 'and', 'but', or maybe only a semi-colon to connect the two statements and form a compound statement in this way.
- the disjunction usually refers to sentences compounded by the connective 'or', e.g., "Rick likes the taste of the coffee or he likes the hot drinks."

However, to formalize conditions, one needs some special care, as there are various differences in formal logic that is used in mathematics and programming and the 'logic' used in natural languages.

- The connective ‘and’ may have the meaning ‘and then’, e.g., “Bob went to the supermarket and he has bought some drinks.” In this sense, this connective is not always commutative. It is also dangerous to abbreviate the statements and put the ‘and’ between some parts of the sentences without repeating the other parts of the statement. Ambiguity occurs, e.g., “You are allowed to distribute the softwares I wrote and I licensed.” (It may not be clear if it is only about the softwares which I have both written and licensed or also those which I only have written or only have licensed...)

- The connective ‘or’ frequently has ‘xor’ meaning, in the sense that we want to allow only one of the options, e.g., “Jack is drinking a coffee or he is drinking a tea”; “Bob will do his homework or he will fail in the course.” Thus in some cases instead of simply writing ‘or’, in some text ‘and/or’ is written in the usual meaning of the disjunction. (Sometimes to highlight the ‘xor’ feature, the format, “either he will do the homework or he will fail” is used, but the usage of ‘either’ is optional in the language, even if the meaning of the sentences should be the same.)

Now, we turn to give some notes on the usage of a base with only one operator. The options would be to use only NAND or only NOR. However, this would make the writing of the conditions long and hardly understandable causing extra care and possible faults and bugs in coding. Although mathematically and theoretically, these would be options, in practice it would be not a good way to use the high level programming languages in any of these ways. As computers become widespread, to allow more and more people to learn programming and write their own codes it would be very difficult to learn.

Although, in the hardware industry, it could be a good decision, as the connection of the logic gates can be easily checked by simulations and also computers support the design of the circuits, the programmers have not been trained to convert any types of Boolean logical conditions/statements to formulae using only one connective.

4. Expanded Logic

Now, since we have seen why it is not a good idea to use to small number of logical connectives, we show how the set of used connectives can be expanded to allow some more of the usual connectives.

As we have shown in Table 2, the following well-known and widely used operations, i.e., connectives have three true and one false values in their truth table: disjunction, implication, NAND. The connectives conjunction and NOR are defined in the opposite way, i.e., by only one true and three false values. The other two well-known operations, the equivalence and the XOR have two-two true and false values. These facts turn to be important in the sequel.

4.1. Practical implementation

A preprocessor or a macro may substitute the formulae containing any of the IMP, NAND, NOR, EQU, XOR operations to formulae with only conjunction, disjunction and negation. In this way, the code is first transformed to an equivalent code in the traditional/usual programming language, and thus, the programmer may use these larger set of connectives to write a more intuitive and simpler condition, but, in fact, the computer will execute the code after preprocessed in the traditional manner. On the other hand, since programming languages are focusing on performance when making the executable code, and the logical operations on the machine code include e.g., XOR, this operation can be compiled and executed directly.

Nevertheless, as a possible solution to include these new logical operations, we give possible substitution(s) that a preprocessor may do for each of the five operations. Let `a` and `b` abbreviate the two (simple or compound) conditions that are connected by the given operator, their (truth-)value can be 0 and 1.

- `a IMP b` can be written as `NOT a OR b`
- `a NAND b` can be written as `NOT a OR NOT b`
- `a NOR b` can be written as `NOT a AND NOT b`
- `a EQU b` can be written as `(a AND b) OR (NOT a AND NOT b)` or `(a AND b) OR NOT (a OR b)`
- `a XOR b` can be written as `(NOT a AND b) OR (a AND NOT b)`

We may need to use fully bracketed formulae to make their meaning clear. Especially, the substitution of the last two operators, the EQU and XOR seem long. Now some tricks are shown which may be used in the programming language C, where instead of the Boolean type, integers are used. Let `a` and `b` abbreviate the two conditions (which may have values 0 or 1), as before, then we have the following. We start with the operators AND and OR, which do not need to be substituted or interpreted in other ways, as they are built in, however, for the sake of completeness and to feel what types of ideas are behind the scene, we start with those.

- `a AND b` can be written as `a * b`
- `a OR b` can be written as `a + b`
- `a IMP b` can be written as `a <= b`
- `a NAND b` can be written as `!(a * b)`
- `a NOR b` can be written as `!(a + b)`
- `a EQU b` can be written as `a == b`

- `a XOR b` can be written as `a != b` or `!(a == b)`

As here, we are showing rewriting that is close to the style of the programming language C, we used the sign `!` for negation. At the disjunction, as the sum may produce a value that is outside of the targeted set $\{0, 1\}$, in some cases one may need to use it in the form `!!(a+b)` that transforms the value based on the double negation law to the desired set of values. (In some other languages, e.g., in C++, explicit type conversions can also be used to transform the resulted value back to the set of official truth-values.) The sign '`<=`' may seem to be an arrow $\Leftarrow$ or a horseshoe $\subset$, but in fact none of them is used to represent implication in this orientation. On the other hand, the '==' can be seen as a built in equivalence operator, although it is not highlighted in this way. Moreover, it can be used not only in C, but in many other high-level programming languages, as the equality operation is defined usually also on Boolean values. Thus, in this way, we may be happy that, although it is not underlined in the textbooks and courses, in a usual high-level programming language the programmers may use not only negation, conjunction and disjunction, but also the equivalence operator (usually with the lowest priority, therefore bracketing may be needed if one uses it for this purpose).

Another idea could be to use the connectives in functional form, e.g., to write the logical expressions in prefix form (with or without brackets).

Without using any brackets, the so-called Polish notation of formulae, also called prefix form, can be used. This form is invented by Łukasiewicz to avoid brackets and have a unique way to read and evaluate the formulae [12]. In this writing the operators precede their operands, as we show below.

Examples could be:
EQU AND A IMP B, C XOR B NOT A is representing the formula
$((A \wedge (B \supset C)) \equiv (B \bigoplus \neg A))$.
AND OR NOT A, B IMP A, C is representing $(\neg A \vee B) \wedge (A \supset C)$.

With brackets, the connectives can be interpreted as functions, and in this way, they can be programmed in the programming language itself and they can easily be put to a new logical library as well to include them and allow them to be used by the programmers.

Our last example written in this form is:
AND (OR (NOT (A), B), IMP (A, C))

There is also an advantage of this form over the other without brackets, namely, for the associative operations, e.g., for AND, OR, XOR the programmer may use more than two parameters without any problem, misunderstanding or misinterpretation.

We note that although this type of prefix notation is not widely used in logic and mathematics, it is used in computer science, e.g., in the programming language LISP [27]. The reverse Polish notation, the postfix notation, in which all operands precede the operator is also used in computer science, especially in stack-based programming [20, 26].

4.2. Shortcut for the new connectives

On the one hand, the EUQ and XOR operations are of the form of 2-2 (remember to Table 2 and discussion on the number of the occurrences of the truth values). In this way, we cannot do any pruning or short-cut evaluation technique. By knowing the value of the first part one cannot involve the knowledge of the truth-value of the whole formula in any case, the second part must also be evaluated. This phenomenon is related to the fact that, e.g., in Gentzen sequent calculus by working with a formula having main connective as EQU or XOR, the deduction process is branching and in both branch we need to write and use both immediate subformulae instead of the original formula.

Now, on the other hand, let us take a look on our other five operations (including the original AND and OR) we deal with. All of these are 1-3 or 3-1 forms, meaning that by knowing the truth-value of the first part of the expression, we may know the truth-value of the whole expression (in the fortunate case). These cases are listed below.

At operator	if the first part is	then the whole formula is
AND:	false,	false.
OR:	true,	true.
IMP:	false,	true.
NAND:	false,	true.
NOR:	true,	false.

It is clear to see how the short-cut evaluation works for AND and OR, and in fact, these short-cut evaluations are built in features of the programming languages. On the other hand we can also use the new short-cuts listed in the last three rows of the previous table.

We would like to highlight and discuss one interesting issue here: we have listed 5 operators, but there are only 4 possibilities (to have the truth-value of the first part given and to infer from this to the truth-value of the whole formula). Seemingly, in the table IMP and NAND are similar. Actually, if we can use the short-cut, i.e., the evaluation can be done earlier than all parts of the formula are evaluated, then yes, definitely, they work on the same way. However, in case the short-cut evaluation cannot be used, i.e., the first part is true and we need to evaluate the second part, then their difference appears: if the second part is false, then IMP gives false and NAND gives true. Alternatively, if the second part is also true, then IMP gives true and NAND gives a false value to the whole expression.

We note here again the analogy of the possibilities of the usage of the short-cut techniques and the theorem proving methods Gentzen sequent-calculus and Smullyan tableaux. These methods make a branching at some formulae, and if only the first immediate subformula occurs in a branch, then we can make a cut, e.g., at implication, the whole formula evaluates to true, if the first part is false (and we do not need to check the second subformula).

Finally, we highlight that short-cut evaluations are not only used to make the evaluation faster, but they have safety features as well by allowing to shorten some parts of the code.

Consider the following conditional statement with operator IMP and with variables `num`, `divisor`:

```
if ( IMP( divisor > 0 , num/divisor > 5 ) ) return 1 else return 0
```

It will return 1, if the actual value of the `divisor` is negative or in the case when `divisor` has value 0 , without checking the fraction in the second part. Further, it returns also 1 if `divisor` is positive and `num/divisor` is larger than 5. Finally, it returns 0 only if `divisor` is positive and `num/divisor` is at most 5. The second part of the implication, including the division by `divisor` is checked only if the first part was evaluated to true, i.e., the value of the `divisor` is not 0, but it is positive. In this way, the possible error of division by zero is avoided by the short cut evaluation technique. Statements of this type are related to the nature of the material implication widely used in formal logic, namely, if the condition part, the first part of the statement has a false truth-value, then does not matter how strange and weird is the second part, the whole statement is evaluated to be true.

In this way it is very similar to the very usual compound condition with integer variables `num`, `divisor` and `result`.

The double, nested condition

```
if ( divisor !=0 ) if ( num/divisor > 5 ) result = num/divisor
```

can be abbreviated to a sole, but compound condition as

```
if ( divisor != 0 && num/divisor > 5 ) result = num/divisor
```

Note that in the programming language C, the first part can be simplified and the condition (that is in the bracket) can be written as

```
(divisor && num/divisor > 5)
```

The fact that we can write the double nested condition in one compound condition without any risk is related to the fact that, for instance in the programming language C (and in other languages), the logic is not exactly the classical Boolean logic, but a kind of 3-valued not commutative logic (see in [16]). As the condition written in

```
( num/divisor > 5 && divisor != 0 )
```

causes a runtime error in case the value of `divisor` is 0, this is not equivalent to our original form. This already leads to us to the next section.

5. Discussion, conclusion and related works

This study can be seen as a follow up study about logic in programming languages, which we have started in [16]. There we have concentrated on how the logical values are computed and what type of ideas and processes are behind the scene.

In this paper, we give some thoughts about which and how many logical operations can and should be used in a high-level programming language. We argued and give hints on how is possible to include not only the widely used three operations of Boolean algebra, but some other well-known and frequently used operations, like the exclusive or, the equivalence and the implication in our programs to make compound logical conditions. They may allow (beginner) programmers to write their conditions in a simpler way, or in the way that is more closely reflected by the condition stated in natural language. We have also studied the possibilities of short-cut evaluations, which can also be seen, on one hand the generalizations of the very closely related alpha and beta pruning techniques of game theory [21, 23] that are also generalised to games with chance nodes (i.e., with some random events) [14] and for other types of operations [2, 3]. Related works are also done by using and analysing similar techniques in the three most-known and most used fuzzy and many-valued logic systems, in the Gödel type logic [5], in the product logic [6] and in the Łukasiewicz-type logics [4, 17].

Acknowledgements. Comments of the anonymous reviewer are gratefully acknowledged.

References

[1] B. H. Arnold: *Logic and Boolean Algebra*, Dover Publications, 2011, p. 144, isbn: 978-0486483856.

[2] R. Basbous, B. Nagy: *Generalized Game Trees and their Evaluation*, in: CogInfoCom 2014: 5th IEEE International Conference on Cognitive Infocommunications, Vietri sul Mare, Italy, IEEE, 2014, pp. 55–60, doi: https://doi.org/10.1109/CogInfoCom.2014.7020518.

[3] R. Basbous, B. Nagy: *Strategies to Fast Evaluation of Tree Networks*, Acta Polytechnica Hungarica 12.6 (2015), pp. 127–148, doi: https://doi.org/10.12700/APH.12.6.2015.6.8, url: http://acta.uni-obuda.hu/Basbous_Nagy_62.pdf.

[4] R. Basbous, B. Nagy, T. Tajti: *Pruning Techniques in Łukasiewicz Logics*, Acta Polytechnica Hungarica v.n (2022), doi: https://doi.org/10.12700/APH..

[5] R. Basbous, B. Nagy, T. Tajti: *Short Circuit Evaluations in Gödel Type Logic*, in: Ravi V., Panigrahi B., Das S., Suganthan P. (eds) Proceedings of the Fifth International Conference on Fuzzy and Neuro Computing (FANCCO - 2015), vol. 415, Advances in Intelligent Systems and Computing (AISC), Springer, Cham., 2015, pp. 119–138, doi: https://doi.org/10.1007/978-3-319-27212-2_10.

[6] R. Basbous, T. Tajti, B. Nagy: *Fast evaluations in product logic various pruning techniques*, in: 2016 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2016, Vancouver, BC, Canada, July 24-29, 2016, IEEE, 2016, pp. 140–147, doi: https://doi.org/10.1109/FUZZ-IEEE.2016.7737680.

[7] J. L. Bell, M. Machover: *A Course in Mathematical Logic*, North Holland, 1977, p. 599, isbn: 978-0080934747.

[8] D. Flanagan, G. M. Novak: *Java-Script: The Definitive Guide*, American Institute of Physics, 1998.

[9] J. Gosling, B. Joy, G. Steele, G. Bracha: *The Java language specification*, Addison-Wesley Professional, 2000.

[10] E. Horowitz: *Fundamentals of Programming Languages*, Springer, Berlin, Heidelberg, 2012, isbn: 9783642967290.

[11] B. Kernighan, D. Ritchie, C. Tondo: *The C Programming Language*, Prentice-Hall software series, Prentice Hall, 1988, isbn: 9789688802052.

[12] J. Łukasiewicz: *Aristotle's Syllogistic from the Standpoint of Modern Formal Logic*, Oxford University Press, 1951, p. 141.

[13] R. J. McEliece, R. B. Ash, C. Ash: *Introduction to discrete mathematics*, English, New York etc.: Random House, 1989, pp. xv + 514, isbn: 0-394-35819-8.

[14] E. Melkó, B. Nagy: *Optimal strategy in games with chance nodes*, Acta Cybern. 18.2 (2007), pp. 171–192, url: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3712.

[15] E. Mendelson: *Theory and problems of Boolean algebra and switching circuits including 150 solved problems*, English, Schaum's Outline Series. New York etc.: McGraw-Hill Book Comp. 213 p. (1970). 1970.

[16] B. Nagy: *Many-valued Logics and the Logic of the C Programming Language*, in: ITI 2005, 27th International Conference on Information Technology Interfaces, Cavtat/Dubrovnik, Croatia, IEEE, 2005, pp. 657–662, doi: https://doi.org/10.1109/ITI.2005.1491200.

[17] B. Nagy, R. Basbous, T. Tajti: *Lazy evaluations in Łukasiewicz type fuzzy logic*, Fuzzy Sets Syst. 376 (2019), pp. 127–151, doi: https://doi.org/10.1016/j.fss.2018.11.014.

[18] K. Pásztor-Varga, M. Várterész: *A matematikai logika alaklmazásszerű tárgyalása (Mathematical logic from application point of view, in Hungarian, textbook)*, Budapest: Panem, 2003.

[19] F. J. Pelletier, N. M. Martin: *Post's Functional Completeness Theorem*, Notre Dame J. Formal Log. 31.3 (1990), pp. 462–475, doi: https://doi.org/10.1305/ndjfl/1093635508.

[20] A. Puntambekar: *Data Structures*, UNICORN Publishing Group, 2020, isbn: 9789333223911.

[21] E. Rich, K. Knight: *Artificial Intelligence*, Artificial Intelligence Series, McGraw-Hill, 1991, isbn: 9780070522633.

[22] G. van Rossum, The Python Development Team: *Python Tutorial (Release 3.6.6rcl).* CreateSpace Independent Publishing Platform, 2018.

[23] S. Russell, P. Norvig: *Artificial Intelligence: A Modern Approach*, CreateSpace Independent Publishing Platform, 2016, isbn: 9781537600314.

[24] H. M. Sheffer: *A set of five independent postulates for Boolean Algebras, with application to logical constants*, Transactions of the American Mathematical Society 14 (1913), pp. 481–488.

[25] B. Stroustrup: *The C++ programming language*, Pearson Education India, 2000.

[26] M. A. Weiss: *Data Structures and Algorithm Analysis*, Redwood City, CA; Menlo Park, CA; Reading, Ma; New York; Amsterdam; Bonn; Sidney; Singapore; Tokyo; Madrid: The Benjamin/Cummings Publishing Company, Inc., 1995.

[27] P. H. Winston, B. K. P. Horn: *LISP*, United States: Pearson, Jan. 1989.

[28] N. Wirth: *Algorithms & data structures*, Prentice-Hall, Inc., 1985.

Annales Mathematicae et Informaticae
57 (2023) pp. 78–91
DOI: https://doi.org/10.33039/ami.2023.04.001
URL: https://ami.uni-eszterhazy.hu

Application and impact of electronic solutions in teaching programming

József Udvaros, Norbert Forman, Dóra Éva Dobák

Budapest Business School, Faculty of Finance and Accountancy,
Department of Business Information Technology
{udvaros.jozsef,forman.norbert,dobak.dora}@uni-bge.hu

Abstract. The market trends that are determining the electronics industry today point to a sharp increase in the use of IoT devices, sensors are collecting data around us, using wireless data transmission technologies to transmit the measured values to cloud-based databases, which are processed with various software. Low-power microcontrollers developed for battery power, which are widely used today, provide sensor data collection and data transfer control.

In this article, we present a literature search on the technical IT teaching tools in use today, some of which are inherently educational and popular with students and teachers. We pay attention to the educational principles of technical IT methods.

We show with examples how technical IT solutions can provide an appropriate experiential learning opportunity and background in programming education. We focus on teaching methods that use microcontrollers and various sensors to develop programming skills and acquire programming knowledge. By developing both computational and algorithmic thinking, we aim to develop both skills.

Keywords: Robots, microcontrollers, teaching methods

AMS Subject Classification: 94-06

1. Introduction

Today, most of the devices around us are based on electronic solutions which contain a processor and are controlled by a software. With sensors, they are able to convert the physical, chemical and biological signals of the outside world into electronic quantities and then information, which can thus be processed with the help of software. The electronics used allow battery-powered devices that can handle

Submitted: August 10, 2022
Accepted: April 20, 2023
Published online: April 24, 2023

more and more signals to run operating systems and various applications. These electronic tools can be used effectively to improve the quality of secondary and university programming education [12]. In particular, it has a major impact on the development of technical and computer-minded thinking, without which students have difficulty in today's labor market [10].

The aim of this article is to underline the importance of using robots, microcontrollers and IoT (Internet of Thinks) devices in secondary and university education. Using robots, microcontrollers and IoT in education can improve students' algorithmic thinking and familiarize them with programming techniques. The acquisition of tools using real and scientific examples supports the complex development of STEM.

2. Methodology

In the article, we conducted a short literature search, where we focused on technical IT methods, including the project solution method. We determined which electronic devices are used most in education. We will then conduct a research to support our hypothesis according to which the use of robots, microcontrollers and IoT devices in education can improve students' algorithmic thinking and familiarize them with programming techniques.

The contribution of robotics to education to clarify new disciplines is remarkable. In fact, there is a need for experimental examples that facilitate the acquisition of students 'professional knowledge and thus meet the potential of the actual systems used in various modern disciplines. In [6], the authors discuss laboratory experience in implementing an automatic airflow control system for remote configuration and monitoring of convincing size and role. An example is a non-traditional robot. Built-in electromechanical equipment from old farms are being exploited and revived using modern, widely available microcontrollers, smartphones, tablets, network transceivers, motor drives and some low-cost and custom sensors.

Teachers of Technical University of Košice in their article describes the implementation of IoT technology in the teaching of microprocessor technology. The method presented in this article combines the reality and virtualization of a microprocessor technology laboratory. A built-in IoT monitoring device monitors students 'microcontroller needles and sends the data through the control application to the server to which the teacher is connected. The teacher has the opportunity to monitor the development of the program tasks and student code, where the functionality of these tasks can be checked. Thanks to the remote laboratory implementation of IoT, students' lesson tasks have improved [5].

Programming using microcontrollers is becoming increasingly popular in teaching to help learners gain a deeper understanding of programming principles. Using sensors, motors, and various electronic components with microcontrollers, we can create impressive results in teaching programming, such as movement, flashing, etc. It engages students and gets them interested in programming [15]. We can design applications visually with the help of visual programming, a new trend within coding. The use of visual programming is growing in popularity today. TinkerCad is an excellent tool for visual programming. Using TinkerCad, we can assemble the circuit, write the software code and simulate the results. This application displays the result of each step. An application such as this can be used well during a time of pandemic, when students are being educated online. TinkerCad supports a variety of predefined components, such as Arduino, Raspberry PI, Micro:Bit, and more [4, 11, 13, 14]. The authors of this article describe a method for visualizing programming instructions using the TinkerCad online application.

In [8], the authors detail the development of an approach that provides students with an integrated coursework and laboratory experience. The increased performance and functionality of modern microcontrollers is both an opportunity and a challenge for educators. Increased complexity, the need to integrate hands-on laboratory experience, and declining pressures on curriculum hours require a significant investment of time to modernize microcontroller instruction that few instructors can afford. However, a successful microcontroller course offers a unique opportunity to prepare students for large, complex systems.

In his article, Cubero describes how to direct students to become their own best teachers, able to test their newly acquired skills without receiving minimal or no help from an instructor. They describe “closed-loop” student-centred learning and problem-based learning approaches that include weekly lectures and hands-on laboratory activities that maintain students ’curiosity, motivation, and participation in self-regulated learning. Students must design and test their own original circuits and software code by modifying, extending, or expanding the sample circuits and sample codes described in the lecture notes in order to meet and demonstrate the specific objectives or requirements of each weekly laboratory session. These “closed-loop” student-centred learning labs ensure that all teams of students reach a general or minimum acceptable level of practical skills that appropriately prepares them for the competition of “design and construction”. This learning style also contributes to the development of general lifelong learning skills such as problem research and identification (problem definition and analysis), independent research and experimentation, decision making, communication and teamwork. Even without prior hands-on experience in electronic circuit design, programming, and microcontrollers, all student teams were able to apply and demonstrate new knowledge and skills, design and test original circuit and software designs unsupervised; solve and correct complex problems successfully and confidently; build a workable remote-controlled electric vehicle or mobile robot for the ultimate race. Some went even further and built sensor-controlled, fully autonomous mobile robots [3].

In most articles, the authors suggest using the project method or the problem-solving method in teaching. In their article, Mendoza and his colleagues present the content, teaching, and assessment methods of a mandatory course in the design of microprocessor-based real-time embedded systems in the final years of undergraduate telecommunications engineering. The method used was project-based learning and assessment was based on the skills learned. Finally, the article reflects on how well the course has achieved its objectives using a project-oriented approach [7].

Amiel describes a six-year experiment with his peers based on a project-oriented learning approach to teaching the basics of electronics. The proposed teaching framework has a dynamic structure as it adapts and modifies the conditions for annual assessment to help motivate and interest students. The authors present the effectiveness and value of this approach in terms of student motivation [1].

According to Sari et al., Taking advantage of the benefits of information technology in the digital age of the twenty-first century is increasing in every economy to overcome problems and difficulties and find the solutions you want. So the development of algorithmic thinking is important as a skill that requires the application of knowledge from different disciplines, especially the natural sciences, technology, engineering and mathematics, and improves the solution of real problems. Therefore, practical studies are needed on how to develop algorithmic thinking and what activities and learning contents can be used in classrooms. The impact of STEM-centric physical computing activities with Arduino on teacher candidates 'algorithmic thinking skills and STEM awareness was investigated using mixed-method research. In addition, the student-teacher roles in the activities and the pros and cons of the activities were discussed, taking into account the views of the future teachers. The results showed that STEM-centric physical computing activities improve the algorithmic thinking skills of prospective teachers. Therefore, it can be said that the activities raised the awareness of future teachers about STEM [9].

Angeli describes in his article that those working in science, technology, engineering, and mathematics play a significant role in the sustainable growth and stability of the global economy and thus play a key role in the prosperity of all countries in the world. In this context, computer thinking is an important skill that allows workers to develop creative solutions to complex problems. However, all economies in the world need more workers who are able to think computationally about problems, challenges, and solutions. Therefore, integrating the teaching of computer thinking into secondary and university education is extremely important in order to reduce the skills gap between education and the workplace. An important and crucial question arises as to whether teachers have the knowledge and skills that will teach students to think computationally. Existing research shows that teacher-education classes do not currently have the knowledge to facilitate computer thinking in their programs. This study focuses on two aspects of computational thinking, such as algorithmic thinking and debugging skills, using scaffolding programming scripts in an undergraduate training in educational technology. The results show a statistically significant improvement in learning in the algorithmic thinking and debugging skills of preparatory teachers in the context of LEGO WeDo robot programming activities[2].

3. Results

The use of robots, microcontrollers and IoT (Internet of Thinks) in education can improve students' algorithmic thinking and familiarize them with programming techniques. And the acquisition of tools through the use of real life and science examples supports the complex development of STEM. In the following we will show some of the robots and microcontrollers used in education.

Then, with the results of our research, we confirm that students achieve better results on average when learning programming using visual tools (microcontrollers, TinkerCAD). It can be seen that not all tasks had significant results, but we supported our claims based on these. From the analysis of the results of the 4 task groups, we can conclude that the students achieved significantly better results in the case of the conditional branching (if..else) and the do..while conditional loop instruction task groups. While in the case of the counting loop instruction (for loop instruction) and the while..do conditional loop instruction task groups, there were no significantly better results, although on average the students taught with the help of visual tools performed better here as well.

3.1. LEGO robots

LEGO robots are very useful in education: programming with their help, measurement of various signals (use of sensors), communication between devices, construction of software-controlled mechanical systems (robots) can be learned and taught in a playful and experiential way. However, in addition to the many benefits, it is also important to point out that the transparency of the components is rather low, and due to the integrity, no details are revealed. The goal of the developers of LEGO robots was to make the devices as compact as possible, even without background knowledge.

3.2. Single board computers – microcontrollers

The Raspberry Pi single-sheet computer, originally developed specifically for education, and the Arduino circuit, which is indispensable for most hobbyists, including students and teachers, are also extremely popular. The Arduino is actually a card that has the contacts of a microcontroller used in the industry connected to more easily accessible connectors. For hobbyists and students, the simple development environment and the extremely rich information available on the Internet, as well as the wide range of additional circuits available cheaply, make it very convenient and easy to use.

These devices are already much closer to the technical systems used in practice, the user needs to know more about digital signals, interfaces, electronic solutions, because they encounter them more directly. In many cases, they are quite simple to use and provide very good transparency, which is especially important for education.

In the educational application of Raspberry Pi and Arduino circuits, there are elaborate solutions for almost every task that, while instructive, often encourage more than just copying. Unfortunately, the vast majority of the available knowledge (which can be considered as a curriculum in the case of educational use) was

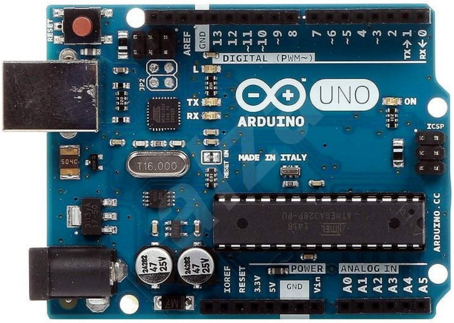


Figure 1. Arduino Uno microcontroller.

produced by professionals with in-depth technical knowledge and attitudes, as well as professional and didactic reliability.

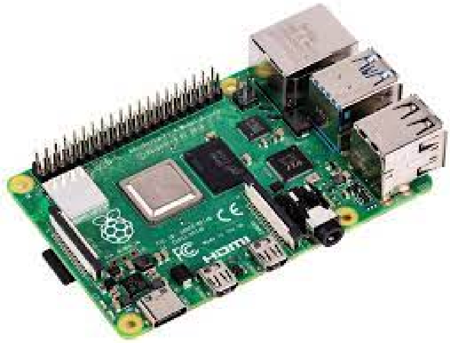


Figure 2. Rasberry PI microcontroller.

3.3. PIC controller

PIC stands for Peripheral Interface Controller. The PIC microcontroller is the smallest microcontroller in the world and is programmed to perform a large number of operations. These were originally designed to support PDP (Programmed Data Processing) computers to control peripheral devices. It is based on RISC architecture.

3.4. Micro:Bit

The BBC Micro:Bit is a small, programmable panel with built-in sensors (compass, accelerometer, light sensor), LED matrix display, I/O connectors, Bluetooth technology. The tool can also be programmed using an easy-to-use graphical block language, similar to the Scratch environment.

There are other educational tools on the market that can be considered robots, which are especially useful in kindergartens and elementary schools to develop algorithmic thinking in education. Such robots, resp. bots include Code&Go, Ozobot, Bee-Bot, etc.

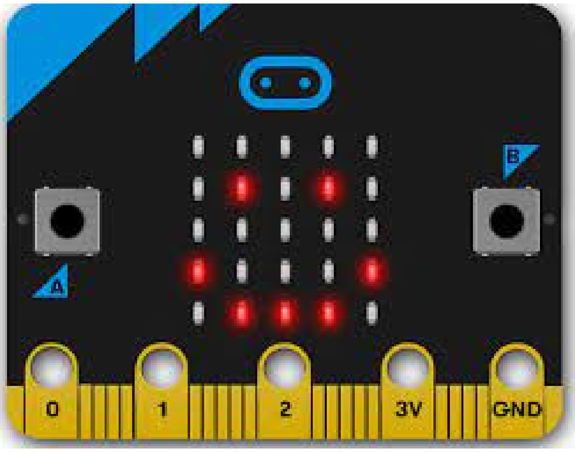


Figure 3. Micro:Bit microcontroller.

3.5. Characteristics of technical IT methods

Practice orientation

The real operation of technical IT teaching tools, the handling of real signals and the creation of effects ensure a practice-orientation. Easy and inexpensive tools help with experimental education, whether it's a teacher demonstration or school or home student experiments. This can improve the retention of interest and provide an opportunity to work together.

Task orientation

With the help of electronic-electrical sensors, teachers can create projects and tasks where students can learn how to use the tools and operating principles of the tools needed to solve the task.

Professionally correct application

Nowadays, technical (electronic) IT tools are used by many people, many people also share application suggestions and educational materials on the Internet, which from a pedagogical point of view may not be suitable for achieving the goal. There are plenty of imaginative and varied solutions to specific problems on the Internet. We must strive to develop critical thinking, the ability to override, the right attitude and demandingness. This requires the acquisition of certain basic professional knowledge, an appropriate level of confidence, and knowledge of the most important operating principles.

Multidisciplinarity

Technical IT methods can be used in almost all science lessons: in addition to IT education, physics class (distance, time, acceleration, pressure, speed measurement, ...), chemistry class (CO, CO2 measurement, ...) and biology class (blood pressure, heart rate, ...).

Transparency

From a pedagogical point of view, it is necessary for the students to understand the structure of the tools and the operating principles as much as possible. This is ensured by the transparency feature. It is usually difficult to find a balance, both superficially and in detail, between problem presentation and solution. This is well influenced by teachers' experience in using the tools directly.

Scalability

The tasks in the lessons should be such that everyone can have a sense of success, everyone can develop, as the skills of the students participating in the lessons in the field of IT can be especially diverse. Most of the time, their interest in the subject is not the same. In the case of technical IT methods, this is quite feasible; the student can solve the given task on many levels, with different additions.

Visual programming is a new trend within programming that allows us to develop applications. Nowadays, visual programming is becoming very popular. TinkerCad online application is very suitable for visual programming. Using the TinkerCad online application, we can assemble our circuit and then simulate the results after writing the program. In fact, the app visualizes the steps taken. The app can also be used during a pandemic period when students are being educated online. TinkerCad can use a lot of predefined tools (parts, sensors) for visualization on different platforms, such as Arduino, Raspberry PI, Micro: Bit,... [1, 2]. Here are some projects to visualize your programming education using the TinkerCad online application:

- Password-protected access
- Distance measurement
- Digital clock
- Temperature monitoring – measurement
- Engine control
- Remote control – Bluetooth, Wifi
- Brightness measurement
- Motion detection
- Line tracking
- Moisture measurement

- Moving lights using LEDs
- Digital sandstone
- Parking system
- RFID identification
- Qcode, barcode reading
- Using the display
- Maze problem

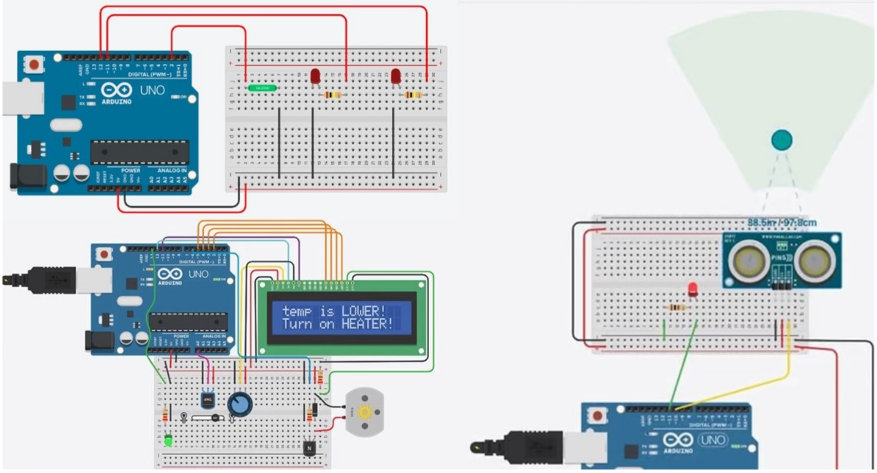


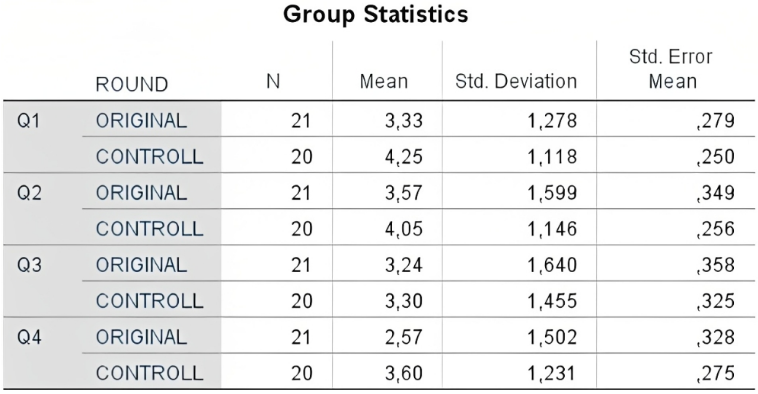
Figure 4. Circuits with Arduino Uno microcontroller created in TinkerCAD.

3.6. Research

We conducted our research in the second year of a secondary school in Slovakia (which is not IT oriented). According to the curriculum, we had 15 hours for teaching programming. The research was carried out in two classes, where only a minimal difference can be observed between students' knowledge level. This difference does not affect the research results. In this research, the results of a programming basics survey taught by an instructor are investigated. In this way, we rule out the possibility that teaching is influenced by multiple teaching methods and personal factors. We would like to examine the results from several perspectives. We would like to draw the appropriate conclusions from the results that can be used in further teaching work. The studied group is composed of 41 students, 21 students were taught the basics of programming using the classical method (ORIGINAL GROUP), while the other 20 students were taught using microcontrollers and TinkerCAD software (CONTROLL GROUP). The aim of the teaching was to learn and correctly use control structures. For both groups, 4 groups of tasks were evaluated in the assessment, which assessed the knowledge of: conditional branching (if..else), counter loop instructions (for loop instructions), conditional loop instructions (while and do..while loop instructions). In all four cases, the maximum score available in the assessment was 5 and the minimum was 0. In our research, we do not consider gender as it is not relevant in our case.

We examined the answers given by the original and the control group in SPSS with a hypothesis test (Independent Samples Test), based on which, on the one hand, the standard deviations of the two groups are the same, so among the tests offered by SPSS, the results obtained along the same standard deviation should be taken as a basis.

Table 1. The averages of the responses to the different questions vary between the two groups.

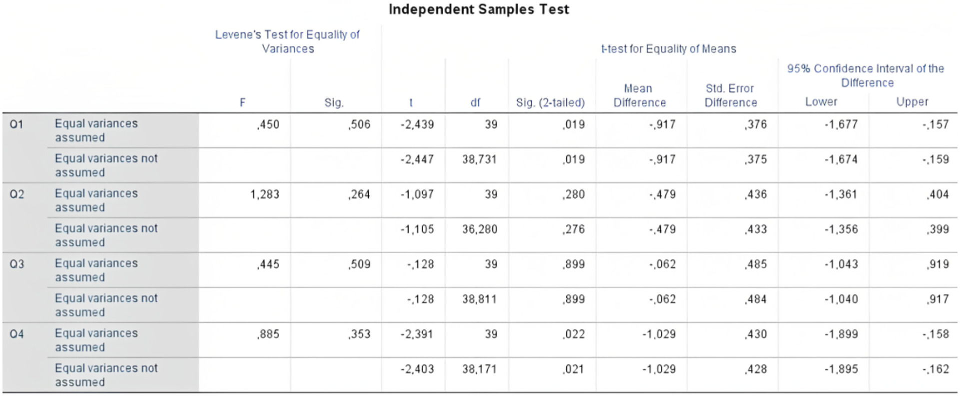


Group Statistics

	ROUND	N	Mean	Std. Deviation	Std. Error Mean
Q1	ORIGINAL	21	3,33	1,278	,279
	CONTROLL	20	4,25	1,118	,250
Q2	ORIGINAL	21	3,57	1,599	,349
	CONTROLL	20	4,05	1,146	,256
Q3	ORIGINAL	21	3,24	1,640	,358
	CONTROLL	20	3,30	1,455	,325
Q4	ORIGINAL	21	2,57	1,502	,328
	CONTROLL	20	3,60	1,231	,275

Table 1 shows how the averages of the responses to the different questions vary between the two groups. Based on this the average responses for the original recording are lower compared to the control group.

Table 2. Independent Sample Test of the responses of the two groups.



Independent Samples Test

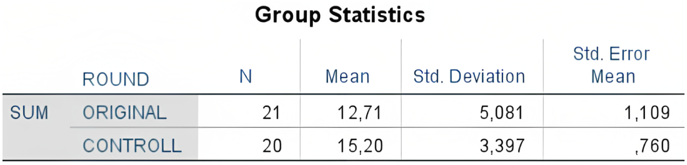
		Levene's Test for Equality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
Q1	Equal variances assumed	,450	,506	-2,439	39	,019	-,917	,376	-1,677	-,157
	Equal variances not assumed			-2,447	38,731	,019	-,917	,375	-1,674	-,159
Q2	Equal variances assumed	1,283	,264	-1,097	39	,280	-,479	,436	-1,361	,404
	Equal variances not assumed			-1,105	36,280	,276	-,479	,433	-1,356	,399
Q3	Equal variances assumed	,445	,509	-,128	39	,899	-,062	,485	-1,043	,919
	Equal variances not assumed			-,128	38,811	,899	-,062	,484	-1,040	,917
Q4	Equal variances assumed	,885	,353	-2,391	39	,022	-1,029	,430	-1,899	-,158
	Equal variances not assumed			-2,403	38,171	,021	-1,029	,428	-1,895	-,162

Table 2 (Independent Sample Test) gives an answer to the similarity/difference of the variances of the two groups and the similarity/difference of the responses of the two groups.

Based on this, since Sig > 5% for all questions, the standard deviation of the responses of the two groups can be considered to be the same for all questions, so the first row should be looked at for all questions when testing similarity/dissimilarity. Since the Sig (2-tailed) > 5% for all questions, the hypothesis H0 is accepted, which states that there is no significant difference between the responses of the two groups.

In addition, we have also considered the case where we do not examine the two groups separately for each question, but on the basis of the sum score (SUM value).

Table 3. The averages of the total response value (SUM) between the two groups.

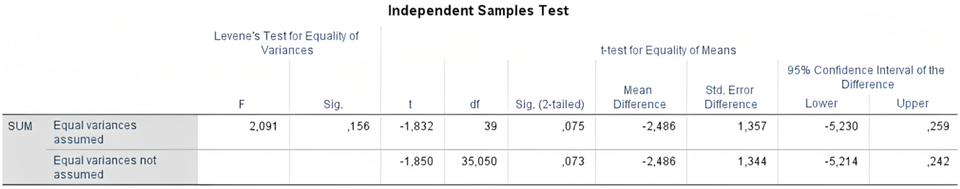


Group Statistics

	ROUND	N	Mean	Std. Deviation	Std. Error Mean
SUM	ORIGINAL	21	12,71	5,081	1,109
	CONTROLL	20	15,20	3,397	,760

Table 3 shows, similar to the first study (when we looked at each question), the control group performed better on average in terms of the total response value (SUM).

Table 4. Independent Sample Test of total response value (SUM).



Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
SUM	Equal variances assumed	2,091	,156	-1,832	39	,075	-2,486	1,357	-5,230	,259
	Equal variances not assumed			-1,850	35,050	,073	-2,486	1,344	-5,214	,242

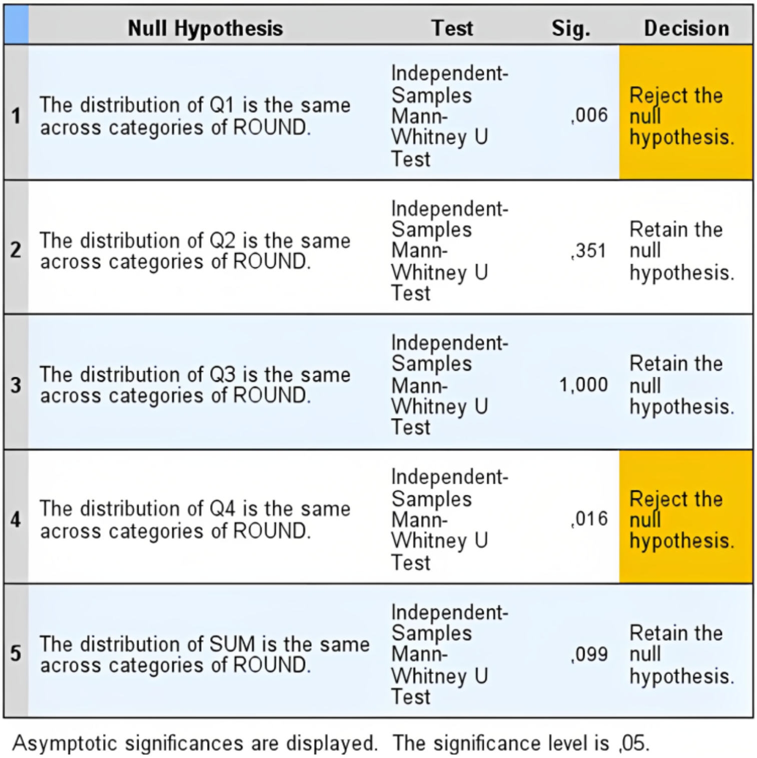
Table 4 shows the responses of both groups (original and controll) which not show significant differences here either (because Sig (2-tailed) > 5%).

These would have been the results if the responses had followed a normal deviation.

However, since the normality condition is not met, it was necessary to continue the tests in a non-parametric direction with a Mann-Whitney test, which does not require a normal deviation of samples.

The main difference between non-parametric and parametric tests (Independent Samples Test) is that they are tested on a median basis instead of a mean basis.

Table 5. Hypothesis Test Summary.



	Null Hypothesis	Test	Sig.	Decision
1	The distribution of Q1 is the same across categories of ROUND.	Independent-Samples Mann-Whitney U Test	,006	Reject the null hypothesis.
2	The distribution of Q2 is the same across categories of ROUND.	Independent-Samples Mann-Whitney U Test	,351	Retain the null hypothesis.
3	The distribution of Q3 is the same across categories of ROUND.	Independent-Samples Mann-Whitney U Test	1,000	Retain the null hypothesis.
4	The distribution of Q4 is the same across categories of ROUND.	Independent-Samples Mann-Whitney U Test	,016	Reject the null hypothesis.
5	The distribution of SUM is the same across categories of ROUND.	Independent-Samples Mann-Whitney U Test	,099	Retain the null hypothesis.

Asymptotic significances are displayed. The significance level is ,05.

Based on results of Table 5, the only significant difference between the two groups' responses is for the first and fourth questions. For the other questions and SUM value, no significant difference is found at the 5% significance level.

4. Conclusion

Most of the devices around us are based on electronic solutions. The results of our research show that students on average achieve better results when using visual tools (microcontrollers, TinkerCAD) to learn programming. It can be seen that there were not significant results for all tasks, but they also support our claims Electronic tools can be used effectively to raise the standard of secondary and university programming education. It has a major impact on the development of technical and computer thinking, without which it will be difficult for students to succeed in today's job market. The market trends that are determining the electronics industry today point in the direction of a sharp increase in the use of IoT devices. With the help of electronics and technical IT methods, the following skills can be effectively developed: algorithmic thinking, technical and computer thinking, problem solving, project team thinking. We can further increase efficiency with visual programming. TinkerCad online application is very suitable for visual programming.

References

[1] F. Amiel, D. Abboud, M. Trocan: *A Project Oriented Learning Experience for Teaching Electronics Fundamentals*, Communications Magazine, IEEE 52 (Dec. 2014), pp. 98–100, doi: https://doi.org/10.1109/MCOM.2014.6979959.

[2] C. Angeli: *The effects of scaffolded programming scripts on pre-service teachers' computational thinking: Developing algorithmic thinking through programming robots*, International Journal of Child-Computer Interaction (June 2021), p. 100329, doi: https://doi.org/10.1016/j.ijcci.2021.100329.

[3] N. S. Cubero: *Fun and effective self-learning approach to teaching microcontrollers and mobile robotics*, International Journal of Electrical Engineering Education 52.4 (2015), pp. 298–319.

[4] M. Fülöp, J. Udvaros, Á. Gubán, Á. Sándor: *Development of Computational Thinking Using Microcontrollers Integrated into OOP (Object-Oriented Programming)*, Sustainability (2022), doi: https://doi.org/10.3390/su14127218.

[5] P. Jacko, M. Bereš, I. Kováčová, J. Molnár, T. Vince, J. Dziak, B. Fecko, Š. Gans, D. Kováč: *Remote IoT Education Laboratory for Microcontrollers Based on the STM32 Chips*, Sensors, MDPI 22.4 (2022), issn: 14248220, doi: https://doi.org/10.3390/s22041440.

[6] D. Loukatos, N. Androulidakis, K. Arvanitis, K. Peppas, E. Chondrogiannis: *Using Open Tools to Transform Retired Equipment into Powerful engineering Education Instruments: A Smart Agri-IoT Control Example*, Electronics, MDPI 11.6 (2022), issn: 20799292, doi: https://doi.org/10.3390/electronics11060855.

[7] J. Pastor, J. M. Villadangos, F. Rodríguez: *Project based learning experiences for embedded systems design*, in: June 2016, pp. 1–6, doi: https://doi.org/10.1109/TAEE.2016.7528370.

[8] R. B. Reese, B. A. Jones: *Improving the Effectiveness of Microcontroller Education*, IEEE Xplore, doi: https://doi.org/10.1109/SECON.2010.5453894.

[9] U. Sari, H. Pektaş, Ö. Şen, H. Çelik: *Algorithmic thinking development through physical computing activities with Arduino in STEM education*, Education and Information Technologies (Jan. 2022), doi: https://doi.org/10.1007/s10639-022-10893-0.

[10] J. Udvaros, K. Czakóová: *Developing Of Computational Thinking Using Microcontrollers And Simulations*, in: EDULEARN21 Proceedings, 13th International Conference on Education and New Learning Technologies, Online Conference: IATED, May 2021, pp. 7945–7951, isbn: 978-84-09-31267-2, doi: https://doi.org/10.21125/edulearn.2021.1619.

[11] J. Udvaros, K. Czakóová: *Using Teaching Methods Based On Visualizing By Tinkercad In Teaching Programming*, in: ICERI2021 Proceedings, 14th annual International Conference of Education, Research and Innovation, Online Conference: IATED, Aug. 2021, pp. 5913–5917, isbn: 978-84-09-34549-6, doi: https://doi.org/10.21125/iceri.2021.1333.

[12] J. Udvaros, O. Takáč: *Developing Computational Thinking By Microcontrollers*, in: ICERI 2020 Proceedings, 13th annual International Conference of Education, Research and Innovation, Online Conference: IATED, Sept. 2020, pp. 6877–6882, isbn: 978-84-09-24232-0, doi: https://doi.org/10.21125/iceri.2020.1474.

[13] J. Udvaros, L. Végh: *New Teaching Methods By Using Microcontrollers In Teaching Programming*, in: eLearning sustainment for never-ending learning, Proceedings of the 16th International Scientific Conference "eLearning and Software for Education", Bucharest: Editura Universitara, Apr. 2020, pp. 630–637, doi: https://doi.org/10.12753/2066-026X-20-082.

[14] J. Udvaros, L. Végh: *Possibilities of Creating Interactive 2D Animations for Education Using HTML5 Canvas JavaScript Libraries*, in: Proceedings of the 16th International Scientific Conference "eLearning and Software for Education", Bucharest: Editura Universitara, Apr. 2020, pp. 269–274, doi: https://doi.org/10.12753/2066-026X-20-119.

[15] J. Udvaros: *Mikrokontrollerek programozásának oktatása TinkerCAD segítségével*, Logisztika – Informatika – Menedzsment 2021 (2021), pp. 22–22.

Annales Mathematicae et Informaticae
57 (2023) pp. 92–106
DOI: https://doi.org/10.33039/ami.2022.11.001
URL: https://ami.uni-eszterhazy.hu

Solving Hungarian natural language processing tasks with multilingual generative models

Zijian Győző Yang, László János Laki

Hungarian Research Centre for Linguistics
{yang.zijian.gyozo,laki.laszlo}@nytud.hu
MTA-PPKE Hungarian Language Technology Research Group
Pázmány Péter Catholic University,
Faculty of Information Technology and Bionics
{yang.zijian.gyozo,laki.laszlo}@itk.ppke.hu

Abstract. Generative ability is a crucial need for artificial intelligence applications, such as chatbots, virtual assistants, machine translation systems etc. In recent years, the transformer-based neural architectures gave a huge boost to generate human-like English texts. In our research we did experiments to create pre-trained generative transformer models for Hungarian language and fine-tune them for multiple types of natural language processing tasks.

In our focus, multilingual models were trained. We have pre-trained a multilingual BART, then fine-tuned it to various NLP tasks, such as text classification, abstractive summarization. In our experiments, we focused on transfer learning techniques to increase the performance. Furthermore, a M2M100 multilingual model was fine-tuned for a 12-lingual Hungarian-Centric machine translation. Last but not least, a Marian NMT based machine translation system was also built from scratch for the 12-lingual Hungarian-Centric machine translation task.

In our results, using the cross-lingual transfer method we could achieve higher performance in all of our tasks. In our machine translation experiment, using our fine-tuned M2M100 model we could outperform the Google Translate, Microsoft Translator and eTranslation.

Keywords: natural language processing, multilingual model, sentiment analysis, abstractive summarization, machine translation, Marian NMT, M2M100

AMS Subject Classification: 68T07, 68T09, 68T50

Submitted: May 24, 2022
Accepted: November 2, 2022
Published online: November 9, 2022

1. Background

Several efforts have been made to analyze the tremendous amount of data that is currently available with the long-term goal to understand and analyze patterns. A highly promising approach towards that direction is the creation of generative models, that can generate new data instances similar to the original dataset. Recent advancements in artificial intelligence promote the development of systems with generative ability.

One aim of this research is to facilitate the work of administrators by processing human language. The members of the consortium that established the Infocommunication and Information Technology National Laboratory (ICT & IT National Laboratory) (the National Security Service and IdomSoft Zrt.) have set a dual goal: to support the safe introduction and use of emerging infocommunication and information technologies and the digital transformation of public administration.

One of IdomSoft LLC.'s[1] key objectives is to research and apply the potential of Artificial Intelligence (AI) based technologies for public administration applications, enabling customers to be exempted from the provision of all data already available in public administrations. The developments will save customers from all the organisational and administrative tasks that can be solved by internal administrative organisation between public administrations. The aim is also to create a secure and seamless contactless, fully digitised and automated administration.

This strategic innovation includes, among other things, the feasibility of public administration services that can handle the specificities of the Hungarian language at a high level of proficiency and meet the expectations of the 21st century. In order to achieve these objectives, IdomSoft LLC. cooperates with Hungarian universities to apply their products, which have been implemented in the R&D process, in practice in connection with the public administration IT solutions it develops.

Neural Machine Translation (NMT) is an important task in the area of Natural Language Processing (NLP), which is clearly highlighted by the fact that there is an increasing demand from the side of both academic and industrial stakeholders to push the limits of model performance and to come up with new, resource-efficient solutions. It is getting increasingly important to establish multilingual models that are able to handle dozens or even more than hundred languages simultaneously. The implementation of these multilingual models in certain directions and their application to NLP tasks in novel settings can promote the progress of machine translation in medium- or low-resourced languages.

Transfer learning represents a key strategy in enhancing model performance. It offers a solution to exploit the capabilities of a model that is trained for a certain task in order to use this knowledge to tackle other related problems. For example, cross-lingual knowledge transfer can substantially increase abstractive summmarization quality.

Our major research focus is to train multilingual models to NLP tasks followed by fine-tuning to specific tasks like text classification and abstractive summariza-

[1] https://idomsoft.hu

tion. We apply cross-lingual knowledge transfer to investigate how it can enhance model performance in our experimental settings.

Here we report that we could achieve highly superior performance with the models when cross-lingual knowledge transfer was applied. This further confirms that the application of transfer learning principles in NLP tasks can represent an outstanding opportunity to boost model performance and to establish competitive new approaches in the field of multilingual natural language processing.

2. Related work

The BART [16] is a transformer model developed by Fairseq (Facebook AI Research Sequence-to-Sequence Toolkit). The architecture of BART is based on two types of Transformers: the bidirectional encoder and the auto-regressive decoder. BART can be seen as a hybrid of a BERT- [8] and a GPT-type model [24]. The combination of the different features makes BART especially powerful and offers a unique opportunity to apply it for various purposes. For example, BERT models achieve impressive results in word- and sentence-level classification, while GPT models are well-suited for text generation tasks, such as summarization. BART can be applied with high success in machine translation, since it brings together the advantageous properties of both BERT-based and auto-regressive models.

The mBART (multilingual BART) is based on the seq2seq concept and it is a denoising autoencoder model pre-trained on corpora in multiple languages [20]. The application of mBART can significantly enhance the performance of both supervised and unsupervised machine translation, which can be especially promising in the case of translation of low- or medium-resourced languages. The mBART follows a sequence-to-sequence Transformer architecture [33] with 12 encoder and 12 decoder layers completed with an additional normalization layer. The conceptual framework of mBART is based on multilingual pre-training followed by fine-tuning to given language pairs. To pre-train the model, the CC25 corpus was applied [7] [15], which is a dataset consisting of 25 languages from different families. The texts were extracted from the CommonCrawl database and went through tokenization as a pre-processing step. The application of mBART could significantly improve the quality of both sentence-level and document-level machine translation, for example, in the case of low resource language pairs like English-Vietnamese or English-Turkish, more than 12 BLEU gains could be reached. On the contrary, for high resource language pairs, this performance gain was not observable or even resulted in a slightly worse performance. The results acquired by seq2seq-based approaches represent a significant improvement in the area of machine translation in comparison to previous efforts [20] [17]. The mBART was later expanded to mBART50 by incorporating additional 25 languages in the pipeline (doubling the number of the included languages), which resulted in remarkable BLEU improvements (up to 15 BLEU improvement in the case of some low resource languages) [30]. Taken together, the performance enhancement observed using mBART models suggests that there is transfer learning potential from the representations acquired during multilingual pre-training. The mBART does not contains Hungarian language knowledge, thus we have pre-trained own English-Hungarian bilingual BART models.

Cross-lingual knowledge transfer can significantly improve model performance. For instance, Kahla et al. pre-trained a multilingual BERT model on a Hungarian corpus, then fine-tuned for abstractive summarization in Arabic. Similarly, the learned representations from pre-training on English corpus were transferred to Arabic in an attempt to improve the quality of summarization. The results indicate that it is possible to significantly elevate the quality of abstractive summarization by applying multilingual models pre-trained on a given language and transfer the acquired knowledge to another language [14]. The work by Artetxe et al. revealed important insights into the generalization ability of multilingual models and found that these models could achieve outstanding results on cross-lingual transfer benchmarks [2]. Additionally, cross-lingual knowledge transfer has been applied successfully in a variety of different areas, such as temporal expression extraction [6], name entity recognition [11], and utterance interpretation [26].

In machine translation research, there are only a few examples of multilingual models that can translate from any languages to Hungarian and vice versa. For research purposes, M2M100 [10] contains many languages including Hungarian, but it is an English-Centric model and it cannot translate from different languages to Hungarian. Among the industrial solutions, there are some multilingual translation systems, for instance Google Translate, Microsoft Translator or eTranslation, which use multilingual or bilingual models to translate from different languages to Hungarian.

The M2M100 project aimed at developing a translation tool comprising 103 different languages and 204 translation directions. A key proposition of the project was to initiate a paradigm shift in machine translation from English-Centric approaches towards multilingual model-based solutions [1]. Machine translation from multiple languages to multiple languages requires large datasets. This gave rise to a series of improvements in the generation of repositories with large data volume, including data mining [3] and reverse translation [28]. Hungarian translation capability is covered in M2M100, therefore it can be exploited in our projects as well.

The Marian NMT framework [13] is written in C++ language, which is an easy to install and well-annotated machine translation tool. Furthermore, its efficiency regarding memory usage and resource requirements makes it especially competent. Additionally, its minimal dependency on other technical solutions facilitates its application on a wider scale [12]. Due to its highly advantageous characteristics, Marian NMT is the most commonly used machine translation tool by academic users and developers [4]. Marian NMT operates using an attention model supported by an encoder-decoder architecture. Marian NMT is based on a neural machine translation model and it can reach the fastest runtime learning without the use of pre-training. In our experiment, a Marian large model was trained with the following specifications: 6 encoder layers and 6 decoder layers; 16 heads of attention; words embedding dimension: 1024; input length: 1024 token; pre-attached mesh size: 4096.

The Google Translate [35] was launched in 2003. During the first phase, its operating principle was restricted to statistical machine translation, which was superseded by neural network-based machine translation in 2016. The quality of the translation has been significantly improved with the introduction of the neural network-based approach. This largely affected the performance in terms of inferences on a broader context and consequently more authentic translations. The Google Translate provides a record of results with several types of different translated versions, for example in the case of languages with gender distinction (e.g. French or Spanish), the feminine version is listed first followed by the masculine version [25]. Google Translate has the ability to handle 109 different languages with the add-on feature of translating spoken texts since 2020.

The Bing Translator is a machine translation solution developed by Microsoft Cognitive Services. It is capable of translating texts in more than 100 different languages and it even provides a solution for translating entire documents. Initially, it applied statistical machine translation, which was replaced by a neural network-based approach in 2018. Xu Tan et al. have developed a tool [29] to overcome the difference in the accuracy between multilingual and monolingual models, which is based on the knowledge distillation principle [5]. The core principle behind knowledge distillation is to increase efficiency and model performance by designating a ‘student model’, that can achieve the performance of a ‘teacher model’ or a set of models. The way this concept is implemented to machine translation means that there are language pair-specific teacher models that are used to train the student model that acquire the capability of handling all the languages by the teacher models. The effectiveness of this methodology is represented by its advanced performance in translation of TED talk transcripts from 44 languages to English, during which a BLEU-score improvement of 1 or even higher was achieved [29].

eTranslation[2] is an automated translation solution that can be applied to translate texts or entire documents written in any of the official languages of the Member States of the European Union, as well as Icelandic, Norwegian, Russian and simplified Chinese. The aim of the European Commission with the launch of eTranslation was to support small and medium-sized companies in the European Union, moreover to facilitate the interaction between public service providers, administrative officials and SMEs. The eTranslation tool can be especially useful, when translation capability is required during administrative and bureaucratic tasks. It is important to highlight that it can be easily integrated with other supporting digital solutions. To further support the machine translation procedure, several processing steps and text filtering options are also available under the CEF eTranslation Building Block project. A good example of that is the built-in option, which first divides long sentences into smaller parts before translation, which are later reconstructed to a coherent text. The eTranslation system has been trained on texts with subject-

[2] https://ec.europa.eu/info/resources-partners/machine-translation-public-administrations-etranslation_en

specific content, such as tenders, legal and medical texts, etc. The model has been trained in 24 different languages on more than 1 billion sentences.

3. Corpora

In order to train our bilingual BART models, two different corpora were used: Hungarian and English Wikipedia. In Table 1, you can see the characteristics of the two corpora.

Table 1. Characteristics of the pre-training corpora for BART.

	Segment	Token	Type	Paragraph sentence # (median)	Paragraph token # (median)
English WikiText-103	707,391	96,534,563	596,820	5	125
Hungarian Wikipedia	1,098,156	90,349.849	3,137,980	4	69

For fine-tuning our BART models to sentiment analysis task, we used the Hungarian Twitter Sentiment Corpus[3] that is created by Precognox[4]. According to the international benchmarks [34] we created two subcorpora from this corpus:

- 2-class (HTS2): binary classification subcorpus. We have converted the scores 1 and 2 to 0 as negative, scores 4 and 5 to 1 as positive. Score 3 was ignored to avoid the ambiguities. Training corpus: 2,468 segments. Test corpus: 269 segments.
- 5-class (HTS5): original five-point likert scaled corpus. 1: very negative, 2: negative, 3: neutral, 4: positive, 5: very positive. Training corpus: 3,600 segments. Test corpus: 400 segments.

For the zeroshot and transfer sentiment analysis experiments, we used the SST2 and SST5 corpora from GLUE [34] benchmark.

For the summarization task, we used the H+I corpus that Yang et al. used in their research [36], NOL (Népszabadság online corpus; nol.hu online articles (art) and its' leads from 1999 to 2016) and MARCELL [32] (law documents (doc) and its' one line descriptions (desc) from 1991 to 2019) corpora. Table 2 shows the characteristics of the fine-tuning corpora. For the zeroshot and transfer summarization experiments, we used the CNN/Daily Mail [27] corpora.

In our machine translation task, we built Hungarian-Centric translation models with 12 languages, which means the source text can be in 12 different languages and the target language is Hungarian (hu) in all cases. The 12 different source languages are the following:

[3] http://opendata.hu/dataset/hungarian-twitter-sentiment-corpus
[4] https://www.precognox.com

Table 2. Characteristics of the fine-tuning corpora.

	Segment	Token #	Type #	Avg. token #
HTS2	2,737	42,797	13,713	15.62
HTS5	4,000	59,997	18,423	14.99
H+I	559,162	147,099,485 (art) 16,699,600 (lead)	2,949,173 (art) 749,586 (lead)	263.07 (art) 29.87 (lead)
NOL	397,343	153,003,164 (art) 15,786,166 (lead)	2,482,398 (art) 623,445 (lead)	384.52 (art) 39.71 (lead)
MARCELL	24,747	27,834,358 (doc) 277,732 (desc)	444,352 (doc) 29,189 (desc)	1124.82 (doc) 11.59 (desc)

- Bulgarian (bg), Czech (cs), German (de), English (en), Croatian (hr), Polish (pl), Romanian (ro), Russian (ru), Slovak (sk), Slovene (sl), Serbian (sr), Ukrainian (uk)

In order to build a machine translation from scratch, a huge amount of data is required. In contrast, for fine-tuning task, smaller amount of data is enough. Thus, we created two corpora for our task. First one contains 8 million (8M) segments per language (except for Ukrainian, due to lack of data it contains only 5,805,144 segments), the second one is a sub-corpus of the 8M corpus that contains 3 million (3M) segments for each language. The data were collected from OPUS [31] that is composed of the following sub-corpora:

- Bible, Bible-uedin, Books, CCAligned, CCMatrix, DGT, ECB, ELITR, ELITR-ECA, ELRC_2922, ELRC_2923, ELRC_3382, EMEA, EUbookshop, EUconst, Europarl, GNOME, GlobalVoices, JRC, JRC-Acquis, KDE4, KDEdoc, MultiCCAligned, MultiParaCrawl, OpenSubtitles, PHP, ParaCrawl, QED, TED2020, Tatoeba, TildeMODEL, Ubuntu, WMT-News, WikiMatrix, Wikimedia, Wikipedia, XLEnt

The different language pairs contain different composite of the sub-corpora. In Table 3, you can see the characteristics of the training sub-corpora for the machine translation task.

4. Experiments

In our pre-training experiment, we have trained two bilingual BART models of different size:

- **BART-base**: base size BART model trained on English and Hungarian Wikipedia. Main hyper-parameters: 6 encoder layers and 6 decoder layers; 12 attention heads; word embedding dimensions: 768; input length: 512; 140 million parameters.

Table 3. Characteristics of the machine translation corpora.

	Token	Type	Avg. token / sent
	8M / 3M	8M / 3M	8M / 3M
bg	101,701,016 / 38,149,260	998,060 / 586,926	12.71 / 12.72
hu	93,370,875 / 35,023,413	1,843,452 / 1,057,434	11.67 / 11.68
cs	96,854,637 / 36,345,169	1,369,081 / 797,557	12.11 / 12.12
hu	96,313,811 / 36,125,748	2,008,769 / 1,141,009	12.04 / 12.04
de	123,826,131 / 46,407,141	1,708,615 /957,634	15.48 / 15.47
hu	113,026,306 / 42,365,265	2,215,093 / 1,267,205	14.13 / 14.12
en	118,593,896 / 44,440,629	1,112,914 / 593,035	14.82 / 14.81
hu	104,287,145 / 39,072,921	2,375,910 / 1,331,924	13.04 / 13.02
hr	78,932,860 / 29,601,947	1,075,070 / 631,246	9.87 / 9.87
hu	78,540,254 / 29,445,821	1,685,025 / 961,367	9.82 / 9.82
pl	97,533,671 / 36,584,480	1,350,775 / 793,299	12.19 / 12.20
hu	98,984,434 / 37,126,013	2,062,157 / 1,166,764	12.37 / 12.38
ro	110,276,300 / 41,357,056	952,906 / 555,642	13.79 / 13.79
hu	93,431,714 / 35,058,265	1,906,878 / 1,091,748	11.68 / 11.69
ru	88,227,629 / 33,085,548	1,376,699 / 807,518	11.03 / 11.03
hu	85,205,960 / 31,956,481	1,838,741 / 1,049,578	10.65 / 10.65
sk	122,935,150 / 46,085,577	1,567,148 / 920,586	15.37 / 15.36
hu	123,016,834 / 46,105,105	2,225,916 / 1,278,686	15.38 / 15.37
sl	106,838,393 / 40,042,349	1,195,476 / 703,052	13.36 / 13.35
hu	106,714,770 / 40,013,573	1,973,244 / 1,138,862	13.34 / 13.34
sr	72,647,210 / 27,237,077	1,185,523 / 710,495	9.08 / 9.08
hu	71,058,803 / 26,642,218	1,446,568 / 832,887	8.88 / 8.88
uk	70,816,656 / 36,581,363	1,306,774 / 927,544	12.20 / 12.19
hu	69,564,268 / 35,933,267	1,556,554 / 1,088,340	11.98 / 11.98

- **BART-large**: large size BART model that trained on English and Hungarian Wikipedia. Main hyper-parameters: 12 encoder layers and 12 decoder layers; 16 attention heads; word embedding dimensions: 1024; input length: 1024; 400 million parameters.

In our fine-tuning experiments, we performed three different tasks:

1. **Sequence classification**: Using our pre-trained bilingual BART models and two multilingual BERT-based models (mBERT [9] and XLM-RoBERTa [19]), we carried out three different experiments in sentence-level sentiment analysis:

    - *baseline*: We fine-tuned and tested the four models on HTS2 and HTS5.
    - *zeroshot*: We fine-tuned the four models on SST2 and SST5, then tested on HTS2 and HTS5.

  - *transfer*: We fine-tuned the four models on SST2 and SST5, then further fine-tuned on HTS2 and HTS5, finally tested on HTS2 and HTS5.

2. **Text summarization**: We fine-tuned the BART base model on three different corpora: H+I, NOL and MARCELL. Because of hardware limits, we could not fine-tuned our BART large model on summarization task. We carried out two different experiments in text summarization task:

   - *baseline*: We fine-tuned and tested our model on the three corpora.
   - *transfer (tf)*: We fine-tuned our model on CNN/Daily Mail, then further fine-tuned and tested on the three Hungarian corpora.

3. **Machine translation**: We fine-tuned the M2M100 large model (facebook/m2m100_1.2B[5]) on the 3M sub-corpus for machine translation. The source text can be in 12 different languages, the target text is Hungarian. In this experiment, we fine-tuned our model with only 1 epoch.

   - *From scratch*: In the case of machine translation, we also trained a multilingual translation model from scratch. For this task, we used the Marian NMT [13] framework. For training Marian NMT model, we used the 8M corpus for machine translation. Similar to the M2M100 experiment, the source text can be in 12 different languages, the target text is Hungarian. To help the translation model, we inserted the language code at the beginning of the source segments in the following format (lang is the ISO language code): ___lang___. A Marian large model was trained with 66 epoch.

5. Results

In order to evaluate our experiments, the following metrics were used:

- Accuracy: In the case of sentiment analysis tasks, accuracy metrics were used.
- ROUGE [18] : For summarization tasks, we used the ROUGE metrics in the following format: ROUGE-1/ROUGE-2/ROUGE-L.
- BLEU [21], chrF [22]: For word-level and character-level evaluation of machine translation, SacreBLEU [23] and chrF-6 metrics were used in the following format: BLEU/chrF-6.

In Table 4, you can see the results of the sentiment analysis experiments. For transfer and zeroshot tasks, first, we fine-tuned the models on the English SST corpora. Under the double line, you can see the results of the SST fine-tuning. Above the double line, you can see the results of our experiments. In all cases, the transfer task could increase the result of the models. It can prove that adding relevant data to model could increase performance, even if it is in another language.

[5] https://huggingface.co/facebook/m2m100_1.2B

Table 4. Sentiment analysis results.

	HTS2	HTS5
BART-base (baseline)	74.44	56.75
BART-base (zeroshot)	42.96	28.75
BART-base (transfer)	**74.81**	**57.25**
BART-large (baseline)	74.07	56.00
BART-large (zeroshot)	44.81	23.50
BART-large (transfer)	**74.59**	**56.74**
mBERT (baseline)	78.51	57.74
mBERT (zeroshot)	47.41	30.50
mBERT (transfer)	**80.37**	**57.99**
XLM (baseline)	83.33	63.49
XLM (zeroshot)	68.88	40.99
XLM (transfer)	**84.81**	**79.79**
	SST2	SST5
BART-base	79.01	36.72
BART-large	80.27	36.36
mBERT	90.59	49.57
XLM	93.34	50.43

In Table 5, you can see the results of the summarization task. Similar to the classification task, under the double line, you can see the result of the fine-tuning on the English CNN/Daily Mail corpora. Above the double line, you can see our experiment. As you can see in the Table 5, transfer method in this case could also increase the performance.

Table 5. Abstractive summarization results.

	H+I	NOL	MARCELL
BART-base (baseline)	31.4/14.3/23.5	42.7/27.6/35.4	71.5/63.0/69.9
BART-base-tf	**31.8/14.5/23.5**	**45.1/30.5/37.6**	**77.1/70.6/76.0**
	CNN/Daily Mail		
BART-base	40.1/17.6/27.4		
BART en original	44.2/21.3/40.9		

In Table 6, you can see the results of the machine translation experiments. We have compared our Marian and M2M100 models with Google Translate, Microsoft Translator and eTranslation (the eTranslation cannot translate serbian, thus this results is missing).

The M2M100 fine-tuning results in significantly higher scores then any other tools included in our experimental analysis. Compared to Marian, M2M100 uses only 3 million segments for each language, and only 1 epoch for fine-tuning. It means, the model could transfer significant amount of knowledge from the pre-

Table 6. Comparision of performance of different machine translation models.

	Marian	**M2M100**	Google	Microsoft	eTranslation
bg	21.3/43.9	**26.6/48.0**	20.0/43.6	20.8/44.2	22.3/45.6
cs	22.5/46.0	**28.9/50.3**	22.6/45.3	23.1/45.9	24.7/47.4
de	21.9/46.2	**28.3/51.4**	22.7/48.0	22.8/47.8	24.0/48.8
en	27.7/49.6	**34.4/54.7**	25.3/49.1	26.3/50.3	28.3/51.3
hr	19.2/42.7	**26.2/47.3**	19.6/42.5	20.1/43.1	20.9/43.7
pl	21.2/45.2	**28.3/50.2**	21.4/45.4	22.2/45.7	23.9/47.2
ro	19.5/43.8	**26.4/48.7**	21.0/44.9	21.8/45.7	23.6/46.9
ru	19.7/43.9	**25.1/48.1**	19.8/44.5	21.0/45.6	20.3/44.8
sk	23.1/48.9	**30.9/53.9**	22.6/47.7	23.1/48.5	26.4/50.8
sl	22.7/45.8	**27.7/50.5**	14.4/34.4	21.5/45.0	26.0/48.4
sr	18.0/40.5	**23.4/44.7**	18.0/40.7	19.2/41.5	-
uk	24.2/49.8	**32.6/55.2**	21.8/47.0	23.3/48.2	22.9/48.5
avg	21.8/45.5	**28.2/50.3**	20.8/44.4	22.1/46.0	23.9/47.6

trained 100 language. Thus, less amount data and training steps are enough to achieve higher results. Our Marian experiment used 2.5x larger corpora and 66x more epoch and still gained lower performance than our fine-tuned M2M100 model, but still better than the Google Translate, for instance. Our Marian model could not outperformed the eTranslation, which is not surprising, because the eTranslation uses different bilingual models to translate, and a bilingual model is more accurate than a 12-lingual model. Therefore, our M2M100 model is an outstanding result, because it uses only one model that can gain better results than the bilingual models.

6. Conclusion

In our research, we pre-trained and fine-tuned different transformer-based multilingual generative models for Hungarian natural language processing tasks. We have carried out four different experiments. For pre-training language model, encoder-decoder autoregressive BART models were applied. As classification task, we fine-tuned four different models for sentiment analysis. For summarization task, our pre-trained BART base model was fine-tuned on three different corpora. We also did experiments in zero-shot and cross-lingual transfer learning settings. Last but not least, we built the first (two at once) 12-lingual Hungarian-Centric machine translation model, which uses only one model to translate from 12 languages to Hungarian. In this task, we trained a model from scratch and the M2M100 model was fine-tuned. Our fine-tuned M2M100 used much less data and training steps and yet, it could outperform the Google Translate, the Microsoft Translator and the eTranslation.

Acknowledgements. The research reported in the current publication was carried out by affiliated members of the Pázmány Péter Catholic University and the IdomSoft Ltd, and it was supported by the Ministry of Innovation and Technology and the National Research, Development and Innovation Office within the framework of the National Laboratory of Infocommunication and Information Technology.

References

[1] R. Aharoni, M. Johnson, O. Firat: *Massively Multilingual Neural Machine Translation*, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 3874–3884, doi: https://doi.org/10.18653/v1/N19-1388.

[2] M. Artetxe, S. Ruder, D. Yogatama: *On the Cross-lingual Transferability of Monolingual Representations*, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2020, doi: https://doi.org/10.18653/v1/2020.acl-main.421.

[3] M. Artetxe, H. Schwenk: *Massively Multilingual Sentence Embeddings for Zero-Shot Cross-Lingual Transfer and Beyond*, Transactions of the Association for Computational Linguistics 7 (2019), pp. 597–610, doi: https://doi.org/10.1162/tacl_a_00288.

[4] L. Barrault, O. Bojar, M. R. Costa-jussà, C. Federmann, M. Fishel, Y. Graham, B. Haddow, M. Huck, P. Koehn, S. Malmasi, C. Monz, M. MĂźller, S. Pal, M. Post, M. Zampieri: *Findings of the 2019 Conference on Machine Translation (WMT19)*, in: Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1), Florence, Italy: Association for Computational Linguistics, 2019, pp. 1–61.

[5] C. Bucila, R. Caruana, A. Niculescu-Mizil: *Model Compression*, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '06, Philadelphia, PA, USA: Association for Computing Machinery, 2006, pp. 535–541, isbn: 1595933395.

[6] Y. Cao, W. Groves, T. K. Saha, J. R. Tetreault, A. Jaimes, H. Peng, P. S. Yu: *XLTime: A Cross-Lingual Knowledge Transfer Framework for Temporal Expression Extraction*, in: arXiv, 2022, doi: https://doi.org/10.48550/ARXIV.2205.01757.

[7] P.-J. Chen, J. Shen, M. Le, V. Chaudhary, A. El-Kishky, G. Wenzek, M. Ott, M. Ranzato: *Facebook AI's WAT19 Myanmar-English Translation Task Submission*, 2019, doi: https://doi.org/10.48550/ARXIV.1910.06848.

[8] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova: *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186, doi: https://doi.org/10.18653/v1/N19-1423.

[9] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova: *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186.

[10] A. Fan, S. Bhosale, H. Schwenk, Z. Ma, A. El-Kishky, S. Goyal, M. Baines, O. Çelebi, G. Wenzek, V. Chaudhary, N. Goyal, T. Birch, V. Liptchinsky, S. Edunov, E. Grave, M. Auli, A. Joulin: *Beyond English-Centric Multilingual Machine Translation*, ArXiv abs/2010.11125 (2020).

[11] X. Feng, X. Feng, B. Qin, Z. Feng, T. Liu: *Improving Low Resource Named Entity Recognition using Cross-lingual Knowledge Transfer*, in: July 2018, pp. 4071–4077, doi: https://doi.org/10.24963/ijcai.2018/566.

[12] M. Junczys-Dowmunt, R. Grundkiewicz, T. Dwojak, H. Hoang, K. Heafield, T. Neckermann, F. Seide, U. Germann, A. F. Aji, N. Bogoychev, A. F. T. Martins, A. Birch: *Marian: Fast Neural Machine Translation in C++*, in: Proceedings of ACL 2018, System Demonstrations, Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 116–121, doi: https://doi.org/10.18653/v1/P18-4020, url: https://aclanthology.org/P18-4020.

[13] M. Junczys-Dowmunt, R. Grundkiewicz, T. Dwojak, H. Hoang, K. Heafield, T. Neckermann, F. Seide, U. Germann, A. Fikri Aji, N. Bogoychev, A. F. T. Martins, A. Birch: *Marian: Fast Neural Machine Translation in C++*, in: Proceedings of ACL 2018, System Demonstrations, Melbourne, Australia: Association for Computational Linguistics, 2018, pp. 116–121.

[14] M. Kahla, Z. G. Yang, A. Novák: *Cross-lingual Fine-tuning for Abstractive Arabic Text Summarization*, in: Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021), Held Online: INCOMA Ltd., Sept. 2021, pp. 655–663.

[15] G. Lample, A. Conneau: *Cross-lingual Language Model Pretraining*, 2019, doi: https://doi.org/10.48550/ARXIV.1901.07291.

[16] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, L. Zettlemoyer: *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Online: Association for Computational Linguistics, July 2020, pp. 7871–7880, doi: https://doi.org/10.18653/v1/2020.acl-main.703.

[17] X. Li, G. Li, L. Liu, M. Meng, S. Shi: *On the Word Alignment from Neural Machine Translation*, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy: Association for Computational Linguistics, July 2019, pp. 1293–1303, doi: https://doi.org/10.18653/v1/P19-1124.

[18] C.-Y. Lin: *ROUGE: A Package for Automatic Evaluation of Summaries*, in: Text Summarization Branches Out, Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81.

[19] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov: *RoBERTa: A Robustly Optimized BERT Pretraining Approach*, CoRR (2019).

[20] L. Miculicich, D. Ram, N. Pappas, J. Henderson: *Document-Level Neural Machine Translation with Hierarchical Attention Networks*, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 2947–2954, doi: https://doi.org/10.18653/v1/D18-1325.

[21] K. Papineni, S. Roukos, T. Ward, W.-J. Zhu: *Bleu: a Method for Automatic Evaluation of Machine Translation*, in: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318, doi: https://doi.org/10.3115/1073083.1073135.

[22] M. Popović: *chrF: character n-gram F-score for automatic MT evaluation*, in: Proceedings of the Tenth Workshop on Statistical Machine Translation, Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 392–395, doi: https://doi.org/10.18653/v1/W15-3049.

[23] M. Post: *A Call for Clarity in Reporting BLEU Scores*, in: Proceedings of the Third Conference on Machine Translation: Research Papers, Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 186–191, doi: https://doi.org/10.18653/v1/W18-6319.

[24] A. Radford, K. Narasimhan: *Improving Language Understanding by Generative Pre-Training*, in: 2018.

[25] A. A. Rescigno, J. Monti, A. Way, E. Vanmassenhove: *A Case Study of Natural Gender Phenomena in Translation: A Comparison of Google Translate, Bing Microsoft Translator and DeepL for English to Italian, French and Spanish*, in: Workshop on the Impact of Machine Translation (iMpacT 2020), Virtual: Association for Machine Translation in the Americas, Oct. 2020, pp. 62–90.

[26] S. Schuster, S. Gupta, R. Shah, M. Lewis: *Cross-lingual Transfer Learning for Multilingual Task Oriented Dialog*, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 3795–3805, doi: https://doi.org/10.18653/v1/N19-1380.

[27] A. See, P. J. Liu, C. D. Manning: *Get To The Point: Summarization with Pointer-Generator Networks*, in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 1073–1083, doi: https://doi.org/10.18653/v1/P17-1099.

[28] R. Sennrich, B. Haddow, A. Birch: *Neural Machine Translation of Rare Words with Subword Units*, in: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725, doi: https://doi.org/10.18653/v1/P16-1162, url: https://aclanthology.org/P16-1162.

[29] X. Tan, Y. Ren, D. He, T. Qin, T.-Y. Liu: *Multilingual Neural Machine Translation with Knowledge Distillation*, in: International Conference on Learning Representations, 2019.

[30] Y. Tang, C. Tran, X. Li, P.-J. Chen, N. Goyal, V. Chaudhary, J. Gu, A. Fan: *Multilingual Translation with Extensible Multilingual Pretraining and Finetuning*, 2020, doi: https://doi.org/10.48550/ARXIV.2008.00401.

[31] J. Tiedemann: *Parallel Data, Tools and Interfaces in OPUS*, in: Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), ed. by N. C. (Chair), K. Choukri, T. Declerck, M. U. Dogan, B. Maegaard, J. Mariani, J. Odijk, S. Piperidis, Istanbul, Turkey: European Language Resources Association (ELRA), 2012, isbn: 978-2-9517408-7-7.

[32] T. Váradi, S. Koeva, M. Yamalov, M. Tadić, B. Sass, B. Nitoń, M. Ogrodniczuk, P. Pęzik, V. Barbu Mititelu, R. Ion, E. Irimia, M. Mitrofan, V. Păiș, D. Tufiș, R. Garabík, S. Krek, A. Repar, M. Rihtar, J. Brank: *The MARCELL Legislative Corpus*, English, in: Proceedings of the 12th Language Resources and Evaluation Conference, Marseille, France: European Language Resources Association, May 2020, pp. 3761–3768, isbn: 979-10-95546-34-4.

[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin: *Attention is All you Need*, in: Advances in Neural Information Processing Systems 30, ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett, Curran Associates, Inc., 2017, pp. 5998–6008.

[34] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, S. Bowman: *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*, in: Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 353–355, doi: https://doi.org/10.18653/v1/W18-5446.

[35] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, J. Dean: *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*, CoRR abs/1609.08144 (2016).

[36] Z. G. Yang, Á. Agócs, G. Kusper, T. Váradi: *Abstractive text summarization for Hungarian*, Annales Mathematicae et Informaticae 53 (2021), pp. 299–316.

Annales Mathematicae et Informaticae
57 (2023) pp. 107–123
DOI: https://doi.org/10.33039/ami.2023.03.001
URL: https://ami.uni-eszterhazy.hu

Building machine reading comprehension model from scratch

Zijian Győző Yang, Noémi Ligeti-Nagy

Hungarian Research Centre for Linguistics
{yang.zijian.gyozo,ligeti-nagy.noemi}@nytud.hu

Abstract. In this paper, we introduce a machine reading comprehension model and how we built this model from scratch. Reading comprehension is a crucial requisite for artificial intelligence applications, such as Question-Answering systems, chatbots, virtual assistants etc. Reading comprehension task requires the highest complexity of natural language processing methods. In recent years, the transformer neural architecture could achieve the ability to solve high complexity tasks. To make these applications available in Hungarian as well it is inevitable to design a Hungarian corpus of reading comprehension so that the pretrained models can be fine-tuned on this dataset.

In our research, we have created the HuRC (Hungarian Reading Comprehension) corpus, which is the first dataset in Hungarian aiming to train, test and evaluate language models on a reading comprehension task. We built such a dataset based on the English ReCoRD corpus. This is a dataset of 120,000 examples consisting of news articles containing a passage and a close-style query, where a named entity is masked and the reference answer has to be found in a list.

Using the evaluated dataset and transformers' question-answering library, we have built the first neural machine reading comprehension models in commonsense reasoning task for Hungarian.

1. Introduction

Machine (Reading) Comprehension is the field of NLP where we teach machines to understand and answer questions using unstructured text. Reading comprehension (RC)—in contrast to information retrieval—requires integrating information and reasoning about events, entities, and their relations across a full document. Question answering is conventionally used to assess RC ability.

Submitted: April 8, 2022
Accepted: March 2, 2023
Published online: March 10, 2023

For English, there are many reading comprehension datasets, many of them included in benchmark collections (ReCoRD and MultiRC in SuperGLUE, for example, [24]) or used as a standalone benchmark dataset (SQuAD, [20]). Models trained on these datasets approximate, or sometimes even surpass human performance.

With a slight delay, but the pre-training of the transformer-based architectures on Hungarian data has begun [5, 14]. Some multilingual models, such as XLM-RoBERTa [2] and mT5 [25] also contain Hungarian data. In the future, it is expected that more models will be taught in Hungarian, and it will be necessary to measure and compare the comprehension of these models as well.

On the other hand, we still lack Hungarian datasets to train and test these models. Recently, a Hungarian benchmark kit has been developed [12] containing 4 datasets at the time of submitting this paper. Here we present one of those datasets, HuRC, which is a large-scale, partly automatically, but partly manually annotated dataset aiming to test machine reading comprehension. We trained three different models on the dataset and evaluated their performance on many ways to illustrate the difficulty of this task in Hungarian. Furthermore, using ensemble method, we could combine the advantages of our models to achieve the highest performance.

2. Related work

Current English datasets often frame the task of question answering as reading comprehension: the question is about a paragraph or a document and the answer is a span in the document.

Dzendzik et al. [4] provides a deep summary of English machine reading comprehension (MRC) datasets. Based on the answer type, they differentiate cloze answer (the question is a sentence with a missing word which has to be inserted, e.g. ReCoRD [28]), selective or multiple choice (a number of options is given, and the correct one(s) should be selected, e.g. MultiRC [9]), boolean (a yes/no answer is expected, e.g. BoolQ [1]) extractive or span extractive (the answer is a substring of the passage, e.g. SQuAD [20]) and generative or free form answer (the answer has to be generated, e.g. NarrativeQA [10]).

The DeepMind Q&A datasets [7] consist of documents from news articles from CNN and Daily Mail, 90k and 197k documents with 380k and 879k questions, respectively. News portals have begun to add summary points with each news piece in recent years, apparently to accommodate online readers' short attention spans. These summary points are not simply text extractions from the article, but rather summary points that can be used to automatically create inquiries that may require comprehension of the news story to answer. The query is built by removing an entity from the statement and asking the reader to fill in the most relevant entity from the text. In pre-processing, entities are detected and coreferenced, and the text is completely masked. This is done to avoid the model relying on external knowledge about the entities when deciding on an answer, instead relying only on its understanding of the context.

A collection of children's books was assembled from the Project Gutenberg archives for the Children's Book Test at Facebook [8]. Each question is made up of 20 consecutive sentences from the book text, with the 21st sentence serving as the query statement. A word from the query is selected and masked. The reader has to decide which word from the text (of the chosen kind) should be used to fill the placeholder in the query. Here not merely entities are masked: named entities, common nouns, verbs and prepositions may be placeholders.

StanfordNLP created the SQuAD (Stanford Question Answering Dataset) in 2016 [20], which included over 100,000 question-answer pairs derived from Wikipedia articles. The task was to build a machine learning model to answer questions using a contextual document as input. The model would return the subset of the text most likely to answer the query when given a contextual document (free form text) and a question. The answers do not have to be entities, and no sets of candidate answers are offered. SQuAD is the first large-scale QA dataset in which answers are text spans that must be identified without any extra information. Human annotators achieved an exact match score of 82.304% and a F1-score of 91.221%. No model has been able to surpass the human results on SQuAD for 2 years. In 2018, BERT was introduced [3], and the original BERT model achieved an exact match score of 85.083% and a F1-score of 91.835%.

MultiRC (Multi-Sentence Reading Comprehension) [9] is a dataset of short paragraphs and multi-sentence questions, which are questions that may be solved by combining information from numerous paragraph phrases. The dataset was created with three main objectives in mind: i) for each question, the number of right response possibilities is not pre-determined. This eliminates the model's reliance on answer possibilities and forces them to judge the validity of each answer independently of the others; ii) It is not necessary for the correct answer(s) to be a span in the text; iii) The texts come from a variety of sources, including news, fiction, and historical documents, thus ensuring diversity across domains.

BoolQ contains 15942 examples with naturally occurring questions [1]. Each example consists of a question, a passage and an answer. The authors sampled questions from a distribution of information-seeking queries. They assume this method results in significantly more challenging examples compared to existing datasets where the text pairs (the questions or the answers) were constructed by annotators.

Kočiský et al. [10] states that existing RC datasets do not test the essential integrative aspect of reading comprehension as their questions can be solved relying upon superficial information, such as local context similarity or global term frequency. They present a novel dataset to tackle this problem. In these tasks the reader must answer questions about stories by reading entire books or movie scripts. A successful answer requires understanding the underlying narrative. There are two tasks proposed in the paper: "summaries only" and "stories only", depending on whether the human-generated summary or the full story text is used to answer the question. NarrativeQA still proves to be challenging for language models: the

SOTA result is that of Masque [15]: a Rouge score of 59.87.

Zhang et al. [28] extracted their examples (more than 120 000 entries) from the CNN/Daily Mail[1] corpus to create the Reading Comprehension with Commonsense Reasoning (ReCoRD) dataset. These news articles were divided into multiple units: passage, cloze-style query (containing the masked entity) and the reference answer. The last paragraph must contain the reference answer, a proper noun which can be found in the passage. As a reading comprehension task, this named entity is masked and the model must predict the masked entity from a list of possible entities in the provided passage, where the same entity may be expressed with multiple different surface forms, which are all correct. ReCoRD is part of the SuperGLUE benchmark [24]. The results are evaluated with max (over all mentions) token-level F1 and accuracy. The best result so far on the ReCoRD dataset is an F-score of 96.4% and an accuracy of 95.9% of the Turing NLR v5 model submitted in 2021.

Most recently, ESTER was introduced [6], which is an MRC dataset for Event Semantic Relation Reasoning. The dataset contains natural language queries to reason about the five most common event semantic relations. The current SOTA systems achieve 22.1%, 63.3%, and 83.5% for token-based exact-match, F1, and event-based HIT@1 scores, which are all significantly below human performances (36.0%, 79.6%, 100% respectively).

Natural language processing has seen spectacular progress with the application of neural network technology, in particular, the Transformer model [23]. Tasks like machine reading comprehension, can be solved with high performance, if a pre-trained language model is fine-tuned. The first breakthrough model based on transformer architecture was the BERT (abbreviation of Bidirectional Encoder Representations from Transformer) model [3]. The BERT model is pre-trained on two language modeling tasks: word masking and next sentence prediction. The first native BERT model in Hungarian was published by Nemeskey [14], named as huBERT, which is the state of the art neural language model for Hungarian.

Cross-Language Understanding (XLU) is key challenge and serves as an accelerator to the development of multilingual models. In 2020, the Facebook AI team published an article presenting XLM-RoBERTa (abbreviated as XLM-R as well) [2], which is a transformer-based multilingual masked language model. XLM-R outperforms mBERT (multilingual BERT) on cross-lingual classification in the case of languages with moderate resources available. XLM-R contains Hungarian language knowledge.

T5 (Text-To-Text Transfer Transformer) [19] is a model and framework developed by the Google research team, which offers a new perspective to solve natural language processing tasks. The T5 project applies transfer learning principles in the context of the sequence-to-sequence approach. The initial idea was that all language processing tasks (translation, question answering, classification) should be considered as a text-to-text issue, therefore the input is a text and the output will be another text. mT5 [25] extends the T5 to several languages that including Hungarian. In our research, huBERT, XLM-R and mT5 models were fine-tuned

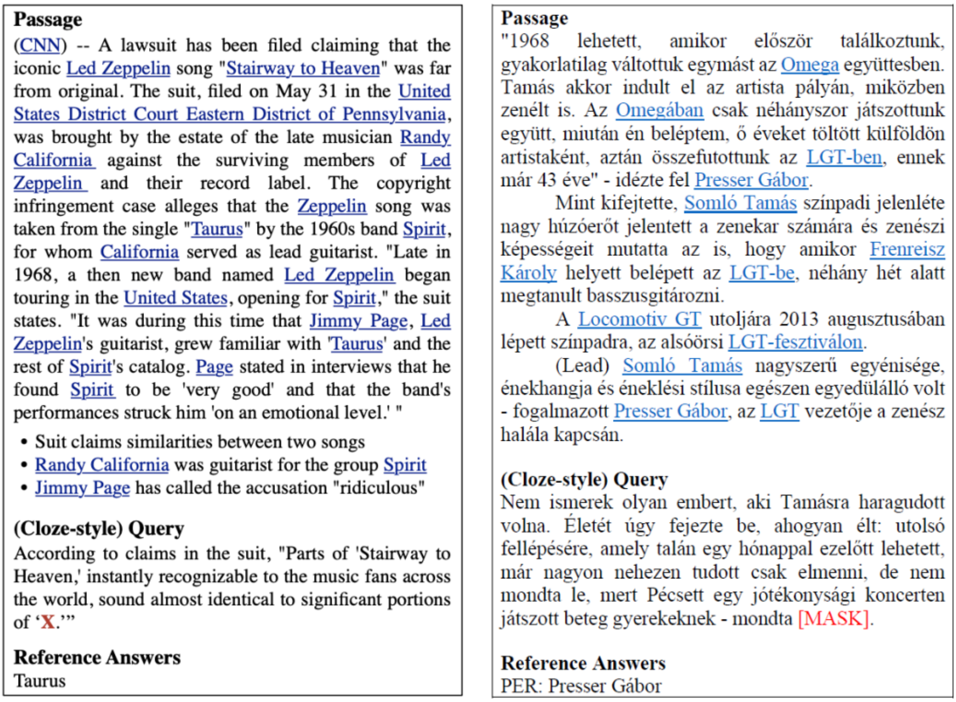
[1] https://github.com/abisee/cnn-dailymail

for the RC task.

Generative Pre-Training (GPT) designates the concept of pre-training a language model on large datasets. The application of the GPT paradigm can foster significant advancements in natural language processing, especially in the area of classification, question-answering and investigation of semantic similarity. GPT models use a Transformer Decoder architecture. A key question behind GPT experimentation is how training on large datasets can improve the performance of language models. GPT-2 achieved significant performance in several tasks already in a zero-shot setting [18]. For Hungarian, Yang trained the first GPT-2 language models [26].

Tajti proved that using ensemble approach could achieve higher system performance [22]. He defined new voting function variants for ensemble learner committee machine algorithms which can be used as competitors of the well-known voting functions. In our research, we used the GPT-2 model as language model to combine our different fine-tuned RC models to gain higher system performance.

3. Building the HuRC Corpus



Passage
(CNN) -- A lawsuit has been filed claiming that the iconic Led Zeppelin song "Stairway to Heaven" was far from original. The suit, filed on May 31 in the United States District Court Eastern District of Pennsylvania, was brought by the estate of the late musician Randy California against the surviving members of Led Zeppelin and their record label. The copyright infringement case alleges that the Zeppelin song was taken from the single "Taurus" by the 1960s band Spirit, for whom California served as lead guitarist. "Late in 1968, a then new band named Led Zeppelin began touring in the United States, opening for Spirit," the suit states. "It was during this time that Jimmy Page, Led Zeppelin's guitarist, grew familiar with 'Taurus' and the rest of Spirit's catalog. Page stated in interviews that he found Spirit to be 'very good' and that the band's performances struck him 'on an emotional level.' "

- Suit claims similarities between two songs
- Randy California was guitarist for the group Spirit
- Jimmy Page has called the accusation "ridiculous"

(Cloze-style) Query
According to claims in the suit, "Parts of 'Stairway to Heaven,' instantly recognizable to the music fans across the world, sound almost identical to significant portions of '**X**.'"

Reference Answers
Taurus

Passage
"1968 lehetett, amikor először találkoztunk, gyakorlatilag váltottuk egymást az Omega együttesben. Tamás akkor indult el az artista pályán, miközben zenélt is. Az Omegában csak néhányszor játszottunk együtt, miután én beléptem, ő éveket töltött külföldön artistaként, aztán összefutottunk az LGT-ben, ennek már 43 éve" - idézte fel Presser Gábor.

Mint kifejtette, Somló Tamás színpadi jelenléte nagy húzóerőt jelentett a zenekar számára és zenészi képességeit mutatta az is, hogy amikor Frenreisz Károly helyett belépett az LGT-be, néhány hét alatt megtanult basszusgitározni.

A Locomotiv GT utoljára 2013 augusztusában lépett színpadra, az alsóörsi LGT-fesztiválon.

(Lead) Somló Tamás nagyszerű egyénisége, énekhangja és éneklési stílusa egészen egyedülálló volt - fogalmazott Presser Gábor, az LGT vezetője a zenész halála kapcsán.

(Cloze-style) Query
Nem ismerek olyan embert, aki Tamásra haragudott volna. Életét úgy fejezte be, ahogyan élt: utolsó fellépésére, amely talán egy hónappal ezelőtt lehetett, már nagyon nehezen tudott csak elmenni, de nem mondta le, mert Pécsett egy jótékonysági koncerten játszott beteg gyerekeknek - mondta [MASK].

Reference Answers
PER: Presser Gábor

Figure 1. A ReCoRD [28] and a HuRC sample.

We created HuRC based on ReCoRD. To create the Hungarian counterpart of

ReCoRD, we used the daily news articles from Népszabadság Online[2] that had titles and summaries as well, in addition to the main text (396 886 articles). If a component was missing from an article, it was discarded. We then selected articles consisting of 3-6 paragraphs. An important criterion was that both the main text and the query (the last paragraph) contained a proper noun.

We trained a NER model using huBERT [14] for detecting proper nouns. For training NER models, the largest Hungarian NER corpus, the NYTK-NerKor (NerKor) corpus [21] was used. NerKor contains 67,524 segments, 1,028,114 tokens and 128,168 type. To fine-tune the models, we used the code provided by huggingface transformers token classification library[3]. The following modified parameters were used: learning rate = 1e-4, batch size: 4, max sequence length: 128. As for the evaluation, the IOB-based seqeval [13] method and F-score were used. In our experiments, we trained the models with 5 epoch number. At each epoch, we have saved a checkpoint and evaluated it. Our model (the checkpoint at epoch 1) achieved an F-score of **90.18** on the test set.

As a final step, we looked for proper names which are present both in the main article and the summary. Several pairs of proper names could occur in one article. In our example (see the example on the right in Figure 1), *Presser Gábor* and *Tamás* are present in both the question and the main text. In such cases, a given article is included in the database several times, with different proper name pairs. Thus, a total of 49 782 articles of different types were selected, of which a total of 88 655 instances constitute our dataset due to the phenomenon of multiple proper name pairs. Table 1 summarizes the quantitative properties of our corpus.

Table 1. Characteristics of the corpora.

	nol.hu	Silver	Gold
Segments	396,886	88,655	80,614
Segment type	-	49,782	47,199
Token	146,816,535	27,703,631	25,218,760
Type	4,361,301	1,115,260	1,078,467
Passage avg. length (word)	(article) 330.09	249.42	215.53
Query avg. length (word)	-	63.07	63.28

Our NER model did not handle some cases as expected: Table 2 shows the phenomena we corrected. Hungarian is an agglutinative language, where the majority of syntactic relations is expressed with suffixes. Most of the incorrect cases of NER were due to the fact that the model separated the suffixes from the proper name. These had to be re-attached to the proper name afterwards. In many cases, the word had a punctuation mark attached to it, but these had to be separated from the named entity. In this sense, 6 different groups of errors were distinguished. The

[2] http://nol.hu

[3] https://github.com/huggingface/transformers/tree/master/examples/pytorch/token-classification

first group was called “all”, where there was no punctuation mark on the proper noun, and the tokens in question had to be combined into one. The other cases are where some punctuation mark was either before the word (“front”) or after the word (“back”). There could be more than one of these punctuation marks (1,2). In addition to problems with punctuation, there were also cases, such as *NAME-[MASK]* in Table 2, where hyphenated proper nouns were split into several parts.

Table 2. Some examples for the errors of the NER corrected manually afterwards.

	examples for NER errors		Modified
“all”	[MASK]-ak [MASK]ában Észak-[MASK]	→	[MASK]
front-1	„[MASK]tel ([MASK]mal +[MASK]nak	→	„[MASK] ([MASK] +[MASK]
back-1	[MASK]-vel, [MASK]ban) [MASK]áról:	→	[MASK], [MASK]) [MASK]:
back-2	[MASK]ához.) [MASK]ban!” [MASK]ának),	→	[MASK].) [MASK]!” [MASK]),
front-1 back-1	([MASK]ban) „[MASK]t, „[MASK]ban”	→	([MASK]) „[MASK], „[MASK]”
front-1 back-2	([MASK]ában),	→	([MASK]),

In general, the main issue was caused by the feature of our NER model; namely that it marks strictly the lemma of the named entities, however, the suffixes are also integral parts of the words in Hungarian. Furthermore, in the surface form of the words, punctuation marks may be attached to the words as well. In this task, we needed the entire named entity with suffixes, but without the punctuation marks. Thus, we had to include the suffixes in the masked words, and to detach the punctuation marks from them. We could separate the following cases:

- no punctuation mark on the word (all),
- one punctuation mark before the word (front-1),
- one punctuation mark after the word (back-1),
- two punctuation marks after the word (back-2),

- one punctuation mark before the word and one punctuation mark after the word (front-1 back-1),
- one punctuation mark before the word and two punctuation marks after the word (front-1 back-2).

We then made a few small improvements to the corpus we created. The resulting corrected dataset was checked by one annotator per 100 units. For the annotation process, we provided a self-made demo interface. The automatic masking had to be validated against the following criteria: i) whether the named entity recognition and masking was correct (i.e. *Pope Francis* was masked and not just *Francis*, and *Gödöllőre* 'Gödöllő.SUB' was masked as [MASK] instead of [MASK]re), and ii) whether the masked proper name was also present in the previous parts of the article.[4] As a result of the validation, 80 614 automatically generated, manually validated text units are in the database. The dataset is already splitted into training, validation and test sets (64 614, 8 000 and 8 000 instances, respectively).[5]

3.1. The test set

Many studies reported that a small flaw in the test set may result in very biased models and may ruin the evaluation easily (see for example [16]). As HuRC was created mainly automatically, the chance of erroneous labels or masking is certainly high. We aimed to provide a test set as clean and accurate as possible, therefore the 8 000 instances of the test set were manually validated again against the following criteria: i) whether the named entity recognition and masking was correct,[6] ii) whether each and every named entity in the passage is listed in the list of named entities found by the NER model. This manual validation required >100 work hours of an annotator.[7]

4. Training models and experiments

There are two approaches to train reading comprehension models: extractive and abstractive. In the case of extractive reading comprehension task, the model identifies the answer to a given question from a document context by 'extracting' the corresponding correct answer. This approach can only produce answers which occur in the given document. But in our task, the masked phrase could be different from the found answer in grammatical form. Thus, this method, in certain cases could only give an approximate answer and may not produce the appropriate accurate answer that fit the masked token. The second approach, the abstractive

[4] A total of 12 annotators worked on the corpus.

[5] https://github.com/nytud/HuRC, https://huggingface.co/datasets/NYTK/HuRC

[6] This is only a double-check of the first annotation process. Two erroneous masking were found in the 8 000 instances of the test set.

[7] By the time this article is submitted, 50% of the test set has been validated.

method, can solve this problem. The abstractive model, based on the given document context, can generate answer from scratch, which could fit exactly to the masked token.

The extractive model learns the start and the end indices of the answers. It calculates the probability of word i being the start/end of the answer span as a dot product between *ith* input token and *start/end vector* followed by a softmax over all of the words in the paragraph. The training objective is the log-likelihood of the correct start and end position. For this task an encoder-only transformer architecture is enough to solve the problem. It is important that the model has to be equipped with Hungarian language knowledge. Thus, in our experiment, we used the state of the art Hungarian huBERT and the XLM-RoBERTa multilingual models.

The abstractive model needs text generation feature, hence an encoder-decoder transformer architecture should be applied. The task can be solved as a text-to-text task, where the input text is the concatenation of document context and question with masked token, the output text is the answer with the correct grammatical form. Since there is no Hungarian fully pre-trained encoder-decoder model, in our experiment, we used the mT5 [25] multilingual model that contains Hungarian knowledge.

To fine-tune our models, first, we have converted our collected data into format SQuAD [20], then, for training models, we used the Question answering libraries[8] that were provided by Hugging Face.

For the extractive experiments, we used 4 x GeForce GTX 1080Ti GPU (11 GB) cards and for the abstractive experiments, we used 4 x NVIDIA A100 GPU (80 GB) cards.

We have trained three different transformer models for the neural reading comprehension (NRC) task, with the following modified hyperparameters:

- Extractive Models:
  - **huBERT** (fine-tuned huBERT model): max_seq_length=512; doc_stride=5; max_answer_length=16; learning_rate=2e-5; epoch=10; batch_size=10;
  - **XLM-R** (Fine-tuned XLM-RoBERTa base model): max_seq_length=512; doc_stride=5; max_answer_length=16; learning_rate=2e-5; epoch=10; batch_size=4;
- Abstractive Model:
  - **mT5** (Fine-tuned mT5 base model): max_seq_length=1024; doc_stride=2; max_answer_length=16; learning_rate=2e-5; epoch=10; batch_size=4;

[8] https://github.com/huggingface/transformers/tree/master/examples/pytorch/question-answering

- Ensemble Model: Using the two extractive and the abstractive models, we combined them to achieve higher output results. In this experiment, in the query, we replaced the [MASK] with the predicted answers that were generated by our NRC models, then using a Hungarian GPT-2 model, we counted the perplexity values of the different queries. The final output is the query which has the lowest perplexity. For this task we used the NYTK/text-generation-news-gpt2-small-hungarian [27] model.

5. Results and evaluation

To evaluate our models, we used different kinds of approach. First, we used the official SQuAD evaluation metrics [20], *exact match* (Match) and (macro-averaged) *F1 score* (F1) respectively. Secondly, we have used the chrF-3 and chrf-6 that are commonly used in machine translation experiments [17]. In the case of Hungarian RC task, the answer could be different only in the suffices of the word, thus a character based evaluation metric could present the more accurate performance of the models.

Table 3. Results.

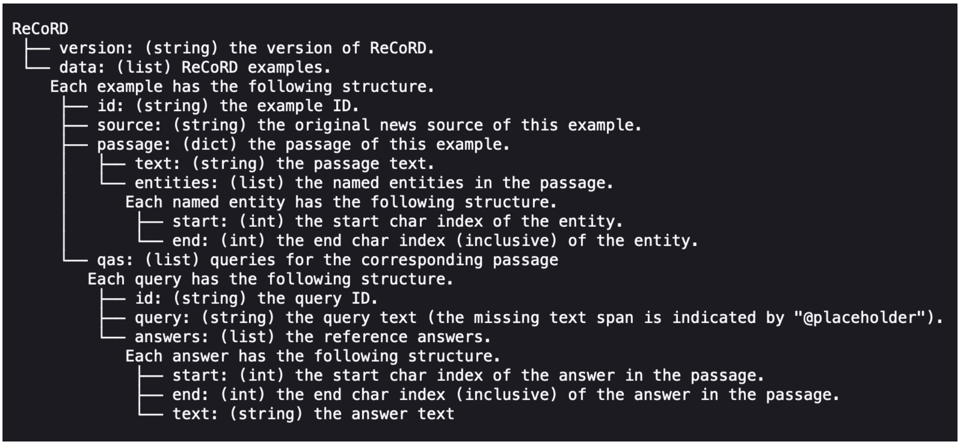
	Match	F1	chrF-3/chrF-6
	Extractive		
huBERT	64.50	69.03	73.12/72.43
XLM-R	58.98	63.59	67.19/66.04
	Abstractive		
mT5	69.51	76.26	82.96/83.28
ensemble	74.04	77.57	80.54/79.97

In Table 3, you can see the results of the models. As expected, mT5 could gain higher performance than the extractive method, because the abstractive method can formulate an answer in the appropriate grammatical form as opposed to the extractive. Furthermore, using the ensemble method, we could achieve the highest exact match and F1-score results by exploiting the advantages of all models. As for the chrF values the mT5 gained the highest performance, it may be because the abstractive method can generate longer answers, resulting in higher matches at the character level, but lower efficiency at the word level. The ensemble approach could keep control this “over-generation” feature of the abstractive method.

In the case of the test set of 8000 instances, 46.35% of the results were predicted correctly (exact match) by all models at the same time and 17.34% were predicted falsely. In the remaining cases at least one model could predict correctly. In Hungarian the masked entity may differ in grammatical form from the reference names entity in the context, thus for instance, in the case of the extractive method we could not expect that the model gives an exact matched answer. Therefore a deeper evaluation method and error analysis is needed for the erroneously predicted answers.

5.1. Special evaluation method

To understand the complexity of this task for Hungarian, first we have to understand ReCoRD's original evaluation method (as it is applied in SuperGLUE, [24]). As can be seen in Figure 2, multiple reference answers are provided for one masked named entity: these are the named entities that were found in the passage and refer to the same entity. For example, if *Manchester United*, *United* and *Manchester* are found in the text of the passage, and *United* is the masked entity in the query, all the appearances of the three named entities are listed as answers.[9] In SuperGLUE, models' performance is evaluated with max (over all mentions) token-level F1 and exact match (EM).



```
ReCoRD
 ├── version: (string) the version of ReCoRD.
 └── data: (list) ReCoRD examples.
     Each example has the following structure.
     ├── id: (string) the example ID.
     ├── source: (string) the original news source of this example.
     ├── passage: (dict) the passage of this example.
     │   ├── text: (string) the passage text.
     │   └── entities: (list) the named entities in the passage.
     │       Each named entity has the following structure.
     │       ├── start: (int) the start char index of the entity.
     │       └── end: (int) the end char index (inclusive) of the entity.
     └── qas: (list) queries for the corresponding passage
         Each query has the following structure.
         ├── id: (string) the query ID.
         ├── query: (string) the query text (the missing text span is indicated by "@placeholder").
         └── answers: (list) the reference answers.
             Each answer has the following structure.
             ├── start: (int) the start char index of the answer in the passage.
             ├── end: (int) the end char index (inclusive) of the answer in the passage.
             └── text: (string) the answer text
```

Figure 2. Format of the ReCoRD dataset.

But if we try to adapt this to Hungarian data, we face a serious problem: the masked named entity may appear earlier in the text referring to the same entity of the word, but it is very likely to have a different surface form depending on the given syntactic function it bears in the query's sentence. Staying with the previous example, *Manchester United* may appear in the passage in multiple various forms, such as *Manchester Unitedet* 'Manchester United.Acc', *Manchester Unitedről* 'Manchester United-Del' etc, and the same goes for *United* (*Unitednek* 'United.Dat', for example) and *Manchester* as well. On top of that, in the query, *United* may appear in a form that was not present in the passage, *Unitedban* 'United.Ine', for example. If we expect the models to give back a list of entities derived from the list of named entities in the passage, the list would look like *Manchester Unitedet* 'Manchester

[9] Only if they refer to the football club in the given context: if *Manchester* is present in the text as the city itself, that occurrence will not be listed among the answers.

United.ACC', *Unitednek* 'United.DAT' etc., which means word forms that definitely do not fit into the sentence in the place of the masked entity.

On the other hand, it may be quite difficult for a language model that is not inherently a generative one to pick the correct lemmas and conjugate correctly at the same time. To overcome this difficulty raised by the grammatical complexity of Hungarian, we decided to insert two lists into the instances. The first one is similar to the answer list of the ReCoRD dataset: it contains the surface forms of the named entities of the passage that refer to the same entity as the masked one in the query. However, they are only listed once: if a given surface form appears more than once in the passage, it still gets into the list once. The second list contains all the lemmas of these surface forms the suffix of the masked entity applied to them: they all fit into the sentence correctly, but are not necessarily present in the passage in their current form. We call the first list "MATCH", and the second one "MATCH_SUFFIX". We evaluate the models on both lists with F-score: this way we reward correct answers and punish incorrect ones, but a non-generative model may also have a chance to perform well on this task (on the MATCH list).

To experiment further with the evaluation options and the capabilities of the models, we have also compiled a merged list of the two lists mentioned above. By the time this paper is submitted, 25% of the test set (2000 instances) is supplied with these lists. The evaluation presented below is based on this test set of 2000 instances.

6. Discussion

As can be seen in Table 4, "MATCH" list, where the reference answers are all word forms appearing in the passage, seems to be easier for huBERT and XLM-Roberta, while mT5 and the ensemble model perform better on the more advanced list, where the word forms have to fit into the masked place perfectly (thus have to be conjugated). The best overall result is that of the ensemble model, 79.58% F-score on the "MATCH_SUFFIX" list. huBERT has the best result on the MATCH list, 76.59% F-score, which is not significantly better than the ensemble model's result on this list (76.19%).

If we look at the merged list, which is really permissive, each model's performance is better than its performance on the other two lists. The ensemble model is again better than the other 3, with an F-score of 81.82%. However, huBERT beats the abstractive mT5 on this merged list (78.09%).

For half of the instances of the test set each model could predict the correct answer. These seem to be "easy" questions for them. In these cases the surface form of the masked entity is almost always suffixless (it is the nominative form of the lemma, without any case suffix on it), and if not, the given surface form appears in the passage as well.

On the other hand, in 19.2% of the cases, none of the models could predict a correct answer (on the MATCH list – this rate is 15.15% for the MATCH_SUFFIX

Table 4. Results of the special evaluation.

	MATCH	MATCH SUFFIX	MERGED
huBERT (F1)	**76.59**	71.88	78.09
XLM-R (F1)	69.99	65.82	71.46
mT5 (F1)	71.08	76.29	77.34
ensemble (F1)	76.19	**79.58**	**81.82**
each model	49.30%	49.70%	51.05%
none of the models	19.2%	15.15%	12.75%
only huBERT	5.90%	5.55%	-
only XLM-R	2.75%	2.15%	-
only mT5	4.85%	10.35%	-

list and 12.75% for the merged list). Table 5 shows some examples with the reference answers (of the merged list) and the answers of the models.

Table 5. Some examples for wrong prediction.

Reference	huBERT	XLM-R	mT5
Kissen ’Kiss.Sup’, Kiss-sel ’Kiss.Ins’, Kiss	Alekszandrovna	Alekszandrovna	Aleks
Balogh Levente	Varga Zoltán	Varga Zoltán	Varggh Levente
Neuer	Thiago	Dante	Ribeer
MVM	MFB	MFB	MFB
Juhászék ’Juhász.FamPl’ Juhász	Juhász kérés ’Juhász question’	Lázár János már ’Lázár János already’	Tuászsék ’Tuászs.FamPL’[10]
Washington	Washingtonnak ’Washington.Dat’	Washingtonnak ’Washington.Dat’	Washingtonban ’Washington.Ine’
Indexnek ’Index.Dat’	Index	Eximbank	Index
Törökország, Törökországnak ’Turkey.Dat’, Törökországból ’Turkey.Ela’	Törökország közötti ’Turkey in.between’	Törökország közötti ’Turkey in.between’	Törökországba

In the first half of the table examples (see Table 5) show cases when models have erroneously predicted a named entity regardless of the suffixes. These cases can be seen as complete mistakes. The second half of the table shows some mixed cases: the models often hallucinate, either by adding extra (common) nouns to the

[10] *Tuászs* is not a valid Hungarian proper name.

proper name, or adding some adverbs or other function words, or by generating non-existing lemmas.

As mentioned earlier, the dataset may contain an article more than once with different named entities masked in the query. We examined the articles in the test set that appear multiple times. Models are able to predict the correct answer in the different appearances of an article. Table 6 shows cases where the article has 3 different instances in the test set with different masked named entities, and the majority of the models happen to predict the correct one in all of the cases. It is quite interesting that in the case of the last example, in two instances *Pence* / *Mike Pence* is the masked entity, and in one case the models predict it well (except for XLM-RoBERTa, which happens to insist on *Putyin*). In the other case, mT5 also hallucinates an answer (*Put Pence*). For some reasons, in one case, the models rely on the surname of the politician (*Pence*), and in the other, they all use the first name of him as well (*Mike*, and *Put* can be seen as a hallucinated first name in the case of mT5).

Table 6. Some examples for the results on articles appearing three times in the test set with different masked named entities in their query.

reference	XLM-R	huBERT	mT5	ensemble
Napi Gazdaság	Magyar Nemzet	Napi Gazdaság	Magyar Gazdaság	Napi Gazdaság
Fidesz	Magyar Nemzet	Fidesz	Fidesz	Fidesz
Fidesz	Magyar Nemzet	Fidesz	Fidesz	Fidesz
Trump	Trump	Trump	Donald	Trump
Pence, Mike Pence	Putyin	Mike Pence	Put Pence	Mike Pence
Pence, Mike Pence	Putyin	Pence	Pence	Pence

As for the important role of cloze questions in NLP, one has to mention the research of Lewis et al. [11]. Their paper is a nice and clear presentation of how cloze-stlye query databases may be exploited for a broader range of studies. First they trained a model to create cloze questions from sample documents. Afterwards, they trained a standard extractive QA model on their generated data. Their results demonstrate that self-supervised extractive QA is achievable with highly competitive results. As their training data is automatically generated, the method makes the creation of extractive QA models possible for other languages and more domains as well.

7. Conclusion

In this paper we presented the first neural machine reading comprehension models in commonsense reasoning task for Hungarian. We trained the multilingual models XLM-R and mT5, and the Hungarian model huBERT on a reading comprehension dataset (HuRC) designed based on the ReCoRD dataset. We tested to extractive (hubERT and XLM-R) and an abstractive (mT5) model to be able to compare their performance with regard to their different architectures as well. We also implemented an ensemble method by using a Hungarian GPT-2 model to count the perplexity values of the different queries built up by the predictions of the three models. We applied a complex and thorough evaluation methodology. Our result show that the reading comprehension task in Hungarian is still challenging for the different models. Extractive models seemed to be perform better in giving back already seen surface forms of the masked named entities, but the abstractive model, mt5 beats them in conjugating the words correctly. The ensemble model reached promising results in all evaluation configurations. We hope that our results will advance neural models trained for reading comprehension task for Hungarian.

References

[1] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, K. Toutanova: *BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions*, in: NAACL, 2019.

[2] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, V. Stoyanov: *Unsupervised Cross-lingual Representation Learning at Scale*, CoRR abs/1911.02116 (2019), arXiv: 1911.02116, url: http://arxiv.org/abs/1911.02116.

[3] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova: *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186, doi: https://doi.org/10.18653/v1/N19-1423, url: https://aclanthology.org/N19-1423.

[4] D. Dzendzik, C. Vogel, J. Foster: *English Machine Reading Comprehension Datasets: A Survey*, in: EMNLP, 2021.

[5] Á. Feldmann, R. Hajdu, B. Indig, B. Sass, M. Makrai, I. Mittelholcz, D. Halász, Z. G. Yang, T. Váradi: *HILBERT, magyar nyelvű BERT-large modell tanítása felhő környezetben*, in: XVII. Magyar Számítógépes Nyelvészeti Konferencia, Szeged, Magyarország: Szegedi Tudományegyetem, Informatikai Intézet, 2021, pp. 29–36.

[6] R. Han, I.-H. Hsu, J. Sun, J. Baylon, Q. Ning, D. Roth, N. Peng: *ESTER: A Machine Reading Comprehension Dataset for Event Semantic Relation Reasoning*, 2021, doi: https://doi.org/10.48550/ARXIV.2104.08350, url: https://arxiv.org/abs/2104.08350.

[7] K. M. Hermann, T. Kočiský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, P. Blunsom: *Teaching Machines to Read and Comprehend*, in: Advances in Neural Information Processing Systems (NIPS), 2015, url: http://arxiv.org/abs/1506.03340.

[8] F. Hill, A. Bordes, S. Chopra, J. Weston: *The Goldilocks Principle: Reading Children's Books with Explicit Memory Representations*, CoRR abs/1511.02301 (2016).

[9] D. Khashabi, S. Chaturvedi, M. Roth, S. Upadhyay, D. Roth: *Looking Beyond the Surface: A Challenge Set for Reading Comprehension over Multiple Sentences*, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 252–262, doi: https://doi.org/10.18653/v1/N18-1023, url: https://aclanthology.org/N18-1023.

[10] T. Kočiský, J. Schwarz, P. Blunsom, C. Dyer, K. M. Hermann, G. Melis, E. Grefenstette: *The NarrativeQA Reading Comprehension Challenge*, Transactions of the Association for Computational Linguistics TBD (2018), TBD, url: https://TBD.

[11] P. Lewis, L. Denoyer, S. Riedel: *Unsupervised Question Answering by Cloze Translation*, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy: Association for Computational Linguistics, July 2019, pp. 4896–4910, doi: https://doi.org/10.18653/v1/P19-1484, url: https://aclanthology.org/P19-1484.

[12] N. Ligeti-Nagy, G. Ferenczi, E. Héja, K. Jelencsik-Mátyus, L. J. Laki, N. Vadász, Z. G. Yang, T. Váradi: *HuLU: magyar nyelvű benchmark adatbázis kiépítése a neurális nyelvmodellek kiértékelése céljából*, in: XVIII. Magyar Számítógépes Nyelvészeti Konferencia, Szeged: JATEPress, 2022, pp. 431–446.

[13] H. Nakayama: *seqeval: A Python framework for sequence labeling evaluation*, Software available from https://github.com/chakki-works/seqeval, 2018, url: https://github.com/chakki-works/seqeval.

[14] D. M. Nemeskey: *Introducing huBERT*, in: XVII. Magyar Számítógépes Nyelvészeti Konferencia, Szeged, Magyarország: Szegedi Tudományegyetem, Informatikai Intézet, 2021, pp. 3–14.

[15] K. Nishida, I. Saito, K. Nishida, K. Shinoda, A. Otsuka, H. Asano, J. Tomita: *Multi-style Generative Reading Comprehension*, 2019, arXiv: 1901.02262 [cs.CL].

[16] C. G. Northcutt, A. Athalye, J. Mueller: *Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks*, ArXiv abs/2103.14749 (2021).

[17] M. Popović: *chrF: character n-gram F-score for automatic MT evaluation*, in: Proceedings of the Tenth Workshop on Statistical Machine Translation, Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 392–395, doi: https://doi.org/10.18653/v1/W15-3049, url: https://aclanthology.org/W15-3049.

[18] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever: *Language Models are Unsupervised Multitask Learners* (2019).

[19] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu: *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*, Journal of Machine Learning Research 21.140 (2020), pp. 1–67.

[20] P. Rajpurkar, J. Zhang, K. Lopyrev, P. Liang: *SQuAD: 100,000+ Questions for Machine Comprehension of Text*, in: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2383–2392, doi: https://doi.org/10.18653/v1/D16-1264, url: https://aclanthology.org/D16-1264.

[21] E. Simon, N. Vadász: *Introducing NYTK-NerKor, A Gold Standard Hungarian Named Entity Annotated Corpus*, in: Text, Speech, and Dialogue - 24th International Conference, TSD 2021, Olomouc, Czech Republic, September 6-9, 2021, Proceedings, ed. by K. Ekstein, F. Pártl, M. Konopík, vol. 12848, Lecture Notes in Computer Science, Springer, 2021, pp. 222–234.

[22] T. G. Tajti: *New voting functions for neural network algorithms*, Annales Mathematicae et Informaticae 52 (2020), pp. 229–242.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin: *Attention is All you Need*, in: Advances in Neural Information Processing Systems 30, ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett, Curran Associates, Inc., 2017, pp. 5998–6008.

[24] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, S. R. Bowman: *SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems*, 2020, arXiv: 1905.00537 [cs.CL].

[25] L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua, C. Raffel: *mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer*, in: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Online: Association for Computational Linguistics, June 2021, pp. 483–498, doi: https://doi.org/10.18653/v1/2021.naacl-main.41, url: https://aclanthology.org/2021.naacl-main.41.

[26] Z. G. Yang: *"Az invazív medvék nem tolerálják a suzukis agressziót" - Magyar GPT-2 kísérleti modell*, in: XVIII. Magyar Számítógépes Nyelvészeti Konferencia, Szeged, Magyarország: Szegedi Tudományegyetem, Informatikai Intézet, 2022, pp. 463–476.

[27] Yang Zijian Győző: *"Az invazív medvék nem tolerálják a suzukis agressziót" - Magyar GPT-2 kísérleti modell*, in: XVIII. Magyar Számítógépes Nyelvészeti Konferencia, Szeged, Magyarország: Szegedi Tudományegyetem, Informatikai Intézet, 2022, pp. 463–476.

[28] S. Zhang, X. Liu, J. Liu, J. Gao, K. Duh, B. V. Durme: *ReCoRD: Bridging the Gap between Human and Machine Commonsense Reading Comprehension*, 2018, arXiv: 1810.12885 [cs.CL].