# Proceedings of the

# 29th International Workshop on Concurrency, Specification and Programming (CS&P 2021)

**Berlin, September 27-28, 2021**

**Holger Schlingloff, Thomas Vogel (Eds.)**

# Preface of the 29th International Workshop on Concurrency, Specification and Programming (CS&P 2021)

Holger Schlingloff[1], Thomas Vogel[1]

[1]*Humboldt Universität zu Berlin, Germany*

The 29th International Workshop on Concurrency, Specification, and Programming 2021 (CS&P'21) is one of a series of seminars formerly organized every even year by Humboldt Universität zu Berlin and every odd year by Warsaw University. Due to the corona pandemic, this order has now been inverted. CS&P deals with the formal specification of concurrent and parallel systems, mathematical models for describing such systems, and programming and verification concepts for their implementation.

The workshop has a tradition dating back to the mid-seventies; since 1993 it was named CS&P. During almost 30 years, CS&P has become an important forum for researchers from European and Asian countries.

In 2020, the tradition was interrupted; there was no CS&P conference, and CS&P'20 was postponed to September 2021 and renamed to be CS&P'21. The event was held hybrid, both as an online event and as an on-site conference with physical meeting of the participants. The on-site meeting took place at GFaI e.V., the society for the advancement of applied informatics (Gesellschaft zur Förderung der angewandten Informatik), in Berlin-Adlershof, Germany.

This volume contains the 16 talks selected by the program committee, plus abstracts of the two invited talks of the workshop. The program of CS&P'21 reflects the current trends in its field: there are "classical" contributions on the theory of concurrency, specification and programming such as event/data-based systems, cause-effect structures, granular computing, and time Petri nets. However, more and more papers are also concerned with artificial intelligence and machine learning techniques, e.g., in the set of pairs of objects, or with sparse neural networks. Moreover, application areas such as the prediction of football games results, the classification of dry beans, or the care for honeybees are in the focus of attention. Furthermore, this volume contains several papers on software quality assurance, e.g., fault localization and automated testing of software-based systems. Altogether, we hope that you will find the collection to constitute an interesting read and to offer many connecting factors for further research.

Berlin, in September 2021
Holger Schlingloff and Thomas Vogel

*29th International Workshop on Concurrency, Specification and Programming (CS&P'21)*
hs@informatik.hu-berlin.de (H. Schlingloff); thomas.vogel@informatik.hu-berlin.de (T. Vogel)
http://www.informatik.hu-berlin.de/~hs (H. Schlingloff); https://thomas-vogel.github.io/ (T. Vogel)
0000-0002-7127-352X (T. Vogel)

## Table of Contents

# Specifying Event/Data-based Systems (keynote)

Alexander Knapp

*Augsburg University, Germany*

## Abstract

Event/data-based systems are controlled by events, their local data state may change in reaction to events. Numerous methods and notations for specifying such reactive systems have been designed, though with varying focus on the different development steps and their refinement relations. We first briefly review some of such methods, like temporal/modal logic, TLA, UML state machines, symbolic transition systems, CSP, synchronous languages, and Event-B with their support for parallel composition and refinement. We then present E↓-logic for covering a broad range of abstraction levels of event/data-based systems from abstract requirements to constructive specifications in a uniform foundation. E↓-logic uses diamond and box modalities over structured events adopted from dynamic logic, for recursive process specifications it offers (control) state variables and binders from hybrid logic. The semantic interpretation relies on event/data transition systems; specification refinement is defined by model class inclusion. Constructive operational specifications given by state transition graphs can be characterised by a single E↓-sentence. Also a variety of implementation constructors is available in E↓-logic to support, among others, event refinement and parallel composition. Thus the whole development process can rely on E↓-logic and its semantics as a common basis.

Acknowledgments: Joint work with Rolf Hennicker and Alexandre Madeira.

*29th International Workshop on Concurrency, Specification and Programming (CS&P'21)*
knapp@informatik.uni-augsburg.de (A. Knapp)
https://www.uni-augsburg.de/de/fakultaet/fai/informatik/prof/swtsse/ (A. Knapp)
0000-0002-4050-3249 (A. Knapp)

# Evaluating Fault Localization Techniques with Bug Signatures and Joined Predicates

Roman Milewski[1], Simon Heiden[1] and Lars Grunske[1]

[1]*Software Engineering, Humboldt-Universität zu Berlin, Germany*

**Abstract**

Predicate-based fault localization techniques have an advantage over other approaches by not only determining the location of a fault but also potentially giving the developer additional information to understand it. In this paper, we evaluate the accuracy of predicate-based bug signatures based on the Defects4J benchmark. Additionally, we try to improve the predicate-based approach by extending it with joined predicates, a technique for combining multiple predicates, to extract even more information. To validate our results, we compare our approaches with established spectrum-based fault localization methods.

**Keywords**

SBFL, predicate-based fault localization, bug signatures

## 1. Introduction

Searching for bugs, debugging, and fixing bugs are important parts of the software development cycle. A lot of time and effort is spent on minimizing the amount of bugs in a program, but this is usually a costly and difficult process [1].

While some bugs can be found by static analysis, e.g., compilers fail to compile a program, a lot of bugs only manifest under special conditions or are simply not detectable at all by compilers or static analysis [2]. For those bugs, the only detection approach is dynamic analysis, e.g., extracting information about the program during execution. Basic methods for findings bugs include the usage of print statements to extract information about the state of variables or the path taken through the program. More advanced methods include using a debugger or slicing [3]. All these methods give the programmer more information about the program states which lead to the bug, or reduce the amount of code that needs to be examined, thus helping the programmer to identify the faulty code.

There have been multiple approaches to automate parts of the fault localization process. A popular approach, shared by most techniques, is collecting data during correct and faulty executions of the program code and then analyzing it to produce information about the possible bug locations [4]. Tarantula, one of the first automated fault localization techniques, compares the executed lines in failed and successful program executions and assigns lines executed by more failed runs a higher score [5]. There exist more advanced statistics-based approaches

*29th international Workshop on Concurrency, Specification and Programming (CS&P'21)*

milewskr@hu-berlin.de (R. Milewski); heiden@informatik.hu-berlin.de (S. Heiden); grunske@informatik.hu-berlin.de (L. Grunske)

that use, e.g., path spectra [6, 7], def-use pairs [8, 9], information-flow pairs [9], dependence chains [10], (potential) invariants [11, 12, 13, 14, 15, 16], predicates [17, 18, 19, 20], predicate pairs [21] or method call sequences [22]. Many of these techniques provide the user with more information than only the potential locations of the bug and, thus, provide more context to help the user understand the cause of the bug.

One leading the way to providing this context is statistical debugging, a technique introduced by Liblit et al. [17], to isolate bugs by profiling several runs of the buggy program and then using statistical analysis to pinpoint the likely cause of failure. It uses predicates to provide extra information, as each predicate can hold information about a state of the program or information about the control flow of the program. Predicates can be used to generate bug signatures [23], which are sets of said predicates, providing information about the cause or effect of a bug or other information helpful for debugging. This provides the *context* for where and how the bug occurs and, thus, can be used to understand and locate the bug itself.

As an example, when confronted with a bug which leads to a program crash, the usual entry point for the programmer is the stack trace (a report of the active stack frames) and, thus, the location the program failed in. Often, the programmer is now presented with a scenario, in which it is obvious *why* the code failed at this location (e.g., a null-reference) but not *how* the program managed to arrive at this state. The programmer now has to work backwards through the stack trace, trying to find the location of the bug that caused the failure. However, often the bug is not directly part of the stack-trace. It may be in a function that manipulated some variables but already returned or – in a multi-threaded scenario – had happened in another thread.

A bug signature tries to explain a bug by showing the programmer a set of relevant predicates that have been observed during the execution of the program prior to the crash. These sets are extracted by comparing failed program executions to normal executions (i.e., without a crash) and have a high chance of being correlated to the cause of the program failure [23]. This gives the programmer additional entry points into the code, at locations with some relevance to the bug, and may thus expedite the process of identifying the bug. Additionally, the bug signatures proposed by Hsu et al. [23] consist of predicates which provide information about either the state of the program, its control flow or its data flow prior to the crash. This is additional information that the programmer can use to reconstruct the program execution more easily and which may help them locate the bug.

In this paper, we want to explore the usage of predicates in a java environment. We implemented a tool for instrumenting the java bytecode to collect the predicate data during execution, a mining algorithm that can find the top-k bug signatures from this data and, finally, compare the bug localization performance of our approaches with a state-of-the-art bug localization approach. Additionally, we explore the idea of *joined predicates*, i.e., predicate chains consisting of one or more predicates, as an improvement for predicated bug signatures.

**RQ1:** Can a bug signature based approach be used for bug localization?

**RQ2:** Can a bug signature based approach be enhanced by using *joined predicates*?

# 2. Background

In this section, we explain important concepts of predicates, itemsets and generators used in this work.

## 2.1. Predicates

Predicates are part of an approach developed by Liblit et al. [17]. Here, a program is instrumented at prior defined *instrumentation sites*. At each of these sites, one or multiple *predicates* are evaluated (as true or false) and the evaluation results are tracked. Liblit et al. [17] used the following instrumentations:

> **branches:** At each conditional statement, one *predicate* tracks whether the true branch is taken at runtime, and one *predicate* tracks the false branch.
>
> **returns:** Sometimes, function return values are used to track success or failure (either directly as boolean or with a numeric value). At each scalar returning method call, six *predicates* are used: $< 0$, $\leq 0$, $= 0 \neq 0$, $\geq 0$, $> 0$.
>
> **scalar pairs**: At each assignment to a scalar variable $x$, for each other in scope scalar variable $y_i$ the following six *predicates* are possible: $x < y_i$, $x \leq y_i$, $x = y_i$, $x \neq y_i$, $x \geq y_i$ and $x > y_i$.

Additionally, we define and track the following predicates:

> **nullness:** At each assignment of an object to a variable, we track whether the object equals `null`.

## 2.2. Itemsets, Generators and Gr-Tree

In this work, we will use the following terminology for itemsets and generators:

- An *itemset* $I$ is an unordered set of distinct items $I = \{i_1, i_2, ..., i_m\}$.
- A *transaction* $t$ is a set of items, and $t \subseteq I$.
- A *transaction database* $T$ is a set of transactions $T = \{t_1, t_2, ..., t_n\}$.
- A *frequent itemset* is an itemset that appears in at least $x$ transactions from a transaction database.
- The *support* of an itemset is the number of transactions in a transaction database that contain the itemset.
- A *generator* is an itemset $X$ such that there does not exist an itemset $Y$ strictly included in $X$ that has the same support.

Frequent itemset mining algorithms are a data mining technique used to find all frequent itemsets for a given transaction database. In a naive approach, the search space is exponential to the number of items in the transaction database. Starting with the Apriori algorithm [24], better algorithms for finding frequent itemsets have been developed. One important difference between most frequent itemset mining algorithms and the one used by our approach based on [25] is the specification of *support* as *positive support* and *negative support.* A transaction has positive support if it is the result of a successful run, and it has negative support if it is the result of a failed run.

Gr-trees (generator trees) and a depth-first Gr-growth algorithm for mining frequent generators were introduced in [26]. A Gr-tree is a typical trie structure, providing a compact representation of a transaction database. Each Gr-tree has a prefix that is the distinct itemset prefixing all items in the Gr-tree. In [25] and [27], the authors provide improved algorithms, albeit still using a Gr-tree as basis. We base our implementation on the work done in [25].

## 3. Fault Localization with Mined Bug Signatures via Predicates and Joined Predicates

### 3.1. Generating Predicates

In this first step, the goal is to modify the existing java code in such a way that we can gather predicate information during later execution.

#### 3.1.1. Instrumentation with ASM

The ASM library[1] [28] is one of the more popular frameworks for instrumenting java code. We used its ability to manipulate the bytecode of already compiled java classes to *instrument* java programs and to generate predicate data if the program is run afterward. The core ASM library provides an *event-based* representation of a compiled java class, with each element of the class being represented by an event. The core ASM library uses the visitor design pattern in which each class is passed to one (or multiple) `ClassVisitors` which can modify and transform it. The same happens for fields and methods. We implement our own visitors, which react to the events emitted by ASM while parsing the bytecode. Now, when ASM is reading a class and visits a relevant instruction, we can generate predicates and insert instructions for triggering the evaluation of those predicates: if the predicate evaluates as true during the execution of the code, the program saves this information.

#### 3.1.2. Joined Predicates

In addition to the predicates introduced in [17], our approach aims to evaluate the usefulness of *joined predicates.* In our context, joined predicates are composed of multiple "traditional" predicates and carry information about their order of appearance in the program flow.

As a motivating example, in the buggy code shown in Listing 1, a classical approach would place predicates in lines 3 and 5, among others. The included bug only manifests if *both* branches

[1]https://asm.ow2.io

Listing 1: Example code with tests

```
1   private boolean anyTrue(boolean first, boolean second) {
2      int x = 0;
3      if (first)
4         x++;
5      if (second)
6         x++;
7      return x == 1; // bug, should be: x >= 1
8   }
9   public Test1() { assert(!anyTrue(false,false)); }
10  public Test2() { assert(anyTrue(true,false)); assert(anyTrue(false,true)); }
11  public Test3() { assert(anyTrue(true,true)); } // this fails
```

at line 3 and 5 are true, but not in any other combination. If we run all example tests from Listing 1, the predicate database would contain the same predicates L:3[true] and L:5[true] after running either test 2 or test 3, making them impossible to distinguish based on only this information. Our approach would now generate 4 additional joined predicates: 1) L:3[true] $\succ$ L:5[true], 2) L:3[false] $\succ$ L:5[true], 3) L:3[true] $\succ$ L:5[false], and 4) L:3[false] $\succ$ L:5[false].

These joined predicates are different from the two separate predicates, as they additionally encode the order of execution. For example, L:3[true] $\succ$ L:5[true] means: the branch at line 3 was evaluated to true, and then, the *next* branch statement was at line 5 and was true, as well. This joined predicate is only evaluated to true in the failed test case, allowing us to distinguish it from the other test cases.

### 3.2. Gathering execution data

In the next step, we execute the instrumented code to generate the trace data. During execution, we collect predicate execution data for, e.g, each test in a test suite, while also classifying each trace as successful or failed, depending on the test outcome. The approach expects multiple execution profiles and at least one failed execution to work correctly.

The current prototype of our approach uses a simple rule for generating joined predicates: each pair of two simple predicates is a possible joined predicate. During execution of the instrumented code, each time a predicate gets evaluated, we dynamically create a new joined predicate from the last evaluated simple predicate and the currently triggered one and add it to our list of joined predicates, if it has not been encountered, previously. It is possible to use more complex rules for the creation of joined predicates, e.g., using the previous two predicates to create a joined predicate consisting of three simple ones or even only considering specific types of predicates.

During the execution of tests, all predicates for an instrumented location are directly evaluated by the executing code, and the results are stored in a list. This list contains all predicates and joined predicates that got triggered during a run and is exported at the end of the run.

### 3.3. Mining bug signatures

In this step, we try to find the top-k discriminative *bug signatures*. For this, we use an adaptation of the mining algorithm presented in [25].

The input is a database of transactions with each transaction containing a single run of the program and the information if that run was successful. A *Gr-tree* (see subsection 2.2) is constructed from the database. The Gr-tree stores all its items in its head table and links them to their corresponding nodes in the tree structure. The output is a list containing the top-k discriminative signatures.

Our mining algorithm implements the pseudo code mining algorithm described in [25]. Most differences stem from the java implementation and do not alter the basic idea. The algorithm also has some parameters controlling the size of the mined bug signatures, the cutoff point for significance, and k, the size of the returned list. We used the same default parameters as defined in [25].

## 4. Experimental Setup

### 4.1. Choosing a base for a comparison

The result of bug signature identification is different from other fault localization techniques. Other fault localization techniques output a list of program elements ordered by suspiciousness, while a bug signature approach instead returns a ranked list of bug signatures. Each bug signature consists of one or multiple program elements, describing a supposed bug context. The advantage of this additional information is difficult to quantify, as each programmer may process the information from such a bug context in a unique way and most likely different than a program would. A bug signature can therefore consist of program elements that do not have an immediately obvious relation to the bug under examination.

We decided to quantitatively compare our approaches to *spectrum-based fault localization (SBFL)*, a popular fault localization technique. Thus, we compare the following approaches:

| | |
|---|---|
| *SBFL* | state-of-the-art *SBFL* as implemented in *BugLoRD*[2], using JaCoCo[3]to collect execution profiles (test coverage data) and using the DStar metric [29] to calculate suspiciousness scores |
| Predicates | our approach using predicates |
| Joined Predicates | our approach using predicates and joined predicates (see subsubsection 3.1.2) |

### 4.2. The scoring algorithm

The result of our approach, described in section 3, is a ranked list of *bug signatures*. The result of SBFL is a ranked list of source code lines. As a bug signature can contain information about one or more code locations (note that a bug signature may also contain additional information

[2]https://github.com/hub-se/BugLoRD
[3]https://www.eclemma.org/jacoco/index.html

that has no equivalent in SBFL), we developed an algorithm to calculate suspiciousness scores for a bug signature. A scoring approach for fault localization experiments is a metric, first proposed by Renieres and Reiss in [30] and used by others [31, 32], based on static program dependencies. In the first step, a *program dependence graph (PDG)*, a graph that contains a node for each expression in the program, is constructed from the source code. Next, nodes in the PDG get marked ”faulty” for being related to the bug under inspection. Using the results of the fault localization, for each prediction, one or more nodes can be marked as ”reported” if they contain the predicted program element. Now, a score can be calculated as the fraction of the *PDG* that would need to be examined to get from a ”reported” node to a ”faulty” one by shortest path.

We used Soot[4] [33, 34] to generate intra-procedural PDGs. Soot stores the code for each method in a `Body`. Each `Body` contains a chain of `Units` which represent the actual code inside of a method. Each `Unit` represents a statement in the original source code. We now define a `codelocation` as the line of code in the source code and all following lines not belonging to another `Unit`.

---

**Definition 1 - codelocation**

---

Given a reference to a line of java source code (e.g., `MyClass:7`) and a corresponding java method in a Soot representation, a `codelocation` is the source code line of a `Unit` corresponding to the referenced line and all lines after that, until the next `Unit` or the end of its method.

The result of our algorithm is a ranked list of bug signatures. Each bug signature contains one or multiple predicates. Each predicate has a reference to the line of source code, where it was evaluated. Now, we can generate `codelocations` from a bug signature by using all references to lines of source code inside the bug signature. We generate the `codelocations` for the bug under examination (using the source code lines associated with the bug) and the `codelocations` from the bug signatures reported by our algorithm (or the *SBFL* results).

---

**Definition 2 - path cost**

---

The `path cost` is the sum of all edge costs on the shortest path between two `codelocations`.

In our setting, each edge between two classes weighs 25, each edge between two methods weighs 10 and each edge between two `Units` inside a method weighs 1. The higher cost of class and method transitions represents the additional mental effort while debugging. The numbers are chosen based on personal experience and can be adjusted.

Both SBFL and our approach rank their output by an internal suspiciousness score (see [25] for a definition of *discriminative significance (DS)* and [35] for a definition of the used SBFL metrics). Often, multiple elements get assigned the exact same score, due to the nature of the used formulas, the limited availability of diverse enough execution data (i.e., test cases with diverse execution profile) or simply due to identical execution behavior that is dictated by the implementation itself. Even worse (from an evaluation approach), a bug signature, by

[4]https://soot-oss.github.io/soot/

definition, usually contains multiple predicates and/or joined predicates and thus contains multiple `codelocations`. This means that the actual position of a `codelocation` in a ranking is nondeterministic. For each of such cases, we therefore have a *best* and *worst* case. In the best case, the `codelocation` that will lead to the smallest score is evaluated first, and in the worst case, it is evaluated last among all `codelocations` with the same suspiciousness score.

$$minCodeLocations_{before}(c_i) = |\{c_j \in C \mid susp_{DS}(c_j) > susp_{DS}(c_i)\}|$$
$$maxCodeLocations_{before}(c_i) = |\{c_j \in C \mid susp_{DS}(c_j) \geq susp_{DS}(c_i)\}|$$

An alternative, more correct, definition for $maxCodeLocations_{before}(c_i)$ would subtract 1 to not count $c_i$ twice. Now, by combining the `path cost` and `codelocation`, we can define our `EvaluationScore`:

$$Score_{best} = minCodeLocations_{before} + \left(pathCost * \sqrt{minCodeLocations_{before} + 1}\right)$$
$$Score_{worst} = maxCodeLocations_{before} + \left(pathCost * \sqrt{maxCodeLocations_{before} + 1}\right)$$
$$Score_{avg} = \frac{1}{2} * (Score_{best} + Score_{worst})$$

Note that $Score_{avg}$ is just the average of $Score_{best}$ and $Score_{worst}$. To calculate a real average `EvaluationScore`, one would use $averageMinCodeLocations_{before}$ and `path cost`. In our definition, the `path cost` is also included in the average.

The `EvaluationScore` is based on the usefulness of a bug prediction to a programmer while debugging. The `EvaluationScore` is 0, a perfect score, if the first reported `codelocation` is exactly the line of source code associated with the bug. The `EvaluationScore` grows by one for every additional `codelocation` to check. Additionally, the `EvaluationScore` is scaled with the number of `codelocations` already checked, as a programmer checking the first few locations will be motivated to look deeper, but when having already looked at multiple locations before, might stop his investigation sooner. So, while there might be an incredibly long path in the `Unit`-graph linking the suspected `codelocation` and the bug location, it is highly unlikely that this connection would be useful to the programmer. I.e., the more the `path cost` between code locations and bug increases, the less likely it is to track down the bug.

## 4.3. Evaluation Subjects

The Defects4J Benchmark[5] [36] is a collection of open source projects with each being available in multiple buggy versions. Each buggy version consists of the respective source code and a change set with changes *exclusively* related to the bug. This omission of other changes (e.g., refactorings) in the change set makes it more reliable as a source for the lines of code related to the bug. Sobreira et al. [37] did an analysis of many Defects4J bugs and showed a method for linking a bug to lines of code. The big number of real (not artificially created) bugs from many different projects make this a sensible benchmark choice for our purposes.

We used the bugs in the version 2.0.0 from the projects in Table 1. From the original set of bugs, we had to exclude 21 bugs from our final results. The most common reasons were problems

[5]https://github.com/rjust/defects4j

**Table 1**
Used projects from the Defects4J Benchmark

| project | size[loc] | #bugs |
|---|---|---|
| jfreechart (Chart) | 96k | 26 |
| commons-cli (Cli) | 2k | 39 |
| commons-codec (Codec) | 3k | 18 |
| commons-csv (Csv) | 1k | 16 |
| gson (Gson) | 6k | 18 |
| commons-lang (Lang) | 22k | 64 |
| commons-math (Math) | 84k | 106 |
| joda-time (Time) | 90k | 26 |
| Total | | 313 |
| Applicable after Instrumentation | | 292 |
| Applicable after Runtime Eval. | | 162 |

with compilation of the source code, crashes of the instrumenter and *JVM* crashes during test execution. During the runtime evaluation, we encountered some additional problems, because we could not relate the `Unit` containing the line of source code to the given bug/prediction, or because the evaluation produced a time out. This leaves us with 162 bugs which were used for the following graphs.

## 5. Results

In Figure 1, we see that $Score_{best}$ (`EvaluationScore` if the reported location is examined first among all locations ranked equally; see section 4 for more details) for both joined predicates and predicates is lower (better) than for SBFL. $Score_{worst}$ is more similar for all three approaches, with the joined predicates being significantly worse than simple predicates. If we look at the included p-values, we can see that the `EvaluationScores` for both predicate approaches are significantly different in our data set. The only non significant difference is between the worst `EvaluationScore` for joined predicates and SBFL.

In Table 2, we can see the mean values for $Score_{best}$ and $Score_{worst}$. For a better visualization, we can look at Figure 3(left), where the total `EvaluationScores` for each approach are summed up. Both predicate approaches have significantly lower total `EvaluationScores` than the SBFL approach when comparing $Score_{best}$. When comparing $Score_{worst}$, the results are more even, with only simple predicates being significantly different than the other two. If we look at the median values in Table 2, we can see the very small difference between all three approaches for $Score_{best}$. This shows us that a lot of bugs have similar, very small scores and most of the

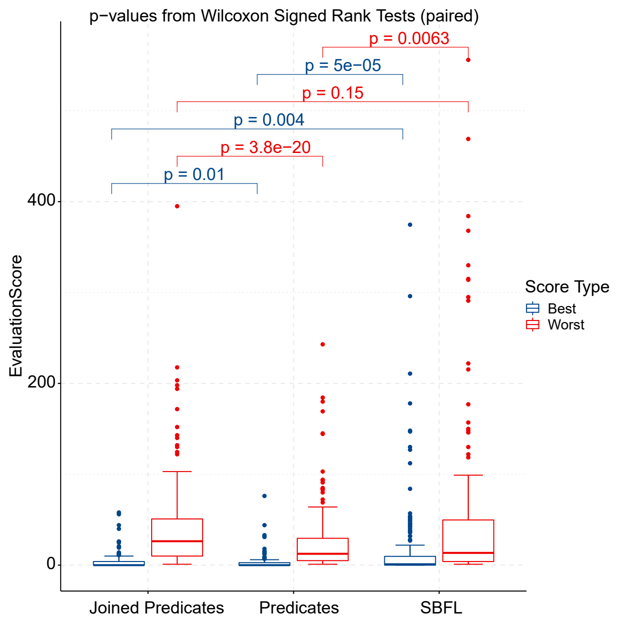


**Figure 1:** Summary of results

differences stem from a few bugs with high (i.e. bad) scores. When comparing the medians for $Score_{worst}$, the median for joined predicates is nearly double as big as the medians of the other approaches.

In Figure 3(right), we further split up our results regarding the different Defects4J projects used. It shows the different mean values for $Score_{avg}$ separated by project. Here, we can see that both predicate approaches have lower (i.e., better) results for nearly all projects, with only 'cli' in favor of SBFL.

Because a lot of the scores are close to or equal to $0$, i.e., the best result, we additionally looked at the density of scores. Figure 2 shows the density distribution of $Score_{avg}$ from 0 to $75$. The dashed lines show the mean values for each approach. Here, we see that the simple predicate approach has the highest density in the 0-10-range. This represents a lot of very low $Score_{avg}$ results – results where the correct `codelocation` was highly ranked, while not having too many similarly ranked `codelocations`. SBFL seems to have both very good scores, but also a lot of very bad ones, leading to a significantly worse mean value.

**Table 2**
Mean and median values for $Score_{best}$ and $Score_{worst}$

| Approach | $\overline{Score_{best}}$ | $\widetilde{Score_{best}}$ | $\overline{Score_{worst}}$ | $\widetilde{Score_{worst}}$ |
|---:|---:|---:|---:|---:|
| Joined Predicates | 3.97 | 0 | 41.25 | 26.33 |
| Predicates | 2.94 | 0 | 26.14 | 12.50 |
| *SBFL* | 18.94 | 1 | 48.34 | 13.50 |

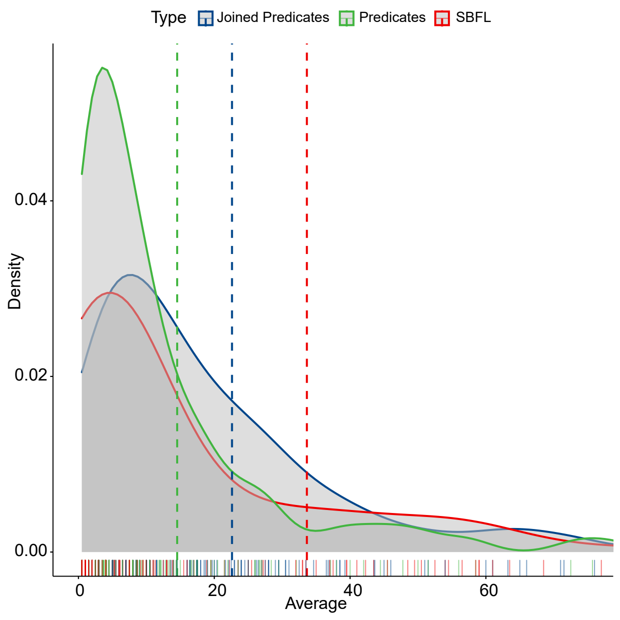


**Figure 2:** Density plot for $Score_{avg}$ with x-axis limited to 75

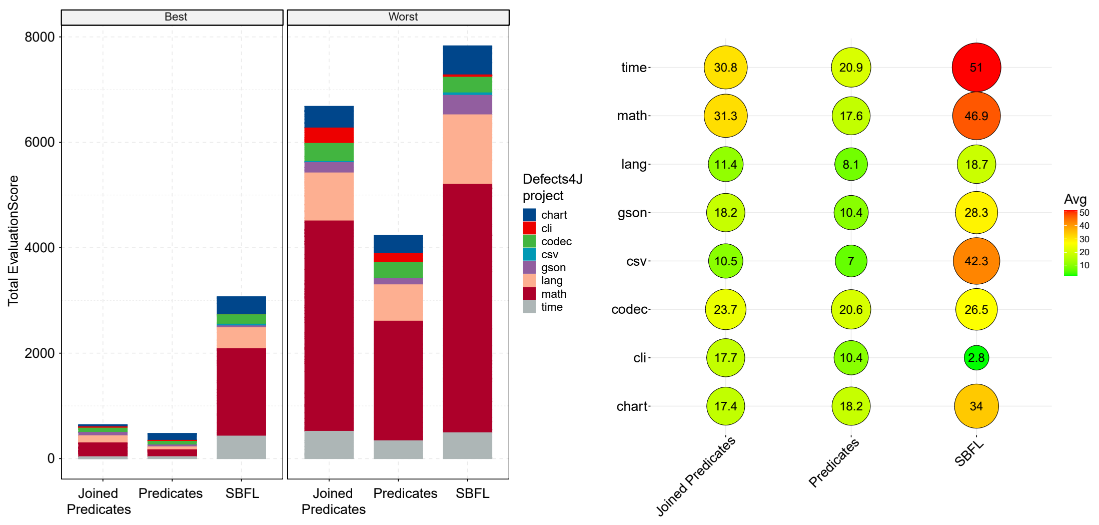


**Figure 3:** Left: Best and Worst `EvaluationScore`, Right: Mean values for $Score_{avg}$, separated by Defects4J project

## 5.1. Spread and Path Cost

Next, we analyze the `EvaluationScore` in more detail to better understand the differences between the predicate based and SBFL approaches. For this, we split up the `EvaluationScore` into its two main components:

- *path cost*: number of `Units` in the graph between the reported `codelocation` and the buggy `codelocation`
- *penalty*: number of `codelocations` to be examined before finding the reported `codelocation` in the ranking

We see in Figure 4(left), that SBFL has a *significantly* lower `path cost` than the two predicate based approaches. This is expected, as SBFL directly outputs a line of code as result, while a bug signature from a predicate based approach also contains additional information about the state of the program in a faulty run. The code lines we extract and use are actually the locations of the instrumentation sites for the predicates that the bug signature consists of. Since not every executable line is a valid instrumentation site for a predicate, but some of the evaluated bugs are located on such lines of code, the predicate based approaches can not reach a `path cost` of 0 in those cases.

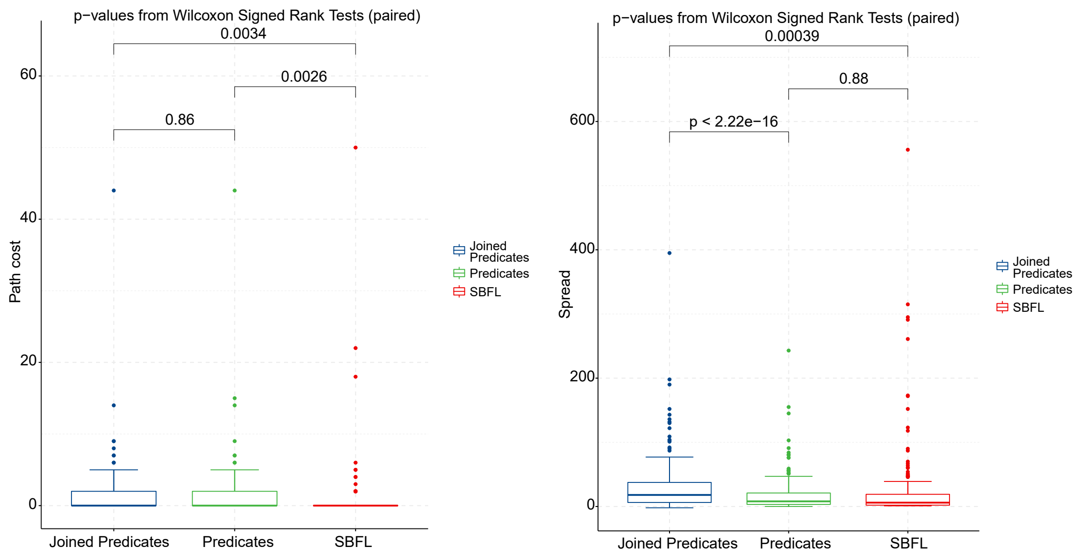


**Figure 4:** Left: Comparison of `path cost` (number of `Units` in the graph between the two `codelocations`) Right: Comparison of spread $(maxCodeLocations_{before} - minCodeLocations_{before})$

Comparing the spread in Figure 4(right), one can see that joined predicates differ *significantly* from the other approaches. This leads to the conclusion that joined predicates have more ties in the ranking of its results which makes sense, as in addition to bug signatures having the same rank, all predicates in the bug signature will have the same rank as the bug signature, itself. Additionally, a bug signature consisting of three predicates can have up to three `codelocations` , while a bug signature with joined predicates increases this number by one for each joined predicate that is part of the bug signature. For example, a bug signature consisting of 2 joined and one simple predicate can have up to five `codelocations`.

An additional observation we made during our experiments is the partly unfeasibly high run time of our experiments. Some bug signature mining runs took multiple days of computation time, making the approach in its current form largely unusable in a real world scenario – at least without further optimization.

## 6. Conclusions

After having analyzed the data from section 5, we summarize the following conclusions:

**RQ1:** Can a bug signature based approach be used for bug localization?

**Yes**, as we have seen in, e.g., Figure 1 and Figure 3(left), a bug signature (predicate) based approach consistently gets a lower `EvaluationScore` than *SBFL*.

These are welcome results, because while we expected it, as bug signature approaches have been shown as effective in other languages, e.g., [23, 25] showed predicate based approaches in C, there were multiple additional, java-specific difficulties with regard to the instrumentation.

**RQ2:** Can a bug signature based approach be enhanced by using *joined predicates*?

**Maybe**, but we did not manage to find any *significant* improvement by using our (rather simple) joined predicate approach over the single predicate approach. In our studies, we only evaluated joined predicate with a simple ”two-following” generation rule, and we did not exhaust other possible rules for generating joined predicates, yet. While a three-following rule might deem too time-intensive, we have many ideas for more elaborate rules concerning two predicates. Another factor could be our definition of the `EvaluationScore`. As we discussed in section 4, we did not find a way to use an established scoring algorithm to evaluate bug signatures. While the problems we faced were beyond the scope of this project, most of them could be solved in the future, allowing for a new evaluation of our results or future results.

We still strongly believe that a bug signature based approach can be enhanced by using joined predicates, but with other joined predicates beyond a simple combination of two following predicates into a joined one.

Another point is the differences between SBFL and predicate approaches visible in Figure 4(left). SBFL has a *significantly* lower average `path cost` than the two predicate approaches. We can explain this with the different output formats used by the two techniques. The fact that the predicate based approaches still get better overall scores thus must mean they perform better in the non `path cost` part of our `EvaluationScore`.

Furthermore, the bug signature mining algorithm described in [25] which our approaches are based on, too, actually has a parameter for controlling how many predicates can be part of a single bug signature. A bigger size limit significantly increases the computation time, which is why we used a size limit of 3, as recommended in [25]. This could have been chosen too small, as in the joined predicate approach, the joined predicates now compete with the others for a spot inside a bug signature. On the one hand, while a bigger bug signature may potentially be more helpful to a real programmer, it would negatively impact our score, as it does not differentiate between processing multiple predicates inside one bug signature or multiple small bug signatures containing the same predicates. Even worse for our evaluation (and likely any other based on lines of source code), just one ”correct” predicate inside a bug signature is all that is needed to score it positively, while ignoring all the others inside the bug signature. On the other hand, in [25], the authors noted that increasing the size of the mined bug signatures increases the predictive power but increases the computation time.

Especially noteworthy is that *all* the extra information a programmer is supposed to get out of a bug signature, e.g., it containing a predicate with $var = null$ for a $var$ where $null$ is not expected, can lead to another investigation path than just the information to investigate that line. An evaluation method considering those aspects could have completely different results than ours.

**Acknowledgements** Lars Grunske would like to thank the intensive care unit of the Achenbach hospital in Königs Wusterhausen for the tremendous help during his Covid-19 case.

## References

[1] M. Zhivich, R. K. Cunningham, The real cost of software errors, IEEE Security & Privacy Magazine 7 (2009) 87–90. doi:10.1109/msp.2009.56.

[2] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler, J. Penix, Using static analysis to find bugs, IEEE Software 25 (2008) 22–29. doi:10.1109/ms.2008.130.

[3] M. Perscheid, B. Siegmund, M. Taeumel, R. Hirschfeld, Studying the advancement in debugging practice of professional software developers, Software Quality Journal 25 (2016) 83–110. doi:10.1007/s11219-015-9294-2.

[4] W. E. Wong, R. Gao, Y. Li, R. Abreu, F. Wotawa, A survey on software fault localization, IEEE Transactions on Software Engineering 42 (2016) 707–740. doi:10.1109/tse.2016.2521368.

[5] J. A. Jones, M. J. Harrold, Empirical evaluation of the tarantula automatic fault-localization technique, in: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering - ASE '05, ACM Press, 2005, pp. 273–282. doi:10.1145/1101908.1101949.

[6] T. Reps, T. Ball, M. Das, J. Larus, The use of program profiling for software maintenance with applications to the year 2000 problem, ACM SIGSOFT Software Engineering Notes 22 (1997) 432–449. doi:10.1145/267896.267925.

[7] T. M. Chilimbi, B. Liblit, K. Mehra, A. V. Nori, K. Vaswani, Holmes: Effective statistical debugging via efficient path profiling, in: 2009 IEEE 31st International Conference on Software Engineering, IEEE, IEEE, 2009, pp. 34–44. doi:10.1109/icse.2009.5070506.

[8] R. Santelices, J. A. Jones, Y. Yu, M. J. Harrold, Lightweight fault-localization using multiple coverage types, in: 2009 IEEE 31st International Conference on Software Engineering, IEEE, IEEE, 2009, pp. 56–66. doi:10.1109/icse.2009.5070508.

[9] W. Masri, Fault localization based on information flow coverage, Software Testing, Verification and Reliability 20 (2009) 121–147. doi:10.1002/stvr.409.

[10] R. A. Assi, W. Masri, Identifying Failure-Correlated Dependence Chains, in: 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, 2011, pp. 607–616. doi:10.1109/ICSTW.2011.42.

[11] T. B. Le, D. Lo, C. L. Goues, L. Grunske, A learning-to-rank based fault localization

approach using likely invariants, in: Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, ACM, 2016, pp. 177–188. doi:10.1145/2931037.2931049.

[12] M. D. Ernst, J. Cockrell, W. G. Griswold, D. Notkin, Dynamically discovering likely program invariants to support program evolution, IEEE Transactions on Software Engineering 27 (2001) 99–123. doi:10.1109/32.908957.

[13] S. Hangal, M. S. Lam, Tracking down software bugs using automatic anomaly detection, in: Proceedings of the 24th International Conference on Software Engineering, ICSE '02, Association for Computing Machinery, New York, NY, USA, 2002, pp. 291–301. doi:10.1145/581339.581377.

[14] B. Pytlik, M. Renieris, S. Krishnamurthi, S. P. Reiss, Automated fault localization using potential invariants, CoRR cs.SE/0310040 (2003). arXiv:preprintcs/0310040.

[15] S. K. Sahoo, J. Criswell, C. Geigle, V. Adve, Using likely invariants for automated software fault localization, in: Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems, 2013, pp. 139–152. doi:10.1145/2451116.2451131.

[16] M. A. Alipour, A. Groce, Extended program invariants: applications in testing and fault localization, in: Proceedings of the Ninth International Workshop on Dynamic Analysis, 2012, pp. 7–11. doi:10.1145/2338966.2336799.

[17] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, M. I. Jordan, Scalable statistical bug isolation, ACM SIGPLAN Notices 40 (2005) 15–26. doi:10.1145/1064978.1065014.

[18] C. Liu, X. Yan, L. Fei, J. Han, S. P. Midkiff, Sober: Statistical model-based bug localization, in: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering - ESEC/FSE-13, Proceedings of 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM Press, 2005, pp. 286–295. doi:10.1145/1081706.1081753.

[19] C. Liu, L. Fei, X. Yan, J. Han, S. P. Midkiff, Statistical debugging: A hypothesis testing-based approach, IEEE Transactions on Software Engineering 32 (2006) 831–848. doi:10.1109/TSE.2006.105.

[20] P. A. Nainar, T. Chen, J. Rosin, B. Liblit, Statistical debugging using compound boolean predicates, in: D. S. Rosenblum, S. G. Elbaum (Eds.), Proceedings of the 2007 international symposium on Software testing and analysis - ISSTA '07, ACM Press, 2007, pp. 5–15. doi:10.1145/1273463.1273467.

[21] Z. You, Z. Qin, Z. Zheng, Statistical fault localization using execution sequence, in: 2012 International Conference on Machine Learning and Cybernetics, volume 3, IEEE, IEEE, 2012, pp. 899–905. doi:10.1109/icmlc.2012.6359473.

[22] V. Dallmeier, C. Lindig, A. Zeller, Lightweight defect localization for java, in: ECOOP 2005 - Object-Oriented Programming, Springer Berlin Heidelberg, 2005, pp. 528–550. doi:10.1007/11531142_23.

[23] H.-Y. Hsu, J. A. Jones, A. Orso, Rapid: Identifying bug signatures to support debugging activities, in: 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, IEEE, IEEE, 2008, pp. 439–442. doi:10.1109/ase.2008.68.

[24] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases,

in: VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile, Morgan Kaufmann, 1994, pp. 487–499. URL: http://www.vldb.org/conf/1994/P487.PDF. doi:10.5555/645920.672836.

[25] C. Sun, S.-C. Khoo, Mining succinct predicated bug signatures, in: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013, ACM Press, 2013, pp. 576–586. doi:10.1145/2491411.2491449.

[26] J. Li, H. Li, L. Wong, J. Pei, G. Dong, Minimum description length principle: Generators are preferable to closed patterns., in: Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1, volume 1 of *AAAI'06*, 2006, pp. 409–414.

[27] Z. Zuo, S.-C. Khoo, C. Sun, Efficient predicated bug signature mining via hierarchical instrumentation, in: Proceedings of the 2014 International Symposium on Software Testing and Analysis - ISSTA 2014, ACM Press, 2014, pp. 215–224. doi:10.1145/2610384.2610400.

[28] E. Bruneton, ASM 4.0 A Java bytecode engineering library, 2007.

[29] W. E. Wong, V. Debroy, R. Gao, Y. Li, The DStar method for effective software fault localization, IEEE Transactions on Reliability 63 (2014) 290–308. doi:10.1109/tr.2013.2285319.

[30] M. Renieres, S. Reiss, Fault localization with nearest neighbor queries, in: 18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings., ASE'03, IEEE Comput. Soc, 2003, pp. 30–39. doi:10.1109/ase.2003.1240292.

[31] H. Cleve, A. Zeller, Locating causes of program failures, in: Proceedings of the 27th international conference on Software engineering - ICSE '05, ICSE '05, ACM Press, 2005, pp. 342–351. doi:10.1145/1062455.1062522.

[32] P. A. Nainar, T. Chen, J. Rosin, B. Liblit, Statistical debugging using compound boolean predicates, in: D. S. Rosenblum, S. G. Elbaum (Eds.), Proceedings of the 2007 international symposium on Software testing and analysis - ISSTA '07, ACM Press, 2007, pp. 5–15. doi:10.1145/1273463.1273467.

[33] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, V. Sundaresan, Soot, in: CASCON First Decade High Impact Papers on - CASCON '10, ACM Press, Mississauga, Ontario, Canada, 2010, p. 13. doi:10.1145/1925805.1925818.

[34] P. Lam, E. Bodden, O. Lhoták, L. Hendren, The soot framework for java program analysis: a retrospective, in: Cetus Users and Compiler Infastructure Workshop (CETUS 2011), volume 15, 2011, p. 35.

[35] S. Heiden, L. Grunske, T. Kehrer, F. Keller, A. Hoorn, A. Filieri, D. Lo, An evaluation of pure spectrum-based fault localization techniques for large-scale software systems, Software: Practice and Experience 49 (2019) 1197–1224. doi:10.1002/spe.2703.

[36] R. Just, D. Jalali, M. D. Ernst, Defects4j: a database of existing faults to enable controlled testing studies for java programs, in: Proceedings of the 2014 International Symposium on Software Testing and Analysis - ISSTA 2014, ACM Press, 2014, pp. 437–440. doi:10.1145/2610384.2628055.

[37] V. Sobreira, T. Durieux, F. Madeiral, M. Monperrus, M. de Almeida Maia, Dissection of a bug dataset: Anatomy of 395 patches from defects4j, in: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2018, pp. 130–140. doi:10.1109/saner.2018.8330203.

# NLP-Based Requirements Formalization for Automatic Test Case Generation

Robin Gröpler[1], Viju Sudhi[1], Emilio José Calleja García[2] and Andre Bergmann[2]

[1]*ifak Institut für Automation und Kommunikation e.V., 39106 Magdeburg, Germany*
[2]*AKKA Germany GmbH, 80807 München, Germany*

**Abstract**
Due to the growing complexity and rapid changes of software systems, the assurance of their quality becomes increasingly difficult. Model-based testing in agile development is a way to overcome these difficulties. However, major effort is still required to create specification models from a large set of functional requirements provided in natural language. This paper presents an approach for a machine-aided requirements formalization technique based on Natural Language Processing (NLP) to be used for an automatic test case generation. The goal of the presented method is to automate the process of model creation from requirements in natural language by utilizing appropriate algorithms, thus reducing cost and effort. The application of our procedure will be demonstrated using an industry example from the e-mobility domain. In this example, requirement models are generated for a charging approval system within a larger vehicle battery charging application. Additionally, existing tools for automated model synthesis and test case generation are applied to our models to evaluate whether valid test cases can be generated.

**Keywords**
Requirements analysis, natural language processing, test generation

## 1. Introduction

In the life cycle of a device, component or system in industrial use, a rapidly changing and growing number of requirements and the associated increase in features and feature changes inevitably lead to an increasing effort for verifying requirements and testing of the implementation. To manage test complexity and reduce necessary test effort and cost, agile methods for model-based testing have been developed [1]. The effectiveness of model-based testing highly depends on the quality of the used specification model. The creation and maintenance of well-defined specification models is therefore crucial and usually comes with high effort and cost. This is especially true in agile development, where requirements are subject to frequent changes.

In this context, an approach for requirements-based testing was developed that enables efficient test processes, see Fig. 1. Model synthesis and model-based test generation methods are used to systematically and efficiently create a test suite that contains suitable test cases. This approach is based on behavioral requirements that serve as input for model synthesis. The only

*29th International Workshop on Concurrency, Specification and Programming (CS&P'21)*

robin.groepler@ifak.eu (R. Gröpler); viju.sudhi@ifak.eu (V. Sudhi); emiliojose.calleja@gmail.com (E. J. Calleja García); andre.bergmann@akka.eu (A. Bergmann)

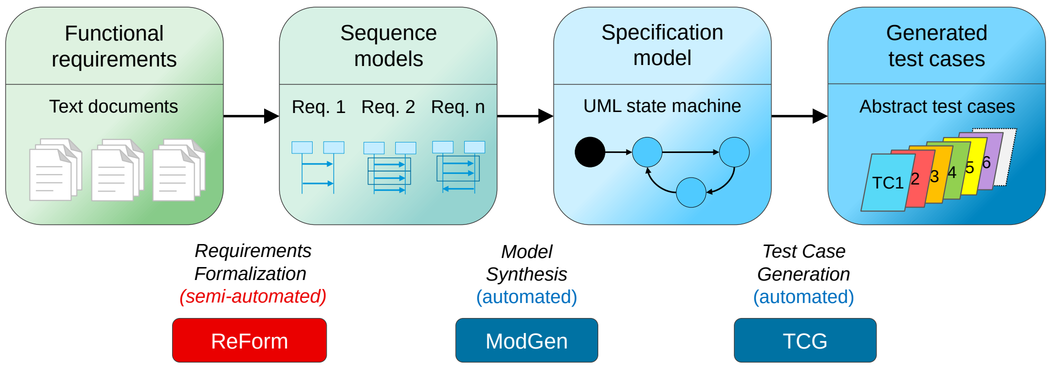


**Figure 1:** Toolchain for requirements-based test case generation.

time-consuming manual step is the creation of requirement models from textual requirements documents.

Recent advances in natural language processing show promising results to organize and identify desired information from raw text. As a result, NLP techniques show a growing interest in automating various software development activities like test case generation. Several NLP approaches and tools have been investigated in recent years aiming to generate test cases from preliminary requirements documents [2, 3, 4]. A major drawback of existing methods is the use of controlled natural language or templates that force the requirements engineer or designer not only to concentrate on the content but also on the syntax of the requirement. Furthermore, those algorithms are in general not applicable to existing requirements.

In this work, we propose a *new, semi-automated technique for requirements-based model generation* that reduces human effort and supports frequent requirements changes and extensions. The aim of our approach is to develop a method that

1) can handle an extended range of domains and formats of requirements, i.e. it is not limited to a specific template or controlled natural language, and
2) provides enhanced but easily interpretable intermediate results in the form of a textual and graphical representation of UML sequence diagrams.

Our approach utilizes an existing NLP parser to obtain basic syntactic information about the words and their relationship to each other. Based upon this information, several rule-based steps are performed in order to identify relevant syntactic entities which are then mapped to semantic entities. Finally, these entities are used to form requirement models as UML sequence diagrams. The main contributions of this work are

1) the development of a rule-based approach based on NLP information that automates the various steps involved in deriving requirement models, and
2) the evaluation on an industrial use case using meaningful metrics that demonstrates the good quality of our approach.

The paper is structured as follows. In Section 2, we briefly outline related work on NLP-based requirements formalization methods. In Section 3, we present the individual steps of our methodology for deriving requirement models from textual descriptions. The method is applied to the battery charging approval system presented in Section 4. In Section 5, we define several evaluation metrics and demonstrate the results of the application. Finally, a conclusion and outlook is given in Section 6.

## 2. Related work

In order to circumvent the challenges of analyzing highly complex requirements, many authors restrict their NLP approaches to a specific domain or a prescribed format. In [5], the authors propose an algorithm creating activity diagrams from requirements following a predefined structure. They consider the SOPHIST method which performs a refinement and formalization of structured texts by introducing text templates with a defined syntactical structure [6]. In [7], a small set of structural rules was developed to address common requirement problems including ambiguity, complexity and vagueness. In [8], requirements are expected to be written in a controlled natural language and are supposed to be from Data-Flow Reactive Systems (DFRS). The approach in [9] is to generate test cases from use cases or user stories, both of which have to comply with a specified format. In [10], requirements engineers shall be supported with formalization of use case descriptions by writing pre-conditions and post-conditions in a predefined format from which test cases can be generated automatically. Likewise [11] aims to find and complete missing test cases by parsing exceptional behaviors from Javadoc comments written in natural language, provided the documentation is in a specified template. [12] relies on the artifacts that the programmers create while developing the product which belong to a smaller subset of specifications.

Even for simple syntactical structures of requirements it is still necessary to enable the requirements engineer to review the intermediate results, i.e. the generated model artifacts, and to adjust them where necessary. The toolchain of [13] involves eliciting requirements according to Restricted Use Case Modeling (RUCM) specifications. This applies to the work of [14], where the authors attempt to generate executable test cases from a Restricted Test Case Modeling (RTCM) language which restricts the style of writing test cases. This becomes an additional overhead to the requirement engineers who draft formal requirements. Additionally, the users are expected to inspect the generated OCL constraints before proceeding to test case generation. Similarly, in [15] the authors explore the possibility of test case generation using Petri Net simulation; however the interpretability of Colored Petri Nets as proposed in the approach may vary depending on the user's level of expertise. These intermediate results may not be easily understood by the user and it may be cumbersome for him to fine-tune or modify the predictions before generating reliable test cases.

A notable work from the authors of [16] makes use of recursive dependency matching to formulate test cases. Though our approach aligns with theirs in this step, we attempt to generate test cases from a broader set of functional requirements while they restrict themselves with user stories from which a cause-effect relationship can be learnt.

# 3. Methodology

We utilize an existing NLP parser and use a rule-based algorithm to perform the transformation from requirements written in natural language to requirement models. Our rule set tries to conceive all relevant rules that could satisfactorily parse the input behavioral requirement and extract its semantic content.

## 3.1. Linguistic pre-processing

The behavioral requirements are, in general, complex by nature. In order to reliably extract the syntactic and semantic content of these requirements, a thorough linguistic pre-processing is indispensable. For this stage, we rely on spaCy (v2.1.8) [17] - a free, open-source library for advanced Natural Language Processing. We follow the basic NLP pipeline including tokenization, lemmatization, part-of-speech (POS) tagging and dependency parsing in various stages of the algorithm.

### 3.1.1. Pronoun resolution

Though the formal requirements tend to avoid first person (I, me, etc.) or second person (you, your, etc.) pronouns, they may contain third person neutral pronouns (it, they, etc.) [18]. These pronouns are identified and resolved with the farthest subject, inline with the algorithm proposed in [19] and [20]. Owing to the simplicity of the task, we assume there is no particular need to use more sophisticated algorithms checking grammatical gender and person while resolving pronouns. However, we attempt to resolve pronouns only if the grammatical number of the pronoun agrees with that of the antecedent. Since pleonastic pronouns (pronouns without a direct antecedent) do not affect the algorithm, they are cited but not replaced.

**Example:** Consider the requirement "If the temperature of the battery is below Tmin or it exceeds Tmax, charging approval has to be withdrawn". Here, the pronoun *it* is resolved with its antecedent *the temperature of the battery*.

### 3.1.2. Decomposition

Textual requirements with multiple conditions and conjunctions are hard to be transformed and mapped to individual relations. This demands decomposition of complex requirements into simple clauses [21]. Multiple conditions (sentences with multiple *if*s, *while*s, etc.), root conjunctions (sentences with multiple *roots* connected with a conjunction) and noun phrase conjunctions (sentences with multiple subjects and/or objects connected with a conjunction) are decomposed to simple primitive clauses.

We resort to the syntactic dependencies obtained from the parser to decompose requirements. The algorithm considers the token(s) with dependency *mark* to decompose multiple conditions and dependency *conj* for decomposing root and noun phrase conjunctions. The span of the sub-requirement can then be defined by identifying the *edges* (for e.g. the *left-most edge* refers to the token towards the left of the dependency graph with which the parent token holds a syntactic dependency) of the token of interest.

**Example:** In the requirement ”If the temperature of the battery is below Tmin or the temperature of the battery exceeds Tmax, charging approval has to be withdrawn”, the root conjunction (arising from the two roots *is* and *exceeds*) and the subsequent multiple conditions (arising from *if*) are decomposed to three sub-requirements as ”[if the temperature of the battery is below Tmin] *or* [if the temperature of the battery exceeds Tmax], [charging approval has to be withdrawn]”.

## 3.2. Syntactic entity identification

Almost all behavioral requirements describe a particular action (linguistically, *verb*) done by an agent (linguistically, *subject*) on the system of interest (linguistically, *object*). This motivates the idea of identifying syntactic entities from the requirements. The algorithm identifies these syntactic entities by checking the dependencies of tokens with the root.

1) Action: The main action verb in the requirement (mostly, with the dependency *ROOT*) is identified and called an action. The algorithm particularly distinguishes the type of actions as: *Nominal action* which has a noun and a verb together (e.g. *send a message*), *Boolean action* which can take a Boolean constraint (e.g. *is withdrawn*) and *Simple action* which has only an action verb (e.g. *send*).
   In addition, the algorithm also tries to identify the verb type(s) (transitive, dative, prepositional, etc.) as suggested in [21] to supplement the syntactic significance of action types. This is essential particularly when we rely on action types for relation formulation.
2) Subjects and Objects: The tokens with dependencies *subj* and *obj* (and their variants like *nsubj*, *pobj*, *dobj*, etc.) are identified mostly as Subjects and Objects, respectively. They can be noun chunks (e.g. *temperature of the battery*), compound nouns (e.g. *battery temperature*) or single tokens (e.g. *battery*) in the requirement.

Also, we noted that there are several requirements involving a logical comparison (identified as an adjective or an adverb) between the expressed quantities. In order to identify comparisons (e.g. *greater than*, *exceeds*, etc.) in the requirement, we utilize the exhaustive synonym hyperlinks from Roget’s Thesaurus [22] and map them to the corresponding equality (=), inequality (!=), inferiority (<, <=) and superiority (>, >=) symbols.

**Example:** From the sub-requirements ”[if the temperature of the battery is below Tmin] *or* [if the temperature of the battery exceeds Tmax], [charging approval has to be withdrawn]”, the system identifies *Battery_Temperature* and *Charging_Approval* as Subjects, *Tmin* and *Tmax* as Objects and *withdrawn* as a Boolean Action. Also, the comparison term *below* is mapped as < and *exceeds* is mapped as >.

## 3.3. Semantic entity identification

Semantic entities are tightly coupled with the end application which translates the parsed syntactic information to sequence diagrams and then to abstract test cases. The semantic

**Table 1**
Mapping of syntactic to semantic entities

| Syntactic entities | Semantic entities |
|---|---|
| Action | Signal |
| Action constraints | Attributes |
| Subject / Object | Actor / Component |

entities are defined from the perspective of interactions in a sequence diagram and are outlined below. The algorithm derives these entities from their syntactic counterparts[1].

1) Actor or Component: The participants involved in an interaction are defined as actors and components. To differentiate other participants from the system under test (SUT), component is always considered as the SUT.
2) Signal: The interaction between different participants is defined as a signal.
3) Attributes: The variables holding the status at different points of interaction are defined as attributes.
4) State: This refers to the initial, intermediate and final states of an interaction.

Semantic entities demand additional details for completeness. For example, if the value of an attribute is not given, it can not be initialized in its corresponding test case. Likewise, for each signal the corresponding actor needs to be identified. For each requirement, the direction of communication (*incoming*: towards the system under test or *outgoing*: from the system under test) should be identified. In cases where the algorithm lacks the desired ontology information, user input is demanded to update these values.

It is worth noting that the separation of the entities as syntactic (application independent but grammar dependent) and semantic (application dependent but grammar independent) gives more flexibility to the algorithm to be used in parts also in a different environment than the description language considered here. However, the mapping from the syntactic entities to their semantic counterparts can be completely automated with stricter rules or can be accomplished with user intervention and validation.

**Example:** From the sub-requirements "[if the temperature of the battery is below Tmin] *or* [if the temperature of the battery exceeds Tmax], [charging approval has to be withdrawn]", the identified Subjects (*Battery_Temperature* and *Charging_Approval*) are mapped as Signals and the identified Objects (*Tmin* and *Tmax*) are mapped as Attributes. Here, the identified Action *withdrawn* is also considered as an Attribute owing to the semantics of its corresponding Boolean Signal. Additionally, we can arrive at the Actor for *Battery_Temperature* as *battery*. However, the Actor of *Charging_Approval* is ambiguous (or rather unknown). Likewise, Attribute values should either be passed by the user or they remain uninitialized in the resulting test case.

[1]Note that the algorithm maps syntactic to semantic entities with more complex rules (including action types and verb types). In Table 1, we have presented only the most primitive ones for brevity. This difference is also detailed in the example where an Action is considered as an Attribute and a Subject is mapped to a Signal.

## 3.4. Transformation to requirement model

For the description of the formal requirements a simple text-based domain-specific language (DSL) is used, the ifak requirements description language (IRDL) [23]. This notation for requirement models was developed on the basis of UML sequence diagrams and is specially adapted to the needs of describing requirements as sequences. The IRDL defines a series of model elements (e.g. components, messages) with associated attributes (name, description, recipient, sender, etc.) and special model structures (behavior based on logical operators or time). Functional, behavior-based requirements are described textually using IRDL and can then be visualized graphically as sequence diagrams (Fig. 2).

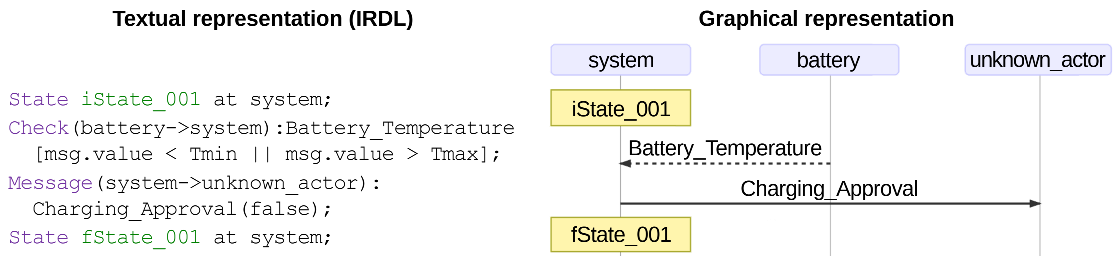
Once the entities are mapped and validated, the algorithm forms IRDL relations for each clause and then combines them together to form relations for the whole requirement. IRDL defines mainly two types of relations:

1) Incoming messages: SUT receives these messages provided the guard expression evaluates to be true and then continues to the next sequence in an interaction. IRDL defines this class of messages as 'Check'.
2) Outgoing messages: SUT sends these messages to other interaction participants with the content defined in the signal. In IRDL, these messages are denoted as 'Message'.

```
Check(Actor->Component): Signal[guard expression];
Message(Component->Actor): Signal(signal content);
```

As an intermediate result, the user is shown the formulated IRDL relations along with the sequence diagram corresponding to the requirement and is asked if the IRDL and the corresponding sequence diagram are correct. In case the user wants to further modify the relation formulation, the algorithm repeats from the mapping of syntactic entities to semantic entities. This continues until the user confirms the model is satisfactory.

**Example:** IRDL relations for the example requirement "If the temperature of the battery is below Tmin or it exceeds Tmax, charging approval has to be withdrawn", after the above-mentioned steps is shown in Fig. 2.



```
State iState_001 at system;
Check(battery->system):Battery_Temperature
  [msg.value < Tmin || msg.value > Tmax];
Message(system->unknown_actor):
  Charging_Approval(false);
State fState_001 at system;
```

**Figure 2:** Visualization of a requirement model in IRDL and as sequence diagram.

### 3.5. Model synthesis and test generation

The formalized requirements of the SUT can be combined to a specification model using existing methods for model synthesis [23]. The sequence elements described before, are transformed using a rule-based algorithm into equivalent elements of a UML state machine.

After model synthesis, test cases can be automatically generated from the state machine using an existing method for model-based test generation [24]. Selecting a specific graph-based coverage criteria such as ”all paths”, ”all decisions”, ”all places” or ”all transitions”, the state machine is transformed into a special form of a Petri net from which abstract test cases in the form of sequence diagrams can be generated. In this way, the approach allows modeling of even complex system behavior and applying graph-based coverage criteria to the entire system model.

## 4. Application

The toolchain for requirements-based model and test case generation presented in the previous section is applied to an industrial use case from the e-mobility domain. The use case describes a system for charging approval of an electric vehicle in interaction with a charging station. The industrial use case was defined by AKKA and aims to provide a typical basic scenario and development workflow in software development for an automotive electronic control unit (ECU). It does so by defining requirements, using model-based software development and deploying the functionality on an ECU.

The use case has to be seen in the context of an electric car battery that is supposed to be charged. The function “charging approval” implements a simple function, which decides upon specific input signals, if the charging process of the battery is allowed or not. For example, charging approval is given or withdrawn depending on the battery temperature, voltage or state of charge, the requested current is adjusted according to the battery temperature, and error behavior is handled for certain conditions. This is a continuous process, i.e. the signal values may change over time. A more detailed technical description of the industrial use case can be found in [25]. To fulfill the requirement of model-based software development, the module is implemented in Matlab Simulink. Matlab Simulink Coder is used to generate C/C++ code that can be compiled and deployed to the target. A Raspberry Pi is used to simulate some but not all aspects of an ECU. A basic overview of the charging approval system and its interfaces to the environment is given in Fig. 3.

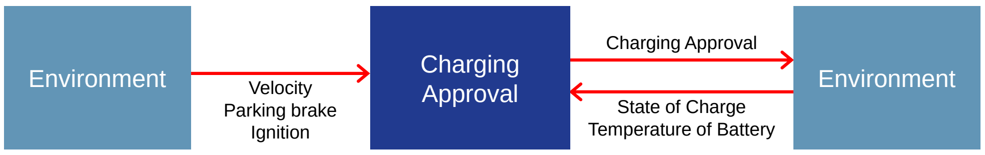


**Figure 3:** Process overview of charging approval system.

## 5. Results

The battery charging approval system described in the former section is used to evaluate the proposed method. We first define the used evaluation metrics and then demonstrate the results. To our knowledge, there are no available tools with similar input and output properties as our tool that enable a direct comparison.

### 5.1. Evaluation metrics

Let $R$ be the set of textual requirements. For a requirement $r \in R$, let $X_r$ be the set of expected artifacts and $Y_r$ be the set of generated artifacts. Here, *artifacts* refer to all the semantic entities including the relation indicators. Let $X = \bigcup_{r \in R} X_r$ denote the set of expected artifacts in all requirements and $Y = \bigcup_{r \in R} Y_r$ the set of generated artifacts in all requirements. Then we define the following metrics to measure the performance of the method.

1) **Completeness:** For an individual requirement, this metric denotes the number of expected artifacts $x \in X_r$ for which a corresponding (not necessarily identical) generated artifact $y \in Y_r$ exists, in relation to the total number of expected artifacts $|X_r|$.
2) **Correctness:** For an individual requirement, this metric denotes the number of generated artifacts $y \in Y_r$ for which a corresponding, semantically identical (up to naming conventions) expected artifact $x \in X_r$ exists, in relation to the total number of generated artifacts $|Y_r|$.
3) **Consistency:** This metric denotes the number of generated artifacts $y \in Y$ for which a corresponding expected artifact $x \in X$ exists and is used identically in all requirements $r \in R$, in relation to the total number of generated artifacts $|Y|$.

The *macro average* for completeness and correctness, respectively, is then given by the mean value of all individual percentage values for all $r \in R$. The *micro average* is given by the sum of all values in the numerator divided by the sum of all values in the denominator for all $r \in R$.

**Example:** In order to assert the evaluation metrics in detail, consider the requirement clause *'if the SoC of the battery is below SoC_max'*.

Expected: `Check(charging_management->system): Battery_SoC[msg.Ialue < SoC_max];`
Generated: `Check(battery->system): battery_SoC[msg.value < SoC_max];`

For the metric *completeness*, we check if all the expected artifacts (i.e. *Check*, *charging_management*, *Battery_SoC*, etc.) are generated by the algorithm. In this case, we can see that all of them were generated. For obtaining the *correctness*, we check if those generated artifacts are semantically correct. In this case, though we expect the actor *charging_management*, the algorithm generates *battery*. This reduces the value of correctness. If the algorithm generates *battery_SoC* for every occurrence of 'SoC of battery' across all the requirements, it is considered *consistent* for this artifact.

**Table 2**
Evaluation of the algorithm on the charging approval system

| | without domain knowledge | | with domain knowledge | |
|---|---|---|---|---|
| | **macro avg.** | **micro avg.** | **macro avg.** | **micro avg.** |
| **Completeness** | 78.2% | 79.8% | 81.4% | 84.1% |
| **Correctness** | 74.9% | 78.8% | 78.3% | 82.1% |
| **Consistency** | 94.1% | | 96.4% | |

## 5.2. Requirements formalization

As part of the demonstrated use case, AKKA has provided functional requirement documents describing the expected behavior for the relevant SUT. To apply the NLP-based requirements formalization method, each statement is treated as a separate entity for which a well-defined requirement model is created. Overall, the charging approval SUT is described by 14 separate requirement statements.

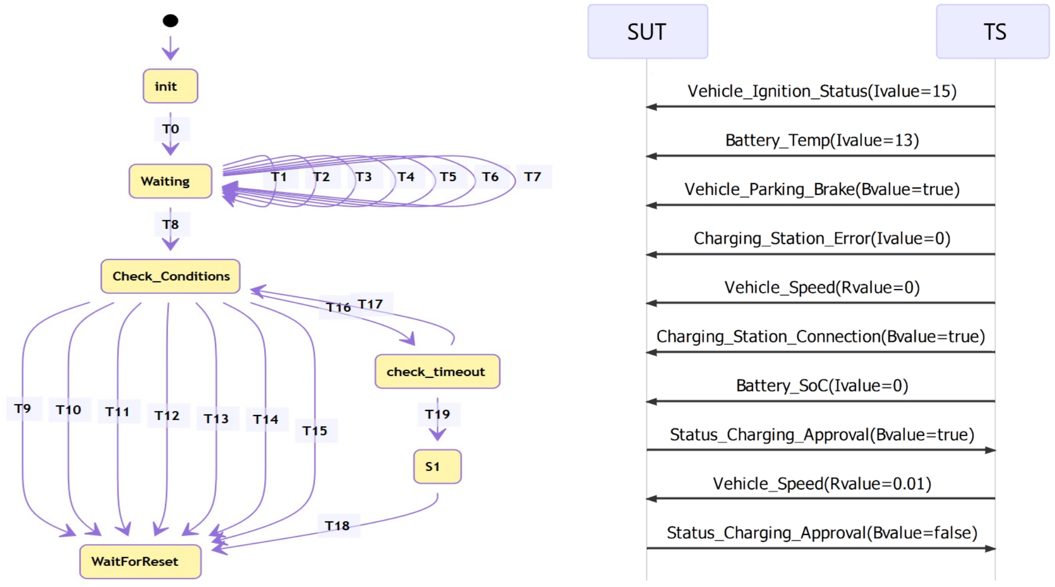
The results of our evaluation are shown in Table 2. We have determined the individual correctness and completeness values and calculated the macro and micro average for them. We avoided double counting of identical entity detections as not to skew the results. As mentioned above, if an actor or value of an attribute is not explicitly mentioned in the textual requirement, it cannot be detected by the algorithm. Therefore we also show the results using domain knowledge, which could be in the form of a predefined list of signals, attributes, etc. or integrated by direct user interaction from an expert with knowledge about the system.

As one can observe, the method shows good results, most of the signals and other artifacts were detected correctly and completely. Having a list of artifact declarations in advance produces even more accurate predictions. Thus, our NLP-based approach shows a good quality and supports the generation of the formal requirement model to a significant extent.

A comparison of the time for its creation, both with and without the provided tool is not measured directly. However, from our experience of former and the presented use case it takes a lot of time for a requirements engineer to get into the description language for sequence diagrams by reading documentations and having discussions, to create the logical structure and to add all the details to the model manually. The new semi-automated approach supports the user in a great manner. It gives a first proposal of the requirement model in a textual and graphical view and provides options for handeling unclear points. This should therefore save a lot of time, even though a manual review of the created model is still required.

## 5.3. Model synthesis and test generation

For the next step, the requirement models of the charging approval system were used as the input for model synthesis using ifak's prototypical tool ModGen. Since the NLP-based approach treats every requirement as a separate entity, it is also necessary to connect each requirement by modelling the boundaries explicitly. As a result, a graph-based representation of the functionality as described by the requirements was generated in the form of a UML state machine (Fig. 4). The generated model contains 6 states and 20 transitions with appropriate signals, guards and actions. The semantic as well as syntactic validity of the generated UML state machine could



init
T0
Waiting
T1
T2
T3
T4
T5
T6
T7
T8
Check_Conditions
T16
T17
check_timeout
T19
S1
T9
T10
T11
T12
T13
T14
T15
T18
WaitForReset
SUT
TS
Vehicle_Ignition_Status(Ivalue=15)
Battery_Temp(Ivalue=13)
Vehicle_Parking_Brake(Bvalue=true)
Charging_Station_Error(Ivalue=0)
Vehicle_Speed(Rvalue=0)
Charging_Station_Connection(Bvalue=true)
Battery_SoC(Ivalue=0)
Status_Charging_Approval(Bvalue=true)
Vehicle_Speed(Rvalue=0.01)
Status_Charging_Approval(Bvalue=false)

**Figure 4:** UML state machine of charging approval (left) and a test case visualized as a sequence diagram (right).

be confirmed by a thorough evaluation based on the initial requirement documents and by checking for deadlocks and livelocks. It could be shown that no further manual editing of the model is required for a full description of the behavior of the system.

In this evaluation, ifak's prototypical tool TCG with the coverage criteria "all-paths" was selected, which ensures that each possible path in the utilized model is covered by at least one test case. By utilizing the existing algorithm for test generation, a total of 73 test cases were generated. In Fig. 4, one of the generated test cases is visualized in the form of a sequence diagram. Here, a test system (TS) interacts with the SUT (charging approval) and provides a number of parameters, upon which the system decides if charging approval is given.

Overall, it can be shown that valid abstract test cases are generated based on the specification model. Using an appropriate framework for test case execution and a suitable test adapter, the generated test cases could be used for validation of the functional behavior of the SUT.

## 6. Conclusion

In this work, an NLP-based method for machine-aided model generation from textual requirements is presented. The method is designed to cover a wide range of requirements formulations without being restricted to a specific domain or format. Further, the generated requirement models are given in a user-friendly, comprehensible textual and graphical representation in the form of sequence diagrams.

We evaluated our approach on the industrial use case of a battery charging approval system and showed that the algorithm can produce complete, correct and consistent artifacts to a high degree. We have also shown how these artifacts are then used to create sequence diagrams for each requirement and transformed into a state machine for the entire specification model to finally generate abstract test cases. With the proposed semi-automated approach, we aim to reduce the human effort of creating test cases from textual requirements to validating the generated requirement models. In future versions of this prototypical implementation, we intend to refine the rule-based approach further, thus reducing the need for manual modifications. One possible solution to this regard could be training a Named Entity Recognition (NER) algorithm to identify the semantic entities, however at the cost of intensive labelling work. Another solution could be to rely on (pre-trained) Semantic Role Labels (SRL).

This study is still research-in-progress, since even more complex textual requirements have to be considered for future applications. The use of the methodology is also conceivable in other domains, such as in rail, industrial communication and automotive. In future work, we therefore intend to analyze how we can improve the method to cover more application domains.

## Acknowledgments

This research was funded by the German Federal Ministry of Education and Research (BMBF) within the ITEA 3 projects TESTOMAT under grant no. 01IS17026G and XIVT under grant no. 01IS18059E. We thank our former colleague Martin Reider and our research assistant Libin Kutty from ifak for the valuable contributions to this paper. We also thank AKKA Germany GmbH for providing an industrial use case for the evaluation of the presented method.

## References

[1] P. Ammann, J. Offutt, Introduction to Software Testing, 2nd ed., Cambridge University Press, 2017.

[2] M. J. Escalona, J. J. Gutierrez, M. Mejías, G. Aragón, I. Ramos, J. Torres, F. J. Domínguez, An overview on test generation from functional requirements, Journal of Systems and Software 84 (2011) 1379–1393.

[3] I. Ahsan, W. H. Butt, M. A. Ahmed, M. W. Anwar, A comprehensive investigation of natural language processing techniques and tools to generate automated test cases, in: ICC, 2017, pp. 1–10.

[4] V. Garousi, S. Bauer, M. Felderer, NLP-assisted software testing: A systematic mapping of the literature, Information and Software Technology 126 (2020).

[5] M. Riebisch, M. Hubner, Traceability-Driven Model Refinement for Test Case Generation, in: ECBS, 2005, pp. 113–120.

[6] C. Rupp, Requirements-Engineering und -Management: Aus der Praxis von klassisch bis agil, 6th ed., Hanser, 2014.

[7] A. Mavin, P. Wilkinson, A. Harwood, M. Novak, Easy Approach to Requirements Syntax (EARS), in: RE, 2009, pp. 317–322.

[8] G. Carvalho, F. Barros, A. Carvalho, A. Cavalcanti, A. Mota, A. Sampaio, NAT2TEST Tool: From Natural Language Requirements to Test Cases Based on CSP, in: SEFM, 2015, pp. 283–290.

[9] S. C. Allala, J. P. Sotomayor, D. Santiago, T. M. King, P. J. Clarke, Towards Transforming User Requirements to Test Cases Using MDE and NLP, in: COMPSAC, 2019, pp. 350–355.

[10] C. Nebut, F. Fleurey, Y. Le Traon, J.-M. Jezequel, Automatic test generation: A use case driven approach, IEEE Transactions on Software Engineering 32 (2006) 140–155.

[11] A. Goffi, A. Gorla, M. D. Ernst, M. Pezzè, Automatic generation of oracles for exceptional behaviors, in: ISSTA, 2016, pp. 213–224.

[12] A. Blasi, A. Goffi, K. Kuznetsov, A. Gorla, M. D. Ernst, M. Pezzè, S. D. Castellanos, Translating code comments to procedure specifications, in: ISSTA, 2018, pp. 242–253.

[13] C. Wang, F. Pastore, A. Goknil, L. Briand, Automatic Generation of Acceptance Test Cases from Use Case Specifications: an NLP-based Approach, IEEE Transactions on Software Engineering (2020) 1–38.

[14] T. Yue, S. Ali, M. Zhang, RTCM: a natural language based, automated, and practical test case generation framework, in: ISSTA, 2015, pp. 397–408.

[15] B. C. F. Silva, G. Carvalho, A. Sampaio, Test Case Generation from Natural Language Requirements Using CPN Simulation, in: SBMF, 2015, pp. 178–193.

[16] J. Fischbach, A. Vogelsang, D. Spies, A. Wehrle, M. Junker, D. Freudenstein, Specmate: Automated creation of test cases from acceptance criteria, in: ICST, 2020, pp. 321–331.

[17] spaCy, Industrial-strength Natural Language Processing in Python, 2020. URL: https://spacy.io/.

[18] H. Yang, A. de Roeck, V. Gervasi, A. Willis, B. Nuseibeh, Analysing anaphoric ambiguity in natural language requirements, Requirements Engineering 16 (2011) 163–189.

[19] S. Lappin, H. J. Leass, An algorithm for pronominal anaphora resolution, Computational Linguistics 20 (1994) 535–561.

[20] L. Qiu, M.-Y. Kan, T.-S. Chua, A Public Reference Implementation of the RAP Anaphora Resolution Algorithm, in: LREC, 2004, pp. 291–294.

[21] D. K. Deeptimahanti, R. Sanyal, An Innovative Approach for Generating Static UML Models from Natural Language Requirements, in: ASEA, 2008, pp. 147–163.

[22] Roget's Hyperlinked Thesaurus, Categories of notions, 2020. URL: http://www.roget.org/scripts/hier.php/?class=I&division=0&section=III.

[23] S. Magnus, T. Ruß, J. Krause, C. Diedrich, Modellsynthese für die Testfallgenerierung sowie Testdurchführung unter Nutzung von Methoden zur Netzwerkanalyse, at - Automatisierungstechnik 65 (2017) 73–86.

[24] J. Krause, Testfallgenerierung aus modellbasierten Systemspezifikationen auf der Basis von Petrinetzentfaltungen, Ph.D. thesis, Otto-von-Guericke-Universität Magdeburg, 2012.

[25] D. Grujic, T. Henning, E. J. C. García, A. Bergmann, Testing a Battery Management System via Criticality-based Rare Event Simulation, preprint, arXiv:2107.00530 [cs.SE], 2021.

# Cause-Effect Structures Behaving like Reaction Systems

Ludwik Czaja[1,2]

[1]Vistula University, Warsaw, Poland
[2]Institute of Informatics, The University of Warsaw, Poland

**Abstract**
Cause-effect (c-e) structures, a net-like algebraic formalism for describing and analysing systems, primarily parallel, may be adapted to work as Reaction Systems. This is acquired by a simple modification of the c-e structures' semantics

**Keywords**
cause-effect structure, reaction system, quasi semiring

## 1. Summary of elementary cause-effect structures

Cause-effect structures (c-e) are objects of an algebraic calculus called a *quasi-semiring*[1], for describing and analysing systems built up as nets. Among a number of similarities to Petri nets, the noticeable is graphic presentation of systems' dynamics. The complete presentation of c-e structures is in [Cza 2019].

**Definition 1.1** (set $\boldsymbol{F}[\mathbb{X}]$, quasi-semiring of formal polynomials)

Let $\mathbb{X}$ be a non-empty enumerable set. Its elements, called *nodes*, are counterparts of places in Petri nets. $\theta \notin \mathbb{X}$ is a symbol called *neutral*. It plays role of neutral element for operations on terms, called *formal polynomials over* $\mathbb{X}$. The names of nodes, symbol $\theta$, operators +, •, called addition and multiplication, and parentheses are symbols out of which formal polynomials are formed. A node's name and $\theta$ is a formal polynomial; if $K$ and $L$ are formal polynomials, then $(K + L)$ and $(K \bullet L)$ are too. Their set is denoted by $\boldsymbol{F}[\mathbb{X}]$. Assume stronger binding of • than +, which allows for dropping some parentheses. Addition and multiplication of formal polynomials is defined as follows: $K \oplus L = (K + L)$, $K \odot L = (K \bullet L)$. To simplify notation, let us use + and • instead of $\oplus$ and $\odot$. It is required that the system $\langle \boldsymbol{F}[\mathbb{X}], +, \bullet, \theta \rangle$ obeys the following equality axioms for all $K, L, M \in \boldsymbol{F}[\mathbb{X}]$, $x \in \mathbb{X}$:

(+) $\theta + K = K + \theta = K$ (•) $\theta \bullet K = K \bullet \theta = K$

29th International Workshop on Concurrency, Specification and Programming (CS&P'21)
lczaja@mimuw.edu.pl (L. Czaja)
0000-0003-4675-816X (L. Czaja)

[1]This calculus differs from the standard semiring by restricted distributivity law: $a \cdot (b + c) = a \cdot b + a \cdot c$ satisfied provided that $b \neq \theta$ if and only if $c \neq \theta$ where $\theta$ is a neutral element for both operations (simultaneity and nondeterministic choice). Hence the preposition "quasi". Note that due to the conditional distributivity, the calculus neither reduces to the single element $\theta$, nor makes both operations coincide.

(++) $K + K = K$ (••) $x \bullet x = x$
(+++) $K + L = L + K$ (• • •) $K \bullet L = L \bullet K$
(++++) $K + (L + M) = (K + L) + M$ (• • ••) $K \bullet (L \bullet M) = (K \bullet L) \bullet M$
(+•) If $L \neq \theta \Leftrightarrow M \neq \theta$ then $K \bullet (L + M) = K \bullet L + K \bullet M$

Algebraic system which obeys these axioms will be referred to as a *quasi-semiring of formal polynomials.* □

**Definition 1.2** (cause-effect structure, carrier, set ***CE***[$\mathbb{X}$])

A cause-effect structure (c-e structure) over $\mathbb{X}$ is a pair $U = \langle C, E\rangle$ of total functions:

$C$: $\mathbb{X} \to \boldsymbol{F}[\mathbb{X}]$ (*cause function*; nodes occuring in $C(x)$ are *causes* of $x$)
$E$: $\mathbb{X} \to \boldsymbol{F}[\mathbb{X}]$ (*effect function*; nodes occuring in $E(x)$ are *effects* of $x$)

such that $x$ occurs in the formal polynomial $C(y)$ iff $y$ occurs in $E(x)$. *Carrier* of $U$ is the set $car(U) = \{x \in \mathbb{X} : C(x) \neq \theta \vee E(x) \neq \theta\}$. $U$ is of *finite carrier* iff $|car(U)| < \infty$ (|...| denotes cardinality). The set of all c-e structures over $\mathbb{X}$ is denoted by ***CE***[$\mathbb{X}$]. Since $\mathbb{X}$ is fixed, we write just ***CE***. □

$C$ and $E$ are total, thus each c-e structure comprises all nodes from $\mathbb{X}$, also the isolated ones - those from outside of its carrier. In presenting c-e structures graphically, only their carriers are drawn. A representation of a c-e structure $U = \langle C, E\rangle$ as a set of annotated nodes is $\{x^{C(x)}_{E(x)} : x \in car(U)\}$. $U$ is also presented as a directed graph with $car(U)$ as set of vertices labelled with objects of the form $x^{C(x)}_{E(x)}$ ($x \in car(U)$); there is an edge from $x$ to $y$ iff $y$ occurs in the polynomial $E(x)$. Fig. 1 (a) and (b) depict two graphical presentations of the same c-e structure with carrier $\{a, b, c, d, e, f, g, h\}$; in (a) the encircled nodes comprise groups making products in formal polynomials in (b), where the sums of the products create families of the groups. This c-e structure is the set $\{a^{\theta}_{e}, b^{\theta}_{e}, c^{\theta}_{e}, d^{\theta}_{e}, e^{a+b\cdot c+c\cdot d}_{f\cdot g+h}, f^{e}_{\theta}, g^{e}_{\theta}, h^{e}_{\theta}\}$. Sometimes the empty subscript/superscript $\theta$ by node names is omitted.

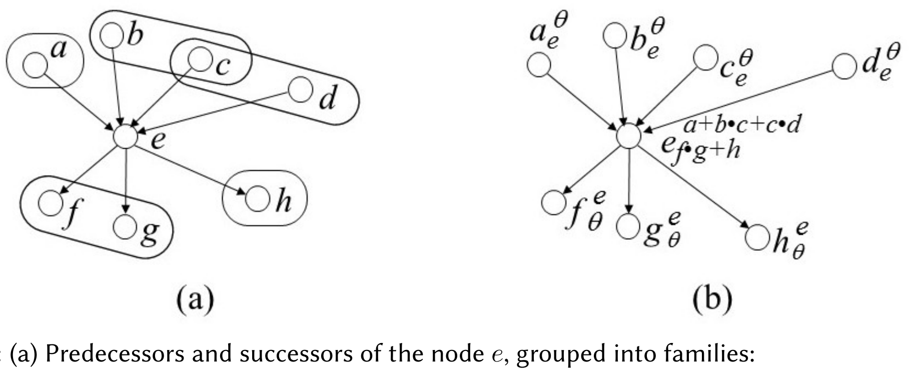


**Figure 1:** (a) Predecessors and successors of the node $e$, grouped into families: $\{\{a\}, \{b, c\}, \{c, d\}\}$ and $\{\{f, g\}, \{h\}\}$. (b) Notation by means of polynomials; the arrows are redundant: used only for graphic presentation of c-e structures.

**Definition 1.3** (addition and multiplication, monomial c-e structure)

For c-e structures $U = \langle C_U, E_U \rangle,\ V = \langle C_V, E_V \rangle$ define:
$U + V = \langle C_{U+V}, E_{U+V} \rangle = \langle C_U + C_V, E_U + E_V \rangle$ where
$(C_U + C_V)(x) = C_U(x) + C_V(x)$ and similarly for $E$
$U \bullet V = (C_{U \bullet V}, E_{U \bullet V}) = (C_U \bullet C_V, E_U \bullet E_V)$ where
$(C_U \bullet C_V)(x) = C_U(x) \bullet C_V(x)$ and similarly for $E$

(The same symbols "+" and "•" are used for operations on c-e structures and formal polynomials). $U$ is a *monomial* c-e structure iff each polynomial $C_U(x)$ and $E_U(x)$ is a monomial, i.e. does not comprise non-reducible (relative to equations in Definition 1.1) operation “+“. $\square$

Apart from the representation of c-e structures as a set $\{x^{C(x)}_{E(x)} : x \in car(U)\}$, their linear notation is used as the so-called "*arrow-expressions*": $\{x^\theta_y,\ y^x_\theta\}$ is an arrow, denoted as $x \to y$ and, consequently, $\{x^\theta_y, y^x_\theta\} \bullet \{y^\theta_z, z^y_\theta\} \bullet \{z^\theta_u, u^z_\theta\}..... = \{x^\theta_y, y^x_z, z^y_u, u^z_{....}.....\}$ is a chain, denoted as $x \to y \to z \to u....$ . Bidirectional arrow $x \leftrightarrow y$ denotes $x \to y \to x$ (equivalent to $y \to x \to y$), that is, the close cycle $\{x^y_y, y^x_x\}$. Chains and arrows in particular, may be combined into “arrow expressions” representing some c-e structures. For instance c-e structure $\{a^\theta_{x+y}, b^\theta_{x \bullet y}, x^{a \bullet b}_\theta, y^{a \bullet b}_\theta\}$ may be written as $(a \to x + a \to y) \bullet (b \to x) \bullet (b \to y)$.

The set ***CE*** with addition, multiplication and a distinguished element denoted also by $\theta$ and understood as the empty c-e structure $(\theta, \theta)$, where $\theta$ is a constant function $\theta(x) = \theta$ for all $x \in \mathbb{X}$, makes an algebraic system similar to that in Definition 1.1, the quasi-semiring of c-e structures. Therefore the Proposition 1.1:

**Proposition 1.1** The system $\langle \boldsymbol{CE}[\mathbb{X}], +, \bullet, \theta \rangle$ obeys the following equations for all $U, V, W \in \boldsymbol{CE}[\mathbb{X}],\ x, y \in \mathbb{X}$:

(+) $\theta + U = U + \theta = U$ (•) $\theta \bullet U = U \bullet \theta = U$
(++) $U + U = U$ (••) $(x \to y) \bullet (x \to y) = x \to y$
(+++) $U + V = U + V$ (• • •) $U \bullet V = V \bullet U$
(++++) $U + (V + W) = (U + V) + W$ (• • ••) $U \bullet (V \bullet W) = (U \bullet V) \bullet W$
(+•) If $C_V(x) \neq \theta \Leftrightarrow C_W(x) \neq \theta$ and $E_V(x) \neq \theta \Leftrightarrow E_W(x) \neq \theta$ then
$U \bullet (V + W) = U \bullet V + U \bullet W$ $\square$

**Definition 1.4** (partial order $\leq$; substructure, set ***SUB***$[V]$, firing component, set ***FC***, pre-set and post-set)

For $U, V \in$ ***CE*** let $U \leq V \Leftrightarrow V = U + V$; $\leq$ is a partial order in ***CE***. If $U \leq V$ then $U$ is a *substructure* of $V$; ***SUB***$[V]$= $\{U : U \leq V\}$ is the set of all substructures of $V$. For $A \subseteq$***CE***: $V \in A$ is *minimal* (wrt $\leq$) in $A$ iff $\forall W \in A$: $(W \leq V \Rightarrow W = V)$.
A minimal in ***CE***$\backslash\{\theta\}$ c-e structure $Q = \langle C_Q, E_Q \rangle$ is a *firing component* iff $Q$ is a monomial c-e structure and $C_Q(x) = \theta \Leftrightarrow E_Q(x) \neq \theta$ for any $x \in car(Q)$. The set of all firing components

is denoted by ***FC***, thus the set of all firing components of $U \in$***CE*** is ***FC***$[U] =$ ***SUB***$[U] \cap$ ***FC***. Let for $Q \in$ ***FC*** and $G \subseteq$ ***FC***:

$^\bullet Q = \{x \in car(Q) : C_Q(x) = \theta\}$ (*pre-set* or causes of $Q$)
$Q^\bullet = \{x \in car(Q) : E_Q(x) = \theta\}$ (*post-set* or effects of $Q$)
$^\bullet Q^\bullet = {}^\bullet Q \cup Q^\bullet$
$^\bullet G = \bigcup\limits_{Q \in G} {}^\bullet Q$ (*pre-set* or causes of $G$)
$G^\bullet = \bigcup\limits_{Q \in G} Q^\bullet$ (*post-set* or effects of $G$)
$^\bullet G^\bullet = {}^\bullet G \cup G^\bullet$ □

Thus, the firing component is a connected graph, due to the required minimality. Elements of the pre-set are its *causes* and elements of the post-set are its *effects*.

**Definition 1.5** (**salvo** - pairwise detached firing components; family ***FCS***)

Firing components $Q$ and $P$ are *detached* if and only if $^\bullet Q^\bullet \cap {}^\bullet P^\bullet = \varnothing$. Any set $G \subseteq$***FC*** of pairwise detached firing components is called their ***salvo***. The family of salvos is denoted by ***FCS***. So, if $G \subseteq$***FC***$[U]$ then ***FCS***$[U] \subseteq 2^{\mathbf{FC}[U]}$ for a c-e structure $U$, denotes a collection of salvos in $U$. The intention is that firing components in a salvo are capable of acting ("firing") simultaneously. This notion will be used in definition of parallel semantics of c-e structures.□

**Definition 1.6** (state of elementary c-e structures)

A *state* is a subset of the set of nodes: $s \subseteq \mathbb{X}$. The set of all states: $\mathbb{S} = 2^{\mathbb{X}}$. A node $x$ is *active* in the state $s$ iff $x \in s$ and *passive* otherwise. As in case of Petri nets, we say "$x$ holds a token" when $x$ is active. Obviously, the state might be defined equivalently as a two-valued function $s$: $\mathbb{X} \rightarrow \{0, 1\}$. □

**Definition 1.7** (sequential and parallel semantics of elementary c-e structures)

**Sequential**. For $Q \in$***FC***$[U]$, $s \in \mathbb{S}$, let $[[Q]] \subseteq \mathbb{S} \times \mathbb{S}$ be a binary relation defined as:
$(s,t) \in [[Q]]$ iff $^\bullet Q \subseteq s \wedge Q^\bullet \cap s = \varnothing \wedge t = (s \backslash {}^\bullet Q) \cup Q^\bullet$ ($Q$ transforms state $s$ into $t$).
*Semantics* $[[U]]$ of $U \in$***CE*** is: $[[U]] = \bigcup\limits_{Q \in FC[U]} [[Q]]$. $[[U]]^*$ is its reflexive and transitive closure, that is $(s,t) \in [[U]]^*$ if and only if $s = t$ or there exists a sequence of states $s_0, s_1, ..., s_n$ with $s = s_0$, $t = s_n$ and $(s_j, s_{j+1}) \in [[U]]$ for $j = 0, 1, ..., n-1$. State $t$ is *reachable* from $s$ in semantics [[ ]] and the sequence $s_0, s_1, ..., s_n$ is a *computation* performed by $U$.
**Parallel**. For a salvo $G \in$***FCS***$[U]$, $G \neq \varnothing$, relations $[[G]]$ and $[[U]]_{par}$ are defined in the same way as $[[Q]]$ and $[[U]]$ in the sequential case but with $Q$ replaced with $G$ and ***FC***$[U]$ replaced with ***FCS***$[U]$. Closure $[[U]]^*_{par}$, reachability and computation are defined as in the sequential case. □

Note that $[[U]] = \varnothing$ iff ***FC***$[U] = \varnothing$. Behaviour of elementary c-e structures may be thought of as a token game: if each node in a firing component's pre-set holds a token and none in its post-set does, then remove tokens from the pre-set and put them in the post-set. This is illustrated in Fig. 2. Properties inferred from above definitions are proved in [Cza 2019].

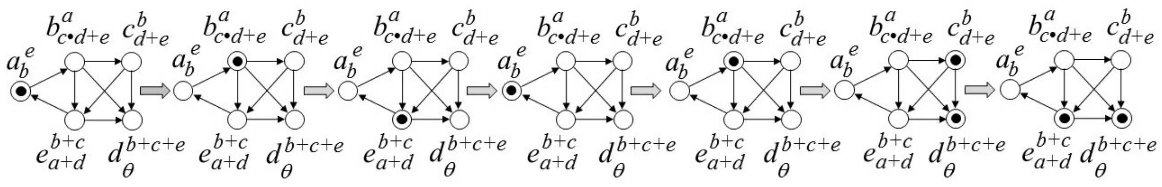


**Figure 2:** Example of activity (successive transformations) of elementary c-e structure. Its linear notation as an "arrow expression" is the following:
$(a \to b) + (b \to c) \bullet (b \to d) + (b \to e) + (c \to e) + (c \to d) + (e \to d) + (e \to a)$.

## 2. Summary of extended cause-effect structures

The structure of extended c-e structures, their firing component in particular, is the same as in Definitions 1.1 - 1.4. The extensions consist in redefining the state, treating the pre and post sets of firing components as multisets, and redefining semantics. It is assumed that with a given c-e structure $U \in$***CE***$[\mathbb{X}]$ (i.e. already constructed by operations introduced in Definition 1.3) and the set of its firing components ***FC***$[U] =$ ***SUB***$[U] \cap$***FC***, some additional information is associated. The following extensions of elementary c-e structures with this information will be obtained: multi-valued nodes, capacity of nodes and coefficients of monomials in polynomials annotating nodes (counterparts of weight of arrows in Petri nets), in particular a coefficient $\omega$ which represents inhibiting. To this end, a notation for multisets is convenient: let $\mathbb{N}$ be the set of natural numbers including 0 and $\mathbb{N}_\omega = \mathbb{N} \cup \{\omega\}$, where the value $\omega$ means infinity, that is $\omega > n$ for each $n \in \mathbb{N}$. A *multiset* over a base set $X$ is a (total) function $f\colon X \to \mathbb{N}_\omega$. If the set $\{x\colon f(x) \neq 0\}$ is finite then the linear-form notation is adopted for multisets , e.g.

| $X$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $f(X)$ | 2 | 0 | 3 | 1 | $\omega$ |

is denoted by $2 \otimes a + 3 \otimes c + d + \omega \otimes e$. A multiset is *zero* $\mathbb{O}$, when $\mathbb{O}(x) = 0$ for all $x$. Addition, subtraction and multiplication on multisets is defined: $(f+g)(x) = f(x)+g(x)$, $(f-g)(x) = f(x)-g(x)$ for $f(x) \geq g(x)$, $(f\cdot g)(x) = f(x)\cdot g(x)$, comparison of multisets: $f \leq g$ iff $f(x) \leq g(x)$ for all $x$. Assume the customary arithmetic of $\omega$: $\omega + n = \omega$, $\omega - n = \omega$, $\omega + \omega = \omega$ and additionally $0 - \omega = 0$.

**Definition 2.1** (state of extended c-e structures)

A state of extended c-e structure $U$ is a total function $s : car(U) \to \mathbb{N}$, thus a multiset over $car(U)$. The set of all states of $U$ is denoted by $\mathbb{S}$. □

**Definition 2.2.** (weights of monomials and capacity of nodes)

For a c-e structure $U = \langle C, E\rangle$ and its firing component $Q \in$***FC***$[U]$, let with the pre-set ${}^\bullet Q$ and post-set $Q^\bullet$ of $Q$, some multisets $\overline{{}^\bullet Q}\colon {}^\bullet Q \to \mathbb{N}_\omega$ and $\overline{Q^\bullet}\colon Q^\bullet \to \mathbb{N}_\omega$ be given as additional information. The value $\overline{{}^\bullet Q}(x)$ is a *weight* (or *multiplicity*) of monomial $E_Q(x)$ and the value

$\overline{Q^\bullet}(x)$ - a *weight* (or *multiplicity*) of monomial $C_Q(x)$. For $E_Q(x) = \theta$ or $C_Q(x) = \theta$ assume respectively $\overline{{}^\bullet Q}(x) = 0$ or $\overline{Q^\bullet}(x) = 0$ and additionally let $\overline{{}^\bullet Q}(x) = 0$ for $x \notin {}^\bullet Q$ and $\overline{Q^\bullet}(x) = 0$ for $x \notin Q^\bullet$. Let $cap(U)$ be a total function $cap(U) : car(U) \to \mathbb{N}_\omega \backslash \{0\}$, assigning a *capacity* to nodes in the set $car(U)$. A c-e structure endowed with such information is a *c-e structure-with-weighted monomials and capacity of nodes.* Note that this definition extends directly from the firing components onto their salvos. Indeed, For any non-empty salvo $G \in \textbf{\textit{FCS}}[U]$ and any $x \in {}^\bullet G^\bullet$ there exists exactly one firing component $Q \in G$, with $x \in {}^\bullet Q^\bullet$ (because firing components in $G$ are pairwise detached). Thus one may define $\overline{{}^\bullet G} = \overline{{}^\bullet Q}$ and $\overline{G^\bullet} = \overline{Q^\bullet}$. It should also be noticed that weights of cause or effect monomials in identical firing components appearing in different c-e structures, may be different. □

An effect monomial $E_Q(a)$ of a node $a \in {}^\bullet Q$, with weight $\overline{{}^\bullet Q}(a)$, is denoted by $\overline{{}^\bullet Q}(a) \otimes E_Q(a)$. Similarly for a cause monomial $C_Q(x)$ of a node $x \in Q^\bullet$ with weight $\overline{Q^\bullet}(x)$: $\overline{Q^\bullet}(x) \otimes C_Q(x)$. The coefficient representing weights will be abandoned if they are 1.

**Definition 2.3** (inhibitors)

For a firing component $Q \in \textbf{\textit{FC}}[U]$, let $inh[Q] = \{x \in {}^\bullet Q : \overline{{}^\bullet Q}(x) = \omega\}$, thus a set of nodes in the pre-set of $Q$, whose effect monomials $E_Q(x)$ are of weight $\omega$. The nodes in $inh[Q]$ will play role of *inhibiting nodes* of firing component $Q$. In accordance with the note in Definition 1.5 (pairwise detached firing components in salvos), the concept of inhibiting nodes extends directly onto the salvos: $inh[G] = \{x \in {}^\bullet G : \overline{{}^\bullet G}(x) = \omega\}$ where $G \in \textbf{\textit{FCS}}[U]$. The inhibiting nodes will be called *inhibitors.* □

**Definition 2.4.** (enabled firing components and enabled salvos)

For a firing component $Q \in \textbf{\textit{FC}}[U]$ and state $s$ let us define the formula: $enabled[Q](s)$ if and only if:

$$\begin{aligned}
&\forall x \in inh[Q] : \ s(x) = 0 \wedge \\
&\forall x \in {}^\bullet Q \backslash inh[Q] : \ s(x) > 0 \wedge \overline{{}^\bullet Q}(x) \leq s(x) \leq cap(U)(x) \wedge \\
&\forall x \in Q^\bullet : \overline{Q^\bullet}(x) + s(x) \leq cap(U)(x)
\end{aligned}$$

So, $Q$ is enabled in the state $s$ iff none of inhibiting nodes $x \in {}^\bullet Q$ contains a token and each remaining node in ${}^\bullet Q$ contains, with no fewer tokens than is the weight of its effect monomial $E_Q(x)$ and no more than capacity of each $x \in {}^\bullet Q$. Moreover, none of $x \in Q^\bullet$ holds more tokens than their number when increased by the weight of the cause monomial $C_Q(x)$, exceeds capacity of $x$. Evidently, for a salvo $G \in \textbf{\textit{FCS}}[U]$, the formula $enabled[G](s)$ is defined as above by replacing $Q$ with $G$ and $\textbf{\textit{FC}}[U]$ with $\textbf{\textit{FCS}}[U]$. □

**Definition 2.5.** (Sequential and parallel semantics of extended c-e structures)

**Sequential**. For $Q \in \textbf{\textit{FC}}[U]$ , let $[[Q]] \subseteq \mathbb{S} \times \mathbb{S}$ be a binary relation defined as:

$(s,t) \in [[Q]]$ iff $enabled[Q](s) \wedge t = (s - \overline{{}^{\bullet}Q}) + \overline{Q^{\bullet}} \leq cap(U)$ ($Q$ transforms state $s$ into $t$). *Semantics* $[[U]]$ of $U \in$***CE*** is $[[U]] = \bigcup_{Q \in \boldsymbol{FC}[U]} [[Q]]$. Closure $[[U]]^*$ and reachability and computation are defined as in Definition 1.7.

**Parallel**. For a salvo $G \in$***FCS***$[U]$, $G \neq \varnothing$, the relations $[[G]]$ and $[[U]]_{par}$ are defined in the same way as $[[Q]]$ and $[[U]]$ in the sequential case, but with $Q$ replaced with $G$ and ***FC***$[U]$ replaced with ***FCS***$[U]$. Closure $[[U]]^*_{par}$, reachability and computation are defined as in the sequential case. □

An example of using inhibitor is in Fig. 3. The weight of one effect (i.e. subscript) monomial $\omega \otimes z$ of the node $T$ is $\omega$, thus traversing arrow from $T$ to $z$ would require infinite number of tokens at $T$, which is impossible. Thus, $T$ is the inhibiting node in firing component $Q_0$ and ordinary - in firing component $Q_1$. In accordance with aforesaid notation of multisets, the weighted monomials are denoted by $w \otimes M$, where $w$ is the weight of $M$ - an unweighted monomial. The weight is skipped if $w = 1$. The corresponding arrows are dashed in the figures if $w = \omega$. In all graphic presentations of c-e structures, nodes of unbounded capacity $\omega$, for storing data (sets of tokens), will be drawn as bigger circles with double edge. Remaining nodes, the "control", are of capacity 1 and drawn as smaller circles.

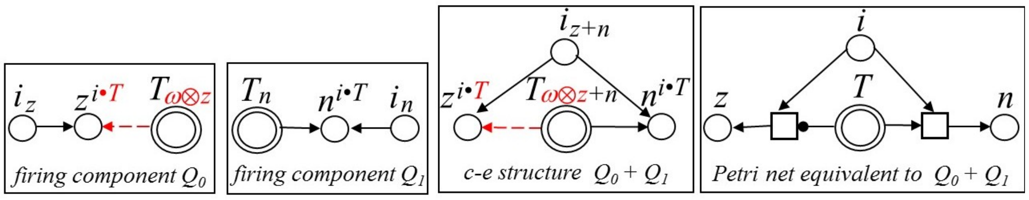


**Figure 3:** Construction of c-e structure $Q_0 + Q_1$ implementing "test zero". A token at the node $i$ starts testing contents of $T$; on termination, the token appears at $z$ (zero) if $T$ is empty and at $n$ (not zero) otherwise. The tested node $T$ plays two roles: inhibiting in $Q_0$ and ordinary in $Q_1$. Capacity of $T$ is infinite, whereas of remaining nodes is 1. The empty subscript/superscript $\theta$ is skipped. In the Petri net counterpart, the inhibiting arrow enters the left transition.

## 3. Basic notions of reaction systems

This outline of reaction systems [Ehr 2007], [Ehr 2017] is limited to the primary definitions, not their extensions, generalizations and properties established in this theory. The outline serves only as a reference base for the next section concerned with a translation of reaction systems into a version of cause-effect structures. That is why the original (in the literature) denotation of certain constituents of reaction systems, has been somewhat changed, in order to avoid naming collision with the cause-effect structures. But the new presentation is completely equivalent to original and only tailored to the purpose of the next section. The reaction systems will be referred to as ***RE***. The reaction systems as well as cause-effect structures, though devised to describe interactions or cooperation in a network of active objects, both are certain models of computing - in the broadest meaning of this word. Reaction systems are very inspiring new model of computing, if "computing" is to encompass interaction, not only "number crunching". That is why a certain attempt to translate the reaction systems into c-e structures is undertaken, though for a price of modifying semantics of the latter, but retaining their structure. There have been also other endeavours to relate reaction systems with some formal models, for instance [Kle 2011], [Dut 2018], [Bar 2018], [Gor 2018].

**Definition 3.1** (reaction system)

A reaction system $A$ is created of two sets: $A = (\mathbb{B}, \mathbb{R})$ where $\mathbb{B}$ is a set called a **background** (intended to comprise the so-called **entities)**, $\mathbb{R}$ is a set of **reactions**, each reaction $r \in \mathbb{R}$ created of three sets $r = (R_r, I_r, P_r)$ where $R_r \subseteq \mathbb{B}$ is a set of **reactants**, $I_r \subseteq \mathbb{B}$ is a set of **inhibitors** with $R_r \cap I_r = \varnothing$ and $P_r \subseteq \mathbb{B}$ is a set of **products**. The **initial state** of the system $A$ is a set $\mathbb{S}_0 \subseteq \mathbb{B}$. $\square$

In the literature, all the sets in Definition 3.1 are required to be finite. Though the concept of "entity" has not been formally adopted in the definitions, for the purpose of this section it will be understood as an object associated in a given state with an element of the background, likewise a "token", an abstract object, of intuitive, informal status in the c-e structures or Petri nets. Hence, the set phrase "entity present in..." in writings on the reaction systems. Since the notion of "state" of reaction systems is analogous to that in elementary Petri nets (and elementary c-e structures), meaning of the word "entity" may be thought of as an element of state.

**Definition 3.2** (state and enabled reactions)

A state $\mathbb{S}$ of reaction system $A$ is a subset of its background: $\mathbb{S} \subseteq \mathbb{B}$. The intention is that $\mathbb{S}$ be the set of those background's members, which comprise entities. A reaction $r = (R_r, I_r, P_r)$ is **enabled** in a state $\mathbb{S}$ iff $R_r \subseteq \mathbb{S}$ and $I_r \cap \mathbb{S} = \varnothing$. $\square$

**Definition 3.3** (semantics of reaction systems - change of state)

The **result** of a reaction $r$ in a state $\mathbb{S}$ is the set $P_r$ if $r$ is enabled at $\mathbb{S}$ and the empty set $\varnothing$ otherwise. This result is denoted by $res_r(\mathbb{S})$. The result of the reaction system $A$ in a state $\mathbb{S}$ is the union of all its reactions in this state:

$$res_A(\mathbb{S}) = \bigcup_{r \in \mathbb{R}} res_r(\mathbb{S})$$

$\square$

Note that on completion (if it exists) of reaction system work, the entities in the difference of sets $\mathbb{S} \backslash res_A(\mathbb{S})$ disappear. Note also two features differing the reaction systems from cause-effect structures (or Petri nets): the lack of conflicts and possible absorption of reactants by products.

These features will be taken into account in definition of semantics of reaction c-e structures in the next section.

**Example 3.1** (test zero)

The "test zero" task (Fig. 3) may be realized in the reaction system $A = (\mathbb{B}, \mathbb{R})$ with $\mathbb{B} = \{i, z, n, T\}$, $\mathbb{R} = \{r1, r2\}$ where $r1 = (\{i\}, \{T\}, \{z\})$, $r2 = (\{i, T\}, \varnothing, \{n\})$. Result of this system's work depends on the initial state $\mathbb{S}_0$ (i.e. what the environment supplies): if $\mathbb{S}_0 = \{i, T\}$ then the result is $\{n\}$ ("not zero" - presence of entity at $T$), if $\mathbb{S}_0 = \{i\}$ then the result is $\{z\}$ ("zero" - absence of entity at $T$). Reactant $i$ initiates work of the system, while $T$ is tested for presence/absence of entity. Evolution of the system $A = (\{i, z, n, T\}, \{r1, r2\})$ with initial state $\{i, T\}$ is the following:

$res_{r1}(\{i, T\}) = \varnothing$ (because $\{i\} \subseteq \{i, T\}$ and $\{T\} \cap \{i, T\} \neq \varnothing$)
$res_{r2}(\{i, T\}) = \{n\}$ (because $\{i, T\} \subseteq \{i, T\}$ and $\varnothing \cap \{i, T\} = \varnothing$) thus
$res_A(\{i, T\}) = res_{r1}(\{i, T\}) \cup res_{r2}(\{i, T\}) = \{n\}$ ("not zero")

Evolution of the system $A$ with initial state $\{i\}$ is the following:

$res_{r1}(\{i\}) = \{z\}$ (because $\{i\} \subseteq \{i\}$ and $\{T\} \cap \{i\} = \varnothing$)
$res_{r2}(\{i\}) = \varnothing$ (because $\{i, T\} \nsubseteq \{i\}$ and $\varnothing \cap \{i\} = \varnothing$) thus
$res_A(\{i\}) = res_{r1}(\{i\}) \cup res_{r2}(\{i\}) = \{z\}$ ("zero")

More complex example of evolution of a reaction system, that is its consecutive state changes, is in section 4.

## 4. Cause-effect structures working similarly to reaction systems

The objective is to build a system structurally identical with c-e structures but working like reaction systems. Let us call them "reaction c-e structures" and denote by ***RECE***. The evident counterparts of some objects in reaction c-e structures and reaction systems are the following:

*nodes*$\longleftrightarrow$*elements of background*,
*firing components*$\longleftrightarrow$*reactions*,
*causes in a firing component*$\longleftrightarrow$*reactants in a reaction*,
*effects in a firing component*$\longleftrightarrow$*products in a reaction*,
*inhibitors in a firing component*$\longleftrightarrow$*inhibitors in a reaction*,
*tokens*$\longleftrightarrow$*entities.*

The state of reaction c-e structures is a restriction of the state introduced in Definition 2.1 with added "$\omega$" to the range (codomain).

**Definition 4.1** (state)

A state of reaction c-e structure $U$ is a total function $s : car(U) \to \{0,1,\omega\}$. The set of all states of $U$ is denoted by $\mathbb{S}$. In the following, symbols 0 and 1 will be treated as logical values of *false* and *true* respectively and operations of propositional calculus on them will be applied. Moreover, operations $\vee$, $\wedge$ on $\omega$, are defined as: $0 \vee \omega = \omega \vee 0 = \omega$, $0 \wedge \omega = \omega \wedge 0 = 0$, $1 \vee \omega = \omega \vee 1 = \omega$, $1 \wedge \omega = \omega \wedge 1 = 1$, $\omega \vee \omega = \omega \wedge \omega = \omega$, $\neg\omega = 0$. As formerly, $\omega$ will be used for inhibiting actions. Interpretation of 0 and 1 as *false* and *true* is justified by absorption property of entities in reaction systems and will be made formal in Definition 4.6. □

The lack of conflicts requires introducing for reaction c-e structures concept called here "volley", to differ it from "salvo" for c-e structures introduced in Definition 1.5.

**Definition 4.2** (weights of monomials and inconsistent firing components)

Given a c-e structure $U = \langle C, E\rangle$ and its firing component $Q \in$***FC***$[U]$, let along with the pre-set $^\bullet Q$ and post-set $Q^\bullet$ of $Q$, some functions
$\overline{^\bullet Q}$: $^\bullet Q \to$ {0,1,$\omega$} and $\overline{Q^\bullet}$: $Q^\bullet \to$ {0,1,$\omega$} be given as additional information. The value $\overline{^\bullet Q}(x)$ is called a *weight* of monomial $E_Q(x)$ and the value $\overline{Q^\bullet}(x)$ - a *weight* of monomial $C_Q(x)$. For $E_Q(x) = \theta$ or $C_Q(x) = \theta$ assume respectively $\overline{^\bullet Q}(x) = 0$ or $\overline{Q^\bullet}(x) = 0$. Additionally, $\overline{^\bullet Q}(x) = 0$ for $x \notin {}^\bullet Q$ and $\overline{Q^\bullet}(x) = 0$ for $x \notin Q^\bullet$. As formerly, $\omega$ is interpreted as a "disable signal" and used for defining inhibiting nodes. Firing components $Q$ and $P$ are *inconsistent* if for a certain $x \in {}^\bullet Q^\bullet \cap {}^\bullet P^\bullet$, weights $\overline{^\bullet Q}(x)$ and $\overline{^\bullet P}(x)$ or $\overline{Q^\bullet}(x)$ and $\overline{P^\bullet}(x)$ are different: $\overline{^\bullet Q}(x) \neq \overline{^\bullet P}(x)$ or $\overline{Q^\bullet}(x) \neq \overline{P^\bullet}(x)$. Note that detached $Q$ and $P$ (Definition 1.5) are not inconsistent. □

**Remark**. Inconsistency of firing components must be avoided so that to obtain free of conflicts behaviour of the reaction c-e structures. The inconsistency is exemplified by the following c-e structure: $U = \{x_{0\otimes y+z},\ y^x,\ z^x\}$ containing two firing components: $Q = \{x_{0\otimes y},\ y^x\}$, $P = \{x_z\ z^x\}$, thus $\overline{^\bullet Q}(x) = 0$, $\overline{^\bullet P}(x) = 1$. In the initial state $s$ with $s(x) = 1$, $s(y) = s(z) = 0$ the parallel execution of the set of firing components $G = \{Q, P\}$ yields undetermined state of node $x$: it might be either 1 or 0. $U$ is a direct translation of the reaction system $(\{x, y, z\}, \{Q, P\}, \{x\})$ with $Q = (\{x\}, \varnothing, \{x, y\})$, $P = (\{x\}, \varnothing, \{z\})$, which performs state transformation $\{x\} \to \{x, y, z\}$. The same transformation of state is performed by the reaction c-e structure $V = \{x_{0\otimes y+0\otimes z},\ y^x,\ z^x\}$, leading to $t(x) = t(y) = t(z) = 1$.

**Definition 4.3 (volley** - simultaneous firing, family ***FCV***, extension of weight functions)

Any set $G \subseteq$***FC*** without inconsistent firing components is called their *volley*. The family of volleys is denoted by ***FCV***. So, if $G \subseteq$***FC***$[U]$ then ***FCV***$[U] \subseteq 2^{\mathbf{FC}[U]}$ for a c-e structure $U$, denotes a collection of volleys in $U$. The pre-set $^\bullet G$ and post-set $G^\bullet$ of a volley $G$ are defined as in Definition 1.4. Extension of the weight functions $\overline{^\bullet Q}$ and $\overline{Q^\bullet}$ onto the volley $G$ are defined as follows:

$$\overline{^\bullet G}(x) = \begin{cases} \overline{^\bullet Q}(x) & \text{for arbitrary } Q \text{ if it belongs to } G \\ 0 & \text{else} \end{cases}$$

$$\overline{G^\bullet}(x) = \begin{cases} \overline{Q^\bullet}(x) & \text{for arbitrary } Q \text{ if it belongs to } G \\ 0 & \text{else} \end{cases}$$

This is a correct definition, since for any firing components $Q$ and $P$ in $G$: $\overline{{}^\bullet Q}(x) = \overline{{}^\bullet P}(x)$ if $x \in {}^\bullet G$ and $\overline{Q^\bullet}(x) = \overline{P^\bullet}(x)$ if $x \in G^\bullet$. Functions $\overline{{}^\bullet G}$ and $\overline{G^\bullet}$ will be used in definition of reaction c-e structures semantics. $\square$

**Definition 4.4** (inhibitors)

As in Definition 2.3, for a firing component $Q \in \boldsymbol{FC}[U]$, the set $inh[Q] = \{x \in {}^\bullet Q : \overline{{}^\bullet Q}(x) = \omega\}$ comprises all nodes in the pre-set of $Q$, whose effect monomials $E_Q(x)$ are of weight $\omega$. This also extends onto the volleys: $inh[G] = \{x \in {}^\bullet G : \overline{{}^\bullet G}(x) = \omega\}$ where $G \in \boldsymbol{FCV}[U]$. $\square$

**Definition 4.5** (enabled firing components and enabled volleys)

For a firing component $Q \in \mathbf{FC}[U]$ and state $s$ let the formula $enabled[Q](\mathbf{s})$ be defined as:

$\forall x \in inh[Q] : \; s(x) = 0 \wedge \forall x \in \; {}^\bullet Q \backslash inh[Q] : s(x) = 1$

For a volley $G \in \boldsymbol{FCV}[U]$, the formula $enabled[G](s)$ is defined as above by replacing $Q$ with $G$ and $\boldsymbol{FC}[U]$ with $\boldsymbol{FCV}[U]$.

In the "token game" metaphor, $Q$ (or $G$) is enabled in the state $s$ iff none of inhibiting nodes $x \in {}^\bullet Q$ (or $x \in {}^\bullet G$) contains a token and each remaining node contains. As formerly, the inhibiting nodes will be called *inhibitors*. $\square$

**Definition 4.6** (semantics [[ ]] of reaction c-e structures)

For a volley $G \in \boldsymbol{FCV}[U], G \neq \varnothing$ let $[[G]] \subseteq \mathbb{S} \times \mathbb{S}$ be a binary relation defined as:
$(s,t) \in [[G]]$ iff $enabled[G](s) \wedge \forall x \in car(U) : \; t(x) = (s(x) \wedge \neg \overline{{}^\bullet G}(x)) \vee \overline{G^\bullet}(x)$
(say: $G$ transforms state $s$ into $t$). *Semantics* $[[U]]$ of $U \in \boldsymbol{RECE}$ is $\bigcup\limits_{G \in FCV[U]} [[G]]$, for any maximal volley $G$, i.e. if $G \subseteq G' \in \boldsymbol{FCV}[U]$ and
$(s,t) \in [[G']]$ then $G = G'$. Closure $[[U]]^*$ and reachability and computation are defined as in Definition 1.7. $\square$

In Definition 4.6, if $x \in {}^\bullet G$ then $\overline{{}^\bullet G}(x) \neq \omega$, because the volley $G$ is enabled in the state $s$ thus $\overline{{}^\bullet G}(x)$ is 0 or 1. If $x \notin {}^\bullet G^\bullet$ then $\overline{{}^\bullet G}(x) = 0$ and $\overline{G^\bullet}(x) = 0$ hence $\neg \overline{{}^\bullet G}(x) = 1$, thus $t(x) = s(x)$ (strictly, "=" means equivalence). Evidently, formula $(s(x) \wedge \neg \overline{{}^\bullet G}(x)) \vee \overline{G^\bullet}(x)$ is equivalent to $(s(x) \Rightarrow \overline{{}^\bullet G}(x)) \Rightarrow \overline{G^\bullet}(x)$. Note also that description of semantics by means of this propositional formula is justified by the property of reaction systems: presence of token ("entity") at a node absorbs another token arriving in this node. Another property of reaction systems, the lack of conflicts between different reactions, takes place in reaction c-e structures due to the lack of inconsistent firing components in $G \in \boldsymbol{FCV}[U]$.

**Example 4.1** (reaction c-e structure assembling a chemical molecule)

A description of creating some chemical molecules presents the reaction system

$$A = (\{C, H, O, U, V, W, X, Y, Z\}, \{r1, r2, r3, r4, r5, r6\}),$$

starting with initial state $\{C, H, O\}$ and with reactions defined as follows:

| | | | |
|---|---|---|---|
| $r1 = (\{C, H\}, \varnothing, \{U\})$ | $U$ contains molecule | $CH$ | (methylidyne) |
| $r2 = (\{U, C, H\}, \varnothing, \{V\})$ | $V$ contains molecule | $C_2H_2$ | (acetylene) |
| $r3 = (\{V, H\}, \varnothing, \{W\})$ | $W$ contains molecule | $C_2H_3$ | (ethylenyl) |
| $r4 = (\{W, H\}, \varnothing, \{X\})$ | $X$ contains molecule | $C_2H_4$ | (ethylene) |
| $r5 = (\{X, H, O\}, \varnothing, \{Y\})$ | $Y$ contains molecule | $C_2H_5O$ | (ethoxide) |
| $r6 = (\{Y, H\}, \varnothing, \{Z\})$ | $Z$ contains molecule | $C_2H_5OH$ | (ethanol) |

A diagram of the final result, that is the ethanol molecule, is in Fig. 4 and a certain translation of this reaction system into a reaction c-e structure is depicted in Fig. 5. Successive steps of the reaction system evolution are the following:

$res_{r1}(\{C, H, O\}) = \{U\}$ (since $\{C, H\} \subseteq \{C, H, O\}$ and $\varnothing \cap \{C, H, O\} = \varnothing$)
$res_{r2}(\{C, H, O\}) = res_{r3}(\{C, H, O\}) = res_{r4}(\{C, H, O\}) = res_{r5}(\{C, H, O\}) =$
$res_{r6}(\{C, H, O\}) = \varnothing$ thus
$res_A(\{C, H, O\}) = \{U\}$

$res_{r1}(\{C, H, O, U\}) = \{U\}$ (since $\{C, H\} \subseteq \{C, H, O, U\}$ and
$\varnothing \cap \{C, H, O, U\} = \varnothing$)
$res_{r2}(\{C, H, O, U\}) = \{V\}$ (since $\{C, H, U\} \subseteq \{C, H, O, U\}$ and
$\varnothing \cap \{C, H, O, U\} = \varnothing$)
$res_{r3}(\{C, H, O, U\}) = res_{r4}(\{C, H, U\}) = res_{r5}(\{C, H, O, U\}) =$
$res_{r6}(\{C, H, O, U\}) = \varnothing$ thus
$res_A(\{C, H, O, U\}) = \{U, V\}$

$res_{r1}(\{C, H, O, U, V\}) = \{U\}$ (since $\{C, H\} \subseteq \{C, H, O, U, V\}$ and
$\varnothing \cap \{C, H, O, U, V\} = \varnothing$)
$res_{r2}(\{C, H, O, U, V\}) = \{V\}$ (since $\{C, H, U\} \subseteq \{C, H, O, U, V\}$ and
$\varnothing \cap \{C, H, O, U, V\} = \varnothing$)
$res_{r3}(\{C, H, O, U, V\}) = \{W\}$ (since $\{H, V\} \subseteq \{C, H, O, U, V\}$ and
$\varnothing \cap \{C, H, O, U, V\} = \varnothing$)
$res_{r4}(\{C, H, O, U, V\}) = res_{r5}(\{C, H, O, U, V\}) = res_{r6}(\{C, H, O, U, V\}) = \varnothing$ thus
$res_A(\{C, H, O, U, V\}) = \{U, V, W\}$

$res_{r1}(\{C, H, O, U, V, W\}) = \{U\}$ (since $\{C, H\} \subseteq \{C, H, O, U, V, W\}$ and $\varnothing \cap \{C, H, O, U, V, W\} =$
$\varnothing$)
$res_{r2}(\{C, H, O, U, V, W\}) = \{V\}$ (since $\{C, H, U\} \subseteq \{C, H, O, U, V, W\}$ and $\varnothing \cap$
$\{C, H, O, U, V, W\} = \varnothing$)

$res_{r3}(\{C,H,O,U,V,W\}) = \{W\}$ (since $\{H,V\} \subseteq \{C,H,O,U,V,W\}$ and $\varnothing \cap \{C,H,O,U,V,W\} = \varnothing$)
$res_{r4}(\{C,H,O,U,V,W\}) = \{X\}$ (since $\{H,W\} \subseteq \{C,H,O,U,V,W\}$ and $\varnothing \cap \{C,H,O,U,V,W\} = \varnothing$)
$res_{r5}(\{C,H,O,U,V,W\}) = res_{r6}(\{C,H,O,U,V,W\}) = \varnothing$ thus
$res_A(\{C,H,O,U,V,W\}) = \{U,V,W,X\}$

$res_{r1}(\{C,H,O,U,V,W,X\}) = \{U\}$ (since $\{C,H\} \subseteq \{C,H,O,U,V,W,X\}$ and $\varnothing \cap \{C,H,O,U,V,W,X\} = \varnothing$)
$res_{r2}(\{C,H,O,U,V,W,X\}) = \{V\}$ (since $\{C,H,U\} \subseteq \{C,H,O,U,V,W,X\}$ and $\varnothing \cap \{C,H,O,U,V,W,X\} = \varnothing$)
$res_{r3}(\{C,H,O,U,V,W,X\}) = \{W\}$ (since $\{H,V\} \subseteq \{C,H,O,U,V,W,X\}$ and $\varnothing \cap \{C,H,O,U,V,W,X\} = \varnothing$)
$res_{r4}(\{C,H,O,U,V,W,X\}) = \{X\}$ (since $\{H,W\} \subseteq \{C,H,O,U,V,W,X\}$ and $\varnothing \cap \{C,H,O,U,V,W,X\} = \varnothing$)
$res_{r5}(\{C,H,O,U,V,W,X\}) = \{Y\}$ (since $\{H,O,X\} \subseteq \{C,H,O,U,V,W,X\}$ and $\varnothing \cap \{C,H,O,U,V,W,X\} = \varnothing$)
$res_{r6}(\{C,H,O,U,V,W,X\}) = \varnothing)$ thus
$res_A(\{C,H,O,U,V,W,X\}) = \{U,V,W,X,Y\}$

$res_{r1}(\{C,H,O,U,V,W,X,Y\}) = \{U\}$ (since $\{C,H\} \subseteq \{C,H,O,U,V,W,X,Y\}$ and $\varnothing \cap \{C,H,O,U,V,W,X,Y\} = \varnothing$)
$res_{r2}(\{C,H,O,U,V,W,X,Y\}) = \{V\}$ (since $\{C,H,U\} \subseteq \{C,H,O,U,V,W,X,Y\}$ and $\varnothing \cap \{C,H,O,U,V,W,X,Y\} = \varnothing$)
$res_{r3}(\{C,H,O,U,V,W,X,Y\}) = \{W\}$ (since $\{H,V\} \subseteq \{C,H,O,U,V,W,X,Y\}$ and $\varnothing \cap \{C,H,O,U,V,W,X,Y\} = \varnothing$)
$res_{r4}(\{C,H,O,U,V,W,X,Y\}) = \{X\}$ (since $\{H,W\} \subseteq \{C,H,O,U,V,W,X,Y\}$ and $\varnothing \cap \{C,H,O,U,V,W,X,Y\} = \varnothing$)
$res_{r5}(\{C,H,O,U,V,W,X,Y\}) = \{Y\}$ (since $\{H,O,X\} \subseteq \{C,H,O,U,V,W,X,Y\}$ and $\varnothing \cap \{C,H,O,U,V,W,X,Y\} = \varnothing$)
$res_{r6}(\{C,H,O,U,V,W,X,Y\}) = \{Z\}$ (since $\{H,Y\} \subseteq \{C,H,O,U,V,W,X,Y\}$ and $\varnothing \cap \{C,H,O,U,V,W,X,Y\} = \varnothing$) thus
$res_A(\{C,H,O,U,V,W,X,Y\}) = \{U,V,W,X,Y,Z\}$

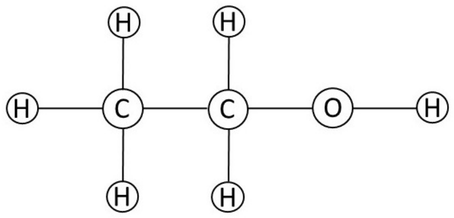


**Figure 4:** Diagram of ethanol. C, H, O - symbols of Carbon, Hydrogen and Oxygen atoms.

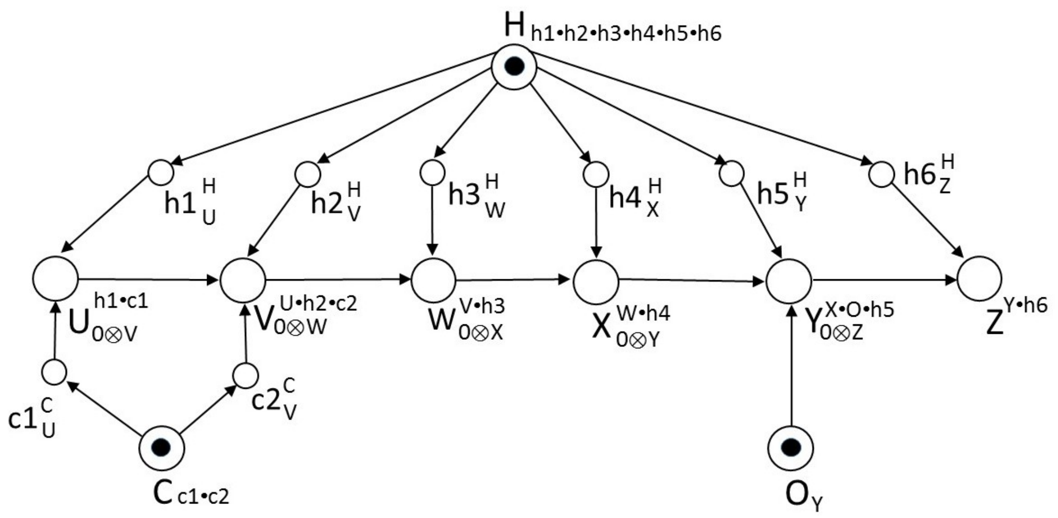


**Figure 5:** A c-e structure with initial state $s(C) = s(H) = s(O) = 1$ (and empty remaining nodes) imitating behaviour of reaction system $A$ specified in Example 4.1. Regarding it as a translation of $A$, note that nodes $c1, c2, h1, h2, h3, h4, h5, h6$ are some "artifacts" of this fairly liberal translation and have no counterparts in $A$. Intuitively, they might be seen as holding single atoms of elements $C$ and $H$.

## References

[Bar 2018] Barbutti R., Bove P., Gori R., Levi F., Milazzo P., *Simulating Gene Regulatory Networks using Reaction Systems*, Proceedings of Concurrency, Specification and Programming 2018, pp. 119-132

[Cza 2019] Czaja L. *Cause-Effect Structures. An Algebra of Nets with Examples of Application*, Lecture Notes in Networks and Systems 45, Springer 2019

[Dut 2018] Dutta S., Jankowski A., Rozenberg G., Skowron A., *Linking Reaction Systems with Rough Sets.* Fundamenta Informaticae 165(3-4): 283-302 (2019)

[Ehr 2007] Ehrenfeucht A., Rozenberg G.: *Reaction systems.* Fundamenta Informaticae 75 (2007), pp. 263-280

[Ehr 2017] Ehrenfeucht A, Petre I., Rozenberg G., *Reaction Systems: A Model of Computation Inspired by the Functioning of the Living Cell*, in: The Role of Theory in Computer Science, Essays Dedicated to Janusz Brzozowski, edited by Stavros Konstantinidis, Nelma Moreira, Rogério Reis and Jeffrey Shallit, World Scientific 2017, pp.1-32

[Gor 2018] Gori R., Gruska D., Milazzo P., *Hidden States in Reaction Systems*, Proceedings of Concurrency, Specification and Programming 2018, pp. 133-144

[Kle 2011] Kleijn J., Kountny M., Rozenberg G., *Modelling Reaction Systems with Petri Nets*, in: Proc. of the International Workshop on Biological Processes & Petri Nets (BioPPN 2011), pp. 36–52

# Interactive Granular Computing Connecting Abstract and Physical Worlds: An Example

Soma Dutta[1], Andrzej Skowron[2,3]

[1]University of Warmia and Mazury in Olsztyn, Słoneczna 54, 10-710 Olsztyn, Poland
[2]Systems Research Institute, Polish Academy of Sciences, Newelska 6, 01-447 Warsaw, Poland
[3]Digital Science and Technology Centre, UKSW, Wóycickiego 1/3; b. 21, 01-938 Warsaw, Poland

**Abstract**

This short paper is an attempt to clarify the role of Interactive Granular Computing (IGrC) as a computation model which respects that a real cognition about a real physical complex phenomenon and making decisions based on that cannot be formalized only being in the language of mathematics. In this regard, the paper focuses on presenting a real life example of computation where in order to move forward, without stumbling over the obstacles, a blind person needs to explore and learn the surrounding environment through interactions with the environment. The paper simply describes different components and features of IGrC model in the light of the concerned example and explains how this computing model has the potential to handle the grounding problem by bridging a connection between the abstract mathematical modeling and the real physical semantics.

**Keywords**

interactions, granular computing, perception, knowledge specification, implementational language, complex granule, informational granule, grounding problem, dynamic transition relation

## 1. Introduction

In a few of our previous papers [8, 9, 19] we already put forward our arguments in favour of a need to develop a model for computing and reasoning which is not purely mathematical and isolated from its real physical semantics, and which has the possibility to learn from the real physical environment through real physical interactions. That such an endeavour is necessary for building an intelligent system, dealing with complex phenomenon, is supported by several opinions of different researchers from different fields of research [1, 2, 3, 4, 6, 7, 10, 11, 12, 13, 16, 18]. Without repeating many such inspiring thoughts of different researchers let us start with citing one from [18].

> [. . .] *the often implicit stand one takes with regard to the question of the bridge between physical and symbolic descriptions determines in a fundamental way how one views the problems of cognition. A primary question here is, Exactly what kind of function is carried out by that part of the organism which is variously called the*

*29th International Workshop on Concurrency, Specification and Programming (CS&P'21)*
soma.dutta@matman.uwm.edu.pl (S. Dutta); skowron@mimuw.edu.pl (A. Skowron)
0000-0002-7670-3154 (S. Dutta); 0000-0002-5271-6559 (A. Skowron)

*sensor, the transducer, or in the case of Gibson, the "information pickup"? One's answer to this question determines the shape of any theory of perception and cognition one subsequently constructs. As I shall argue, unlike the symbolic processes that characterize cognition, the function performed by the transducer cannot be described as a computation, that is, by a purely symbol-manipulation function. Like all primitive operations of the functional architecture, the transducer fundamentally is a physical process; that is, its behavior is explainable (its regularities can be captured) in terms of its intrinsic properties - physical, chemical, biological, and so on. Thus, typically it does not come under study by cognitive psychologists who simply presuppose that an organism is equipped with certain transducer functions. The task of discovering these mechanisms is left for others, for example, "sensory encoding neurophysiologists, biophysicists, and engineers.*

Let us also cite the opinion concerning the need of a new computing model in Cyber-Physical Systems (CPS) [5].

> *The inherent cross-disciplinary nature of CPS requires distinct modelling techniques to be employed, thus prompting for a common background formalism that enables communication between all specialities. However, to this date, no such single super-formalism exists to support the multiple dimensions of the design of a CPS. Indeed, to effectively design a CPS, engineers (in the role of modellers) either need to be versed in multiple formalisms, or* **a fundamentally new modelling approach has to emerge**.

According to the view of Edmund Husserl, the founder of phenomenology [7], non-standard models of computation such as natural computing, reaction systems, Harel's algorithmics and Gurevich's abstract state machines, or neural network computing are closed in the abstract space, in the mathematical manifold.

> *Husserl was frustrated by the idea that science and mathematics were increasingly conducted on an abstract plane* [treating nature itself as a "mathematical manifold"] that was disconnected from human experience and human understanding, independently of questions of truth and applicability. He felt that the sciences increasingly dealt with idealized entities and internal abstractions a world apart from the concrete phenomena of daily life.

Till now, in different works (see, *e.g.*, [8, 9, 14, 19, 20]), we tried to introduce what do we mean by *Interactive Granular Computing* (IGrC) and how it is different from other existing theories from the perspective of modeling computations in a complex system. In a very brief description, *Interactive* symbolizes interaction between the abstract world and the real physical world, and *Granular Computing* symbolizes computation over imperfect, partial, granulated information abstracted about the real physical world. Here, once again we take the opportunity to explain briefly the notion of *complex granule* (c-granule), the basic building block of IGrC.

A c-granule is composed of three parts, known as soft_suit, link_suit and hard_suit. These three parts correspond to three sets of physical objects, together which determines the scope of that particular c-granule[1]. The soft_suit represents the objects from the physical reality which are directly accessible or about which already some information is obtained. The hard_suit corresponds to those objects which are in the scope of the c-granule but not yet accessed or are not in the direct reach at that point of time of the c-granule. The link_suit represents a chain of objects that creates a communication channel between the soft_suit and the hard_suit. A c-granule, when associated with an information layer with it, is known as informational c-granule, in short ic-granule. The information layer of an ic-granule contains different forms of information, such as specification of the already perceived properties of the objects from its soft_suit, specifications of the windows describing where and how some specific part from the scope of the ic-granule can be accessed or reached etc. Based on the purposes and types of the information specifications of an ic-granule there can be different types of ic-granules, such as ic-granule representing perception of the objects from the scope (perception based ic-granule), ic-granule representing domain knowledge (knowledge based ic-granule), ic-granule representing plan of actions (planner ic-granule), ic-granule representing plan into an implementational level language (implementational ic-granule) etc. Here to be emphasized that the ic-granule, responsible for implementation of a plan of actions, serves the task of connecting between the abstract and the real worlds.

A computation process over a c-granule is described by a network of ic-granules lying within the scope of the concerned c-granule. The informational layer of all these ic-granules constitutes the domain knowledge of the *control* of the c-granule, may be also called control of the computation process; this informational content is endowed with a reasoning mechanism. The information content and the reasoning mechanism together designs the control mechanism of a computation process. More specifically, the whole information layer is clustered based on the information relevant to different sub-scopes of the whole scope of the c-granule. For example, in the ic-granules representing the domain knowledge, there can be different sub-clusters in the informational layer corresponding to different aspects of the domain knowledge. That is, an ic-granule may contain several other ic-granules inside its scope. On the other hand the reasoning mechanism of the control of a c-granule is responsible for aggregating, deleting, or generating information from the existing clustered of information layers. Thus new information layers are generated over time based on (i) initial (partial) perception and domain knowledge of the concerned fragment of the environment (ii) initiation of interactions through already accessible objects to access the information about the not directly reachable objects (iii) perception of the physical world after interactions and (iv) verification of the perceived properties of the newly obtained configuration with the expected specifications of the target environment.

Having this much of preliminary relevant details about a computation process over a c-granule, in this paper our target is to present a real life example through which different aspects of computation over a c-granule can be visualised.

In this regard, Section 2 presents an example of computing along with an explanation of how such a real life computation can be modeled in the framework of IGrC. Section 3 presents the concluding remarks explaining how different components and features of IGrC incorporate a

[1]The scope of a c-granule at a given moment $t$ of the local time of the c-granule is the part of the physical space corresponding to the formal specifications of all spatio-temporal windows active at $t$.

possibility of building a mathematical model which is not a simplified static image of its real physical semantics; rather it is grounded in the real physical world.

## 2. Example of a computation over c-granule

The example described in [15] goes well with the idea of how a computation process, based on perception and interactions, should look like according to IGrC model.

> *perceiving is a way of acting.* [...] *Think of a blind person tap-tapping his or her way around a cluttered space, perceiving that space by touch, not all at once, but through time, by skillful probing and movement. This is or ought to be, our paradigm of what perceiving is.*

Let us consider the above cited example as an example of a computation over a c-granule, where the c-granule has in its scope a blind person[2] and its surrounding. More precisely, the person and the top part of the stick are directly accessible part of this environment, and hence belongs to the soft_suit. The part of the stick, which is distant from the direct touch of the person, belongs to the link_suit as a partial information about the end of the stick can be derived based on the part belonging to the soft_suit, and it creates a link to the not directly accessible objects, such as holes or stones lying in the surrounding environment, that is to the hard_suit. The goal of the computation is to have a successful forward movement of the blind person by deriving information about the unseen objects based on the already available knowledge and the perceived information about the directly accessible objects. The whole computation process, leading to the goal, is conducted by the control of the mentioned c-granule. The behaviour of the control is based on transformations of collections (finite families) of the actual ic-granules, or more exactly the actual networks of ic-granules, into the new ones.

All the information layers corresponding to different of ic-granules involved in a computation is clustered in the control of the computation; using this information the reasoning mechanism of the control makes the computation process to happen and dynamically move from one layer to another layer. Below a step-by-step process of the computation is described in the context of this example.

### Layer:0

1. We assume $g_s$ to be an ic-granule having in its scope a blind person or robot with a stick and the objects lying in the surrounding. At $t_0$, the beginning of the control's cycle, $g_s$ is labelled with the perceived information of the directly accessible objects from its soft_suit. Here the directly accessible objects can be the blind person himself and the part of the stick directly in contact with the person (see Figure 1). To be remembered that here $g_s$ represents a perception based ic-granule. The informational layer of $g_s$ contains, in particular information concerning perception of the current perceived situation; in more complex situations it may also contain link to the domain knowledge related to general perception of the environment, formal specifications of the transformations of

[2]One can even consider a robot instead of a human being.

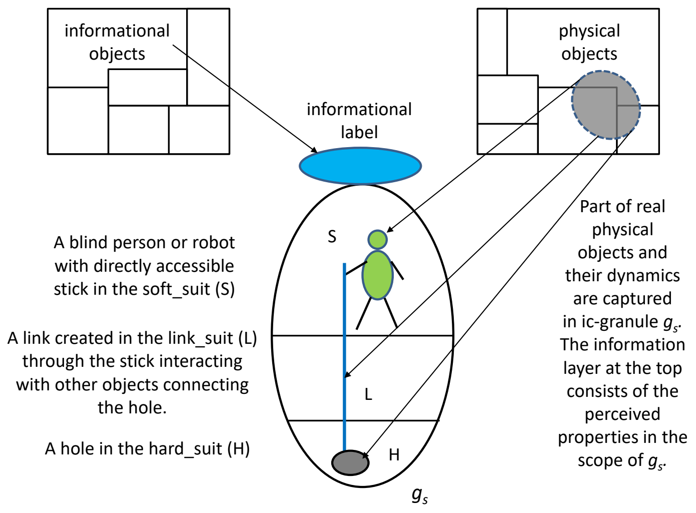

informational objects
physical objects
informational label
A blind person or robot with directly accessible stick in the soft_suit (S)
A link created in the link_suit (L) through the stick interacting with other objects connecting the hole.
A hole in the hard_suit (H)
S
L
H
$g_s$
Part of real physical objects and their dynamics are captured in ic-granule $g_s$. The information layer at the top consists of the perceived properties in the scope of $g_s$.

**Figure 1:** ic-granule operating on a particular scope of the physical world.

the ic-granules within its scope, database with rules for selection of transformation of ic-granules for realisation, etc. In our illustrative example we discuss only a very simplified version of $g_s$.

2. Let the description of the general goal of the computation be attached as the information layer of a planner ic-granule $g_0$. So, here $g_0$ corresponds to those particular cells of a human brain where the goal description is set. So, for $g_0$ the soft_suit, link_suit and the hard_suit can be like different layers of those brain cells where the reachability to more deeper layer in the hard_suit happens through the directly reachable layer in the soft_suit and reactions of the brain cells propagating from the soft_suit to the hard_suit.
3. The role of the knowledge base is represented by another ic-granule $g_{kb}$. Here, $g_{kb}$ can be considered as the brain parts related to the memory locations. The information layer of $g_{kb}$ is labelled with the addresses of different relevant properties of different fragments of the c-granule. The soft_suit of $g_{kb}$ consists of the objects which form the outer box of the memory location whose address is attached to the information layer; in order to access the detailed information about some fragments some more inner boxes, lying in the hard_suit, are to be opened. Such ic-granules representing knowledge base may be called information granules. Their physical parts create local memories for storing and transmitting information. One may also look on $g_{kb}$ as on a compound ic-granule representing a network of ic-granules determining the structure of $g_{kb}$.
4. Now based on the information gathered from the informational layers of $g_s$, $g_0$ and $g_{kb}$, the control with its reasoning mechanism aims to better understand the perceived situation necessary for decomposition; this leads to construction of a more detailed plan of actions. This detailed plan is represented in informational layer of the ic-granule $g_1$ at the next time point $t_1$. For a visual representation the readers are referred to Figure 2.

In our simplified illustrative example, we mention only one mechanism for enriching the

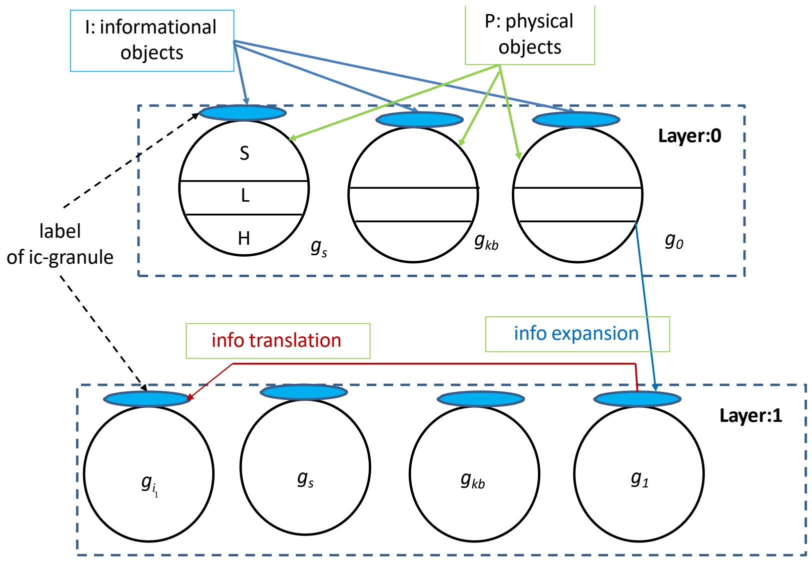


**Figure 2:** Computation over ic-granules passing from layer-0 to layer-1.
(i) $g_s$: Current perception of some objects of P at time $t_0$, indicating the scope. S contains directly perceivable objects, L contains objects creating communication channel to the objects in H where some actions are to be performed.
(ii) $g_{kb}$: Relevant information about general laws related to objects in $g_s$ and specifications of where this information is stored. Here S contains directly accessible part of the storage memory and L contains the objects linking to the directory at H.
(iii) $g_0$: Relevant information regarding a goal that to be implemented in the H part of $g_s$. This specification of goal is stored in some object in the H of $g_0$, and is accessible by some object lying in the S of $g_0$.
(iv) info expansion denotes decomposition of the plan available at $g_0$ to a more detailed plan.

informational layer of $g_s$. In a more realistic example, one should consider other mechanisms, *e.g.*, measuring different parameters by sensors (e.g., stick in our example), recording the perceived results of measurements in the corresponding informational layers of ic-granules etc. In a more general case, a sequence of steps of reasoning is realised, which are based on transformations of ic-granules, leading to better understanding of the currently perceived situation. It should be also noted that in general decomposition problems are challenging and they are related, in particular to the idea of information granulations and computing with words [23, 24, 25, 26].

## Layer:1

1. From the perspective of the example, at $t_0$ the information attached to $g_0$ encodes the general goal of the blind person that primarily gets registered in his brain, *i.e.*, in the soft_suit of $g_0$. At time $t_0$ the hard_suit of $g_0$, such as more deep analytical brain cells, remains still unaccessed. Based on the collected information of $g_s$ and $g_{kb}$, at time $t_1$ the person ponders more analytically; this in a sense activates interaction with the previously unaccessed part of $g_0$. This gradually gives access to the hard_suit of $g_0$, and thus at time

$t_1$ the hard_suit of $g_0$ becomes the soft_suit of $g_1$, labelled with a more detailed plan for the person.

2. Now in order to implement the abstract description of the plan available at $g_1$ through real physical actions, the plan needs to be transformed from the abstract level to an implementational level language. From the perspective of our example, this can be a translation of the plan from the person's analytical brain cells to a language of actuators, like hands, legs, and the stick of the person. So, a new ic-granule is manifested at this layer. We call it as $g_{i_1}$, an implementational ic-granule. To be noted that $g_{i_1}$ does not concern about the actual actuators; rather it is like another hard-drive in the brain of the person where the action plan can be stored in the language of actuators. The information layer of $g_{i_1}$ also contains the specification of the conditions for initiating the implementation plan through a real actuator. Figure 3 presents the computation process described in layer-1.

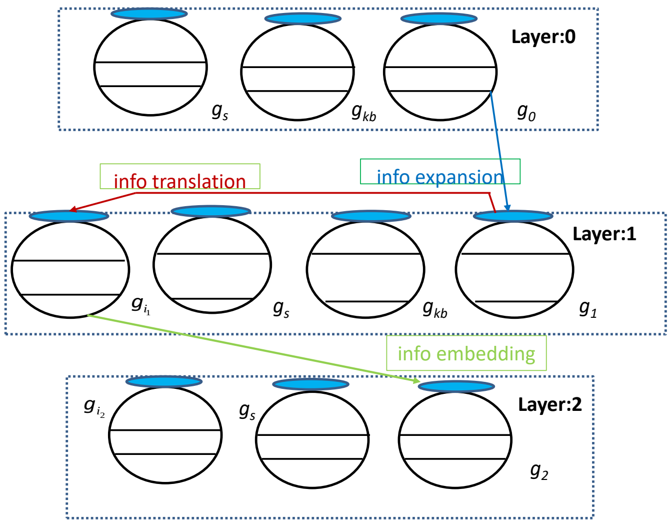


**Figure 3:** Computation over ic-granules passing from layer-1 to layer-2.
(i) $g_1$: at time $t_1$ specification of the plan of $g_0$ is expanded. Detailed specification is generated based on $g_s$, $g_{kb}$ and $g_0$ of Layer:0. In particular the H part of $g_0$ can be now the S of $g_1$ which is gradually reached through the L part of $g_0$.
(ii) $g_{i_1}$: Specification of how the abstract plan of $g_1$ can be implemented in $g_s$, that is the specification of the plan in a lower level language which can be implemented via physical objects. This lower level language is also a built-in language of a hardware lying at the H part of $g_{i_1}$.
(iii) info translation denotes translation of the plan from the level of abstract description to the level of implementational language.
(iv) $g_2$: Specification in lower level language is embedded to a physical object lying in the S part of $g_2$, which prepares the ground to run the plan via a physical object in H part of $g_2$.
(v) info embedding represents embedding the plan of actions on a real physical object.

### Layer:2

1. The specification of the plan of implementation of $g_{i_1}$ is now realized through a physical object at time $t_2$. Let this object belong to the scope of the ic-granule $g_2$. In case of the

example, it can be the stick of the blind person on which the abstract implementation plan is embedded, and $g_2$ represents the ic-granule containing the stick in its scope. The physical interaction of the stick with other objects in $g_2$ is encoded in the information layer of $g_2$. If this information matches to a significant level to the condition for initiating implementation plan stored at $g_{i_1}$ then an action compilation signal is passed to the next implementation granule, may be named as $g_{i_2}$.
2. With the action compilation specification of $g_{i_2}$ the objects lying in its link_suit and hard_suit propagate actions to realize a desired configuration in the hard_suit of $g_s$. In the context of our example, $g_{i_2}$ represents the ic-granule which specifies how to move the stick forward until it touches a stone on its way. This chain of objects between the stick and a stone creates a communication channel.
3. Through this communication channel the computation process enters into the hard_suit of $g_s$, which was inaccessible at time $t_0$. The initiation of the action compilation via $g_{i_2}$ creates a link to the hard_suit of $g_s$. This new interaction gives access to the hard_suit of $g_s$ which was previously inaccessible.
4. A new cycle starts by perceiving properties of the newly accessible part of $g_s$.

Here to be noted, that in the example we only have mentioned about the decomposition of the plan of actions from the initial stage $g_0$ to a stage $g_1$, from where it gets translated to the implementational level language. But in practice decomposition of the action plan, say $\alpha : g_0 \Rightarrow \beta : g_1$ available in the information layer of $g_0$ specifying the target ic-granule $g_1$ with property $\beta$ can have several layers of decomposition in between. For a visual representation the readers are referred to the Figure 4, which will be discussed in Section 3. One should also consider that some actions are lunched in the process of perception while the other ones are related to the main decisions.

The above described idea of computation over a c-granule is in the line of the idea that Luc Steels has characterised in [21]; complex dynamical systems (complex systems, for short) are considered as systems consisting of a set of interacting elements

> [. . .] *where the behavior of the total is an indirect, non-hierarchical consequence of the behavior of the different parts. Complex systems differ in that sense from strictly hierarchical systems* [...] *where the total behavior is a hierarchical composition of the behavior of the parts. In complex systems, global coherence is reached despite purely local non-linear interactions. There is no central control source. Typically the system is open.*

## 3. Beyond pure mathematical modeling: a concluding remark

From the above exemplification of the process of computation over a c-granule, moving from a configuration of ic-granules to another, it is quite clear that the process deals with a set of hunks of real physical matter associated with their information layers; the information layers indicate where, when and how they can be touched, or their properties can be achieved or verified. So, this already clarifies how in the model of IGrC by a c-granule both abstract world, that is the information layer, and the real physical semantics, that is the three-layered hunk of objects, together are referred to. One more point, that needs to be clarified, is how the model designs its real physical implementation by lifting the static description of a process to the level of actions.

The implementational ic-granules create a specific interface between the abstract and the physical world. In particular, at some level of the implementational phase the control of the c-granule launches actions linking the abstract world associated to the informational layer of the control of the c-granule with the real physical world. These actions are not from abstract mathematical space and the model of IGrC keep those action functions free from mathematical formalizations. Their syntactic descriptions can be formalised in the informational layers of the control of the c-granule. However, their implementation should be realised in the real physical world and the model only can mathematically formalize their performance quality by perceiving the changes in the world after initiation of the actions. Of course, the expected properties of the real physical environment, after the implementation of the actions, also can be formalized within the description of the action specification.

Thus the IGrC model keeps the possibility of mismatch between expected and real physical outcome open as we never can a priori formalize all possible outcomes of an action, which is supposed to be initiated in the physical world based on an abstract description of the action specification. We can only expect a desired outcome, specified in the information layer. So, after each implementational phase of a computation over a c-granule the control starts a new cycle again by perceiving the properties of the objects lying in its scope. Then those observed properties are verified with the expected properties. Hence, accordingly there is a need to modify the induced models based on their comparisons with the recorded data.

In Figure 4 we illustrate the idea of decomposition of the description of the action plan in order to transform an ic-granule with a given property, say $\alpha$, into another one with the property $\beta$. After several levels of decomposition it reaches a level at which the relevant actions are launched by the control so that the expected realisation of the whole chain of actions can lead to a situation satisfying, to a satisfactory degree, the property $\beta$. Though the conditions of the chain of actions and the expected properties after the actions are specified by $\alpha$'s and $\beta$'s, the actual actions $ac_1, \ldots ac_k$ are not in the realm of mathematical formulation. The model of IGrC keeps this juncture between abstract and physical bridging free from mathematical formulation. Instead, IGrC proposes to formalize the results of actions by perceiving the properties of outcome configurations and verifying them with the expected property $\beta$.

Let us outline a scheme of the behaviour of the control of a c-granule.

Often the control aims at achieving its target goal, expressed in the information layer of the ic-granule $g_0$, using complex vague concepts, *e.g.*, related to safeness of the perceived situation, from a natural language. Below, we assume that for a given specification a set of rules for selection of the relevant transformations of ic-granules was learned by the control or is given a priori. The pre-condition of each rule is a condition. The degree of matching of this condition by the current status of the perceived situation determines selection of the transformation of ic-granules from the post-condition of the rule.[3]

The outline of the general procedure of a computation in IGrC, realised by the control of a c-granule, is based on searching for relevant transformations of ic-granules and their

[3]Note that other learning paradigms such as lazy learning can be used in inducing models of complex vague concepts different from the discussed here.

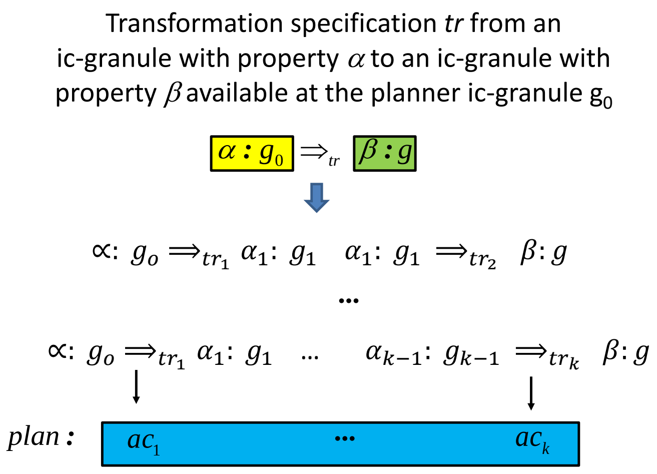

Transformation specification *tr* from an ic-granule with property $\alpha$ to an ic-granule with property $\beta$ available at the planner ic-granule $g_0$
$\alpha : g_0 \Rightarrow_{tr} \beta : g$
$\propto : g_o \Rightarrow_{tr_1} \alpha_1 : g_1 \quad \alpha_1 : g_1 \Rightarrow_{tr_2} \beta : g$
...
$\propto : g_o \Rightarrow_{tr_1} \alpha_1 : g_1 \quad \dots \quad \alpha_{k-1} : g_{k-1} \Rightarrow_{tr_k} \beta : g$
*plan :*
$ac_1$ ... $ac_k$

**Figure 4:** Illustration of a simple case of decomposition of transformation.

implementations; it looks as follows.

perceive accessible parts of the abstract and physical world to understand (up to a satisfactory degree) the current situation for making decision concerning the selection of the formal specification of transformation of ic-granules for implementation; the current status of perception is represented in the informational layer of $g_s$; this is realised by the control in several steps and one of them is listed below

$\Downarrow$

consult domain knowledge and required goal of computation ($g_{kb}$, $g_0$) to enrich the information about the status of currently perceived situation

$\Downarrow$

. . .

(it may be necessary to perform several steps of reasoning before having a satisfactory understanding of the perceived situation; it can be achieved by allowing the control to select the relevant formal specification of transformation of ic-granules for implementation; some of these steps may be related, *e.g.*, to measurements (by sensors) of features of the perceived physical objects in the scope of active ic-granules[4]

. . .

$\Downarrow$

select (from the proper knowledge base) the relevant formal specification of the transformation of the ic-granules for implementation[5]

$\Downarrow$

[4]It should be noted that the mentioned steps of reasoning are also realised by transformations of some ic-granules.

[5]This step is especially compound; details will be discussed in an extended version of our paper.

. . .

the selected formal specification may require several decomposition steps before the proper level for the direct implementation can be achieved (embedding in the physical world)

$\Downarrow$

after reaching the satisfactory level of decomposition generate the formal specification of the action plan; in the simplest case such a plan is represented by a linear order of ic-granules which can be implemented by the considered control of the c-granule in the physical world[6]

$\Downarrow$

initiate (in proper order) implementational granules from the plan responsible for implementation of actions of plan

$\Downarrow$

during plan realisation, perceive the relevant accessible parts of the physical world (through the implementational granules) and store the perceived data in informational layers of the relevant ic-granules

$\Downarrow$

follow the same loop as above taking, in particular into account the results of matching the expected results with the perceived ones[7].

The discussed above issues of decomposition and implementation should be treated as an illustrative example only. In the real-life projects one should take into account many other issues.

Here to be noted that the transition from one configuration of ic-granules to the other, as described above in the general plan of computation, by creating and accessing different ic-granules and their information layers is not also purely mathematical. Usually, the state transition relation is presented by a given family of sets $\{X_i\}_{i\in I}$ where a transition relation is represented as a relation $tr_i \subseteq X_i \times X_i$. Here, this cannot be purely mathematical as we need to incorporate the components which can specify (i) how elements of $X_i$ are perceived in the real physical environment, and (ii) how the transition relation $tr_i$ is implemented in the real physical world. This reasoning mechanism, as described in the Introduction, is conducted by the control of the c-granule over which the computation process is running.

So, in contrary to a static transition relation, in IGrC the control of a c-granule incorporates the possibility of dynamic as well as not purely mathematical formulation of a state transition relation. The structure of a control is designed to have two interacting modules, called the abstract module (AM) and the physical module (PM) (see Figure 5).

Communication between AM and PM is designed by two mechanisms. The first one provides a possibility of encoding given information from AM by a relevant state of a set of physical objects from PM, and the second one provides a possibility of encoding the considered state of

[6]Plans may be generated on different levels of decomposition, *e.g.*, for better understanding the perceived situation helping the control to generate the high quality plans for realisation of the target goals.

[7]In a more compound case of the control of a c-granule, strategies for adaptation of the previously used plan to the new situation are used. Moreover, the decision about the necessity of plan adaptation may be taken often during of the actual plan.

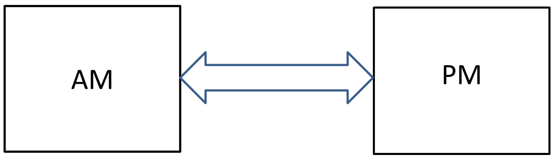


**Figure 5:** Interacting abstract (AM) and physical (PM) modules in control of c-granules.

a set of physical objects from PM by a relevant information represented in the language of AM. These two mechanisms can be implemented by atomic actions.

AM module sends to PM a formal specification of a transformation of a ic-granule as well as formal specifications of some spatio-temporal windows. Some of them are labelled by already perceived information or properties from the scope of the c-granule and others are labelled by formal specifications of the required information from PM. In case when the delivered specification can be directly embedded by PM in the physical world then PM, by creating a network of interacting ic-granules, aims to deliver a network of physical pointers matching information, perceived by AM, with the expected properties, expressed formally in AM. Otherwise, PM sends to AM a message about the necessity of the specification decomposition.

AM may send to PM messages consisting of formal specification(s) of the required transformation(s) of ic-granules together with the expected results provided by AM. The messages sent by AM are encoded by the states of some physical objects in PM. In this way, atomic actions changing states of physical objects to the ones specified by the given information are implemented. However, it is to be noted that the modeling of the behaviour of AM and communication of AM with PM can be based on mathematical modeling; whereas PM is composed out of physical objects which, being from the real world, are outside of the abstract mathematical modeling. However, partial information about properties of these objects and their interactions may be communicated to AM through interaction of PM with AM. Thus, IGrC incorporates both dynamic as well as mathematical modeling grounded in the real physical world.

## References

[1] Hyo-Sung Ahn: Formation Control Approaches for Distributed Agents. Studies in Systems, Decision and Control **205**. SpringerHyo-Sung , Cwitzerland (2020) doi:10.1007/978-3-030-15187-4

[2] Lawrence W. Barsalou: Perceptual symbol systems. Behavioral & Brain Sciences **22** (1999) 577–660 (doi:10.1017/S0140525X99002149)

[3] Franz Brentano: Psychologie vom empirischen Standpunkte. Dunker & Humboldt, Leipzig (1874) (https://archive.org/details/psychologievome02brengoog/page/n4/mode/2up)

[4] Frederick P. Brooks: The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley, Boston (1975) (extended Anniversary Edition in 1995)

[5] Paulo Carreira, Vasco Amaral, Hans Vangheluwe: Multi-paradigm modelling for cyber-physical systems: Foundations. In: Paulo Carreira, Vasco Amaral, Hans Vangheluwe (eds.),

Foundations of Cyber-Physical Systems Multi-Paradigm Modelling for Cyber-Physical Systems. Springer, Cham, Switzerland, pp. 1-14 (2020) (doi:10.1007/978-3-030-43946-0)

[6] David Deutsch, Artur Ekert, Rossella Lupacchini: Machines, logic and quantum physics. Neural Computation **6** (2000) 265–283 (doi:10.2307/421056)

[7] Paul Dourish: Where the Action Is. The Foundations of Embodied Interaction. The MIT Press Cambridge, MA, London (2004)

[8] Soma Dutta, Andrzej Skowron: Toward a computing model dealing with complex phenomena: Interactive granular computing. In: Ngoc Than Nguyen et al (eds): ICCCI 2021 Proceedings, Springer (2021) (to appear)

[9] Soma Dutta, Andrzej Skowron: Interactive Granular Computing Model for Intelligent Systems. In: Shi, Z., Chakraborty, M., Kar, S. (eds): Intelligence Science III. 4th IFIP TC 12 International Conference, ICIS 2020, Durgapur, India, February 24-27, 2021, Revised Selected Papers. IFIP Advances in Information and Communication Technology (IFIPAICT) book series **623**, Springer, Cham, Switzerland, pp. 37-48 (2021) doi:10.1007/978-3-030-74826-5_4

[10] Sean Gerrish: How Smart Machines Think. MIT Press, Cambridge, MA (2018)

[11] Stevan Harnad: The symbol grounding problem. Physica **D42** (1990) 335–346 (doi:10.1016/0167-2789(90)90087-6)

[12] Mark Heller: The Ontology of Physical Objects. Four dimensional hunks of matter. Cambridge Studies in Philosophy , Cambridge University Press, Cambridge, UK (1990)

[13] Andrew Hodges: Alan Turing, Logical and Physical. In: Cooper, S.B., Löwe, B., Sorbi, A., New Computational Paradigms. Changing Conceptions of What is Computable. Springer and Business Media, New York, N.Y., pp. 3-15 (2008) (doi:10.1007/978-0-387-68546-5_1)

[14] Andrzej Jankowski: Interactive Granular Computations in Networks and Systems Engineering: A Practical Perspective. Lecture Notes in Networks and Systems. Springer, Heidelberg (2017) (doi:10.1007/978-3-319-57627-5)

[15] Alva Nöe: Action in perception. MIT Press, Cambridge, MA (2004)

[16] Charles L. Ortiz Jr.: Why we need a physically embodied Turing test and what it might look like. AI Magazine **37** (2016) 55–62 (doi:10.1609/aimag.v37i1.2645 )

[17] Pawlak, Z.: Rough sets. International Journal of Computer and Information Sciences **11** (1982) 341–356 (doi:10.1007/BF01001956)

[18] Zenon W. Pylyshyn: Computation and Cognition. Toward a Foundation for Cognitive Science. The MIT Press Cambridge, MA (1884)

[19] Andrzej Skowron, Andrzej Jankowski, Soma Dutta: Interactive granular computing. Granular Computing **1**(2) (2016) 95–113 (doi:10.1007/s41066-015-0002-1)

[20] Andrzej Skowron, Andrzej Jankowski: Rough sets and interactive granular computing. Fundamenta Informaticae **147** (2016) 371–385 (doi:10.3233/FI-2016-1413)

[21] Luc Steels: The synthetic modeling of language origins. Evolution of Communication **1**(1) (1997) 1–34 (doi:10.1075/eoc.1.1.02ste)

[22] Lotfi A. Zadeh: Fuzzy sets. Information and Control **8**(3) (1965) 338–353 (doi:10.1016/S0019-9958(65)90241-X)

[23] Lotfi A. Zadeh: Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. Fuzzy Sets and Systems **90** (1997) 111–127 (doi:10.1016/S0165-0114(97)00077-8)

[24] Lotfi A. Zadeh: From computing with numbers to computing with words – from manipulation of measurements to manipulation of perceptions. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications **45**(1) 105–119 (1999) (doi:10.1109/81.739259)

[25] Lotfi A. Zadeh: Foreword. In: Pal, S.K., Polkowski, L., Skowron, A. (eds.), Rough-Neuralcomputing: Techniques for Computing with Words, Heidelberg, Springer, pp. ix-xi (2004) (doi:10.1007/978-3-642-18859-6)

[26] Lotfi A. Zadeh: Computing with Words: Principal Concepts and Ideas. Studies in Fuzziness and Soft Computing **277**, Springer, Heidelberg (2012) (doi:10.1007/978-3-642-27473-2)

# On Semantics for Testing in Time Petri Nets

Elena Bozhenkova[1], Irina Virbitskaite[1]

[1]A.P. Ershov Institute of Informatics Systems, SB RAS; 6, Lavrentiev av., Novosibirsk, 630090, Russian Federation

**Abstract**

Dense-Time Petri Nets (TPNs) are now a well-established model to describe and study real time (quantitative) and functional (qualitative) properties of safety-critical, computer-controlled systems. Testing equivalences, used to compare systems' behaviors (processes) and reduce the structure of systems, are defined in terms of tests that processes may or must pass. The intention of the paper is to establish the interrelations between various semantics for must-testing equivalences with extended tests, in the framework of TPNs. This allows for studying in detail the timing behavior in addition to the degree of relative concurrency of processes generated when systems are functioning.

## 1. Introduction

Dense-Time Petri Nets (TPNs) are suitable for qualitative and quantitative modelling and verifying of safety-critical, computer-controlled, real-time systems. Several semantics (behaviors) are explored in the literature for TPNs, that can be classified according to interleaving – partial order dichotomy. The classical interleaving behavior of the TPN is described by runs — sequences of changes in states by time elapsings and/or transition firings. The semantics allows for analyzing some safety and liveness properties of systems, however concurrency between net transitions is reduced to non-deterministic choice between sequences of transitions firings in any possible order. Step semantics of TPNs generalizes the interleaving approach by allowing several concurrent transitions (forming a step) to fire simultaneously. Partial order semantics of TPNs is most often represented by means of the so-called causal net processes which include events and conditions related by causal dependence (the absence of causality means concurrency) and equipped with timing information. Causal tree semantics summarizes the interleaving and partial order approaches by representing the behavior of the TPN in the form of a tree with nodes corresponding to runs, and edges labeled by actions with their times and causal predecessors.

Testing equivalences [1] are explicitly based on a framework of extracting information about the systems' behaviors (processes) by testing them. Two processes are considered equivalent if there is no test that can distinguish them. In the realm of untimed models, interleaving testing was thoroughly investigated and well-understood in the setting of models of transition systems (see [2, 3] among many others). Interleaving, step and partial order testing equivalences for elementary net systems (safe Petri nets without loops) were studied in [4]. There, the authors

*29th International Workshop on Concurrency, Specification and Programming (CS&P'21)*
bozhenko@iis.nsk.su (E. Bozhenkova); virb@iis.nsk.su (I. Virbitskaite)
https://pdb.iis.nsk.su/en/person/220 (E. Bozhenkova); https://persons.iis.nsk.su/en/virbitskaite (I. Virbitskaite)
0000-0002-9291-7451 (E. Bozhenkova); 0000-0002-4475-3480 (I. Virbitskaite)

indicated the location of the testings among other behavioral equivalences from linear time – branching time spectrum, providing their hierarchies for the model under consideration with and without invisible actions. Partial order and non-deterministic semantics for different classes of Petri nets are often represented by means of various models of event structures. In the framework of the models, testing equivalences within interleaving – partial order dichotomy were developed and compared in the papers [5, 6, 7]. Moreover, in [7] special attention was paid to the relationships between partial order and causal tree semantics in the context of testing equivalences. To the best of authors' knowledge, causal tree semantics in the framework of Petri nets has not been studied yet.

As safety-critical applications often require verification of real time characteristics, testing equivalences are expanded in concurrent models with time. In the papers [8, 9, 10, 11], alternative characterizations of timed testing are provided for timed generalizations of interleaving models. In [12], the testing relations along with their alternative characterization and discretization were proposed in the framework of Petri nets with time intervals associated to arcs from places to transitions. At the same time, to the best of our knowledge, there are only few mentions of a fusion of timing and partial order semantics, in testing scenario. In this regard, the work [13] is a welcome exception, where time-sensitive testing is investigated within linear time – branching time spectrum, in the setting of event structure models with time characteristics. Also, our origin is the paper [14] the main result of which is the coincidence of poset and causal tree testing equivalences, with the tests as direct extensions of the experiments, in the setting of TPNs. In this paper, we expand the results of [14] to step, poset and causal tree semantics with extended tests[1], and demonstrate the discriminating/matching power of the semantics in the framework of testing equivalences on contact-free TPNs. The results obtained can be useful in formal verification of systems since partial order semantics allows for reducing the number of systems' states to be analyzed.

## 2. Syntax and Different Semantics of TPNs

In this section, some terminology concerning the model of Petri nets with timing constraints (time intervals on the firings of transitions) and its concurrent semantics in terms of interleaving/step firing sequences, causal net processes and causal trees are defined.

### 2.1. Syntax and Interleaving/Step Semantics of TPNs

We start with recalling the definitions of the structure and interleaving/step behavior of time Petri nets [15, 16]. We use $Act$ as an alphabet of actions and $Act^{\mathbb{N}}$ as a set of multisets over $Act$. Let the domain $\mathbb{T}$ of time values be the set of rational numbers. We denote by $[\tau_1, \tau_2]$ the closed interval between two time values $\tau_1, \tau_2 \in \mathbb{T}$. Infinity is allowed at the upper bounds of intervals. Let $Interv$ be the set of all such intervals.

**Definition 1.** • *A* (labeled over $Act$) time Petri net *is a pair* $\mathcal{TN} = (\mathcal{N}, D)$, *where* $\mathcal{N} = (P, T, F, M_0, L)$ *is a (labeled over Act) underlying Petri net (with a finite set $P$ of places, a*

[1]Testing equivalence with extended tests checks, after the executions of the experiments, the tests that are continuations of the experiments with steps/posets of actions, not with single actions.

*finite set $T$ of transitions such that $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$, a flow relation $F \subseteq (P \times T) \cup (T \times P)$, an initial marking $\emptyset \neq M_0 \subseteq P$, a labeling function $L : T \to Act$) and $D : T \to Interv$ is a* static timing function *associating with each transition a time interval. For a transition $t \in T$, the boundaries of the interval $D(t) \in Interv$ are called the earliest firing time $Eft$ and latest firing time $Lft$ of $t$. For $x \in P \cup T$, let ${}^\bullet x = \{y \mid (y, x) \in F\}$ and $x^\bullet = \{y \mid (x, y) \in F\}$ be the* preset *and* postset *of $x$, respectively. For $X \subseteq P \cup T$, define ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x \in X} x^\bullet$.*

- *A marking $M$ of $\mathcal{TN}$ is any subset of $P$. A transition $t \in T$ is* enabled at a marking *$M$ if ${}^\bullet t \subseteq M$. Let $En(M)$ be the set of transitions enabled at $M$. A non-empty subset $\emptyset \neq U \subseteq T$ is a* step enabled at a marking *$M$, if ($\forall t \in U \diamond t \in En(M)$) and ($\forall t \neq t' \in U$: $({}^\bullet t \cup t^\bullet) \cap ({}^\bullet t' \cup t'^\bullet) = \emptyset$).*

  *A* state *of $\mathcal{TN}$ is a pair $(M, I)$, where $M$ is a marking and $I : En(M) \longrightarrow \mathbb{T}$ is a* dynamic timing function. *The* initial state *of $\mathcal{TN}$ is a pair $S_0 = (M_0, I_0)$, where $M_0$ is the initial marking and $I_0(t) = 0$, for all $t \in En(M_0)$.*

  *A step $U$ enabled at a marking $M$* can fire from a state *$S = (M, I)$* after a delay time *$\theta \in \mathbb{T}$ if ($Eft(t) \leq I(t) + \theta$), for all $t \in U$, and ($I(t') + \theta \leq Lft(t')$), for all $t' \in En(M)$.*

  *The* firing *of a step $U$ that can fire from a state $S = (M, I)$ after a delay time $\theta$* leads *to the new state $S' = (M', I')$ (denoted $S \xrightarrow{(U,\theta)} S'$) given by:*

  $(a)$ $M \xrightarrow{U} M'$,

  $$(b)\ \ \forall t' \in T \diamond I'(t') = \begin{cases} I(t') + \theta, & \text{if } t' \in En(M \setminus {}^\bullet t), \\ 0, & \text{if } t' \in En(M') \setminus En(M \setminus {}^\bullet t), \\ \textit{undefined}, & \textit{otherwise.} \end{cases}$$

  *Then, we write $S \xrightarrow{(A,\theta)} S'$, if $A = L(U) = \Sigma_{t \in U} L(t) \in Act^{\mathbb{N}}$, i.e. $A$ is a multiset over the set $\{a \in Act \mid a = L(t)$ and $t \in U\}$. We use the notation $S \xrightarrow{\sigma} S'$ iff $\sigma = (U_1, \theta_1) \ldots (U_k, \theta_k)$ and $S = S^0 \xrightarrow{(U_1,\theta_1)} S^1 \ldots S^{k-1} \xrightarrow{(U_k,\theta_k)} S^k = S'$ ($k \geq 0$). In this case, $\sigma$ is a* step firing sequence of $\mathcal{TN}$ from $S$ (to $S'$), *and $S'$ is a* reachable state of $\mathcal{TN}$ from $S$. *Whenever $\mid U_i \mid = 1$ for all $1 \leq i \leq k$, we call $\sigma$ an* interleaving firing sequence of $\mathcal{TN}$. *Let $\mathcal{FS}_{s(i)}(\mathcal{TN})$ be the set of all step (interleaving) firing sequences of $\mathcal{TN}$ from $S_0$, and $RS(\mathcal{TN})$ be the set of all reachable states of $\mathcal{TN}$ from $S_0$. For $\sigma = (U_1, \theta_1) \ldots (U_k, \theta_k) \in \mathcal{FS}_{s(i)}(\mathcal{TN})$, $L(\sigma) = (A_1, \theta_1) \ldots (A_k, \theta_k)$ iff $A_i = L(U_i)$ for all $1 \leq i \leq k$.*

  *We call $\mathcal{TN}$ $T$-*restricted *iff ${}^\bullet t \neq \emptyset \neq t^\bullet$, for all transitions $t \in T$;* contact-free *iff whenever a step $U$ can fire from the state $S = (M, I)$ after some delay time $\theta$, then $(M \setminus {}^\bullet U) \cap U^\bullet = \emptyset$, for all $S \in RS(\mathcal{TN})$. In what follows, we shall consider only $T$-restricted and contact-free time Petri nets.*

**Example 1.** *A (labeled over $Act = \{a, b, c\}$) time Petri net $\widetilde{\mathcal{TN}}$ is shown in Figure 1. Here, the places are represented by circles, and transitions — by bars; the names are depicted near the net elements, the flow relation is drawn by the arcs, the initial marking is represented as the set of the places with tokens (bold points), and the values of the labeling and timing functions are*

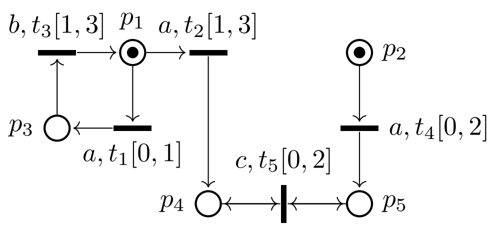


**Figure 1:** The TPN $\widetilde{\mathcal{TN}}$.

*printed next to the transitions. It is not difficult to check that $t_1$, $t_2$ and $t_4$ are transitions enabled at the initial marking $M_0 = \{p_1, p_2\}$, and, moreover, $\{t_1, t_4\}$ and $\{t_2, t_4\}$ are steps enabled at $M_0$. The steps can fire from the initial state $S_0 = (M_0, I_0)$ after time delay $\theta_1 = 1$, where $I_0(t) = \begin{cases} 0, & \text{if } t \in \{t_1, t_2, t_4\}, \\ undefined, & \text{otherwise.} \end{cases}$ The sequence $\sigma = (\{t_1, t_4\}, 0.5)\ (t_3, 1)\ (t_2, 2)\ (t_5, 2)$ is a step firing sequence of $\widetilde{\mathcal{TN}}$ from $S_0$. Also, we have $L(\sigma) = (2'a, 0.5)\ (b, 1)\ (a, 2)\ (c, 2)$. Furthermore, it is easy to see that $\widetilde{\mathcal{TN}}$ is really $T$-restricted and contact-free.*

## 2.2. Causal Net Process Semantics of TPNs

In this subsection, the concept of causality-based net processes is presented in the context of TPNs.

We start with the definition of time causal nets, whose events and conditions are related by causal dependence and concurrency (absence of causality), and whose timing function associates with the events their occurrence times.

**Definition 2.** *A* (labeled over *Act*) time causal net *is a finite, acyclic net $TN = (B, E, G, l, \tau)$ with a set $B$ of conditions; a set $E$ of events; a flow relation $G \subseteq (B \times E) \cup (E \times B)$ such that $|\ ^\bullet b\ | \leq 1 \geq |\ b^\bullet\ |$, for all $b \in B$, and $^\bullet B = E = B^\bullet$; a labeling function $l : E \to Act$, and a timing function $\tau : E \to \mathbb{T}$ such that $e\ G^+\ e' \Rightarrow \tau(e) \leq \tau(e')$.*

For a time causal net $TN = (B, E, G, l, \tau)$ and $e, e' \in E$, define:

- $TN^\bullet = \{b \in B \mid b^\bullet = \emptyset\}$;
- $\prec = G^+$, $\preceq = G^*$ (causality);
- $Predec(e) = \{e' \in E \mid e' \prec e\}$ (causal predecessors of $e$), $Earlier(e) = \{e' \in E \mid \tau(e') < \tau(e)\}$ (time predecessors of $e$), and $Cut(e) = (Earlier(e)^\bullet \cup {}^\bullet TN) \setminus {}^\bullet Earlier(e)$;
- $E'$ is a *downward-closed subset of* $E$ iff $E' \subset E$ and $Predec(e') \subseteq E'$, for all $e' \in E'$; a *timely sound subset of* $E$ iff $E' \subset E$ and $Earlier(e') \subseteq E'$, for all $e' \in E'$;
- $e \smile e' \iff \neg((e \prec e') \vee (e' \prec e))$ (concurrency); $\emptyset \neq V \subseteq E$ is a *step* iff $e \smile e'$ and $\tau(e) = \tau(e')$ for all $e \neq e' \in V$. Let $\tau(V) = \tau(e)$ for some $e \in V$ (time of $V$);

- a sequence $\rho = V_1 \dots V_k$ ($k \geq 0$) of steps of $TN$ is an *s-linearization of* $TN$ iff $\bigcup_{1 \leq i \leq k} V_i = E$ and $\bigcap_{1 \leq i \leq k > 1} V_i = \emptyset$ (i.e. every event of $TN$ appears in the sequence exactly once), and for all $1 \leq i < j \leq k$ it holds: $\bigcup_{1 \leq l \leq i} V_l$ is a downward-closed and timely sound subset of $\bigcup_{1 \leq m \leq j} V_m$ (i.e. both causal and time order are preserved: for all $1 \leq i < j \leq k$, $\neg((e' \prec e) \vee (\tau(e') < \tau(e)))$, for all $e \in V_i$, $e' \in V_j$). Whenever $\mid V_i \mid = 1$ for all $1 \leq i \leq k$, $\rho$ is an $i$-linearization;

- $\eta(TN) = (E_{TN}, \preceq_{TN} \cap (E_{TN} \times E_{TN}), l_{TN}, \tau_{TN})$ is a time poset[2]. For time posets $\eta = (E, \preceq, l, \tau)$ and $\eta' = (E', \preceq', l', \tau')$, $\eta$ is a *pos-extension of* $\eta'$ iff $E'$ is a downward-closed and timely sound subset of $E$, $\preceq' = \preceq \cap E' \times E'$, $l' = l \mid_{E'}$, and $\tau' = \tau \mid_{E'}$.

We are now ready to define the concept of causal net based processes of TPNs, proposed in [15].

**Definition 3.** *Given a time Petri net $\mathcal{TN} = ((P,\ T,\ F,\ M_0,\ L),\ D)$ and a time causal net $TN = (B, E, G, l, \tau)$,*

- *a mapping $\varphi : B \cup E \to P \cup T$ is a* homomorphism *from $TN$ to $\mathcal{TN}$ iff the following hold:*
  - *$\varphi(B) \subseteq P$, $\varphi(E) \subseteq T$;*
  - *the restriction of $\varphi$ to ${}^\bullet e$ is a bijection between ${}^\bullet e$ and ${}^\bullet\varphi(e)$, and the restriction of $\varphi$ to $e^\bullet$ is a bijection between $e^\bullet$ and $\varphi(e)^\bullet$, for all $e \in E$;*
  - *the restriction of $\varphi$ to ${}^\bullet TN$ is a bijection between ${}^\bullet TN$ and $M_0$;*
  - *$l(e) = L(\varphi(e))$, for all $e \in E$.*

- *a pair $\pi = (TN, \varphi)$ is a* time process of a time Petri net *$\mathcal{TN}$ iff $TN$ is a time causal net and $\varphi$ is a homomorphism from $TN$ to $\mathcal{TN}$;*

- *a time process $\pi = (TN, \varphi)$ of $\mathcal{TN}$ is* correct *iff for all $e \in E$ it holds:*

  *(a) $\tau(e) \geq \textbf{\textit{TOE}}_\pi({}^\bullet e, \varphi(e)) + Eft(\varphi(e))$,*

  *(b) $\forall t \in En(\varphi(Cut(e))) \diamond \tau(e) \leq \textbf{\textit{TOE}}_\pi(Cut(e), t) + Lft(t)$.*

  *Here, for a subset $B' \subseteq B_{TN}$ and a transition $t \in En(\varphi(B'))$, the time of enabling (TOE) of $t$, i.e. the latest global time moment when tokens appear in all input places of $t$, is defined as follows: $\textbf{\textit{TOE}}_\pi(B', t) = \max\Big(\{\tau_{TN}({}^\bullet b) \mid b \in B'_{[t]} \setminus {}^\bullet TN\} \cup \{0\}\Big)$, where $B'_{[t]} = \{b \in B' \mid \varphi_{TN}(b) \in {}^\bullet t\}$.*
  *Let $\mathcal{CP}(\mathcal{TN})$ denote the set of correct time processes of $\mathcal{TN}$.*

We now present for the time Petri net the relationships between its correct time processes and its interleaving/step firing sequences from the initial state.

**Proposition 1.** *[15, 16] Let $\mathcal{TN}$ be a time Petri net. Then,*

[2]A *(labeled over Act) time poset (partially ordered set)* is a tuple $\eta = (X, \preceq, \lambda, \tau)$ consisting of a finite set $X$ of elements; a reflexive, antisymmetric and transitive relation $\preceq$; a labeling function $l : X \to Act$; and a timing function $\tau : X \to \mathbb{T}$ such that $x \preceq x' \Rightarrow \tau(x) \leq \tau(x')$.

*(i) for any $\pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})$ with an $s(i)$-linearization $\rho = V_1 \dots V_k$ of $TN$, there is a unique step (interleaving) firing sequence $FS_\pi(\rho) = (\varphi(V_1), \tau(V_1) - 0) \dots (\varphi(V_k), \tau(V_k) - \tau(V_{k-1})) \in \mathcal{FS}_{s(i)}(\mathcal{TN})$;*

*(ii) for any step (interleaving) firing sequence $\sigma \in \mathcal{FS}_{s(i)}(\mathcal{TN})$, there is a unique (up to an isomorphism[3]) correct time process $\pi_\sigma = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})$ with a unique $s(i)$-linearization $\rho_\sigma$ of $TN$ such that $FS_{\pi_\sigma}(\rho_\sigma) = \sigma$.*

For correct time processes $\pi = (TN, \varphi)$, $\pi' = (TN', \varphi') \in \mathcal{CP}(\mathcal{TN})$, we say that $\pi$ is a $pos$-extension of $\pi'$ in $\mathcal{TN}$ iff $\eta(TN)$ is $pos$-extension of $\eta(TN')$, $B' \subset B$, $G' = G \cap (B' \times E' \cup E' \times B')$, and $\varphi' = \varphi\mid_{B' \cup E'}$.

We expand the above results to $pos$-extensions of correct time processes of TPNs.

**Lemma 1.** *Given a time Petri net $\mathcal{TN}$, $\sigma \in \mathcal{FS}_{s(i)}(\mathcal{TN})$, and $\pi \in \mathcal{CP}(\mathcal{TN})$ such that $\sigma = FS_\pi(\rho)$, where $\rho$ is an $s(i)$-linearization of $TN_\pi$,*

*(i) if $\widetilde{\pi}$ is a $pos$-extension of $\pi$ in $\mathcal{TN}$, then there is $\sigma\sigma' \in \mathcal{FS}_{s(i)}(\mathcal{TN})$ such that $\sigma\sigma' = FS_{\widetilde{\pi}}(\rho\rho')$, where $\rho\rho'$ is an $s(i)$-linearization of $TN_{\widetilde{\pi}}$;*

*(ii) if $\sigma\sigma' \in \mathcal{FS}_{s(i)}(\mathcal{TN})$, there is $\widetilde{\pi} \in \mathcal{CP}(\mathcal{TN})$ such that $\widetilde{\pi}$ is a $pos$-extension of $\pi$ in $\mathcal{TN}$ and $\sigma\sigma' = FS_{\widetilde{\pi}}(\rho\rho')$, where $\rho\rho'$ is an $s(i)$-linearization of $TN_{\widetilde{\pi}}$.*

## 2.3. Causal Tree Semantics of TPNs

Causal trees [17] are synchronisation trees which carry in their labels additional information about causes of actions thus providing us with an interleaving description of concurrent processes, which faithfully expresses causality. Time causal trees are generalizations of causal trees by adding timing. In the time causal tree of the TPN, the nodes are the interleaving firing sequences of the TPN, and an edge exists between two nodes if the second one is a direct extension of the first one. The causes in the edge labels are calculated based on the causality relations in the correct time processes of the TPN corresponding to the nodes (the interleaving firing sequences).

**Definition 4.** *The* time causal tree *of the TPN $\mathcal{TN}$, $TCT(\mathcal{TN})$, is a tree $(\mathcal{FS}_i(\mathcal{TN}), Ed, \phi)$, where $\mathcal{FS}_i(\mathcal{TN})$ is the set of nodes with the root $\epsilon$; $Ed = \{(\sigma, \sigma(t,\theta)) \mid \sigma, \sigma(t,\theta) \in \mathcal{FS}(\mathcal{TN})\}$ is the set of edges; $\phi$ is the labeling function such that $\phi(\epsilon) = \epsilon$ and $\phi(\sigma, \sigma(t,\theta)) = (L_{\mathcal{TN}}(t), \theta, K)$, with $\sigma = FS_{\pi_\sigma}(\rho_\sigma = e_1 \dots e_n)$, $\sigma(t,\theta) = FS_{\pi_{\sigma(t,\theta)}}(\rho_{\sigma(t,\theta)} = e_1 \dots e_n e)$, $K = \{n - l + 1 \mid e_l \prec_{TN_{\pi_{\sigma(t,\theta)}}} e\}$. Let $path(\sigma)$ be the path starting from the root and finishing in the node $\sigma$ of $TCT(\mathcal{TN})$[4].*

[3]Time processes $\pi = (TN, \varphi)$ and $\pi' = (TN', \varphi') \in \mathcal{CP}(\mathcal{TN})$ are *isomorphic* (denoted $\pi \simeq \pi'$) iff there exists a bijective mapping $\beta : B \cup E \to B' \cup E'$ such that (i) $\beta(B) = B'$ and $\beta(E) = E'$; (ii) $x\, G\, y \iff \beta(x)\, G'\, \beta(y)$, for all $x, y \in B \cup E$; (iii) $l(e) = l'(\beta(e))$, for all $e \in E$; (iv) $\tau(e) = \tau'(\beta(e))$, for all $e \in E$; (v) $\varphi(x) = \varphi'(\beta(x))$, for all $x \in B \cup E$.

[4]We assume $path(\epsilon) = \epsilon$. Notice that in $TCT(\mathcal{TN})$, for any node $\sigma \in \mathcal{FS}_i(\mathcal{TN})$, there is a path starting from the root and finishing in $\sigma$.

**Example 2.** *Consider the time Petri net $\widetilde{\mathcal{TN}}$ (see Figure 1) and its interleaving firing sequence $\sigma = (t_1, 0.5)\ (t_4, 0)\ (t_3, 1)\ (t_2, 2)\ (t_5, 2) \in \mathcal{FS}_i(\widetilde{\mathcal{TN}})$. It is easy to get that $\phi(path(\sigma)) = (a, 0.5, \emptyset)\ (a, 0.5, \emptyset)\ (b, 1, \{2\})\ (a, 2, \{1\})\ (c, 2, \{1, 3\})$.*

We finally establish some relationships between correct time processes and labeled paths in the time causal trees of two time Petri nets.

**Proposition 2.** *Let $\mathcal{TN}, \mathcal{TN}'$ be time Petri nets. Then,*

*(i) for any $\pi \in \mathcal{CP}(\mathcal{TN})$ and $\pi' \in \mathcal{CP}(\mathcal{TN}')$ with an isomorphism $f : \eta(TN_\pi) \to \eta(TN_{\pi'})$, $\phi(path(FS_\pi(\rho))) = \phi'(path(FS_{\pi'}(f(\rho))))$, for any $i$-linearization $\rho$ of $TN_\pi$;*

*(ii) for any $\sigma \in \mathcal{FS}_i(\mathcal{TN})$ and $\sigma' \in \mathcal{FS}_i(\mathcal{TN}')$ such that $\phi(path(\sigma)) = \phi'(path(\sigma'))$, there is an isomorphism $f : \eta(TN_{\pi_\sigma}) \to \eta(TN_{\pi_{\sigma'}})$ such that $f(\rho_\sigma) = \rho_{\sigma'}$.*

## 3. Testing Equivalences

Interleaving testing equivalence deals with the experiments on the TPN — sequences of actions with their times — and the behaviors which are tested for after the experiments — sets of actions with their times. So, it checks whether actions with times, given as a test, can be executed after a sequence of actions with times, specified as an experiment. Here, both the experiments and tests represent interleaving semantics.

**Definition 5.** *Given time Petri nets $\mathcal{TN}$ and $\mathcal{TN}'$,*

- *for a sequence $w \in (Act \times \mathbb{T})^*$ and a set $W \subseteq (Act \times \mathbb{T})$, $\mathcal{TN}$ **after** $w$ $\mathbf{MUST}^{int}_{int}$ $W$ iff for all firing sequences $\sigma \in \mathcal{FS}_i(\mathcal{TN})$ such that $L(\sigma) = w$, there exists an element $(a, \theta) \in W$ and a firing sequence $\sigma(t, \theta) \in \mathcal{FS}_i(\mathcal{TN})$ such that $L(\sigma(t, \theta)) = w(a, \theta)$;*
- *$\mathcal{TN}$ and $\mathcal{TN}'$ are* interleaving testing equivalent *(denoted $\mathcal{TN} \sim^{int}_{int} \mathcal{TN}'$) iff for all sequences $w \in (Act \times \mathbb{T})^*$ and for all sets $W \subseteq (Act \times \mathbb{T})$, it holds:*

$$\mathcal{TN} \textbf{ after } w\ \mathbf{MUST}^{int}_{int}\ W \iff \mathcal{TN}' \textbf{ after } w\ \mathbf{MUST}^{int}_{int}\ W.$$

In step testing, the experiments on the TPN are sequences of multisets over sets of actions with their times and the tests checked after the experiments are sets of multisets over sets of actions with their times. So, it checks whether multisets over sets of actions with times, given as a test, can be executed after a sequence of multisets over sets of actions with times, specified as an experiment. Thereby, both the experiments and tests respect step semantics.

**Definition 6.** *Given time Petri nets $\mathcal{TN}$ and $\mathcal{TN}'$,*

- *for a sequence $w \in (Act^{\mathbb{N}} \times \mathbb{T})^*$ and a set $W \subseteq (Act^{\mathbb{N}} \times \mathbb{T})$, $\mathcal{TN}$ **after** $w$ $\mathbf{MUST}^{step}_{step}$ $W$ iff for all firing sequences $\sigma \in \mathcal{FS}_s(\mathcal{TN})$ such that $L(\sigma) = w$, there exists an element $(A, \theta) \in W$ and a firing sequence $\sigma(U, \theta) \in \mathcal{FS}_s(\mathcal{TN})$ such that $L(\sigma(U, \theta)) = w(A, \theta)$;*

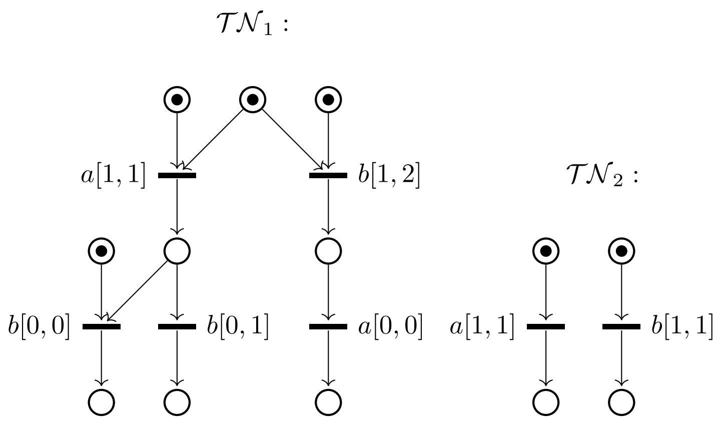


**Figure 2:** The $\sim_{int}^{int}$–equivalent but neither $\sim_{step}^{step}$– nor $\sim_{pos}^{pos}$–equivalent TPNs $\mathcal{TN}_1$ and $\mathcal{TN}_2$.

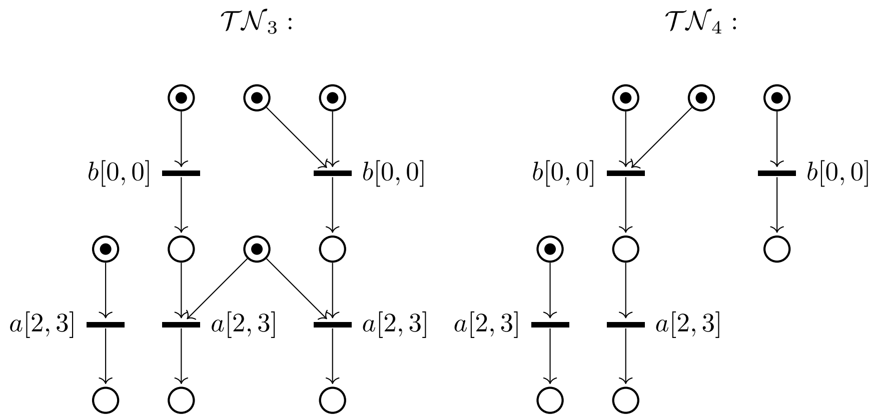


**Figure 3:** The $\sim_{step}^{step}$–equivalent but not $\sim_{pos}^{pos}$–equivalent TPNs $\mathcal{TN}_3$ and $\mathcal{TN}_4$.

- *$\mathcal{TN}$ and $\mathcal{TN}'$ are* step testing equivalent *(denoted $\mathcal{TN} \sim_{step}^{step} \mathcal{TN}'$) iff for all sequences $w \in (Act^{\mathbb{N}} \times \mathbb{T})^*$ and for all sets $W \subseteq (Act^{\mathbb{N}} \times \mathbb{T})$, it holds:*

$$\mathcal{TN} \textbf{ after } w \ \mathbf{MUST}_{step}^{step} \ W \iff \mathcal{TN}' \textbf{ after } w \ \mathbf{MUST}_{step}^{step} \ W.$$

The idea of partial order testing is that the experiments on the TPN are time posets and the tests, that are examined after the experiments, are sets of $pos$-extensions of the experiments. This contrasts with partial order based testing investigated in the paper in [14], where the tests contain sets of time posets extending the experiments by single actions with their times. From now on, we denote $pos$-extensions of a time poset $TP$ by $\mathbf{TP}_{TP}$.

**Definition 7.** *Given time Petri nets $\mathcal{TN}$ and $\mathcal{TN}'$,*

- *for a time poset $TP$ and a set $\mathbf{TP} \subseteq \mathbf{TP}_{TP}$, $\mathcal{TN}$ **after** $TP$ $\mathbf{MUST}_{pos}^{pos}$ $\mathbf{TP}$ iff for all time processes $\pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})$ and for all isomorphisms $f : \eta(TN) \longrightarrow TP$,*

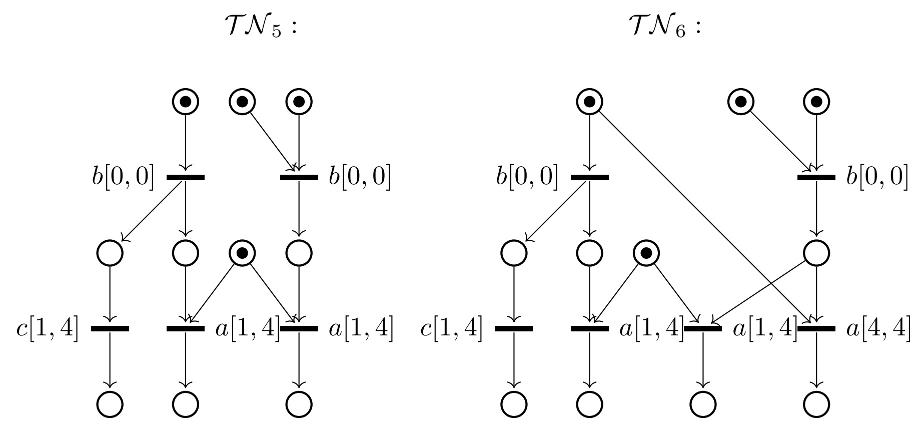


**Figure 4:** The $\sim_\star^\star$–equivalent TPNs $\mathcal{TN}_5$ and $\mathcal{TN}_6$, for $\star \in \{int, step, pos\}$.

*there exists a time poset $TP' \in \mathbf{TP}$, a time process $\pi' = (TN', \varphi') \in \mathcal{CP}(\mathcal{TN})$, and an isomorphism $f' : \eta(TN') \longrightarrow TP'$, such that $\pi'$ is a pos-extension of $\pi$ and $f \subset f'$;*

- *$\mathcal{TN}$ and $\mathcal{TN}'$ are* poset testing equivalent *(denoted $\mathcal{TN} \sim_{pos}^{pos} \mathcal{TN}'$) iff for all time posets $TP$ and for all sets $\mathbf{TP} \subseteq \mathbf{TP}_{TP}$, it holds:*

$$\mathcal{TN}\ \mathbf{after}\ TP\ \mathbf{MUST}_{pos}^{pos}\mathbf{TP} \iff \mathcal{TN}'\ \mathbf{after}\ TP\ \mathbf{MUST}_{pos}^{pos}\mathbf{TP}.$$

**Theorem 1.** *Given time Petri nets $\mathcal{TN}$ and $\mathcal{TN}'$,*

$$\mathcal{TN} \sim_{pos}^{pos} \mathcal{TN}' \implies \mathcal{TN} \sim_{step}^{step} TN \implies \mathcal{TN} \sim_{int}^{int} \mathcal{TN}'.$$

The implications in the theorem above do not hold in the opposite directions, as demonstrated in the example below.

**Example 3.** *The time Petri nets $\mathcal{TN}_1$ and $\mathcal{TN}_2$, shown in Figure 2, are $\sim_{int}^{int}$–equivalent but they are neither $\sim_{step}^{step}$– nor $\sim_{pos}^{pos}$–equivalent. First, check that $\mathcal{TN}_1$ and $\mathcal{TN}_2$ are not $\sim_{step}^{step}$–equivalent. It is easy to see that $\mathcal{TN}_1$ **after** $w = (1'a + 1'b, 1)$ $\mathbf{MUST}_{step}^{step}$ $W = \emptyset$, because in $\mathcal{FS}_s(\mathcal{TN}_1)$ there is no firing sequence $\sigma$ such that $L(\sigma) = w$. However, this is not the case in $\mathcal{TN}_2$, since in $\mathcal{FS}_s(\mathcal{TN}_2)$ there exists a firing sequence $\sigma$ such that $L(\sigma) = w$ and it is impossible to find any element $(A, \theta)$ of $W$ so as to locate in $\mathcal{FS}_s(\mathcal{TN}_2)$ a firing sequence $\sigma(U, \theta)$ such that $L(\sigma(U, \theta)) = w(A, \theta)$. Hence, it hods that $\neg(\mathcal{TN}_2$ **after** $w = (1'a + 1'b, 1)$ $\mathbf{MUST}_{step}^{step}$ $W = \emptyset)$. Second, verify that $\mathcal{TN}_1$ and $\mathcal{TN}_2$ are not $\sim_{pos}^{pos}$–equivalent. Define a poset $TP = (\{x_1, x_2\}, \preceq, \lambda, \tau)$ (with $\preceq = \{(x_1, x_1), (x_2, x_2)\}$, $\lambda(x_1) = a$, $\lambda(x_2) = b$, $\tau(x_1) = \tau(x_2) = 1$). For any time process $\pi_1 = (TN_1, \varphi_1) \in \mathcal{CP}(\mathcal{TN}_1)$, there is no isomorphism $f_1 \colon \eta(TN_1) \longrightarrow TP$. So, it is true that $\mathcal{TN}_1$ **after** $TP$ $\mathbf{MUST}_{pos}^{pos}$ $\mathbf{TP} = \emptyset$. However, this is not the case in $\mathcal{TN}_2$ because there is a time process $\pi_2 = (TN_2, \varphi_2) \in \mathcal{CP}(\mathcal{TN}_2)$, with $E_{TN_2}$ containing two concurrent events labeled by $a$ and $b$, both with time value $1$, and an isomorphism $f_2 \colon \eta(TN_2) \longrightarrow TP$, and we cannot find any pos-extension of $TP$ in $\mathbf{TP}$. Hence, it hods that $\neg(\mathcal{TN}_2$ **after** $TP$ $\mathbf{MUST}_{pos}^{pos}$ $\mathbf{TP} = \emptyset)$.*

*The time Petri nets $\mathcal{TN}_3$ and $\mathcal{TN}_4$, shown in Figure 3, are $\sim^{step}_{step}$–equivalent but not $\sim^{pos}_{pos}$–equivalent. Let's make sure of the latter. Define posets $TP = (\{x_1\}, \preceq, \lambda, \tau)$ (with $\preceq = \{(x_1, x_1)\}$, $\lambda(x_1) = b$, $\tau(x_1) = 0$) and $TP' = (\{x_1, x_2, x_3, x_4\}, \preceq', \lambda', \tau')$ (with $\preceq' = \{(x_i, x_i) \mid 1 \leq i \leq 4\} \cup \{(x_2, x_3)\}$, $\lambda'(x_1) = \lambda'(x_2) = b$, $\lambda'(x_3) = \lambda'(x_4) = a$, $\tau'(x_1) = \tau'(x_2) = 0$, and $\tau'(x_3) = \tau'(x_4) = 2.9$. It is easy to see that $TP'$ is a pos-extension of $TP$. For any time process $\pi_1 = (TN_1, \varphi_1) \in \mathcal{CP}(\mathcal{TN}_3)$, with $E_{TN_1}$ consisting of an event with label $b$ and time value $0$, and any isomorphism $f_1 \colon \eta(TN_1) \longrightarrow TP$, we can find a pos-extension $\pi_1' = (TN_1', \varphi_1') \in \mathcal{CP}(\mathcal{TN}_1)$, with $E_{TN_1'}$ consisting of two concurrent events, both with label $b$ and time value $0$, and two concurrent events, both with label $a$ and time value $2.9$, and, moreover, one of the two events labeled by $a$ is causally preceded by the added event labeled by $b$, and an isomorphism $f_1' \colon \eta(TN_1') \longrightarrow TP'$ such that $f_1 \subset f_1'$. But this is not the case in $\mathcal{TN}_4$.*

*The time Petri nets $\mathcal{TN}_5$ and $\mathcal{TN}_6$, depicted in Figure 4, are $\sim^{\star}_{\star}$–equivalent, for $\star \in \{int, step, pos\}$.* □

At last, the definition of testing equivalence based on the causal trees of TPNs is developed. In doing so the experiments are considered as sequences over the alphabet $(Act \times \mathbb{T} \times 2^{\mathbb{N}})$ (corresponding to labeled paths from the roots in the causal trees) and the tests are specified as sets of non-empty sequences over the same alphabet (corresponding to sets of extensions of the labeled paths in the causal trees). In the paper [14], the tests directly extend the experiments by single elements, not by sequences of elements, from the set $(Act \times \mathbb{T} \times 2^{\mathbb{N}})$.

**Definition 8.** *Given time Petri nets $\mathcal{TN}$ and $\mathcal{TN}'$ with their time causal trees $TCT(\mathcal{TN})$ and $TCT(\mathcal{TN}')$, respectively,*

- *for a sequence $w \in (Act \times \mathbb{T} \times 2^{\mathbb{N}})^*$ and a set $\mathbf{W} \subseteq (Act \times \mathbb{T} \times 2^{\mathbb{N}})^+$, we say $TCT(\mathcal{TN})$* **after** *$w$* $\mathbf{MUST}^{ext}_{ct}$ $\mathbf{W}$ *iff for all paths $u$ in $TCT(\mathcal{TN})$ from its root to a node $n$ such that $\phi(u) = w$, there exists $w' \in \mathbf{W}$ and a path $u'$ starting from the node $n$, such that $\phi(u') = w'$;*

- *$\mathcal{TN}$ and $\mathcal{TN}'$ are causal tree testing equivalent ($\mathcal{TN} \sim^{ext}_{ct} \mathcal{TN}'$) iff for all sequences $w \in (Act \times \mathbb{T} \times 2^{\mathbb{N}})^*$ and sets $\mathbf{W} \subseteq (Act \times \mathbb{T} \times 2^{\mathbb{N}})^+$, it holds:*

$$TCT(\mathcal{TN}) \textbf{ after } w\, \mathbf{MUST}^{ext}_{ct}\mathbf{W} \iff TCT(\mathcal{TN}') \textbf{ after } w\, \mathbf{MUST}^{ext}_{ct}\mathbf{W}.$$

We finally expand the main result of [14] by establishing the coincidence of poset and causal tree testing equivalences with extended tests, in the setting of TPNs.

**Theorem 2.** *Given time Petri nets $\mathcal{TN}$ and $\mathcal{TN}'$,*

$$\mathcal{TN} \sim^{pos}_{pos} \mathcal{TN}' \iff \mathcal{TN} \sim^{ext}_{ct} \mathcal{TN}'.$$

## 4. Concluding Remarks

We have specified and studied several testing equivalences based on concurrent semantics, in the setting of contact-free time Petri nets. In doing so, we dealt with various conceptions of the behavior of the time Petri net: interleaving/step firing sequences, time processes, from causal nets of which partial orders are derived, and time causal tree, constructed from interleaving firing sequences and partial orders. We have demonstrated that interleaving testing equivalence (with the experiments as labeled interleaving firing sequences and with the tests as experiments extensions by single actions with their times) is coarser than step testing (with the experiments as labeled step firing sequences and the tests as sequences extensions by steps of concurrent actions with their times), which, in turn, is coarser than poset testing (with time posets as experiments and their $pos$-extensions as tests). As the main result, the latter equivalence has been established to coincide with causal tree testing (based on labeled paths and their extensions in time causal trees of TPNs).

As for future work, we plan to investigate the equivalences and semantics under consideration in the framework of Petri nets with weak timing policy [18]. Also, it would be interesting to see whether open intervals in the specification of TPNs influence the results obtained here.

## References

[1] R. de Nicola, M. Hennessy, Testing equivalence for processes, in: J. Díaz (Ed.), Automata, Languages and Programming, 10th Colloquium, Barcelona, Spain, July 18-22, 1983, Proceedings, volume 154 of *Lecture Notes in Computer Science*, Springer, 1983, pp. 548–560. URL: https://doi.org/10.1007/BFb0036936. doi:10.1007/BFb0036936.

[2] R. de Nicola, Extensional equivalences for transition systems, Acta Informatica 24 (1987) 211–237. URL: https://doi.org/10.1007/BF00264365. doi:10.1007/BF00264365.

[3] R. Cleaveland, M. Hennessy, Testing equivalence as a bisimulation equivalence, Formal Aspects Comput. 5 (1993) 1–20. URL: https://doi.org/10.1007/BF01211314. doi:10.1007/BF01211314.

[4] L. Pomello, G. Rozenberg, C. Simone, A survey of equivalence notions for net based systems, in: G. Rozenberg (Ed.), Advances in Petri Nets 1992, The DEMON Project, volume 609 of *Lecture Notes in Computer Science*, Springer, 1992, pp. 410–472. URL: https://doi.org/10.1007/3-540-55610-9_180. doi:10.1007/3-540-55610-9_180.

[5] L. Aceto, History preserving, causal and mixed-ordering equivalence over stable event structures, Fundam. Informaticae 17 (1992) 319–331.

[6] L. Aceto, R. de Nicola, A. Fantechi, Testing equivalences for event structures, in: M. V. Zilli (Ed.), Mathematical Models for the Semantics of Parallelism, Advanced School, Rome, Italy, September 24 - October 1, 1986, Proceedings, volume 280 of *Lecture Notes in Computer Science*, Springer, 1986, pp. 1–20. URL: https://doi.org/10.1007/3-540-18419-8_9. doi:10.1007/3-540-18419-8_9.

[7] U. Goltz, H. Wehrheim, Causal testing, in: W. Penczek, A. Szalas (Eds.), Mathematical Foundations of Computer Science 1996, 21st International Symposium, MFCS'96, Cracow, Poland, September 2-6, 1996, Proceedings, volume 1113 of *Lecture Notes in Computer Science*, Springer, 1996, pp. 394–406. URL: https://doi.org/10.1007/3-540-61550-4_165. doi:10.1007/3-540-61550-4_165.

[8] R. Cleaveland, A. E. Zwarico, A theory of testing for real-time, in: Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The

Netherlands, July 15-18, 1991, IEEE Computer Society, 1991, pp. 110–119. URL: https://doi.org/10.1109/LICS.1991.151635. doi:10.1109/LICS.1991.151635.

[9] M. Hennessy, T. Regan, A process algebra for timed systems, Inf. Comput. 117 (1995) 221–239. URL: https://doi.org/10.1006/inco.1995.1041. doi:10.1006/inco.1995.1041.

[10] F. Corradini, W. Vogler, L. Jenner, Comparing the worst-case efficiency of asynchronous systems with PAFAS, Acta Informatica 38 (2002) 735–792. URL: https://doi.org/10.1007/s00236-002-0094-3. doi:10.1007/s00236-002-0094-3.

[11] L. F. L. Díaz, D. de Frutos-Escrig, Denotational semantics for timed testing, in: M. Bertran, T. Rus (Eds.), Transformation-Based Reactive Systems Development, 4th International AMAST Workshop on Real-Time Systems and Concurrent and Distributed Software, ARTS'97, Palma, Mallorca, Spain, May 21-23, 1997, Proceedings, volume 1231 of *Lecture Notes in Computer Science*, Springer, 1997, pp. 368–382. URL: https://doi.org/10.1007/3-540-63010-4_25. doi:10.1007/3-540-63010-4_25.

[12] E. Bihler, W. Vogler, Timed Petri nets: Efficiency of asynchronous systems, in: M. Bernardo, F. Corradini (Eds.), Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures, volume 3185 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 25–58. URL: https://doi.org/10.1007/978-3-540-30080-9_2. doi:10.1007/978-3-540-30080-9_2.

[13] M. V. Andreeva, I. B. Virbitskaite, Observational equivalences for timed stable event structures, Fundam. Informaticae 72 (2006) 1–19. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi72-1-3-02.

[14] E. N. Bozhenkova, I. B. Virbitskaite, Testing equivalences of time Petri nets, Program. Comput. Softw. 46 (2020) 251–260. URL: https://doi.org/10.1134/S0361768820040040. doi:10.1134/S0361768820040040.

[15] T. Aura, J. Lilius, A causal semantics for time Petri nets, Theor. Comput. Sci. 243 (2000) 409–447. URL: https://doi.org/10.1016/S0304-3975(99)00114-0. doi:10.1016/S0304-3975(99)00114-0.

[16] I. B. Virbitskaite, D. Bushin, E. Best, True concurrent equivalences in time Petri nets, Fundam. Informaticae 149 (2016) 401–418. URL: https://doi.org/10.3233/FI-2016-1454. doi:10.3233/FI-2016-1454.

[17] P. Darondeau, P. Degano, Refinement of actions in event structures and causal trees, Theor. Comput. Sci. 118 (1993) 21–48. URL: https://doi.org/10.1016/0304-3975(93)90361-V. doi:10.1016/0304-3975(93)90361-V.

[18] P. Reynier, A. Sangnier, Weak time Petri nets strike back!, in: M. Bravetti, G. Zavattaro (Eds.), CONCUR 2009 - Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings, volume 5710 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 557–571. URL: https://doi.org/10.1007/978-3-642-04081-8_37. doi:10.1007/978-3-642-04081-8_37.

# Left Recursion by Recursive Ascent

Roman R. Redziejowski

**Abstract**
Recursive-descent parsers can not handle left recursion, and several solutions to this problem have been suggested. This paper presents yet another solution. The idea is to modify recursive-descent parser so that it reconstructs left-recursive portions of syntax tree bottom-up, by "recursive ascent".

**Keywords**
parsing, recursive descent, left recursion

## 1. Introduction

Recursive-descent parser is a collection of "parsing procedures" that correspond to different syntactic units and call each other recursively. Some of these procedures must choose what to call next, and the choice is made by looking at the input ahead, on depth-first basis. This process does not work if the grammar is left-recursive: the procedure may indefinitely call itself, facing the same input. One suggested solution [1] counts the number of recursive calls of each procedure and stops when it reaches a pre-set bound. The process is repeated with the bound starting with 1 and gradually increased. Another solution [2, 3] saves the input consumed by invocations of parsing procedures and tricks the parser to use the saved result instead of making recursive call. The third idea sees parsing as reconstruction of input's syntax tree. The classical process builds that tree top-down, but [4] suggests that portions of the tree involved in left recursion can be reconstructed starting from the bottom. We present here an approach based on this idea. To reconstruct portions of the tree bottom-up we use procedures that call each other recursively. These procedures can be regarded as new parsing procedures. We incorporate them into the recursive-descent parser, and the result is recursive-descent parser for a new grammar, referred to as the "dual grammar". The dual grammar is (normally) not left-recursive and defines the same language as the original one.

After the necessary definitions in Section 2, Section 3 gives an example of recursive ascent, and shows that the procedures performing it can be seen as parsing procedures for new non-terminals. Section 4 incorporates these new non-terminals into the original grammar, thus obtaining the dual grammar. Two Propositions state the essential properties of that grammar. Section 5 discusses handling choice expressions, and Section 6 discusses the elements omitted to simplify the presentation. Section 7 contains some final remarks: relation to other solutions and unsolved problems. Proofs of the Propositions are given in the Appendix.

*29th international Workshop on Concurrency, Specification and Programming (CS&P'21), September 2021, Berlin, Germany*
roman@redz.se (R. R. Redziejowski)
http://www.romanredz.se (R. R. Redziejowski)

## 2. Basic concepts

We consider a BNF-like grammar $G = (\mathbb{N}, \Sigma, \mathbb{E}, \mathcal{E}, N_s)$ with finite set $\mathbb{N}$ of *non-terminals*, finite set $\Sigma$ of *terminals*, finite set $\mathbb{E}$ of *expressions*, function $\mathcal{E}$ from non-terminals to expressions, and the *start symbol* $N_s$.
An expression is one of these:

- $a \in \Sigma$ ("terminal"),
- $N \in \mathbb{N}$ ("non-terminal"),
- $e_1 \dots e_n$ ("sequence"),
- $e_1 | \dots | e_n$ ("choice"),

where each of $e_i$ is an expression. The function $\mathcal{E}$ is defined by a set of rules of the form $N \to e$, where $e$ is the expression assigned by $\mathcal{E}$ to non-terminal $N$. We often write $N \to e$ to mean $e = \mathcal{E}(N)$. To simplify the presentation, we did not include the empty string $\varepsilon$ among expressions. In the following, expressions $a \in \Sigma$ and $N \in \mathbb{N}$ will be viewed as special cases of choice expression with $n = 1$.

Non-terminal $N \to e_1 \dots e_n$ *derives* the string $e_1 \dots e_n$ of symbols, while $N \to e_1 | \dots | e_n$ derives one of $e_1, \dots, e_n$. The derivation is repeated to obtain a string of terminals. This process is represented by syntax tree.

Figures 1 and 2 are examples of grammar $G$, showing syntax trees of strings derived from the start symbol. The set of all strings derived from $N \in \mathbb{N}$ is called the *language* of $N$ and is denoted by $\mathcal{L}(N)$.

$$\mathbb{N} = \{\mathtt{Z,A,A1,B,B1,B2}\}$$
$$\Sigma = \{\mathtt{a,b,x,y}\}$$
$$Ns = \mathtt{Z}$$

```
Z  → x A y
A  → A1 | a
A1 → B a
B  → B1 | B2 | b
B1 → A b
B2 → B b
```

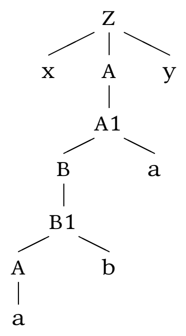


**Figure 1:** Example of grammar $G$ and syntax tree of `'xabay'`

For $N \in \mathbb{N}$ and $e \in \mathbb{N} \cup \Sigma$, define $N \xrightarrow{\text{first}} e$ to mean that parsing procedure for $N$ may call that for $e$ on the same input. We have thus:

- If $N \to e_1 \dots e_n$, $N \xrightarrow{\text{first}} e_1$.
- If $N \to e_1 | \dots | e_n$, $N \xrightarrow{\text{first}} e_i$ for $1 \le i \le n$.

Let $\xrightarrow{\text{First}}$ be the transitive closure of $\xrightarrow{\text{first}}$. Non-terminal $N \in \mathbb{N}$ is *recursive* if $N \xrightarrow{\text{First}} N$. The set of all recursive non-terminals of $G$ is denoted by $\mathbb{R}$. All non-terminals in Figure 1 except Z, and all non-terminals in Figure 2 are recursive.

$$\begin{array}{lcl} \mathbb{N} & = & \{\mathtt{E},\mathtt{E1},\mathtt{F},\mathtt{F1}\} \\ \Sigma & = & \{\mathtt{a},\mathtt{b},\mathtt{x},\mathtt{z}\} \\ Ns & = & \mathtt{E} \end{array}$$

$$\begin{array}{l} \mathtt{E} \rightarrow \mathtt{E1} \mid \mathtt{F} \\ \mathtt{E1} \rightarrow \mathtt{E} + \mathtt{F} \\ \mathtt{F} \rightarrow \mathtt{F1} \mid \mathtt{a} \\ \mathtt{F1} \rightarrow \mathtt{F} * \mathtt{a} \end{array}$$

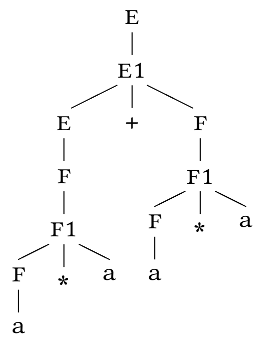


**Figure 2:** Example of grammar $G$ and syntax tree of 'a*a+a*a'

Define relation between recursive $N_1, N_2 \in \mathbb{R}$ that holds if $N_1 \xrightarrow{\text{First}} N_2 \xrightarrow{\text{First}} N_1$. This is an equivalence relation that partitions $\mathbb{R}$ into equivalence classes. We call them *recursion classes*. The recursion class of $N$ is denoted by $\mathbb{C}(N)$. All non-terminals in Figure 1 belong to the same class; the grammar of Figure 2 has two recursion classes: {E, E1} and {F, F1}.

In syntax tree, the leftmost path emanating from any node is a chain of nodes connected by $\xrightarrow{\text{first}}$. Suppose $N_1$ and $N_2$ belonging to the same recursion class $\mathbb{C}$ appear on the same leftmost path. Any non-terminal $N$ between them must also belong to $\mathbb{C}$, which follows from the fact that $N_1 \xrightarrow{\text{First}} N \xrightarrow{\text{First}} N_2 \xrightarrow{\text{First}} N_1$. It means that members of $\mathbb{C}$ appearing on the same leftmost path must form an uninterrupted sequence. We call such sequence a *recursion path* of class $\mathbb{C}$. The only recursion path in Figure 1 is the whole leftmost path from the first A without final a. The syntax tree in Figure 2 has two recursion paths, one starting with E and another with F.

Let $N \rightarrow e_1 \dots e_n$ be on a recursion path. The next item on the leftmost path is $e_1$, and it must belong to $\mathbb{C}(N)$ to ensure $N \xrightarrow{\text{First}} N$. It follows that the last item on a recursion path must be $N \rightarrow e_1 | \dots | e_n$ where at least one of $e_i$ is not a member of $\mathbb{C}(N)$. Such $N$ is called an *exit* of $\mathbb{C}(N)$, and its alternatives outside $\mathbb{C}(N)$ are the *seeds* of $\mathbb{C}(N)$. In Figure 1, both A and B are exits, and the seeds are a and b. In Figure 2, E and F are exits of their respective classes, and the corresponding seeds are F and a.

A recursive $N$ that appears in expression for a non-terminal outside $\mathbb{C}(N)$, or is the start symbol, is an *entry* of its recursion class. It is the only non-terminal that can be first in a recursion path. The recursion class of Figure 1 has entry A, and recursion classes of Figure 2 have E and F as their respective entries.
To simplify presentation, we assume that each class has only one entry.

## 3. Recursive ascent

Recursive descent constructs the syntax tree implicitly, as the structure of its procedure calls. We assume that to serve any purpose, this tree has to be somehow registered. Thus, we assume that parsing procedures include "semantic actions" that actually build data structure representing the tree.

We suggest that parsing procedure for entry $E$ builds its syntax tree in a special way, illustrated below by parsing the string `'xabay'` from Figure 1. The parsing starts with procedure for non-recursive Z that, in the usual way, calls the procedures for `'x'`, A, and `'y'`. After consuming `'x'`, Z applies A to `'abay'`. The procedure for entry A is not the usual choice between A1 and a; instead, it reconstructs the subtree A as follows:

1. The recursion path from A must end with a seed. The seeds are a and b. Decide to try a.
2. Apply expression a to `'abay'`. It consumes `'a'`, leaving `'bay'`. Use a as start of the tree.
3. The only possible parent of seed a in the syntax tree is its exit A. Add A on top of the tree, with a as child.
4. We have a tree for A, but decide to continue.
5. A appears only in B1→A b, so B1 is the only possible parent of A.
6. The tree for A is already constructed. Complete B1 by applying expression b to the remaining `'bay'`. It consumes b, leaving `'ay'`. Add B1 on top of the tree, with A , b as children.
7. B1 appears only in B→B1 | B2 | b, so B is the only possible parent of B1.
8. Add B on top of the tree, with B1 as child.
9. The possible parents of B are A1→B a and B2→B b. Decide for A1.
10. The tree for B is already constructed. Complete A1 by applying expression a to `'ay'`. It consumes `'a'`, leaving `'y'`. Add A1 on top of the tree with B , a as children.
11. The only possible parent of A1 is A→A1.
12. Add A on top of the tree, with A1 as child.
13. We have a tree for A, and decide to stop. Return the tree of the consumed `'aba'` to Z, which continues to consume y and constructs the tree for `'xabay'`.

In general, the procedure for entry $E$ chooses and executes one of procedures that correspond to different seeds. In the example, the choice is made in step 1, and the selected procedure consists of steps 2-13. The procedure for seed $S$ starts with constructing the tree for $S$ and follows by executing a procedure that adds node for the containing exit $X$; then it proceeds to grow the resulting tree towards $E$. These are the steps 2 respectively 3-13 in our example. This may be seen as parsing procedure that implements a new expression for $E$:

$$E \to S_1 \, \$X_1 | \dots | S_n \, \$X_n \tag{1}$$

where $S_i$ are all the seeds of $\mathbb{C}(E)$ and $X_i$ are the exits containing them.

We can see $\$X$ as a special case of procedure $\$R$ that adds a new node $R$ on top of previously constructed tree. This is the case in steps 6, 8, 10, and 12 of our example. The procedure then continues to build the tree, which is in each case done in the subsequent steps up to the step 13.

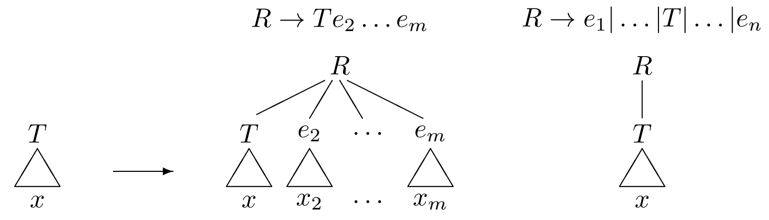


**Figure 3:** `Adding R`

The way of adding $R$ on top of tree $T$ depends on $R$ as illustrated in Figure 3.

If $R \to e_1 e_2 \dots e_m$ (where $e_1 = T$), $\$R$ builds the trees for $e_2, \dots, e_m$, binds them with $T$ into the tree for $R$. The whole procedure can be seen as parsing procedure for new non-terminal:

$$\$R \to e_2 \dots e_m \; \#R \tag{2}$$

where $e_2, \dots, e_m$ are procedures for these expressions and $\#R$ continues the growing.

If $R \to e_1 | \dots | e_m$ (where $T = e_i$ for some $i$), the procedure just adds $R$ on top of $T$ and proceeds to grow that tree. This can be seen as parsing procedure for:

$$\$R \to \#R \, . \tag{3}$$

Procedure $\#R$ consists of choosing a possible parent of node $R$ and adding that parent to the tree. This can be seen as parsing procedure for another new non-terminal:

$$\#R \to \$P_1 | \dots | \$P_n \tag{4}$$

where $P_i$ are all members of $\mathbb{C}$ such that $P_i \xrightarrow{\text{first}} R$.
In our example, the choice (4) is performed in steps 5, 7, 9, and 11.

If $R$ is identical to $E$, $\#R$ can choose to terminate building of the tree. Therefore, $\#E$ must have an alternative to allow that:

$$\#E \to \$P_1 | \dots | \$P_n | \$\varepsilon \tag{5}$$

where procedure $\$\varepsilon$ returns the tree constructed for $E$. It was not chosen in step 4, but terminated the process in step 13.

## 4. The dual grammar

As shown above, parsing procedure for an entry expression can be implemented by a set of procedures that reconstruct portion of syntax tree bottom-up, by "recursive ascent". These procedures can be seen as parsing procedures for new non-terminals. We can add these non-terminals to the original grammar. The result for the grammar of Figure 1 is shown in Figure 4. Here we replaced the expression for entry A by (1), and added the new non-terminals appearing in it, and in their expressions. The other left-recursive non-terminals are no longer used, so they are omitted. The non-recursive non-terminals, here represented by Z, are left unchanged. This

```
Z   → x A z               $B  → #B
A   → a $A | b $B         $B1 → b #B1
$A  → #A                  $B2 → b #B2
$A1 → a #A1               #B  → $A1 | $B2
#A  → $B1 | $ε            #B1 → $B
#A1 → $A                  #B2 → $B
```

**Figure 4:** Dual grammar for grammar of Figure 1

```
E   → F $E                F   → a $F
$E  → #E                  $F  → #F
$E1 → + F #E1             $F1 → * a #F1
#E  → $E1 | $ε            #F  → $F1 | $ε
#E1 → $E                  #F1 → $F
```

**Figure 5:** Dual grammar for grammar of Figure 2

is the "dual grammar" of the grammar in Figure 1. Figure 5 shows the dual grammar obtained in a similar way for the grammar of Figure 2.

In general, the dual grammar is obtained as follows:

- For each entry $E$ replace $\mathcal{E}(E)$ by $S_1\,\$X_1|\dots|S_n\,\$X_n$
  where $S_1,\dots,S_n$ are all seeds of $\mathbb{C}(E)$, and $X_i$ is the exit containing $S_i$.
- Replace each recursive $R \to e_1e_2\dots e_n$ by $\$R \to e_2\dots e_n\ \#R$.
- Replace each recursive $R \to e_1|\dots|e_n$ by $\$R \to \#R$.
- For each $R \in \mathbb{R}$ create:
  - $\#R \to \$P_1|\dots|\$P_n$ if $R \neq E$,
  - $\#R \to \$P_1|\dots|\$P_n|\$\varepsilon$ if $R = E$,

  where $P_1,\dots,P_n$ are all members of $\mathbb{C}(R)$ such that $P_i \xrightarrow{\text{first}} R$, and $E$ is the entry of $\mathbb{C}$.

The dual grammar is an n-tuple $D = (\mathbb{N}_d, \Sigma, \mathbb{E}_d, \mathcal{E}_d, N_s)$. Its set $\mathbb{N}_d$ consists of:

- The non-recursive members of $\mathbb{N}$;
- The entries to recursion classes;
- The \$-expressions;
- The #-expressions.

In the following, the set of all non-recursive members of $\mathbb{N}$ is denoted by $\overline{\mathbb{R}}$, and the set of all entries by $\mathbb{R}_{\mathbb{E}}$. They appear as non-terminals in both $G$ and $D$. The set $\overline{\mathbb{R}} \cup \mathbb{R}_{\mathbb{E}}$ of these common symbols is denoted by $\mathbb{N}_c$. Functions $\mathcal{E}$ and $\mathcal{E}_d$ assign the same expressions to members of $\overline{\mathbb{R}}$.

The two important facts about the dual grammar are:

**Proposition 1.** *The dual grammar is left-recursive only if the original grammar contains a cycle, that is, a non-terminal that derives itself.*

**Proof** is found in the Appendix.

**Proposition 2.** *$\mathcal{L}_D(N) = \mathcal{L}(N)$ for all $N \in \mathbb{N}_c$.*

**Proof** is found in the Appendix.

The start symbol $N_s$ is either non-recursive or an entry, so it appears in both grammars, and thus both grammars define the same language. It means that recursive-descent parser for the dual grammar is a correct parser for the original grammar, and its "semantic actions" build syntax tree for the original grammar.

## 5. Implementing the choices

The construction of dual grammar introduces a number of choice expressions. We assume that they are treated in the same way as those originally present in the grammar. If dual grammar has the LL($k$) property, the choice can be made by looking at the next $k$ input terminals. The dual grammar of Figure 4 is LL(1), so decisions in steps 1, 5, 10, and 14 of the example could well be made by looking at the next terminal.

If the dual grammar is not LL($k$), the possible option is trial-and-error with backtracking. As this may result in exponential processing time, the practical solution is limited backtracking, recently being popular as the core of Parsing Expression Grammar (PEG) [5]. (As a matter of fact, the three solutions named in the Introduction are all suggested as modifications to PEG.)

The problem with limited backtracking is that it may fail to accept some legitimate input strings. For some grammars, which may be called "PEG-complete", limited backtracking *does* accept all legitimate strings. Thus, if the dual grammar is PEG-complete, it provides a correct parser for the original grammar.

There exist a sufficient condition that may be used to check if the dual grammar is PEG-complete [6].

The checks for LL($k$) and PEG-completeness are carried on dual grammar. It is the subject of further research how they can be replaced by checks applied to the original grammar.

## 6. Towards full grammar

We made here two simplifying assumptions. First, we did not include empty string $\varepsilon$ in the grammar. Second, we assumed a unique entry to each recursion class.

The result of adding $\varepsilon$ is that some expressions may derive empty string. These expressions are referred to as *nullable*, and can be identified as such by analyzing the grammar.

Nullable $e_1$ in $N \to e_1 \dots e_n$ invalidates the whole analysis in Section 2. Nullable $e_2 \dots e_n$ in recursive $N \to e_1 e_2 \dots e_n$ invalidates the proof of Proposition 1. Except for these two cases, our approach seems to work with empty string.

The assumption of unique entry per recursion class is not true for many grammars; for example, the class of Primary in Java has four entries.

The exit alternative $\$\varepsilon$ must appear in $\#E$ for the entry $E$ that actually started the ascent. This can be solved by having a separate version of $\#E$ and $\$E$ for each entry, multiplying the number of these non-terminals by the number of entries.

A practical shortcut can exploit the fact that the ascents are nested. One can keep the stack of entries for active ascents and modify the procedure for $\#T$ to check if $T$ is the entry to current ascent.

## 7. Final remarks

The traditional way of eliminating left recursion is to rewrite the grammar so that left recursion is replaced by right recursion. It can be found, for example, in [7], page 177. The process is cumbersome and produces large results; most important, it loses the spirit of the grammar. Our rewriting is straightforward and produces correct syntax tree for the original grammar.

The methods described in [1, 2, 3] use memoization tables and special code to handle them. The procedures are repeatedly applied to the same input. Our approach is in effect just another recursive-descent parser.
Our idea of recursive ascent is in principle the same as that of [4], but this latter uses specially coded "grower" to interpret data structures derived from the grammar.

As indicated in Section 6, our method does not handle some cases of nullable expressions. Among them is the known case of "hidden" left recursion: $A \rightarrow e_1e_2\ldots e_n$ becomes left recursive when $e_1$ derives $\varepsilon$. Another unsolved problem is $E \rightarrow E{+}E\,|\,n$, which results in a right-recursive parse for $E$.

## References

[1] S. Medeiros, F. Mascarenhas, R. Ierusalimschy, Left recursion in Parsing Expression Grammars, Science of Computer Programming 96 (2014) 177–190.

[2] A. Warth, J. R. Douglass, T. D. Millstein, Packrat parsers can support left recursion, in: Proceedings of the 2008 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation, PEPM 2008, San Francisco, California, USA, January 7-8, 2008, pp. 103–110.

[3] L. Tratt, Direct left-recursive parsing expression grammars, Technical Report EIS-10-01, School of Engineering and Information Sciences, Middlesex University, 2010.

[4] O. Hill, Support for Left-Recursive PEGs, 2010.
`https://github.com/orlandohill/peg-left-recursion`.

[5] B. Ford, Parsing expression grammars: A recognition-based syntactic foundation, in: N. D. Jones, X. Leroy (Eds.), Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, ACM, Venice, Italy, 2004, pp. 111–122.

[6] R. R. Redziejowski, More about converting BNF to PEG, Fundamenta Informaticae 133 (2014) 177–191.

[7] A. V. Aho, R. Sethi, J. D. Ullman, Compilers, Principles, Techniques, and Tools, Addison-Wesley, 1987.

## A. Proof of Proposition 1

For $N \in \mathbb{N}_d$ and $e \in \mathbb{N}_d \cup \Sigma$, define $N \xrightarrow{\text{firstD}} e$ to mean that parsing procedure $N$ may call parsing procedure $e$ on the same input:

(a) For $N \in \overline{\mathbb{R}}$, $\xrightarrow{\text{firstD}}$ is the same as $\xrightarrow{\text{first}}$.

(b) For $N \in \mathbb{R}_{\mathbb{E}}$, $N \xrightarrow{\text{firstD}} S$ for each seed $S$ of $\mathbb{C}(N)$.

(c) For $\$R \rightarrow e_2 \dots e_n\#R$, $\$R \xrightarrow{\text{firstD}} e_2$.

(d) For $\$R \rightarrow \#R$, $\$R \xrightarrow{\text{firstD}} \#R$.

(e) $\#R \xrightarrow{\text{firstD}} \$P_i$ for each $P_i \in \mathbb{C}(R)$ such that $P_i \xrightarrow{\text{first}} R$.

Grammar $D$ is left-recursive if $N \xrightarrow{\text{FirstD}} N$ for some $N \in \mathbb{N}_d$, where $\xrightarrow{\text{FirstD}}$ is the transitive closure of $\xrightarrow{\text{firstD}}$.

Suppose $D$ is left-recursive, that is, exist $N_1, N_2, \dots, N_k \in \mathbb{N}_d$ such that $N_1 \xrightarrow{\text{FirstD}} N_2 \xrightarrow{\text{FirstD}} \dots \xrightarrow{\text{FirstD}} N_k$ and $N_k = N_1$. We start by showing that none of them can be in $\mathbb{N}_c$.

Assume, by contradiction, that $N_1 \in \mathbb{N}_c$. We show that in this case $N_2 \in \mathbb{N}_c$ and $N_1 \xrightarrow{\text{First}} N_2$.

(Case 1) $N_1 \in \overline{\mathbb{R}}$. Because $\mathcal{E}_d(N_1) = \mathcal{E}(N_1)$, $N_2$ is in $\mathbb{N}$, and thus in $\mathbb{N}_c$. We have $N_1 \xrightarrow{\text{first}} N_2$.

(Case 2) $N_1 \in \mathbb{R}_{\mathbb{E}}$. According to (b), $N_2$ is a seed of $\mathbb{C}(N_1)$, which is either non-recursive or an entry, and thus is in $\mathbb{N}_c$. For a seed $N_2$ of $\mathbb{C}(N_1)$ holds $N_1 \xrightarrow{\text{First}} N_2$.

(Conclusion) The above can be repeated with $N_2, \dots, N_{k-1}$ to show that if any of $N_i$ is in $\mathbb{N}_c$, then for all $i$, $1 \leq i \leq n$ $N_i \in \mathbb{N}_c$ and $N_i \xrightarrow{\text{First}} N_i$. Thus, all $N_i$ belong to $\mathbb{R}$ and are all in the same recursion class. As none of $N_i$ belongs to $\overline{\mathbb{R}}$, they must all be in $\mathbb{R}_{\mathbb{E}}$. Then, in particular, $N_1 \in \mathbb{R}_{\mathbb{E}}$. But, according to (b), $N_2$ is a seed of $\mathbb{C}(N_1)$, that cannot be a member of $\mathbb{C}(N_1)$. Thus, $N_1 \notin \mathbb{N}_c$.

It follows that each $N_i$ is $\$R_i$ or $\#R_i$ for some $R_i \in \mathbb{R}$. If $R_i \rightarrow e_2 \dots e_n$, we have from (c) $\$R_i \xrightarrow{\text{firstD}} e_2$, and $e_2 \in \mathbb{N}_c$. That means all $R_i$ are choice expressions. Thus, the sequence of $N_i$ is a repetition of $\$R_i, \#R_{i+1}, \$R_{i+2}$ where, according to (d), $R_{i+1} = R_i$, and according to (e), $R_{i+2} \rightarrow \dots |R_{i+1}| \dots$ . That means $R_{i+2}$ derives $R_i$, and in $k/2$ steps derives itself. $\square$

## B. Proof of Proposition 2

The proof is in terms of syntax trees. We write $e \lhd x$ for syntax tree of $x$ with root $e$. We write $e \lhd [e_1 \lhd x_1 + \dots + e_n \lhd x_n]x$ to represent syntax tree with root $e$ and children $e_1 \lhd x_1, \dots, e_n \lhd x_n$ where $x_1 \dots x_n = x$.

To distinguish between the trees according to grammar $G$ ($G$-trees) and those according to grammar $D$ ($D$-trees) we use the symbols $\lhd_G$ respectively $\lhd_D$.

Assuming that a terminal derives itself, we show that every string $x$ derived from $e \in \mathbb{N}_c \cup \Sigma$ according to $G$ can be also derived from $e$ according to $D$ and vice-versa. The proof is by induction on height of syntax trees.

(Induction base) Syntax tree of height 0. This is the syntax tree of a terminal. The terminals and their syntax trees are identical in both grammars.

(Induction step) Assume that the stated property holds for all strings $w$ having syntax tree of height $h \geq 0$ or less. Lemmas 1 and 2 below show that it holds then for syntax trees of height $h + 1$.

**Lemma 1.** *Assume that for each* $e \lhd_G w$ *of height* $h \geq 0$ *with* $e \in \mathbb{N}_c \cup \Sigma$ *exists* $e \lhd_D w$*. Then the same holds for each* $e \lhd_G w$ *of height* $h + 1$*.*

**Proof**. Take any $N \lhd_G w$ of height $h + 1$ where $N \in \mathbb{N}_c$.

(Case 1) $N \in \overline{\mathbb{R}}$. It means $N \to e_1 \dots e_n$ or $N \to e_1 | \dots | e_n$.
We have $N \lhd_G [e_1 \lhd_G w_1 + \dots + e_n \lhd_G w_n]w$ respectively $N \lhd_G [e_j \lhd_G w]w$. Each of the subtrees $e_i \lhd_G w_i$ has height $h$ or less and each $e_i$ is in $\mathbb{N}_c \cup \Sigma$. By assumption, exists $e_i \lhd_D w_i$ for each $i$. The required $D$-tree is $N \lhd_D [e_1 \lhd_D w_1 + \dots + e_n \lhd_D w_n]w$ respectively $N \lhd_D [e_j \lhd_D w]w$.

(Case 2) $N \in \mathbb{R}_\mathbb{E}$. The tree $N \lhd_G w$ has the leftmost path $R_n \xrightarrow{\text{first}} \dots \xrightarrow{\text{first}} R_1 \xrightarrow{\text{first}} R_0$ where $R_n = N$, $R_i \in \mathbb{C}(N)$ for $n \geq i > 0$, and $R_0$ is a seed of $\mathbb{C}(N)$. Define $x_i$ to be the string derived by $G$ from $R_i$, for $0 \leq i \leq n$. We have thus $w = x_n$.

Define $w_0$ be the string derived by $G$ from $R_0$, so the subtree for $R_0$ is $R_0 \lhd_G w_0$.
Let $1 \leq i \leq n$. If $R_i \to e_1 e_2 \dots e_m$, the subtree for $R_i$ is $R_i \lhd_G [R_{i-1} \lhd_G x_{i-1} + e_2 \lhd_G v_2 + \dots + e_m \lhd_G v_m]x_i$. Define $w_i = v_2 \dots v_m$, so $x_i = x_{i-1} w_i$.
if $R_i \to e_1 | \dots | e_m$, the subtree for $R_i$ is $R_i \lhd_G [R_{i-1} \lhd_G x_{i-1}]x_i$ Define $w_i = \varepsilon$, so we have again $x_i = x_{i-1} w_i$.
One can easily see that $w = x_n = w_0 \dots w_n$.

Define $y_i$ to be the string derived by $D$ from $\$R_i$, and $z_i$ to be the string derived by $D$ from $\#R_i$, for $0 \leq i \leq n$.

For entry $R_n$ exists the tree $\#R_n \lhd_D \varepsilon$. If $\$R_n \to \#R_n$, exists the tree $\$R_n \lhd_D [\#R_n \lhd_D \varepsilon]\varepsilon = w_n$. If $\$R_n \to e_2 \dots e_n \#R_n$, exists by assumption D-tree for each of $e_j$, deriving, respectively, $v_j$. Thus, exists the tree $\$R_n \lhd_D [e_2 \lhd_D v_2 + dots + e_m \lhd_D v_m + \#R_n \lhd_D \varepsilon]v_2 \dots v_m \varepsilon = w_n$. Define $y_n = w_n$.

Suppose that exists the tree $\$R_i \lhd_D y_i$. Then exists the tree $\#R_{i-1}[\lhd_D \$R_i \lhd_D y_i]y_i$. By construction similar to the above, w find the D-tree for $\$R_{i-1}$ deriving $y_{i-1} = w_{i-1} y_i$. At the end, we find the D-tree for $\$R_1$ deriving $y_1 = w_1 y_2$.
By assumption there exists D-tree for the seed $R_0$ deriving $w_0$. For entry $R_n$ we have $R_n \to R_0 \$R_1$, which gives the D-tree $R_n \lhd_D [R_0 \lhd_D w_0 + \$R_i \lhd_D y_1]w_0 y_1$. One can easily see that $y_1 = w_1 \dots w_n$, so we have a D-tree for $N = R_n$ deriving $w = w_0 \dots w_n$. □

**Lemma 2.** *Assume that for each* $e \lhd_D w$ *of height* $h \geq 0$ *with* $e \in \mathbb{N}_c \cup \Sigma$ *exists* $e \lhd_G w$*. Then the same holds for each* $e \lhd_D w$ *of height* $h + 1$*.*

**Proof**. Take any $N \lhd_D w$ of height $h + 1$ where $N \in \mathbb{N}_c$.

(Case 1) $N \in \overline{\mathbb{R}}$. It means $N \to e_1 \dots e_n$ or $N \to e_1 | \dots | e_n$. We have $N \lhd_D [e_1 \lhd_D w_1 + \dots + e_n \lhd_D w_n]w$ respectively $N \lhd_D [e_j \lhd_D w]w$. Each of the subtrees $e_i \lhd_D w_i$ has height $h$ or less and each $e_i$ is in $\mathbb{N}_c \cup \Sigma$. By assumption, exists $e_i \lhd_G w_i$ for each $i$. The required $G$-tree is $N \lhd_G [e_1 \lhd_G w_1 + \cdots + e_n \lhd_G w_n] w$ respectively $N \lhd_G [e_j \lhd_G w] w$.

(Case 2) $N \in \mathbb{R}_{\mathbb{E}}$. The D-tree of $N$ has root $N$, node $S$ for seed of $\mathbb{C}(N)$,and nodes $\$R_i$, $\#R_i$ for $1 \leq i \leq n$. Denote $R_n = N$ and $R_0 = S$. Denote $y_i$ the string derived by $\$R_i$ and $z_i$ that derived by $\#R_i$. We have $z_i = y_{i+1}$ for $i < n$ and $z_n = \varepsilon$. Denote by $w_0$ the string derived by $R_0$ and by $w$ one derived by $N = R_n$.
The D-tree for $R_n$ is $R_n \lhd_D [R_0 \lhd_D w_0 + \$R_1 \lhd_D y_1] w_0 y_1 = w$.
If $\$R_i \to e_2 \ldots e_n \# R_i$, we have $y_i = v_2 \ldots v_m z_i$. where $v_j$ is the string derived by $e_j$. Defining $w_i = v_2 \ldots v_m$ by $w_i$, we have $y_i = w_i z_i$.
If $\$R_i \# R_i$, we have $y_i = z_i$. Defining $w_i = \varepsilon$, we have again $y_i = w_i z_i$.
We have $z_i = y_(i+1)$ for $i < n$ and $z_n = \varepsilon$, so $y_i = w_i y_(i+1)$ for $i < n$ and $y_n = w_n$. One can easily see that $w = w_0 \ldots w_n$.

By assumption exists G-tree $R_0 \lhd_G w_0$. Suppose we have the G-tree for $R_{i-1}$ where $1 \leq i \leq n$ that derives $x_{i-1}$.
Suppose $R_i \to R_(i-1) e_2 \ldots e_m$. By assumption exist G-trees that derive $v_2 \ldots v_m$ from $e_2 \ldots e_m$, so exists G-tree for $R_i$ deriving $x_i = x_(i-1) v_2 \ldots v_m = x_(i-1) w_i$. Suppose $R_i \to R_(i-1)$. Then exists G-tree deriving $x_i = x_(i-1)\varepsilon = x_(i-1) w_i$. One can easily see that the string $x_n$ derived by $R_n$ is $w_0 \ldots w_n = w$ $\square$

# Process Opacity and Insertion Functions

Damas P. Gruska[1], M. Carmen Ruiz[2]

[1]Comenius University, Slovak Republic
[2]Universidad de Castilla-La Mancha, Spain

**Abstract**
Time insertion functions as a way how to guarantee state-based security with respect to timing attacks are proposed and studied. As regards the security property, we work with the property called process opacity. First, we define timing attacks and later we show how security with respect to them can be enforced by such functions. The time insertion function can alter the time behaviour of the original system by inserting some time delays to guarantee its security. We investigate conditions under which such functions do exist and also some of their properties.

**Keywords**
state-based security, process opacity, process algebras, information flow, insertion function, timing attack

## 1. Introduction

Formal methods allows us, in many cases, to show, even prove, that a given system is not secure. Then we have a couple of options what to do. We can either re-design its behavior, what might be costly, difficult or even impossible, in the case that it is already part of a hardware solution, proprietary firmware and so on, or we can use supervisory control (see [1]) to restrict system's behaviour in such a way that the system becomes secure. A supervisor can see (some) system's actions and can control (disable or enable) some set of system's action. In this way it restricts system's behaviour to guarantee its security (see also [2]). This is a trade-off between security and functionality. Situation is different in the case of timing attacks. They, as side channel attacks, represent serious threat for many systems. They allow intruders "break" "unbreakable" systems, algorithms, protocols, etc. For example, by carefully measuring the amount of time required to perform private key operations, attackers may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems (see [3]). This idea was developed in [4] were a timing attack against smart card implementation of RSA was conducted. In [5], a timing attack on the RC5 block encryption algorithm is described. The analysis is motivated by the possibility that some implementations of RC5 could result in data-dependent rotations taking a time that is a function of the data. In [6], the vulnerability of two implementations of the Data Encryption Standard (DES) cryptosystem under a timing attack is studied. It is showed that a timing attack yields the Hamming weight of the key used by both DES implementations. Moreover, the attack is computationally inexpensive. A timing attack against an implementation of AES candidate Rijndael is described in [7], and the one against the popular SSH protocol in

*29th International Workshop on Concurrency, Specification and Programming (CS&P'21)*
damas.gruska@fmph.uniba.sk (D. P. Gruska); MCarmen.Ruiz@uclm.es (M. C. Ruiz)

[8]. Even relatively recently discovered possible attacks on most of currently used processors (Meltdown and Spectre) also belong to timing attacks. To protect systems against timing attacks we propose application of inserting functions (see [9, 10, 11, 12]). Such functions can add some idling between actions to enforce process's security. In this paper we investigate conditions under which such functions do exist and also their properties.

As regards formalism, we will work with a timed process algebra and opacity, which is the security property based on an absence of information flow. This formalism enables us to formalize timing attacks. In [13] we have introduced time insertion functions to guarantee language-based security and showed some of their properties. In [14] we studied conditions under which there exists a timed insertion function for a given process and security language-based security property and we have presented some decidability and undecidability results. In this paper we define and study time insertion functions for state-based security property process opacity.

The paper is organized as follows. In Section 2 we describe the timed process algebra TPA which will be used as a basic formalism and information flow state-based security process opacity. The next section is devoted to time insertion functions as a way how to guarantee state this security property with respect to timing attacks.

## 2. Working Formalism

In this section we briefly recall Timed Process Algebra, TPA for short (for more details see [15]). TPA is based on Milner's Calculus of Communicating Systems (for short, CCS, see [16]) but the special time action $t$ which expresses elapsing of (discrete) time is added and hence the set of actions is extended from $Act$ to $Actt$. The presented language is a slight simplification of Timed Security Process Algebra (tSPA) introduced in [17]. We omit an explicit idling operator $\iota$ used in tSPA and instead of this we allow implicit idling of processes. Hence processes can perform either "enforced idling" by performing $t$ actions which are explicitly expressed in their descriptions or "voluntary idling" (i.e. for example, the process $a.Nil$ can perform $t$ action despite the fact that this action is not formally expressed in the process specification). But in both cases internal communications have priority to action $t$ in the parallel composition. Moreover we do not divide actions into private and public ones as it is in tSPA. TPA differs also from the tCryptoSPA (see [18]). TPA does not use value passing and strictly preserves *time determinancy* in case of choice operator $+$ what is not the case of tCryptoSPA (see [15]).

To define the language TPA, we first assume a set of atomic action symbols $A$ not containing symbols $\tau$ and $t$, and such that for every $a \in A$ there exists $\overline{a} \in A$ and $\overline{\overline{a}} = a$. We define $Act = A \cup \{\tau\}, At = A \cup \{t\}, Actt = Act \cup \{t\}$. We assume that $a, b, \ldots$ range over $A$, $u, v, \ldots$ range over $Act$, and $x, y \ldots$ range over $Actt$.

We give a structural operational semantics of terms by means of labeled transition systems. The set of terms represents a set of states, labels are actions from $Actt$. The transition relation $\rightarrow$ is a subset of TPA $\times$ $Actt$ $\times$ TPA. We write $P \stackrel{x}{\rightarrow} P'$ instead of $(P, x, P') \in \rightarrow$ and $P \stackrel{x}{\not\rightarrow}$ if there is no $P'$ such that $P \stackrel{x}{\rightarrow} P'$. The meaning of the expression $P \stackrel{x}{\rightarrow} P'$ is that the term $P$ can evolve to $P'$ by performing action $x$, by $P \stackrel{x}{\rightarrow}$ we will denote that there exists a term $P'$ such that $P \stackrel{x}{\rightarrow} P'$. We define the transition relation as the least relation satisfying the inference rules for CCS plus the following inference rules:

$$\frac{}{Nil \xrightarrow{t} Nil} \; A1 \qquad \frac{}{u.P \xrightarrow{t} u.P} \; A2$$

$$\frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q', P \mid Q \not\xrightarrow{\tau}}{P \mid Q \xrightarrow{t} P' \mid Q'} \; Pa \qquad \frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q'}{P + Q \xrightarrow{t} P' + Q'} \; S$$

For $s = x_1.x_2.\dots.x_n, x_i \in Act$ we write $P \xrightarrow{s}$ instead of $P \xrightarrow{x_1}\xrightarrow{x_2} \dots \xrightarrow{x_n}$ and we say that $s$ is a trace of $P$. The set of all traces of $P$ will be denoted by $Tr(P)$. By $\epsilon$ we denote the empty sequence and by $M^*$ we denote the set of sequences of elements from $M$. We use $\overset{x}{\Rightarrow}$ for transitions including $\tau$ actions (see [16]). By $s|_B$ we will denote the sequence obtained from $s$ by removing all actions not belonging to $B$. By $L(P)$ we will denote a set of actions which can be performed by $P$, i.e. $L(P) = \{x | P \xrightarrow{s.x}, s \in Actt^*\}$.

We define two behavior equivalences trace equivalence and weak bisimulation, respectively (see [16]).

**Definition 1.** *The set of weak traces of process $P$ is defined as $Tr_w(P) = \{s \in At^\star | \exists P'.P \overset{s}{\Rightarrow} P'\}$.*

*Two processes $P$ and $Q$ are weakly trace equivalent iff $Tr_w(P) = Tr_w(Q)$.*

**Definition 2.** *Let $(TPA, Act, \rightarrow)$ be a labelled transition system (LTS). A relation $\Re \subseteq TPA \times TPA$ is called a* weak bisimulation *if it is symmetric and it satisfies the following condition: if $(P, Q) \in \Re$ and $P \xrightarrow{x} P', x \in Actt$ then there exists a process $Q'$ such that $Q \overset{\widehat{x}}{\Rightarrow} Q'$ and $(P', Q') \in \Re$. Two processes $P, Q$ are weakly bisimilar, abbreviated $P \approx Q$, if there exists a weak bisimulation relating $P$ and $Q$.*

To formalize an information flow we do not divide actions into public and private ones at the system description level, as it is done for example in [18], but we use a more general concept of observation and opacity. This concept was exploited in [19] and [20] in a framework of Petri Nets and transition systems, respectively. Firstly we define observation function on sequences from $Act^\star$. Various variants of observation functions differs according to contexts which they take into account. For example, an observation of an action can depend on the previous actions.

**Definition 3 (Observation).** *Let $\Theta$ be a set of elements called observables. Any function $\mathcal{O} : Actt^\star \rightarrow \Theta^\star$ is an observation function. It is called static /dynamic /orwellian / m-orwellian $(m \geq 1)$ if the following conditions hold respectively (below we assume $w = x_1 \dots x_n$):*

- *static if there is a mapping $\mathcal{O}' : Actt \rightarrow \Theta \cup \{\epsilon\}$ such that for every $w \in Act^\star$ it holds $\mathcal{O}(w) = \mathcal{O}'(x_1) \dots \mathcal{O}'(x_n)$,*
- *dynamic if there is a mapping $\mathcal{O}' : Actt^\star \rightarrow \Theta \cup \{\epsilon\}$ such that for every $w \in Actt^\star$ it holds $\mathcal{O}(w) = \mathcal{O}'(x_1).\mathcal{O}'(x_1.x_2) \dots \mathcal{O}'(x_1 \dots x_n)$,*
- *orwellian if there is a mapping $\mathcal{O}' : Actt \times Actt^\star \rightarrow \Theta \cup \{\epsilon\}$ such that for every $w \in Actt^\star$ it holds $\mathcal{O}(w) = \mathcal{O}'(x_1, w).\mathcal{O}'(x_2, w) \dots \mathcal{O}'(x_n, w)$,*

- *m-orwellian if there is a mapping* $\mathcal{O}' : Actt \times Actt^\star \rightarrow \Theta \cup \{\epsilon\}$ *such that for every* $w \in Actt^\star$ *it holds* $\mathcal{O}(w) = \mathcal{O}'(x_1, w_1).\mathcal{O}'(x_2, w_2) \dots \mathcal{O}'(x_n, w_n)$ *where* $w_i = x_{max\{1,i-m+1\}}.x_{max\{1,i-m+1\}+1} \dots x_{min\{n,i+m-1\}}$.

In the case of the static observation function each action is observed independently from its context. In the case of the dynamic observation function an observation of an action depends on the previous ones, in the case of the orwellian and m-orwellian observation function an observation of an action depends on the all and on $m$ previous actions in the sequence, respectively. The static observation function is the special case of m-orwellian one for $m = 1$. Note that from the practical point of view the m-orwellian observation functions are the most interesting ones. An observation expresses what an observer - eavesdropper can see from a system behavior and we will alternatively use both the terms (observation - observer) with the same meaning. Note that the same action can be seen differently during an observation (except static observation function) and this express a possibility to accumulate some knowledge by intruder. For example, an action not visible at the beginning could become somehow observable. From now on we will assume that $\Theta \subseteq Actt$.

Now let us assume hat an intruder is interested whether a given process has reached a state with some given property which is expressed by a (total) predicate. This property might be process deadlock, capability to execute only traces with some given actions, capability to perform at the same actions form a given set, incapacity to idle (to perform $\tau$ action ) etc. We do not put any restriction on such predicates but we only assume that they are consistent with some suitable behavioral equivalence. The formal definition follows.

**Definition 4.** *We say that the predicate* $\phi$ *over processes is consistent with respect to relation* $\cong$ *if whenever* $P \cong P'$ *then* $\phi(P) \Leftrightarrow \phi(P')$.

As the consistency relation $\cong$ we could take bisimulation, weak bisimulation, weak trace equivalence or any other suitable equivalence.

An intruder cannot learn validity of predicate $\phi$ observing process's behaviour iff there are two traces, undistinguished for him (by observation function $\mathcal{O}$), where one leads to a state which satisfy $\phi$ and another one leads to a state for which $\neg\phi$ holds. The formal definition follows.

**Definition 5 (Process Opacity).** *Given process* $P$*, a predicate* $\phi$ *over processes is process opaque w.r.t. the observation function* $\mathcal{O}$ *whenever* $P \xrightarrow{w} P'$ *for* $w \in Actt^*$ *and* $\phi(P')$ *holds then there exists* $P''$ *such that* $P \xrightarrow{w'} P''$ *for some* $w' \in Actt^*$ *and* $\neg\phi(P'')$ *holds and moreover* $\mathcal{O}(w) = \mathcal{O}(w')$*. The set of processes for which the predicate* $\phi$ *is process opaque w.r.t. to the* $\mathcal{O}$ *will be denoted by* $POp^{\phi}_{\mathcal{O}}$.

## 3. Insertion functions

Timing attacks belong to powerful tools for attackers who can observe or interfere with systems in real time. On the other side these techniques is useless for off-line systems and hence they could be consider safe with respect to attackers who cannot observe (real) time behaviour. By the presented formalism we have a way how to distinguish these two cases. First we define untimed version of an observation function, i.e. a function which does not take into account time information. From now on we will consider observation functions $\mathcal{O} : Actt^\star \rightarrow Actt^\star$ for which there exists untimed variants $\mathcal{O}_t$ . Function $\mathcal{O}_t$ is untimed variant of $\mathcal{O}$ iff $\mathcal{O}(w) = \mathcal{O}_t(w|_{Act})$, i.e. untimed variant represents an observer who does not see elapsing of time since both traces, with and without actions $t$, are seen equally. Now we can formally describe situation when a process could be jeopardized by a timing attack i.e. is secure only if an observer cannot see elapsing of time.

**Definition 6 (Timinig Attacks).** *We say that process $P$ is prone to timing attacks with respect to $\phi$ and $\mathcal{O}$ iff $P \notin POp^{\phi}_{\mathcal{O}}$ but $P \in POp^{\phi}_{\mathcal{O}_t}$.*

**Example 1.** *Let us assume an intruder who tries to learn whether a private action $h$ was performed by a given process while (s)he can observe only public action $l$ but not $h$ itself. Then process $P = a.t.R + b.t.t.Q$ is not opaque for static observation function $\mathcal{O}(x) = \epsilon$ for $x \neq t$, $\mathcal{O}(t) = t$ and $\phi(R), \neg\phi(Q)$ hold, i.e. $P \notin POp^{\phi}_{\mathcal{O}}$. But if an observer cannot see elapsing of time this process is opaque, i.e. $P \in POp^{\phi}_{\mathcal{O}_t}$.*

From now on we will consider only processes which are prone to timing attacks (see Definition 6) and moreover we will assume that predicate $\phi$ is decidable. There are basically three ways how to solve vulnerability to timing attacks except putting a system off-line. First, redesign the system, put some monitor or supervisor which prevents dangerous behavior which could leak some classified information (see, for example, [2]) or hide this leakage by inserting some time delays between system's action (see [12, 10] for general insertion functions for non-deterministic finite automata). Now we will define and study this possibility. First we need some notation. For $w, w' \in Actt^*$ and $w = x_1.x_2 \ldots x_n$ we will write $w \ll_S w'$ for $S \subset Actt$ iff $w' = x_1.i_1.x_2 \ldots x_n.i_n$ where $i_k \in S^*$ for every $k, 1 \leq k \leq n$. In general, an insertion function inserts additional actions between original process's actions (given by trace $w$) such that for the resulting trace $w'$ we have $w \ll_S w'$ and $w'$ is still a possible trace of the process. In our case we would consider insertion functions (called time insertion functions) which insert only time actions i.e. such functions that $w \ll_{\{t\}} w'$. Results of an insertion function depends on previous process behaviour. We can define this dependency similarly as it is defined for observation functions.

**Definition 7 (Time Insertion function).** *Any function $\mathcal{F} : Actt^\star \rightarrow Actt^\star$ is an insertion function iff for every $w \in Actt^*$ we have $w \ll_{\{t\}} \mathcal{F}(w)$. It is called static /dynamic /orwellian / m-orwellian $(m \geq 1)$ if the following conditions hold respectively (below we assume $w = x_1 \ldots x_n$):*

- *static if there is a mapping $f : Actt \rightarrow \{t\}^*$ such that for every $w \in Actt^\star$ it holds $\mathcal{F}(w) = x_1.f(x_1).x_2.f(x_2) \ldots x_n.f(x_n)$,*
- *dynamic if there is a mapping $f : Actt^\star \rightarrow \{t\}^*$ such that for every $w \in Act^\star$ it holds $\mathcal{F}(w) = x_1.f(x_1).x_2.f(x_1.x_2) \ldots x_n.f(x_1. \ldots .x_n)$,*
- *orwellian if there is a mapping $f' : Actt \times Actt^\star \rightarrow \{t\}^*$ such that for every $w \in Actt^\star$ it holds $\mathcal{F}(w) = x_1.f(x_1, w).x_2.f(x_2, w) \ldots x_n.f(x_n, w)$,*

- *m-orwellian if there is a mapping $f' : Actt \times Actt^\star \to \{t\}^*$ such that for every $w \in Actt^\star$ it holds $\mathcal{F}(w) = x_1.f(x_1, w_1).x_2.f(x_2, w_2) \dots x_n.f(x_n, w_n)$, $w_i = x_{max\{1,i-m+1\}}.x_{max\{1,i-m+1\}+1} \dots x_{min\{n,i+m-1\}}$.*

Note that contrary to general insertion function (see [12, 10]) inserting time actions is much simpler due to transition rules $A1, A2, S$. The purpose of time insertion function is to guaranty security with respect to process opacity. Let $P \notin POp^{\phi}_{\mathcal{O}}$ but $P \in POp^{\phi}_{\mathcal{O}_t}$ , i.e. the process $P$ is prone to timing attack with respect to $\mathcal{O}$ and $\phi$. If $\mathcal{O}$ and $\phi$ is clear from a context we will omit it. Now we define what it means that process can be immunized by a time insertion function.

**Definition 8.** *We say that process $P$ can be immunized for process opacity with respect to a predicate $\phi$ over $Actt^\star$ and the observation function $\mathcal{O}$ if for every $P'$, $P \overset{w}{\to} P'$ such that $\phi(P')$ holds and there does not exists $P''$ such that $P \overset{w'}{\to} P''$ for some $w'$ such that $\mathcal{O}(w) = \mathcal{O}(w')$ and $\phi(P'')$ does not hold, there exist $w_t$, $w \ll_{\{t\}} w_t$ such that $P \overset{w_t}{\to} P''$ and and there exists $P'''$ and $w''$, such that $P \overset{w''}{\to} P'''$ such that $\neg\phi(P''')$ holds and $\mathcal{O}(w_t) = \mathcal{O}(w'')$.*

In Fig. 1 process immunization is depicted.

$$\begin{array}{cccc}
P & \overset{w}{\Longrightarrow} & \phi(P') & \mathcal{O}(w) \\
 & & & \| \\
P & \overset{w'}{\not\Longrightarrow} & \neg\phi(P'') & \mathcal{O}(w') \\
 & & & \\
P & \overset{w_t}{\Longrightarrow} & \phi(P') & \mathcal{O}(w_t) \\
 & & & \| \\
P & \overset{w''}{\Longrightarrow} & \neg\phi(P''') & \mathcal{O}(w'')
\end{array}$$

**Figure 1:** Process immunization

Now we will study an existence of time insertion functions. First we need some notations. We begin with observational functions which do not see $\tau$ actions.

**Definition 9.** *We say that observational function $\mathcal{O}$ is not sensitive to $\tau$ action iff $\mathcal{O}(w) = \mathcal{O}(w|_{At})$ for every $w \in Act^*$. Otherwise we say that $\mathcal{O}$ is sensitive to $\tau$ action.*

**Example 2.** *Process $P$, $P = t.R + (a.Q|\bar{a}.Nil) \setminus \{a\}$, cannot be immunized if $\mathcal{O}$ is sensitive to $\tau$ action and $\phi(R)$, $\neg\phi(Q)$ hold. An immunization should put a time delay into the trace performed by the right part of the process $P$ but this subprocess cannot perform $t$ action due to the inference rule $Pa$ before communication by means of channel $a$.*

**Lemma 1.** *Let $P$ is prone to timing attack with respect to $\mathcal{O}$ and $\phi$. Let $\tau \notin L(P)$, $P$ is sequential (i.e. does not contain parallel composition) and $\mathcal{O}$ is static. Then $P$ can be immunized.*

Another problem, which causes that processes cannot be immunized, is related to observation of time, namely, if this observation is context sensitive, as it is stated by the following example.

**Example 3.** *Process $P, P = h.R.Nil + t.Q.Nil$ cannot be immunized for dynamic $\mathcal{O}$ such that $\mathcal{O}(w.h.t^*.w') = \mathcal{O}(w.w')$, $\mathcal{O}(w.l.w') = \mathcal{O}(w).l.\mathcal{O}(w')$, if $w$ does not end with action $h$ we have $\mathcal{O}(w.t.w') = \mathcal{O}(w).t.\mathcal{O}(w')$, and $\phi(R)$, $\neg\phi(Q)$ hold.*

Now we define time contextuality formally.

**Definition 10.** *We say that observational function $\mathcal{O}$ is time non-contextual if $\mathcal{O}_t(w) = \mathcal{O}_t(w')$ for every $w, w'$ such that $w \ll_{\{t\}} w'$.*

**Proposition 1.** *Let process $P$ is prone to timing attacks with respect to $\phi$ and time non-contextual observation function $\mathcal{O}$ which does not see $\tau$. Then $P$ can be immunized for opacity with respect to timing attacks.*

**Proof 1.** *The main idea. Let $P \notin POp^{\phi}_{\mathcal{O}}$ but $P \in POp^{\phi}_{\mathcal{O}_t}$. This means that there exists a sequence $w$ and $P'$ such that $P \xrightarrow{w} P'$, $\phi(P')$ holds and there does not exist $P''$ such that $P \xrightarrow{w'} P''$ for some $w'$ such that $\mathcal{O}(w) = \mathcal{O}(w')$ and $\phi(P'')$ does not hold.*

*Suppose that $P$ cannot be immunized, i.e. for every $w_t, w \ll_{\{t\}} w_t$ if $P \xrightarrow{w_t} P'''$, $\phi(P''')$ holds, there does not exist $P''''$ such that $P \xrightarrow{w''} P''''$ for some $w''$ such that $\mathcal{O}(w_t) = \mathcal{O}(w''')$. But due to our assumption we have $\mathcal{O}_t(w_t) = \mathcal{O}_t(w)$ and hence it is with contradiction that $P$ is prone to timing attacks.*

**Corollary 1.** Let $\mathcal{O}$ is a static observation function such that $\mathcal{O}(\tau) = \epsilon$ and $\mathcal{O}(t) = t$. Then process $P$ which is prone to timing attacks with respect to $\phi$ and observation function $\mathcal{O}$ can be immunized for process opacity with respect to timing attacks.

Under some special conditions time insertion functions can be computed effectively a it is stated by the following proposition.

**Proposition 2.** *Let process $P$ is prone to timing attacks with respect to $\phi$ and time non-contextual observation function $\mathcal{O}$ which does not see $\tau$. Then $P$ can be immunized for opacity with respect to timing attacks by a m-orwellian insertion function, moreover such one, which can be emulated by finite state process.*

**Proof 2.** *Sketch. The prove follows an idea from Proposition 3 and Theorem 4.10 and Lemma 4.5.in [13], where insertion functions are modeled by processes run in parallel with $P$.*

No we define what it means that a predicate is time sensitive.

**Definition 11.** *We say that predicate $\phi$ is time sensitive iff whenever $\phi(P)$ holds for $P$ then there exists $n, n > 0$ such that $P \xrightarrow{t^n} P'$ and $\phi(P')$ does not hold.*

**Proposition 3.** *Let process $P$ is prone to timing attacks with respect to time sensitive predicate $\neg\phi$ and time non-contextual observation function $\mathcal{O}$ which does not see $\tau$. Then $P$ can be immunized for opacity with respect to timing attacks, $\mathcal{O}$ and $\phi$.*

**Proof 3.** *Sketch. By inserting some time actions after performing a sequence we can always reach a state for which $\phi$ holds and hence $P$ becomes safe with respect to $POp_{\mathcal{O}}^{\neg\phi}$.*

**Corollary 2.** Let process $P$ is prone to timing attacks with respect $\phi$ and time non-contextual observation function $\mathcal{O}$ which does not see $\tau$. Let $\neg\phi$ is not time sensitive predicate then $P$ can be immunized for opacity with respect to timing attacks and $\mathcal{O}$ and$\phi$.

In general, we cannot decide whether process can be immunized as it is stated by the following proposition. Fortunately, it is decidable for the most important static and m-orwellian observation functions.

**Proposition 4.** *Immunizability is undecidable i.e. it cannot be decided whether $P$ can be immunized for opacity with respect to timing attacks.*

**Proof 4.** *Sketch. Let $T_i$ represents i-th Turing machine under some numeration of all Turing machines. We start with generalized process from Example 3. Let $P = h.l.R + \sum_{i \in N} t^i.l.Q$. Let $\mathcal{O}(w.h.t^i.w') = \mathcal{O}(w.w')$ whenever $T_i$ halts with the empty tape and $\mathcal{O}(w.h.t^i.w') = \mathcal{O}(w).t^i.\mathcal{O}(w')$ otherwise. It is easy to check that immunization of $P$ is undecidable.*

**Proposition 5.** *Immunizability is decidable for static and m-orwellian observation function $\mathcal{O}$.*

**Proof 5.** *According to Proposition 3 it is enough to show that observation function $\mathcal{O}$ is time non-contextual observation function and it does not see $\tau$. Clearly both these properties are decidable for static and m-orwellian observation functions.*

## 4. Conclusions

We have investigated time insertion functions for timed process algebra which enforce the security with respect to timing attack formalized process opacity. Time insertion functions add some delays to system's behaviour to prevent a timing attack. We study an existence of an insertion function for a given process, given observational function and a predicate over processes. In future work we plan to investigate minimal insertion functions, i.e. such functions which add as little as possible time delays to guarantee process's security with respect to timing attacks. The presented approach allows us to exploit also process algebras enriched by operators expressing other "parameters" (space, distribution, networking architecture, power consumption and so on). Hence we could obtain security properties which have not only theoretical but also practical value. Moreover, we can use similar techniques as in [21] to minimize time, as well as other resources, added to process's behaviour. Moreover, we plan to model both observation functions as well as predicates over processes by processes themselves, to obtain some complexity results as it was done in [13] for trace based variant of opacity.

## Acknowledgement

This work was supported by the Slovak Research and Development Agency under the Contract no. APVV-19-0220 (ORBIS) and by the Slovak VEGA agency under Contract no. 1/0778/18 (KATO).

## References

[1] P. Ramadge, W. Wonham, The control of discrete event systems, Proceedings of the IEEE 77 (1989) 81–98. doi:10.1109/5.21072.

[2] D. P. Gruska, M. C. Ruiz, Opacity-enforcing for process algebras, in: B. Schlingloff, S. Akili (Eds.), Proceedings of the 27th International Workshop on Concurrency, Specification and Programming, Berlin, Germany, September 24-26, 2018, volume 2240 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2018. URL: http://ceur-ws.org/Vol-2240/paper1.pdf.

[3] P. C. Kocher, Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems, in: N. Koblitz (Ed.), Advances in Cryptology — CRYPTO '96, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 104–113.

[4] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. MestrÃ©, J.-J. Quisquater, J.-L. Willems, A practical implementation of the timing attack, volume 1820, 1998, pp. 167–182. doi:10.1007/10721064_15.

[5] H. Handschuh, H. M. Heys, A timing attack on rc5, in: Proceedings of the Selected Areas in Cryptography, SAC '98, Springer-Verlag, Berlin, Heidelberg, 1998, p. 306–318.

[6] A. Hevia, M. Kiwi, Strength of two data encryption standard implementations under timing attacks, ACM Trans. Inf. Syst. Secur. 2 (1999) 416–437. URL: https://doi.org/10.1145/330382.330390. doi:10.1145/330382.330390.

[7] F. Koeune, F. Koeune, J.-J. Quisquater, J. jacques Quisquater, A timing attack against Rijndael, Technical Report, 1999.

[8] D. X. Song, D. Wagner, X. Tian, Timing analysis of keystrokes and timing attacks on ssh, in: Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10, SSYM'01, USENIX Association, USA, 2001.

[9] R. Jacob, J.-J. Lesage, J.-M. Faure, Overview of discrete event systems opacity: Models, validation, and quantification, Annual Reviews in Control 41 (2016) 135–146. URL: https://www.sciencedirect.com/science/article/pii/S1367578816300189. doi:https://doi.org/10.1016/j.arcontrol.2016.04.015.

[10] C. Keroglou, L. Ricker, S. Lafortune, Insertion functions with memory for opacity enforcement, IFAC-PapersOnLine 51 (2018) 394–399. URL: https://www.sciencedirect.com/science/article/pii/S240589631830661X. doi:https://doi.org/10.1016/j.ifacol.2018.06.331, 14th IFAC Workshop on Discrete Event Systems WODES 2018.

[11] C. Keroglou, S. Lafortune, Embedded insertion functions for opacity enforcement, IEEE Transactions on Automatic Control 66 (2021) 4184–4191. doi:10.1109/TAC.2020.3037891.

[12] Y.-C. Wu, S. Lafortune, Enforcement of opacity properties using insertion functions, in: 2012 IEEE 51st IEEE Conference on Decision and Control (CDC), 2012, pp. 6722–6728. doi:10.1109/CDC.2012.6426760.

[13] D. P. Gruska, Security and time insertion, Proceedings of the 23rd Pan-Hellenic Conference on Informatics (2019).

[14] D. P. Gruska, Time insertion functions, in: Proceedings CSMML 2021, to appear, 2021.

[15] D. P. Gruska, Process opacity for timed process algebra, in: A. Voronkov, I. B. Virbitskaite (Eds.), Perspectives of System Informatics - 9th International Ershov Informatics Conference, PSI 2014, St. Petersburg, Russia, June 24-27, 2014. Revised Selected Papers,

volume 8974 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 151–160. URL: https://doi.org/10.1007/978-3-662-46823-4_13. doi:10.1007/978-3-662-46823-4_13.

[16] R. Milner, Communication and Concurrency, Prentice-Hall, Inc., USA, 1989.

[17] R. Focardi, R. Gorrieri, F. Martinelli, Information flow analysis in a discrete-time process algebra, in: Proceedings 13th IEEE Computer Security Foundations Workshop. CSFW-13, 2000, pp. 170–184. doi:10.1109/CSFW.2000.856935.

[18] R. Gorrieri, F. Martinelli, A simple framework for real-time cryptographic protocol analysis with compositional proof rules, Science of Computer Programming 50 (2004) 23–49. doi:10.1016/j.scico.2004.01.001.

[19] J. W. Bryans, M. Koutny, P. Y. Ryan, Modelling opacity using petri nets, Electronic Notes in Theoretical Computer Science 121 (2005) 101–115. URL: https://www.sciencedirect.com/science/article/pii/S1571066105000277. doi:https://doi.org/10.1016/j.entcs.2004.10.010, proceedings of the 2nd International Workshop on Security Issues with Petri Nets and other Computational Models (WISP 2004).

[20] J. Bryans, M. Koutny, L. Mazare, P. Ryan, Opacity generalised to transition systems, volume 7, 2008, pp. 421–435. doi:10.1007/11679219_7.

[21] Y. Ji, X. Yin, S. Lafortune, Enforcing opacity by insertion functions under multiple energy constraints, Automatica 108 (2019) 108476. URL: https://www.sciencedirect.com/science/article/pii/S0005109819303243. doi:https://doi.org/10.1016/j.automatica.2019.06.028.

# Attack Trees with Time Constraints

Aliyu Tanko Ali, Damas Gruska

*Comenius University, Mlynska Dolina 842 48, Bratislava, Slovak Republic*

**Abstract**

We propose how attack trees formalism can be extended with time constraints. An attack tree is a basic description of how an attacker can compromise an asset, we refine this basic description by adding time constraints which can prevent an attacker from reaching the root node, if the attack actions performed cannot be completed within the defined time constraint. Adding time to attack trees causes an infinite number of possible states, to overcome this problem, we translate the tree into (an extended version of) timed automata and later use UPPAAL verification tool to analyse.

**Keywords**

Attack trees, timed attack trees, cyber-physical systems, security, threat modelling, timed automata, reachability

## 1. Introduction

*Attack tree's security.* The revolution that brought the introduction of assets such as IoTs, CPS, and industrial control systems etc., in the last decades also brought many security challenges. At its inception, attack trees are used to model how an asset (mostly static) may be compromised and allow a security engineer to plan on how to address the potential security threats. For example, in its early days, attack trees were used to model how to gain access (open) to secure documents, how a PGP encrypted file or password can be cracked, potential ways to infect a system files with a virus, how unauthorized users can obtain admin privileges, and how to gain remote access to a system [1, 2, 3] etc. In recent days, attack trees are used for security and risk assessment of assets such as SCADA systems [4], IoT systems [5], CPS [6], and medical equipment [7] etc. It is important to note that while attack trees remain a powerful graphical security tool that can be used to identify how an asset may be compromised, the shift in the dynamics of the assets i.e. from modelling and analysing single static systems to dealing with complex and dynamic (sometimes run concurrently) raised some questions in the effectiveness of using attack trees to analyse certain assets.

The settings of (traditional) attack trees are to depict (static) varying ways an asset may be compromised. However, most assets nowadays have erratic behaviour and can interact with other (sub)systems. This makes their vulnerabilities change according to the threat environments. Therefore, to capture such dynamism using attack trees, the estimated annotations (i.e. nodes) need to be updated regularly. Attempts have been made to extend attack trees through the introduction of attack-defence trees [8], attack protection trees [9], sequential and parallel attack

*29th International Workshop on Concurrency, Specification and Programming (CS&P'21)*

aliyu.ali@fmph.uniba.sk (A. T. Ali); gruska@fmph.uniba.sk (D. Gruska)

trees [10], and attack countermeasure trees [11] etc. These proposed extensions introduced how potential threats can be refuted. For example, if an attack tree models potential ways an asset may be attacked (i.e. access to a secure sever room), the security engineer having good knowledge of the assets surroundings, will design a security defence (hidden or open) to defend the asset. These concepts are proven effective to assets that can be modelled with finite number of nodes (states). The asset is modelled with an attack tree, and all possible attack paths are blocked with defence or attack countermeasures. However, for a large and complex asset that has infinitely many states, this concept is ineffective. Another important point is that; most assets nowadays are safety-critical. They do require a timely response to threats. This means apart from identifying possible ways an asset may be compromised, the model has to also provide a means to slow down or prevent the attack. In previous papers [9, 12], we investigated the concept of attack trees for stand-alone assets. In [9] we proposed the use of protection nodes as an alternative to defence nodes. However, we realised that such a method is less effective to CPS assets even with a small number of states. The challenge here is, CPS assets interact with a wide range of objects (i.e. routing signs, cameras, buildings) and respond accordingly. Each of these objects has its characteristics, and an attacker can manipulate these objects in different ways (e.g. blur, blocked, add) to compromise the asset (e.g. DoS, message falsification attacks). As such, a single pre-defined countermeasures or defence nodes is not enough to stop potential attacks. In [12] we explored informally an idea to introduce time constraints in the set of parent nodes in addition to the associated gates refinement. In this idea, we assume that even if the threat environment of an asset changed, new vulnerabilities that emerged are connected to an existing parent nodes. Although these vulnerabilities might not be refuted with the already existing defence or protection nodes for other vulnerabilities, the time constraints defined on the parent node will still be apply to the new vulnerabilities that connect to the parent node.

In this paper we push forward our approach, we formally defined attack trees with time constraints, we provided a case study to highlight situations where such model is applicable. Also, we translated the concept into a (weighted) timed automata, and use a formal verification tool UPPAAL to analyse. Let $q$ be a parent node that is associated with a gate refinement, assume we want to prevent an attacker from achieving the parent node. We introduce a set of constant time intervals $\tau$ that is represented by a constant pair $< b, f >$, with $b$ marking the start of an attack and $f$ marking the end of the attack on a parent node. $b, f \in \mathbb{Q}$, and $b \leq f$. We associate the set of attack actions $Act$ with a set of attack time $T$. An attack time defines the attack execution time for each action. To reach to the parent node, the attack time $t \in T$ for each action on a given child node under attack has to be less than or equal to the interval. In other words, if the attack time for child node(s) is more than the interval, the attacker cannot reach to the parent node. We formalized this idea and translated the parent node reachability to state reachability of timed automata.

**Related work.** Currently, threat modelling methods commonly used in industry mainly include graphical models, such as attack trees [1], and attack graph [13]. Among them, attack tree is a systematic attack scenario modelling method proposed by Schneier [1] and formally defined by Mauw and Oostdijk [2]. Since attack tree is a static model (and considered a semiformal model), several analysis frameworks have been proposed to establish its analysis method based on formal methods. These analysis frameworks have been developed based on timed automata [14], petri nets [15], and stochastic games [16] etc. The authors of [17] developed a stochastic framework for quantitative analysis of ADTrees. The framework adopts ADTree methodology to represent attack scenarios in a simple graphical representation and performs quantitative assessment using CTMC analytical approach. The authors of [18] introduced a multi-objective optimization framework for attack trees whereby the leaves of the attack trees were enriched with various parameters such as cost, time, skills and resources. The framework supports the computation of a wide range of security metrics such as attack values, attack paths, and ranking. They translated each attack tree gate and leaf into a priced timed automata and analyse the framework via UPPAAL CORE. In another effort, the work in [19] developed a modelling framework for expressing the temporal behaviour of an attacker as a boolean formula, and use a model checking tools to perform fully automated analyses of the modelled system by performing both qualitative (boolean) and quantitative (probabilistic) analysis. The authors demonstrated an example using UPPAAL tool, a network of timed automata that shows the model of a thief who wants to enter a house while the resident is not at home. The work in [20] defined the semantics for arbitrary attackers in an attack-defence tree using schematic timed automata (STA), and implemented the model by translating it into UPPAAL SMC. The authors modelled the encoding of an AD-Tree with one automaton modelling the defender, one automaton modelling the attacker and a separate one modelling the environment of an outcome, and coordinated their behaviour through synchronising channels. Other related works are [21, 22, 11, 8].

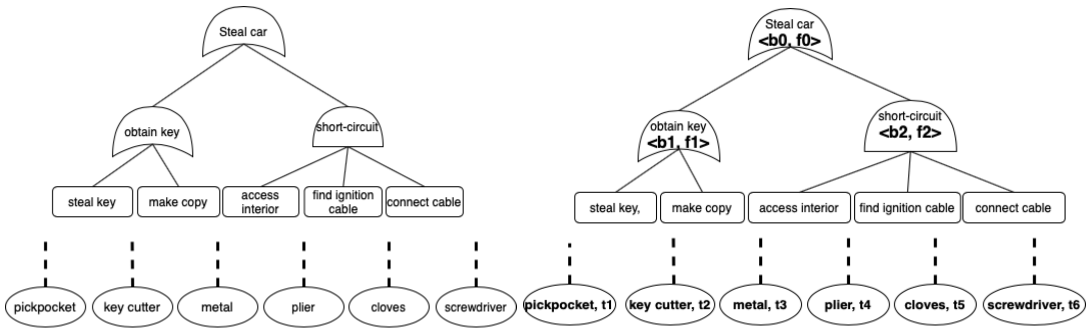
Unlike the works mentioned here, our work focus on traditional attack trees without considering defence nodes/actions. In our view, as mentioned earlier, defence actions are effective only to pre-identified vulnerabilities. As such cannot be applied to a (new) set of attack surface that emerged as a result of a change in the vulnerabilities landscape of the asset; modelled in an attack tree. Therefore, we shift our focus on preventing attacks by introducing a set of time constraints at the parent nodes; in addition to the gate refinement of the tree. We introduce a set of time constraints $\tau$ that is represented by a constant pair $< b, f >$, with $b$ marking the start of an attack and $f$ marking the end of the attack on a parent node. $b, f \in \mathbb{Q}$, and $b \leq f$. We associate the set of attack actions $Act$ with a set of attack time $T$. An attacker can achieve the parent node if and only if the attack action(s) together with the attack time can be executed within the defined constant time intervals. To model the time constraints, we translate the tree into a parallel composition of weighted timed automata (WTA). The sets of nodes of the attack trees are translated to a set of locations in the weighted timed automata. For each leaf node in the attack tree, we have an automaton that represents a linear path from the leaf to the root node. Altogether their areas many WTAs as the leaf nodes in the attack tree. Each location that represents a leaf has a clock that is activated when there is an attack in the corresponding leaf in the tree. A transition is enabled only if a simple attack is successful in the attack tree.

The paper is organized as follows. In Section 2 we describe attack trees formalism. In Section 3 we present the motivation for enhanced restrictions on attack trees model, and why time constraint is a good option. In Section 4 we present time constraint as a form of security in attack trees, and Section 5 timed automata and the translation of attack trees with time constraint into (weighted) timed automata. Section 6 contains discussions and plans for future work.

## 2. Attack trees formalism

In this section, we describe and define attack trees. An attack tree is a graphical way of describing varying ways an asset may be compromised by a malicious user (an attacker). The structure of attack trees we use in this paper is based on the existing model introduced by Schneier [1] but here we introduce a set of attacker's actions $Act$ (explain later) over the tree. $(\mathcal{Q}, \mathcal{E})$ is a tree, where $\mathcal{Q} = \{q_0\} \cup \mathcal{Q}_S \cup \mathcal{Q}_L$. $\{q_0\}$ is the root node of the tree, it represents attackers ultimate goal, $\mathcal{Q}_S$ is a set of internal nodes which we will refer to as sub-goal (also called parent nodes), they represent the decomposition of the root node into smaller units that are easier to solve, and $\mathcal{Q}_L$ is a set of leaf nodes (also called child nodes). The leaf nodes represent atomic nodes or end nodes (vulnerability), indicating an attack step. These sets of nodes are connected by a set of edges $\mathcal{E} \subseteq \mathcal{Q} \times \mathcal{Q}$. Each node in the attack tree (except for leaf nodes) has a gate refinement. A gate indicates (the fashion) how a node can be achieved (compromised). The interpretation of this fashion is on a node at a level above the current node. For this work, we make use of the $AND$ and $OR$ gates refinement. Informally, for a (parent) node with $AND$ gate, it is said to have a set of (child) nodes, that are linked to the parent node by a set of edges and all these (child) nodes must be achieved first before the parent node is reached. For a (parent) node with $OR$ gate, a single node from the set of child nodes when achieved is enough for the parent node to be reached. Formally we associate nodes with gates by the mapping $\mathcal{G} : \{q_0\} \cup \mathcal{Q}_S \to \{AND, OR\}$.

**Definition 1.** *An attack tree is a tuple $\mathcal{T} = (\mathcal{Q}, q_0, \mathcal{Q}_S, \mathcal{Q}_L, \mathcal{E}, \mathcal{G})$ where, $\mathcal{Q}$ is a finite set of nodes, $q_0$ is the root node of the tree, $\mathcal{Q}_S, \mathcal{Q}_L \subseteq \mathcal{Q}$ are two subsets such that $\mathcal{Q}_S \cup \mathcal{Q}_L = \mathcal{Q}$ and $\mathcal{Q}_S \cap \mathcal{Q}_L = \emptyset$, $\mathcal{E} \subseteq \mathcal{Q} \times \mathcal{Q}$ is a set of edges, connecting the nodes, and $\mathcal{G} : \{q_0\} \cup \mathcal{Q}_S \to \{AND, OR\}$ is a mapping that associate some nodes to a gate.*



Steal a car attack tree (A) An attack tree with time constraint (B)

**Figure 1:** Simple attack trees of how to steal a car that could be extended with time restriction

Note, to compromise nodes in the tree, an attacker needs to perform a set of attack actions. These set of (possible) actions are denoted by $Act$, and we defined an attack as a mapping $\mathcal{A} : \mathcal{Q}_L \to Act \cup \{Nil\}$ such that $A(l) = a$ means that an attacker can compromises leaf node $l \in \mathcal{Q}_L$ by executing an action $a \in Act$, $\mathcal{A}(l) = Nil$ when no action is performed. We say that attack $\mathcal{A}$ is simple if $\mathcal{A}(l) \neq Nil$ only for one leaf i.e. only one leaf is attacked. Practically, these set of attack actions are aided by the use of tools or/and techniques to carry out the attack.

Therefore, in this work, we will name a tool that can aid an attacker in executing the attack when referring to attack process in the working examples.

**Example 1.** *Shown in Figure.1 (A) is a simple attack tree that depicts possible ways to steal a car. The car can be stolen by achieving the sub-goal obtain key or short-circuit. To obtain the key, the sub-goal is of $OR$ refinement and therefore, an attacker must either steal the key or make a copy. While to achieve short-circuit, the sub-goal is of $AND$ refinement and the attacker must get access interior, find ignition cable, and connect the cable. Since the nodes must be achieved sequentially, in this case the $AND$ gate is said to be extended to $SAND$ (sequential $AND$). Linked with dotted lines (below the tree) are a set of aided tools or/and techniques, and an attack is performed when a tool is mapped to a node i.e. $\mathcal{A}$(make copy) = key cutter.*

To model the attack propagation (i.e. from leaf nodes, through sub-goals to root), we define a state of attack tree, denoted by $s$ i.e. state after an attacker has performed some actions from $Act$. A state of attack tree is defined as a set of nodes, and for each successful execution of attack actions (depending on the gate refinement of the target state), an attacker progresses to another state. An initial state is denoted by $s_0$. Let $s$ be a state of an attack tree, and let $\mathcal{A}$ be an attack. The transition $\delta : (s, \mathcal{A}) \rightarrow s^{'}$ defines a change in state from $s$, when an attack $\mathcal{A}$ is performed, to state $s^{'}$. We define this for simple attacks but it can be extended for arbitrary ones.

**Definition 2.** *Let $\mathcal{A}$ be a simple attack such that $\mathcal{A}(l_i) = b$ and let $s$ be an attack state. Then the next state $s'$ after attack $\mathcal{A}$ is defined as $s^{'} = r(s \cup l_i)$ where operation $r$ on the set of nodes is defined as follows: $t^{'} = r(t)$ iff $t'$ is the smallest set with the following properties*

- $t \subseteq t'$
- *if $q$ is a sub-goal that is associated with $AND$ gate and all its child nodes are in $r(t)$ then $q \in r(t)$*
- *if $q$ is a sub-goal that is associated with $OR$ gate and at least one of its child node is contained in $r(t)$ then $q \in r(t)$,*

A state is called *final* if it contains the root node. The transitive closure of $\delta$ defines reachability and will consider only states reachable from $s_0$ by some attacks. Henceforth, we will use the notation $s \xrightarrow{\mathcal{A}} s'$ instead of $\delta : (s, \mathcal{A}) \rightarrow s^{'}$. In the following lemma, we can see that an attack can be decomposed into simple attacks, however, the order of executing the attacks is important in succeeding.

**Lemma 1.** *Given an attack tree $T$. Let $\mathcal{A}_1, \mathcal{A}_2$ be two simple attacks and $s$ is a state. Then $s \xrightarrow{\mathcal{A}_1\mathcal{A}_2} s' \neq s \xrightarrow{\mathcal{A}_2\mathcal{A}_1} s'$, i.e. attacks are asymmetry.*

**Proof 1.** *By example. State $s$ is composed of all the nodes needed to reach $s'$, while $s'$ contains all the nodes attacked by $\mathcal{A}_1$ and $\mathcal{A}_2$. As we can see from example 1, it is impossible to connect the cable before accessing the interior and/or before finding the ignition cable. As such ordering of attacks has influences on multiple simple attacks.*

**Example 2.** *Let $s, s^{'}$ be a state and its successor respectively. Suppose from Fig.1 (A) to access the interior, an attacker needs a plier. Then a transition $s \xrightarrow{\mathcal{A}} s'$ iff $\mathcal{A}(\text{access interior}) = \text{plier}$.*

## 3. Motivation for enhanced restrictions

In this section, we will present motivations for a new security concept that will be formally introduced in the next section. We start by presenting why it is important to have *enhanced restrictions* in attack trees that will serve as a form of security in addition to identifying varying ways an asset (modelled with attack trees) may be compromised. But first, we start by explaining enhanced restriction and why it is needed.

An attack tree is a basic description of how an attacker may compromise an asset, without the description of how to prevent or repel the attack. This can play well into potential attackers; by analysing an asset using attack trees to identify the set of vulnerabilities. The gates describe an order (restriction) that guides how a set of parent nodes can be achieved. For example, the $AND$ gate refinement required an attacker to execute attack on all the child nodes (link to a parent node) before the attack succeeds. A more restrictive version of this was proposed in [10], where the attacker is required to achieve the (child) nodes in sequential order. Failing to achieve "all" the child nodes or " accordingly", will result in the attack process failing. Mechanisms like this can be added (hidden) to components that will help prevent an attack. However, assets such as CPS (i.e. interaction with objects from the physical environment which can be observed by the attacker), this can be uncovered easily. The following scenario motivates the need for extending attack trees (gates refinement) with time constraints.

**Attack Scenario:** Assume that an auto company developed an app tool that connects its customers with the service centre for

- *technical analysis*: whereby, some sensors in the vehicle can send data back to the manufacturer for intelligent and autonomous vehicle studies,
- *threat analysis*: whereby, safety or/and varying ways a vehicle can be in danger is identified and cautions message sent to the user.

Combined this, a security threat analysis of the vehicle can be carried out based on the threat environment (i.e. vehicle moving or parked), at each instance; an attack tree identifies potential threats and display either in the vehicles' onboard TFT LCD screen display or/and the user mobile phone as a notification. For each identified possible threat/fault, the app indicates the originating location (leaf node), other components in the vehicle that can be affected (sub-goals) and the resultant effect/damage (attack goal). Potential adversaries to the vehicle are classified as follows:

- *an insider*: someone with close relation to the auto company i.e. rough employee that directly/indirectly misuses his/her privileges,
- *generic attacker*: someone with the intention to exploit vulnerabilities, that can result in putting the vehicle to harm,
- *component failure*: part of vehicle components with rust/ware-out that can lead to damages.

For this paper, we will model working examples from a single threat environment i.e. the vehicle is at a parked position, and in our future work, we will consider working with a dynamic threat environment. Now, consider the vehicle user who is notified (by the app) of potential danger. Even though the attack tree can (correctly) show the originating point and target, the user cannot prevent or slow down the attack process.

**Example 3.** *Let us revisit example 1. let us assume a case whereby an attacker has already made some progress with the attack (i.e. access interior and locate ignition cable) and (s)he is interrupted. By the settings of traditional attack trees model, the attacker is not restricted from returning later in time to complete the attack, or with the previous knowledge, restart the attack process.*

It is important that, apart from identifying possible ways the attacker can achieve the target, a security measure is defined that will constrain the attacker from unlimited attempts.

## 4. Time constraints and security

In this section, we extend the set of gates refinement with a set of time constraints. The time constraints is an addition to the already existing gate refinement that is associated with each parent node. We also associate each attack action with an attack time, indicating the time needed for an attacker to complete executing the action.

Given a set of attack actions $Act$, we introduce a set of attack time $T$. $T$ is the time needed to complete an action that can result in compromising a node, denoted simply by $(a, t) \in Act \times T$ (see Figure.1 (B)). By doing so, we are extending the attack definition (defined in section 2) by $\mathcal{A} : \mathcal{Q}_L \to (Act \times T) \cup \{Nil\}$. The set of gates refinements mapping is also extended with a set of time constraints $\tau$ that is represented by a constant pair $\langle b, f\rangle$, with $b$ marking the start of an attack and $f$ marking the (expected) end of attack on the parent node such that $b, f \in \mathbb{Q}$, and $b \leq f$. This allows us to redefine the gates refinement mapping as $\mathcal{G} : \{q_0\} \cup \mathcal{Q}_S \to \{OR, AND\} \times \tau$. For the sake of simplicity, we denote this as $\langle b, f\rangle$. From the graphical representation shown in Figure.1 (B), one can easily derive these time extensions whereby $\langle b_0, f_0\rangle$, $\langle b_1, f_1\rangle$, and $\langle b_2, f_2\rangle$ is added to the parent nodes (root node and sub-goals), meaning they can only be reached if an attacker can execute the attack within the interval respectively. Also, below with the dotted lines, attack time $t_i$ is added to each action, where $i \in \{1 \ldots 6\}$. Regardless of the gates refinement i.e. $OR, AND$, an attack can only succeed if the action(s) execution time does not exceed the time intervals at the gates of the (parent) node.

**Example 4.** *Let us consider attack tree shown in Figure.1(B), the parent nodes are extended with time intervals represented as pairs $\langle b_0, f_0\rangle$ for the root node, $\langle b_1, f_1\rangle$ for the $OR$ sub-goal, and $\langle b_2, f_2\rangle$ for the $AND$ sub-goal. The attack actions (represented by tools) are also extended with attack time. Each time indicates the time needed to complete the attack. From the initial attack attempt, an attacker has until elapsed of $t_i$ to complete the attack, otherwise the whole attack process is consider failed. If $t_i$ is larger than the constant time interval for the connected sub-goal, the attack cannot succeed.*

**Definition 3.** ***Attack trees with time constraint.*** *Let $(\mathcal{Q}, \mathcal{E})$ be a tree, an attack tree with time constraint is a tuple $\mathcal{T}_r = (\mathcal{Q}, q_0, \mathcal{Q}_S, \mathcal{Q}_L, \mathcal{E}, \tau, \mathcal{G}, T, \mathcal{A})$, where, $\mathcal{Q}$ is a finite set of nodes, $q_0$ is the root node of the tree that represents the attack target, $\mathcal{Q}_S, \mathcal{Q}_L \subseteq \mathcal{Q}$ are two subsets such that $\mathcal{Q}_S \cup \mathcal{Q}_L = \mathcal{Q}$ and $\mathcal{Q}_S \cap \mathcal{Q}_L = \emptyset$, $\mathcal{E} \subseteq \mathcal{Q} \times \mathcal{Q}$ is a set of edges that connect the nodes, $\tau$ is* *a set of time constraints, $\mathcal{G} : \{q_0\} \cup \mathcal{Q}_S \rightarrow \{AND, OR\} \times \tau$ is the mapping that associate some nodes to a gate and time constraint, $T$ is a set of attack time, and $\mathcal{A} : \mathcal{Q}_L \rightarrow (Act \times T) \cup \{Nil\}$.*

**Definition 4.** *Given an attack tree with time constraint $\mathcal{T}_r$, an attack over the tree that can reach to the root node is defined as (assuming $a \in Act, l \in \mathcal{Q}_L$ and $t \in T$)*

- *If the tree has $AND$ gates, $\forall \mathcal{A}_1 \ldots \mathcal{A}_n : \forall t_i \ldots t_n$ are less than or equal to the constant time interval on the parent node,*
- *If the tree has $OR$ gates, $\exists \mathcal{A}_i : t_i$ is less than or equal to the constant time interval on the parent node.*

As an example, let us consider a sub-goal *short-circuit* from Figure. 1 (B), an attacker can succeed in achieving the sub-goal if and only if the summation of attack time $t_3 + t_4 + t_5$ does not exceed the time interval defined at $\langle b, f \rangle$.

From Figure.1 (B), $\mathcal{A}(\mathcal{Q}_L) = Nil$ if the attacker cannot complete the attack within the time constraint at the gates. The following lemma guarantees that under some conditions a parent node remains unreachable to a potential attacker regardless of the gate refinement associated with the parent node.

**Lemma 2.** *Given a sub-goal $S$, a set of time constraint defined at the gate refinement $\langle b, f \rangle$, and a set of attack (actions) $X \subseteq Act$ required to compromise $S$. The sub-goal is unreachable to an attacker if the attack execution time $\forall i \in X$ exceeds the attack time defined at the gate refinement.*

**Proof 2.** *Since each action has an attack time, we need to show that this attack time for all the child nodes of $S$ exceeds the time constraint at the gate. Now if this action cannot be executed within the attack time, the sub-goal cannot be achieved.*

### 4.1. Enhanced restrictions and security

There are some relations between time constraint on the gates refinement and improved security for different kinds of systems; in general. For example, the use of idle time outs for web session is a common practice in the development of high-risk applications such as online banking platforms. Other instances where time constraint is used to improve the security of asset can be found on e-locks (eg. car central locking system) that can automatically locked itself after a specified passage of time [23]. The implementation of time restriction such as action time-out in assets such as ATMs [15], and SMART-doors [24] are further good examples.

In addition to other extensions of attack trees with security mechanism such as defence nodes or attack countermeasures, time constraint can be used to model and analyse different kinds of attack scenarios for different kinds of assets.

## 5. Timed automata

As in [12], we now consider the use of a weighted timed automata to analyse our model. A weighted timed automata (WTA) is an extension of timed automata [14] with cost/price information on both locations and edges that can be used to solve several interesting problems. There exists a formal verification tool UPPAAL [25] that accepts WTA as its modelling language. Before we explain further, we first recall the definition of a timed automata. (henceforth, we refer to a location of timed automata by $l$).

**Definition 5.** *A timed automaton is a tuple $TA = (\mathcal{L}, \mathcal{L}_0, \mathsf{E}, \mathcal{C}, \mathcal{I}, T, f)$, where $\mathcal{L}$ is a finite set of locations, $\mathcal{L}_0 \subseteq \mathcal{L}$ is a set of initial locations, $\mathsf{E}$ is a finite set of synchronization actions (events), $\mathcal{C}$ is a finite set of clocks, $\mathcal{I} : \mathcal{L} \to \Phi(\mathcal{C})$ is an invariant, assigning to every location $l \in \mathcal{L}$ a clock constraint, $T \subseteq \mathcal{L} \times \mathsf{E} \times \Phi(\mathcal{C}) \times 2^{\mathcal{C}} \times \mathcal{L}$ is a set of transition relations such that $\langle l, a, \phi, c, l^{'} \rangle$ represents an edge from location $l$ to location $l^{'}$ on symbol $a$. $\phi$ is a clock constraint, $c \subseteq \mathcal{C}$ is a set of clocks to be reset, and $f$ is a final location.*

The semantics of a timed automaton $TA$ is defined by associating a (labelled) transition system $TA_{LTS}$ (defined in section 1) with it. A state in $TA_{LTS}$ consists of a pair $(l, v)$ whereby $l$ is a location of $TA$ and $v$ indicates that for a clock $c$, $v$ satisfies the label constraint $\mathcal{I}(l)$.

## 5.1. Weighted time automata and attack trees translation

In this subsection, we introduce the basics of a WTA and give the translation of attack trees with time constraint into a WTA.

A Weighted timed automata, otherwise known as price timed automata (PTA), is an extended version of timed automata (TA) with weight/cost information added on both locations and edges. To arrive at a location, the weight value defined at that location has to be satisfied, also, to enabled a transition via an edge, the weight value at that edge has to be satisfied. At the final/desired location, a global weight/cost which is the accumulated weight/cost along the run is calculated. With this accumulative weight/cost value, it is easy to calculate the distance or cost of travelling from point $A$ to point $B$ in different case studies. Formally, a weighted timed automata is defined as follows [25].

**Definition 6.** *The tuple $WTA = (\mathcal{L}, \mathcal{L}_0, \mathsf{E}, \mathcal{C}, \mathcal{I}, \mathcal{W}, \mathcal{WP}, Tf)$ is a weighted timed automaton, where $\mathcal{L}$ is a finite set of locations, $\mathcal{L}_0 \subseteq \mathcal{L}$ is a set of initial locations, $\mathsf{E}$ is a finite set of synchronization actions (events), $\mathcal{C}$ is a finite set of clocks, $\mathcal{I} : \mathcal{L} \to \Phi(\mathcal{C})$ is an invariant, assigning to every location $l \in \mathcal{L}$ a clock constraint, $\mathcal{W} : \mathcal{L} \cup \mathsf{E} \to \mathbb{N}^n_{\geq 0}$ is a function that assigns weight value to location and edge, $w : \mathcal{WP} \to \mathbb{Q}$ is a set of weight parameters that updates the weight value $\alpha : \mathcal{W} \to (\mathcal{W} \cup \mathcal{WP})$, $T$ is a set of transitions such that $\langle l, \phi, a, c, \alpha, l^{'} \rangle$, where $l, l^{'} \in \mathcal{L}$ are the source and target locations, $\phi$ is a clock constraint, $a \in \mathsf{E}$, $c \subseteq \mathcal{C}$ is a set of clocks to be reset, and $\alpha$ is a parametric weight update, and finally $f$ is a final location.*

The trace of a weighted timed automata is a sequence of states with the transitions across the states given as $\pi = l_0 \xrightarrow[c_0]{a_0}^{\alpha_0} l_1 \xrightarrow[c_1]{a_1}^{\alpha_1} l_2 ...$ such that

- there is always an initial location $l_0$ with an initial clock valuation $c_0 = 0$,
- for every $i \in \{1, .., k\}$, there is some transition $(l_i, \phi, a_i, c_i, \alpha_i, l_{i+1}) \in T$,
- a transition is enables only if; for every clock valuation $v$, there exists a constraint $\phi$ such that $v$ satisfies $\phi$,

- for every successful transition, a new clock valuation $c_{i+1}$ is obtained by increasing every clock variable in $c_i$ by a transition $i$ and resetting all previous clocks $0$.

Now, in other to analyse attack trees with time constraint using UPPAAL, we translate the tree into a parallel composition of weighted timed automata. The sets of nodes of the attack trees are translated to a set of locations in the weighted timed automata. For each leaf node in the attack tree, we have a WTA that represents a linear path from the leaf to the root node. Altogether there are as many WTAs as the leaf nodes in the attack tree. Each location that represents a leaf which has a clock that is activated when there is an attack in the corresponding leaf in the tree. An attack on a node in the tree represents an enabled transition in the WTA. Initially, clocks become active when events synchronized, and end with either a success or fail synchronization action.

More general, an attack tree is translated into a parallel composition of weighted timed automata $WTA_1 \ldots WTA_n$ that represents linear path from the leaf to the root node. Given an attack tree with time constraint $\mathcal{T}_r$, the semantics of successfully reaching the root node that satisfies the weighted timed automata WTA can be given as $[\![\mathcal{T}_r]\!] \subseteq$ WTA if

- $[\![q_0]\!] =$ WTA, a final location $f \in$ WTA is always satisfied,
- $[\![\mathcal{Q}]\!]^{OR} = \{\text{WTA}_1 \ldots \text{WTA}_n\}$such that at least $\text{WTA}_i$ reached the final location $f_i$,
- $[\![\mathcal{Q}]\!]^{AND} = \{\text{WTA}_1 \ldots \text{WTA}_n\}$such that for all $\text{A}_i$, the final location $f_i$ reached.

**Lemma 3.** *Let $\mathcal{T}_r$ be an attack tree with time constraint and let $WTA$ be the equivalent product of weighted timed automata. Suppose $S$ is a (target) node, and location $l$ is the (equivalent) translation in $WTA$. The number of the active clock(s) in $WTA$ to reach $l$, is the same as the number of (attack) actions carried out by an attacker before reaching $S$.*

**Proof 3.** *We know that a clock is reset for each enabled transition in WTA, therefore, since the locations of $WTA$ corresponds to the nodes in the attack trees, each enabled clock indicates an active attack on a node. As such, the number of attack executed will correspond to the number of clock(s) reset for events in the $WTA$.*

**Theorem 1.** *Let $WTA$ be a product of weighted timed automata for the translation of $\mathcal{T}_r$. For a transition $(l_i, \phi, a_i, c_i, \alpha_i, l_{i+1})$, the clock $c_i$ is inactive for a corresponding node $q \in \mathcal{Q}$ in the tree, if there is no active attack process on that node.*

**Proof 4.** *Directly from lemma 3.*

We can model an attack $\mathcal{A}$ as a special weighted timed automata, and denote it as $\mathcal{A}WTA$, this weighted timed automata shows the attackers' action and time when they are performed on the attack tree. We use this in the following theorem to check the attack target (root node) reachability for both the attack tree and the WTA.

**Theorem 2.** *Given an attack tree with time constraint $\mathcal{T}_r$ and a time automata $WTA$, an attacker $A$ can only reach the root node of the tree $\{q_0\}$ iff corresponding $WTA$ plus $\mathcal{A}WTA$ running with the attack path in the tree can reach the final location.*

**Proof 5.** *The main idea. We know that a root node can only be reached if an attacker can succeed in completing the attack process. Therefore we have to show that a $WTA$ plus $\mathcal{A}WTA$ can also reach the final location.*

### 5.2. UPPAAL model

Uppaal accepts the synchronization of events using channels (input and output). As such, we modelled a set of events $a?, b?, c?, d?$ (receive) at the initial location to serve as a set of leave nodes. We use two clocks $t$ and $x$, with $t$ being a clock associated with each event while $x$

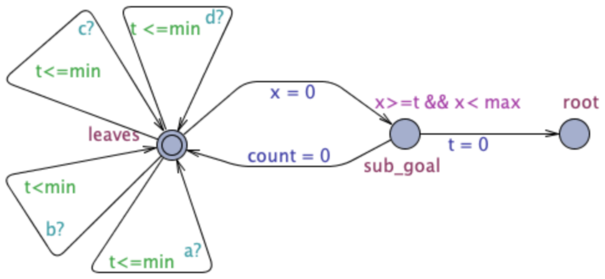


**Figure 2:** A simple UPPAAL model of nodes and time constraints

associated with a target location. Here, $x$ is a clock to track $t$. We also define two invariant $min$ and $max$, to check whether both $t$ and $x$ are satisfied before the transition to the target location is enabled.

The UPPAAL template shown in Figure.2 is a simple model of time constraint in achieving nodes in an attack trees. The *leaves* locations begin by waiting for the activation of signal by one of the events $a?, b?, c?, d?$ An event become activate only when the clock is less than (predefined) $min$. If the *sub_goal* location is of $OR$ gate, the activation of a single event is enough for the transition to be enabled to the target location (*sub_goal*), otherwise all the events (i.e. $AND$ gate) needs to be activated, and the *sub_goal* location can only be reached if the clock $x$ is less than (predefined) $max$.

## 6. Discussion and future work

In this paper, we have presented attack trees with a time constraint to serve as a secured version of attack trees. We discussed how the gates refinement can be extended with time parameters and also propose how the attack trees with time constraint can be modelled using a formal verification tool UPPAAL.

As further work, we plan to study how attack trees can be used to analyse a CPS. A CPS is a special kind of asset that can interact with objects from the physical environment as well as other cyber-systems. This allows its operations to run parallel and concurrent, making it easy for a potential attacker to observe (from the physical components), and perform some dangerous attacks such as message falsification attack, DoS, message spoofing etc. by simply adding, blocking or blurring the objects from the physical environment. This kind of threats cannot be captured using attack trees alone. We plan to investigate *opacity*, a security property formalizing the information leakage of a system to an external observer, namely intruder and study how it can be used with attack trees.

## Acknowledgement

This work was supported by the Slovak Research and Development Agency under the Contract no. APVV-19-0220 (ORBIS) and by the Slovak VEGA agency under Contract no. 1/0778/18 (KATO).

## References

[1] B. Schneier, Attack trees, Dr. Dobb’s journal 24 (1999) 21–29.
[2] S. Mauw, M. Oostdijk, Foundations of attack trees, in: International Conference on Information Security and Cryptology, Springer, 2005, pp. 186–198.
[3] B. Kordy, L. Piètre-Cambacédès, P. Schweitzer, Dag-based attack and defense modeling: Don’t miss the forest for the attack trees, Computer science review 13 (2014) 1–38.
[4] C.-W. Ten, C.-C. Liu, G. Manimaran, Vulnerability assessment of cybersecurity for scada systems, IEEE Transactions on Power Systems 23 (2008) 1836–1846.
[5] D. Beaulaton, N. B. Said, I. Cristescu, S. Sadou, Security analysis of iot systems using attack trees, in: International Workshop on Graphical Models for Security, Springer, 2019, pp. 68–94.
[6] F. Xie, T. Lu, X. Guo, J. Liu, Y. Peng, Y. Gao, Security analysis on cyber-physical system using attack tree, in: 2013 Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IEEE, 2013, pp. 429–432.
[7] M. A. Siddiqi, R. M. Seepers, M. Hamad, V. Prevelakis, C. Strydis, Attack-tree-based threat modeling of medical implants., in: PROOFS@ CHES, 2018, pp. 32–49.
[8] A. Roy, D. S. Kim, K. S. Trivedi, Attack countermeasure trees (act): towards unifying the constructs of attack and defense trees, Security and Communication Networks 5 (2012) 929–943.
[9] A. T. Ali, D. P. Gruska, Attack protection tree., in: CS&P, 2019.
[10] F. Arnold, D. Guck, R. Kumar, M. Stoelinga, Sequential and parallel attack tree modelling, in: International Conference on Computer Safety, Reliability, and Security, Springer, 2014, pp. 291–299.
[11] X. Ji, H. Yu, G. Fan, W. Fu, Attack-defense trees based cyber security analysis for cpss, in: 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), IEEE, 2016, pp. 693–698.
[12] A. T. Ali, Simplified timed attack trees, in: International Conference on Research Challenges in Information Science, Springer, 2021, pp. 653–660.
[13] X. Ou, A. Singhal, Attack graph techniques, in: Quantitative Security Risk Assessment of Enterprise Networks, Springer, 2012, pp. 5–8.

[14] R. Alur, Timed automata, in: International Conference on Computer Aided Verification, Springer, 1999, pp. 8–22.

[15] B. Berthomieu, M. Diaz, Modeling and verification of time dependent systems using time petri nets, IEEE transactions on software engineering 17 (1991) 259.

[16] L. S. Shapley, Stochastic games, Proceedings of the national academy of sciences 39 (1953) 1095–1100.

[17] K. Lounis, S. Ouchani, Modeling attack-defense trees' countermeasures using continuous time markov chains, in: International Conference on Software Engineering and Formal Methods, Springer, 2020, pp. 30–42.

[18] R. Kumar, E. Ruijters, M. Stoelinga, Quantitative attack tree analysis via priced timed automata, in: International Conference on Formal Modeling and Analysis of Timed Systems, Springer, 2015, pp. 156–171.

[19] O. Gadyatskaya, R. R. Hansen, K. G. Larsen, A. Legay, M. C. Olesen, D. B. Poulsen, Modelling attack-defense trees using timed automata, in: International Conference on Formal Modeling and Analysis of Timed Systems, Springer, 2016, pp. 35–50.

[20] R. R. Hansen, P. G. Jensen, K. G. Larsen, A. Legay, D. B. Poulsen, Quantitative evaluation of attack defense trees using stochastic timed automata, in: International Workshop on Graphical Models for Security, Springer, 2017, pp. 75–90.

[21] I. A. Tøndel, M. G. Jaatun, M. B. Line, Threat modeling of ami, in: Critical Information Infrastructures Security, Springer, 2013, pp. 264–275.

[22] A. E. M. AL-Dahasi, B. N. A. Saqib, Attack tree model for potential attacks against the scada system, in: 2019 27th Telecommunications Forum (TELFOR), IEEE, 2019, pp. 1–4.

[23] D. Ray, The time structure of self-enforcing agreements, Econometrica 70 (2002) 547–582.

[24] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, Modeling tcp throughput: A simple model and its empirical validation, in: Proceedings of the ACM SIGCOMM'98 conference on Applications, technologies, architectures, and protocols for computer communication, 1998, pp. 303–314.

[25] P. Bulychev, A. David, K. G. Larsen, M. Mikučionis, D. B. Poulsen, A. Legay, Z. Wang, Uppaal-smc: Statistical model checking for priced timed automata, arXiv preprint arXiv:1207.1272 (2012).

# Extended Abstract: Simulation of Interactions between Beehives

Volha Taliaronak[1], Heinrich Mellmann[1] and Verena V. Hafner[1]

[1]*Humboldt University of Berlin, Unter den Linden 6, Berlin, 10117, Germany*

**Abstract**

The interdisciplinary EU project HIVEOPOLIS aims to develop a new generation of intelligent beehive which might help bees in coping with adverse environmental factors. As a part of the HIVEOPOLIS project, this extended abstract reports on our ongoing work on the simulation of a decision-making process based on interactions between HIVEOPOLIS units and bee colonies which are not equipped with HIVEOPOLIS systems using the Mesa simulation framework.

**Keywords**

Multi-agent systems, Bio-hybrid systems, HIVEOPOLIS

## 1. Introduction

The impact of humans on the environment is difficult to exaggerate. Habitats of different species have been reduced, transformed or damaged as a result of monocultural agriculture, pesticide pollution, etc [1]. These changes in addition to colony diseases dramatically affect insects including bees. As a result, one of the concerning issues has become the sharing of valuable food sources, such as pollen and nectar as well as the limited habitat between native bee species and honeybees, as invasive species [2, 3].

The interdisciplinary EU project HIVEOPOLIS aims to develop a new generation of intelligent beehive which might help bees in coping with these adverse environmental factors and provide a synergistic added value to the colony, to its owner, and to the ecosystem in general [4, 5]. The intelligent beehives will form connected bio-hybrid systems. One concrete example is an active selection of foraging grounds, which could enable a mutually beneficial distribution of resources among several beehives and avoidance of areas affected by pesticides. Bees communicate beneficial foraging locations through a specific *waggle dance* [5, 6]. A HIVEOPOLIS beehive will be equipped with a technology, which enables decoding, suppressing and imitating such dances and allows the system to actively influence the foraging locations of the bees [5, 6]. A prototype of such robot, called RoboBee, was introduced in [6]. And as observed in [7], HIVEOPOLIS unit may incorporate one or more dancing robots which interact with honeybees to communicate them directions to floral resources. From that perspective a bio-hybrid HIVEOPOLIS beehive can be seen as an autonomous robot making autonomous decisions and negotiating with other

*CS&P 2021: 29th International Workshop on Concurrency, Specification and Programming*

taliarov@hu-berlin.de (V. Taliaronak); mellmann@informatik.hu-berlin.de (H. Mellmann); hafner@informatik.hu-berlin.de (V. V. Hafner)
0000-0001-8150-024X (V. Taliaronak)

beehives, e.g., send the bees to a particular region or, potentially, prevent them from harvesting in another.

Bees are able to exhibit complex swarm behaviors like decentralized target selection and workload balancing [8]. A decision mechanism steering the behavior of a whole bee swarm requires a model describing how waggle dances generated by robots which mimic honeybees would influence the behavior of the bees and their interaction with the environment. The first results of such modeling experiments for foraging choices in a bee swarm were introduced in [7] where the authors investigate the effect of robots on the hive's foraging decisions using a mathematical model. Further, such models can form a basis for a decision process based on anticipation as described in [9].

In this paper we introduce a proof of concept rule-based decision-making process and simulate behavior of bees' colonies and their interaction with the environment using agent based modeling approach. We aim to simulate different scenarios and study direct and indirect interactions between native bee colonies and honeybee colonies with integrated external regulation mechanisms and gain insights about possible competitive factors as well as cooperative strategies.

## 2. Modeling Methods

Behavior and interactions are the two key issues for modeling ecosystem organization. Using the Mesa framework [10], we direct our attention on a simulation of a decision-making process based on the interactions between agents (bees) and the environment, while bee agents' behavior is reduced to honeybees' foraging behavior, and the surrounding landscape is modeled as a simplified forage map [11].

### 2.1. Mesa

In comparison to the other well-known simulation tools, like NetLogo [12] and Mason [13], this framework has several competitive advantages. First of all, it is python-based and can be extended with modern python libraries and other python-based tools (e.g., Jupyter Notebook and Pandas tools) in order to create more complex simulations or analyse collected data. The collected data can be stored in a JSON or Pandas DataFrame format for further analysis. Second, Mesa consists of decoupled components, which can be replaced or used independently from each other. Third, visualization is browser-based, which provides additional opportunities for sharing of visualisation via the Internet. Since all components in the Mesa framework are decoupled, visualisation modules can be customized, extended, replaced, or removed.

### 2.2. Simulation

The environment of our simulations is discrete, modeled as a squared grid, with three types of agents: bee, beehive, and field. The beehives are modeled as hierarchical multi-agent system which consists of two levels: bees' level and beehives' (or colonies') level. A bee swarm is considered to be a multi-agent system with a non-hierarchical structure, where every bee is modeled as a separate agent. On the other hand, every beehive itself is considered an agent.

Despite the fact that HIVEOPOLIS unit has been conceived and, in reality, may be designed as a robotic honeybee imitating waggle dancing, in our simulations we implemented HIVEOPOLIS unit on the beehives' level as a separate type of beehive agent. Figure 1 (Right) demonstrates one of our simulations, which consists of two beehives ((1) is a beehive with an integrated HIVEOPOLIS unit and (2) is a wild beehive) with bees (e.g., (6), (7)) and three possible food sources ((3), (4), (5)), green-colored cells represent field agents without available food sources. For the sake of simplicity, we do not model the whole complex social organisation of an individual colony, ignore the diversity of bees' casts (workers, drones, queen) and food sources (nectar, pollen, water). Bee and beehive agents are described with a limited number of parameters (e.g., maximum flying distance, collected amount of food, coordinates of a known field, abundance of a field, etc). Floral sources are described using only three relevant parameters: amount of available resources, blooming tag, flowering period. We simulate agents' movements as discrete events. An activation order of agents is randomized in order to reduce its impact on the model.

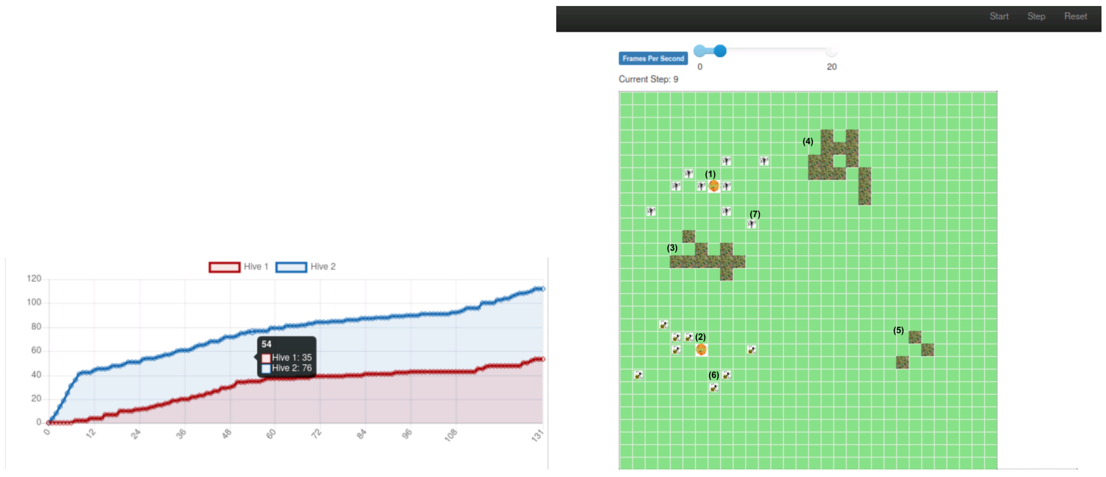


**Figure 1:** Simulation of the environment with two beehives. Left: graphs of collected food for each beehive. Right: a 2D view of the environment.

We implemented two foraging strategies for bees agents: *random foraging search* and *targeted foraging on the known floral patch.* We also considered two different interaction strategies between agents. The first one, a direct interaction, occurs on the bees' level, when the information about known foraging resources is communicated between simulated bees from one colony. So, for instance, the wild bees start with the random search strategy, as seen in Figure 1 (Right). After a discovery of a field with available floral resources (e.g., field (3) in Figure 1 (Right)), the bees return back to the hive and share the information about the found floral resource with other bees which are also in the hive. Bees might follow the communicated directions, in this case, they switch to the targeted foraging strategy, but might also ignore this information about the found food sources. The second one, an indirect interaction, occurs on the beehives' level, where the internal robot is supposed to define the most optimal food source and communicate it to the bees. This type of interactions is implemented only in the beehives with HIVEOPOLIS units. We assume that it will make autonomous decisions regarding optimal foraging sources based on information about the surrounding landscape, weather and other information received from the external sources as well as its predictions about behavior of the other beehives from the surroundings. The implementation details of the HIVEOPOLIS' interior robot are outside the scope of this work. In our simulations, the optimal fields are determined using rule-based decision-making approach which is based on three parameters: a distance parameter, a flowering period of fields, and an abundance of fields. In every simulation step, it is checked if there are any bees in the hive. If there are any, then a new optimal field is being calculated on the beehive's level and communicated to the bees on the bees' level. Also a decision, to follow the communicated coordinates or ignore them is being simulated on the bees' level. Data generated during simulation is displayed in the real-time mode in the live chart (see Figure 1 (Left)), the x axis shows the number of simulation steps, the y axis – the foraging dynamics of the modeled hives.

## 3. Results and Discussion

Bees are not only important pollinators but also a prime example of swarm intelligence [8]. Such modeling tools, like BEESCOUT [11] and BEEHAVE [14], can be useful for better understanding and exploration of the possible realistic scenarios of colony natural dynamics, bees' searching behavior in habitats with different landscape configuration as well as interactions between bees. Nevertheless, these tools are NetLogo based and cannot be applied for simulations of interactions between several colonies.

We are not the first, who is aiming to simulate decision-making processes in bees' swarms. A multi-agent simulation able to simulate the dynamics of honeybee nectar foraging was conducted using NetLogo tool and introduced in [8]. The authors implemented experiments reported in [15] and other works of T. Seeley, who investigated decision-making mechanisms within bee swarm.

Nonetheless, in our work, we aim to model and simulate decision-making processes not only within one colony, as it was done in [15, 8], but in a system of beehives and HIVEOPOLIS units. For this reason, we examined the factors relevant for a selection process of foraging sources. In [15] the authors highlighted three main factors, which are being considered during a process of choosing nectar sources: *distance*, *quality* and *the abundance of the food.* The factors determining the quality of food sources are, for instance, *difficulty of feeding at the source*, *direction in relation to the wind*, and *the colony's need for food*, etc [15]. In our first attempt of simulations, we focus on a distance from hive and abundance of patches. We also added one more parameter, which was not mentioned by [15], but can be relevant for our purposes, – flowering period of floral patches. *Quality* of food resources is a complex factor which is hard to capture and implement without any real data. Nonetheless, we hope to find a way how to integrate this parameter in our further simulation scenarios.

A HIVEOPOLIS bio-hybrid system could serve as a mechanism for implementing interactions on the beehive level and having an influence on the colony decisions regarding chosen food resources. Such centralized control mechanism might be beneficial for cases in which several bee colonies have to share limited floral resources, floral resources are difficult to discover due to the morphology of the beehive surrounding area, or a gentle way to redirect the bees to

the desired fields is required. It might be a feasible path to make safer or ecologically more important food sources more attractive to bees, even if these sources are energetically less profitable [7].
Our simulations don't capture the whole complexity of a decision-making process yet. We are going to continue our work on simulations with different combinations of possible competitive factors such as a food diversity. We consider evaluation of collected data to be a non-trivial task, which will require expertise from other scientific fields. The collected data are strongly affected by the parameters (e.g., number of bees per hive), which are relative values. Nevertheless, all parameters can be changed without much extra effort. Simulation experiments have low computational cost and can be run repeatedly.

## 4. Conclusion

Mesa is a convenient, powerful tool, which provides a solid base functionality for easy and comfortable simulation implementations as well as enough capabilities for customization of created models and their visualisation. Since the framework is based on Python, it might be advantageous and handy for a wide range of researchers.
Multi-agent systems are useful for problems integrating social and spatial aspects and suitable for simulation of complex systems. Models and simulations of beehives have been studied for a long time, so that we can draw experience from a rich library of literature. The novel direction of this work is the study of the simulation scenarios as a basis for a decision mechanism, which would allow a bio-hybrid beehive to act autonomously in a way beneficial to itself and its environment. Our preliminary results have shown coherent behaviors of the whole simulation, nevertheless, the model parameters require further tuning and scientific justification. Our further work will be focused on extension and improving of the existing simulation model. In order to increase credibility of our simulations, we aim to utilise geospatial data. The goal is to integrate an augmented map of a landscape and model the distribution of the floral resources and landscape features more precisely. Further, we are planning on collecting additional data and storing it in DataFrame format for further analysis using modern data science libraries.

## Acknowledgments

This work was supported by the EU H2020 program under grant agreement No. 824069 (HIVEOPOLIS).

## References

[1] O. Komasilova, V. Komasilovs, A. Kviesis, N. Bumanis, H. Mellmann, A. Zacepins, Model for the bee apiary location evaluation, Agronomy Research 18 (2020) 1350–1358. doi:10.15159/AR.20.090.

[2] A. Hudewenz, A.-M. Klein, Competition between honey bees and wild bees and the role of nesting resources in a nature reserve, Journal of Insect Conservation 17 (2013) 1275–1283. doi:10.1007/s10841-013-9609-1.

[3] C. Rasmussen, Y. L. Dupont, H. B. Madsen, P. Bogusch, D. Goulson, L. Herbertsson, K. P. Maia, A. Nielsen, J. M. Olesen, S. G. Potts, S. P. M. Roberts, M. A. K. Sydenham, P. Kryger, Evaluating competition for forage plants between honey bees and wild bees in denmark, PLOS ONE 16 (2021) 1–19. doi:10.1371/journal.pone.0250056.

[4] CORDIS, Futuristic beehives for a smart metropolis, 2019. URL: https://cordis.europa.eu/project/id/824069.

[5] A. Ilgün, K. Angelov, M. Stefanec, S. Schönwetter-Fuchs, V. Stokanic, J. Vollmann, D. N. Hofstadler, M. H. Kärcher, H. Mellmann, V. Taliaronak, A. Kviesis, V. Komasilovs, M. A. Becher, M. Szopek, D. M. Dormagen, R. Barmak, E. Bairaktarov, M. Broisin, R. Thenius, R. Mills, S. C. Nicolis, A. Campo, A. Zacepins, S. Petrov, J.-L. Deneubourg, F. Mondada, T. Landgraf, V. V. Hafner, T. Schmickl, Bio-hybrid systems for ecosystem level effects, in: Proceedings of the ALIFE 2021: The 2021 Conference on Artificial Life, MIT Press Direct, Virtual (formerly Prague), Czech Republic, 2021. doi:10.1162/isal_a_00396.

[6] T. Landgraf, D. Bierbach, A. Kirbach, R. Cusing, M. Oertel, K. Lehmann, U. Greggers, R. Menzel, R. Rojas, Dancing honey bee robot elicits dance-following and recruits foragers, 2018. URL: https://arxiv.org/abs/1803.07126.

[7] D. Lazic, T. Schmickl, Can robots inform a honeybee colony's foraging decision-making?, in: Proceedings of the ALIFE 2021: The 2021 Conference on Artificial Life, MIT Press Direct, Virtual (formerly Prague), Czech Republic, 2021. doi:10.1162/isal_a_00397.

[8] T. Schmickl, K. Crailsheim, Costs of environmental fluctuations and benefits of dynamic decentralized foraging decisions in honey bees, in: C. Anderson, T. Balch (Eds.), The 2nd International Workshop on the Mathematics and algorithms of Social Insects Proceedings, Georgia Institute of Technology, Atlanta, GA, 2003, pp. 145–152.

[9] H. Mellmann, B. Schlotter, L. Musiolek, V. V. Hafner, Anticipation as a mechanism for complex behavior in artificial life, in: Proceedings of the ALIFE 2020: The 2020 Conference on Artificial Life, MIT Press Direct, Virtual (formerly Prague), Czech Republic, 2020, pp. 157–159. doi:10.1162/isal_a_00314.

[10] J. Kazil, D. Masad, A. Crooks, Utilizing python for agent-based modeling: The mesa framework, in: R. Thomson, H. Bisgin, C. Dancy, A. Hyder, M. Hussain (Eds.), Social, Cultural, and Behavioral Modeling, volume 12268 of *Lecture Notes in Computer Science*, Springer, Cham, 2020, pp. 308–317. doi:10.1007/978-3-030-61255-9_30.

[11] M. A. Becher, V. Grimm, J. Knapp, J. Horn, G. Twiston-Davies, J. L. Osborne, Beescout: A model of bee scouting behaviour and a software tool for characterizing nectar/pollen landscapes for beehave, Ecological Modelling 340 (2016) 126–133. doi:10.1016/j.ecolmodel.2016.09.013.

[12] S. Tisue, U. Wilensky, Netlogo: A simple environment for modeling complexity, in: C. Anderson, T. Balch (Eds.), The International Conference on Complex Systems, Boston, MA, USA, 2004.

[13] S. Luke, C. Cioffi-Revilla, L. Panait, K. M. Sullivan, G. Balan, Mason: A multi-agent simulation environment, SIMULATION 81 (2005) 517–527.

[14] M. A. Becher, V. G. P. Thorbek, J. Horn, P. J. Kennedy, J. L. Osborne, Beehave: a systems model of honeybee colony dynamics and foraging to explore multifactorial causes of colony failure, Journal of Applied Ecology 51 (2014) 470–482. doi:10.1111/1365-2664.12222.

[15] T. Seeley, S. Camazine, J. Sneyd, Collective decision-making in honey bees: How colonies

choose among nectar sources, Behavioral Ecology and Sociobiology 28 (1991) 277–290. doi:10.1007/BF00175101.

# Extended Abstract: A Novel Mobile App for the Next Generation of Beekeepers

Eugen Puzynin[1], Heinrich Mellmann[1] and Verena V. Hafner[1]

[1]*Humboldt University of Berlin, Unter den Linden 6, Berlin, 10117, Germany*

**Abstract**

In this extended abstract, the authors report the ongoing work on a new mobile app for beekeepers. It is very important for beekeepers - especially for those who are in search of new locations for their beehives - to know the current situation in the immediate vicinity. This work presents a mobile application suitable for beekeepers to view weather and air quality data and, in particular, what people in the nearby area have annotated.

**Keywords**

Beekeeping, Mobile App, Prototype, HIVEOPOLIS

## 1. Introduction

Beekeepers pollinate many crop plants with their bees. Most important crops depend to some degree of pollination from the most important species for crop pollination: the western honey bee. Honey production is also an important source of income in many rural communities [1]. Often forgotten are the wild bees, who are responsible for pollinating garden plants and wildflowers. Honey bees are linked to the spread of diseases to wild pollinators via shared flowers. They can also out-compete with native pollinators for resources and food [2]. Other modern challenges for bees are pesticides, parasites, climate change and a lack of flowers [3]. According to the World Health Organization, an estimated seven million people die each year from air pollution [4]. Since bees rely on their sense of smell to identify different flowers, air pollution is affecting them too. Air pollution masks the scent molecules from plants. This makes bees forage longer leading them to become ineffective pollinators because of the decreased reproductive output and the amount of pollen flow in flowering plants [5]. Another study observed a reduction in pollinator survival as well as significant molecular and physiological changes [6]. In order to fight these challenges, the authors created a prototype in the form of a mobile app. Beekeepers should monitor climate and air conditions to keep bees and themselves healthy. Managed honey bee hives should not be placed in protected areas, where a risk would exist of wild bees being driven away or attacked. More plants are required to address the plant shortage and help wild insects. The map-based app uses current technologies, with which non-beekeepers can help beekeepers by marking spots on the map. The whole utilization

*CS&P 2021: 29th International Workshop on Concurrency, Specification and Programming*

puzynine@informatik.hu-berlin.de (E. Puzynin); mellmann@informatik.hu-berlin.de (H. Mellmann); hafner@informatik.hu-berlin.de (V. V. Hafner)

process is simplified for all users with features such as identifying plant species via photos and showing which plants are suitable for bees.

The development of a new app is part of the HIVEOPOLIS project [7, 8]. HIVEOPOLIS intends to provide honeybee colonies with technology (internet, databases, satellite data and robots) that would otherwise be unavailable to them.

## 2. Methodology

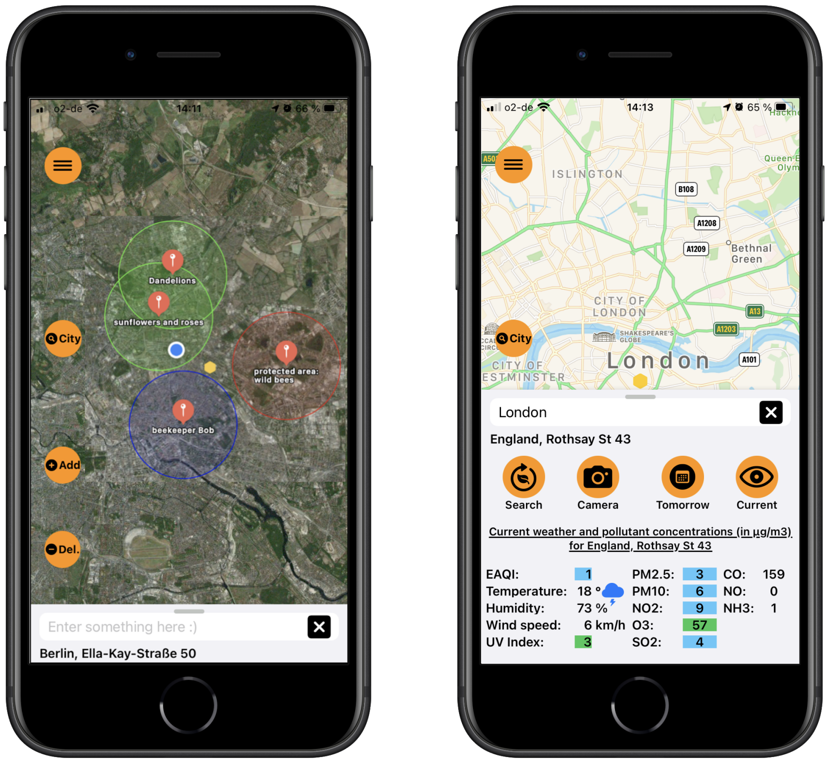
The goals of developing the app consists of three main components:

1. An interactive augmented map shows key elements of weather and air quality for beekeepers.
2. Users must be able to make inputs and help beekeepers with it.
3. It should be simple and swiftly to use.

The application was developed as a native iOS app in the programming language Swift with the integrated development environment Xcode. One reason mobile app development was preferred is access to the camera for various functions, such as photographing plants for plant identification. Furthermore, GPS data is required in case a user of the app wants to get directions to the next destination of his hives. Apple's MapKit was used natively for the app's interactive map. This allows users to see the map as a satellite view in 2D or 3D. OpenWeather provides with its API the current and forecast weather as well as air pollution data for all coordinates. The air pollution values are currently based on the European Air Quality Index. A pretrained convolutional neural network was used for the flower recognition function: Oxford's 102 Flower Dataset with 102 categories, each 40 - 200 images, allows to take a picture of any flower and it tries to recognise the flower's name [9]. Users can have their own account with email and password, which are stored and encrypted on Google's Firestore servers. The cloud database is instantly accessible, which means saved data is instantly available for other users. For example, registered users are able to send chat messages and save annotations. An API from Wikipedia was used to retrieve flower data, allowing users to look up information about any flower.

## 3. Results and Discussion

A preliminary user evaluation had been created as an online survey. A total of 26 people participated in it, 15 were beekeepers and 11 were non-beekeepers. They assisted in revising the design and generated ideas for new functionalities. After this evaluation an alpha test was made. Sergey Petrov from the company Pollenity asked 22 Bulgarian beekeepers to test the app on an iPhone and give feedback. Each member of the group spent between 1 and 10 minutes testing the app. The methodology of the test included a short verbal presentation and a video tutorial on how to use the app. In the following pictures, you can see the app. Figure 1 depicts an impression of the app. (Left) shows markers on the map that have theoretically been set by users. Potential beekeeping locations are depicted here, along with plants of interest to bees. For example, a beekeeper can see where other beekeepers are in the vicinity, if they have marked a chosen location as already been occupied by the beekeeper, and where beekeepers should pay attention to the protection of wild bees. (Right) shows the current weather with temperature, humidity, air speed, UV index, as well as the current air quality values.



Current weather and pollutant concentrations (in µg/m3) for England, Rothsay St 43

| | | | | | |
|---|---|---|---|---|---|
| EAQI: | 1 | PM2.5: | 3 | CO: | 159 |
| Temperature: | 18 ° | PM10: | 6 | NO: | 0 |
| Humidity: | 73 % | NO2: | 9 | NH3: | 1 |
| Wind speed: | 6 km/h | O3: | 57 | | |
| UV Index: | 3 | SO2: | 4 | | |

**Figure 1:** View of the application. Left: overview with annotations. Buttons: hamburger menu with more functions; set focus to a specific city; adding and deleting annotations. Right: Slide up menu with current weather and pollution data. Buttons here are the search button for displaying information of a specific plant; camera for shooting pictures; tomorrow for future weather and air quality; current for weather and air quality at the moment.

## 4. Conclusion

The preliminary user evaluation and the alpha test confirmed that such an app could be of interest to beekeepers and environmentalists. The next steps would be to define some corrections to the design and then test it again as a beta version to get further constructive feedback. Further evaluation is needed to collect more meaningful reactions and constructive criticism from beekeepers to be able to improve the app. Current ideas for improvements are: The most important point is the implementation of Android. Another important point is localization, so that the app works all over the world without language barriers. The authors are also considering how gamification can be integrated to motivate users and bind them to the app. Furthermore, there are still a lot of functions that could be implemented. For example, the flowering times of the respective plants, so that beekeepers know when the flowers are usable for them. And uploadable images so that users can see more of what the location looks like. After some testing and improvements, the app could be made available to the public.

## Acknowledgments

This work was supported by the EU H2020 program under grant agreement No. 824069 (HIVEOPOLIS).

## References

[1] Ashley N. Mortensen at el. European honey bee apis mellifera linnaeus and subspecies (insecta: Hymenoptera: Apidae). UF/IFAS Extension, EENY568, August 2013

[2] Jonas Geldmann and Juan P. González-Varo. Conserving honey bees does not help wildlife. Science, 359(6374):392–393, January 2018

[3] Dave Goulson at el. Bee declines driven by combined stress from parasites, pesticides, and lack of flowers. Science, 347(6229), March 2015

[4] Air pollution in the Western Pacific https://www.who.int/westernpacific/health-topics/air-pollution Last access 1 July 2021

[5] Quinn S. McFrederick et al. Air pollution modifies floral scent trails. Atmospheric Environment, 42(10):2336–2348, March 2008

[6] Geetha G. Thimmegowda et al. A field-based quantitative analysis of sublethal effects of air pollution on pollinators. Proceedings of the National Academy of Sciences, 117(34), August 2020

[7] Futuristic Beehives For A Smart Metropolis https://cordis.europa.eu/project/id/824069 Last access 1 July 2021

[8] Asya Ilgün et al. Bio-Hybrid Systems for Ecosystem Level Effects in Proceedings of the Artificial Life Conference 2021 (ALIFE 2021) (to appear)

[9] Nilsback, M-E. and Zisserman, A. Automated flower classification over a large number of classes Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing (2008) https://www.robots.ox.ac.uk/~vgg/data/flowers/102/

# Efficient Machine Learning Methods over Pairwise Space (keynote)

Hung Son Nguyen

*University of Warsaw*

**Keywords**
Rough sets, Support Vector Machine, Factorization Machine, Distance Metric Learning, Context-Aware Recommendation

## Extended Abstract

In recent years many machine learning concepts and methods were developed on the set of pairs of objects. In this paper, the set of all pairs of objects is called *the pairwise space*. Let us notice that if the set of objects $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ consists of $n$ instances, then the pairwise space contains $O(n^2)$ pairs. Thus why the straightforward implementations of those methods are not applicable for big data sets with millions of objects.

The main concepts in rough set theory (RS) such as reducts, lower and upper approximations, decision rules or discretizations have been defined in term of the discernibility matrix, which is a form of the pairwise space [1]. For example, in minimal decision reduct problem, we are looking for the minimal subset of features that preserves the discernibility between objects from different decision classes [2].

Support Vector Machine (SVM) is also a classification method described as an optimization problem over the pairwise space [3]. The initial idea of looking for the linear classifier with the maximal margin were transformed into the problem of looking for a set of coefficients $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_n)$ related to objects that maximizes an objective function

$$W(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j K(x_i, x_j).$$

defined on the set of dot products of all pairs of objects. In the above formula $y_i$ denotes the decision class of the object $\mathbf{x}_i$ and $K$ is a kernel function chosen by the user.

Distance Metric Learning (DML) [4] is a machine learning discipline that looks for the best distance function (also divergence or similarity ) from certain available information about similarity measures between different pairs or triplets of data. These similarities are determined

*29th International Workshop on Concurrency, Specification and Programming (CS&P'21)*
son@mimuw.edu.pl (H. S. Nguyen)
0000-0002-3236-5456 (H. S. Nguyen)

by the sets

$$\begin{aligned}
S &= \{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{X} \times \mathcal{X} : \ \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are similar.}\} \\
D &= \{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{X} \times \mathcal{X} : \ \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are not similar.}\} \\
R &= \{(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_l) \in \mathcal{X} \times \mathcal{X} \times \mathcal{X} : \ \mathbf{x}_i \text{ is more similar to } \mathbf{x}_j \text{ than to } \mathbf{x}_l.\}
\end{aligned}$$

With these data and similarity constraints, the problem to is to look for those distance functions (belonging to a predefined family of distances $\mathcal{D}$) that minimize a certain loss function $\ell$ determined on the base of the sets $S$, $D$ and $R$. In other words, the objective of DML is to solve the optimization problem

$$\min_{d \in \mathcal{D}} \ell(d, S, D, R)$$

In recent years, many efficient implementations for the mentioned above disciplines have been proposed and developed. Most of them are based either on the approximation idea [5], [6] or deep learning [7].

Context aware recommendation systems is another machine learning concept that were defined on the pairwise space which was in fact defined as a regression problem over the set of transaction pairs [8].

In this talk we compare different techniques for the mentioned above machine learning concepts and we will pay an attention on application of factorization machine (FM). This method has been successfully applied for context aware recommendation systems [9]. The main idea is to transform the optimization problem established on the pairwise space into an equivalent problem where the time complexity for each iteration has been reduced from quadratic time into linear time [10]. We will show that factorization machine can be also applied for some problems in rough set theory, SVM or distance metric learning.

## References

[1] Z. Pawlak, Rough sets, International Journal of Information and Computer Sciences 11 (1982) 341–356.

[2] Z. Pawlak, A. Skowron, Rudiments of rough sets, Information Sciences 177 (2007) 3–27.

[3] C. Cortes, V. Vapnik, Support vector networks, Machine Learning 20 (1995) 273–297.

[4] J.-L. Suárez, S. García, F. Herrera, A tutorial on distance metric learning: Mathematical foundations, algorithms, experimental analysis, prospects and challenges, Neurocomputing 425 (2021) 300–322.

[5] H. S. Nguyen, Approximate Boolean Reasoning: Foundations and Applications in Data Mining, Springer-Verlag, Berlin, Heidelberg, 2006, p. 334–506.

[6] T. Joachims, Learning to Classify Text Using Support Vector Machines – Methods, Theory, and Algorithms, Kluwer/Springer, 2002.

[7] P. H. Barros, F. Queiroz, F. Figueredo, J. A. dos Santos, H. S. Ramos, A new similarity space tailored for supervised deep metric learning, CoRR abs/2011.08325 (2020). URL: https://arxiv.org/abs/2011.08325. arXiv:2011.08325.

[8] S. Rendle, Z. Gantner, C. Freudenthaler, L. Schmidt-Thieme, Fast context-aware recommendations with factorization machines, in: Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 635–644. URL: https://doi.org/10.1145/2009916.2010002. doi:10.1145/2009916.2010002.

[9] X. Xin, B. Chen, X. He, D. Wang, Y. Ding, J. Jose, Cfm: Convolutional factorization machines for context-aware recommendation, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, International Joint Conferences on Artificial Intelligence Organization, 2019, pp. 3926–3932. URL: https://doi.org/10.24963/ijcai.2019/545. doi:10.24963/ijcai.2019/545.

[10] S. Rendle, Factorization machines with libfm, ACM Trans. Intell. Syst. Technol. 3 (2012) 57:1–57:22. URL: http://doi.acm.org/10.1145/2168752.2168771. doi:10.1145/2168752.2168771.

# Influence of Data Dimension Reduction, Feature Scaling and Activation Function on Machine Learning Performance

Grzegorz Słowiński

*University of Technology and Economics, ul. Jagiellońska 82f, 03-301 Warsaw, Poland*

**Abstract**

A dataset containing over 13k samples of dry beans geometric features is being analysed using machine learning (ML) and deep learning (DL) techniques with the goal to automatically classify the bean specie. The obtained geometrical data has quite a lot redundancy. Many features are strongly correlated. This work analyses the influence of data dimension reduction (DDR) (elimination of excess strongly correlated features) and features scaling (FS), often called normalization, on the machine learning performance (measured in terms of accuracy and approximate training time). Additionally also an influence of activation function (sigmoid vs. ReLU) on artificial neural network performance has been checked.

**Keywords**

machine learning, deep learning, data dimension reduction, features scaling, activation function

## 1. Introduction

Classification of dry beans is of some economic importance. Manual classification is labour intensive, etc. Over 13 k samples of dry beans of 7 various species were photographed and their geometry was measured via computer vision techniques in [1]. Then the set was analysed via several machine learning (or data science) and deep learning (or artificial neural network) techniques. The overall accuracy obtained was 87.92-93.13%, depending on technique used.

The dataset used in [1] has been published in the UCI machine learning repository [2]. In this work, a collection of beans was used as material for investigation how machine learning process is influenced by the following factors: 1) data dimension reduction, 2) features scaling (or data normalization) and 3) in case of neural networks, how their performance depends on activation function used (ReLU vs. sigmoid).

The research question examined in this work is: How do data dimension reduction, feature scaling and activation function influence machine learning performance? The above question is related to concurency, specification and programming in the following way. Among topics of CS&P 2021 one can find: Model checking and testing - this work checks different ML models, knowledge discovery and data mining - machine learning belong to this field, soft computing - artificial neural networks are are categorized as a kind of soft-computing.

## 1.1. Data Dimension Reduction

[1]29th International Workshop on Concurrency, Specification and Programming (CS&P'21)
EMAIL: grzegorz.slowinski@uth.edu.pl
ORCID: 0000-0001-9770-5063

In the work [1] data dimension has not been reduced, although many features are strongly correlated. This work investigates the effect of data dimension reduction on performance (computing time and accuracy).

## 1.2. Feature scaling

In the handbook [4], page 72 Aurelien Geron, states: "One of the most important transformations you need to apply to your data is feature scaling. With few exceptions, Machine Learning algorithms don’t perform well when the input numerical attributes have very different scales." This work verifies this statement and investigates what ML methods really needs feature scaling.

## 1.3. Activation Function

In work [1] ANN with sigmoid activation in hidden layers has been applied. This work investigates how ANN performance depends on activation function used. Two activation function are compared: ReLU and sigmoid.

## 2. Tools

The entire analysis was done using Python and its ML frameworks: numpy, pandas, matplotlib, seaborn, scikit-learn and keras. Google Colab a free cloud version of jupyter notebook was used. The reader can find the Python scripts under link [3]. Parameters of compute engine used were: Intel(R) Xeon(R) CPU @ 2.30GHz, 12,69 GB RAM, no graphical processing unit (GPU) acceleration. Majority of experiments performed were shallow learning that do not need GPU support. As the dry beans dataset is relatively simple, the artificial neural network (ANN) applied was also rather simple and GPU support was not crucial for ANN training. Training times were in range from milliseconds to a few minutes.

## 3. Data

The dataset under study consists of 13611 samples. A sample amounts to 16 geometrical features and a label identifying the specie of the bean. The species are as follows: Barbunya, Bombay, Cali, Dermason, Horoz, Seker, and Sira. The features are: Area, Perimeter, MajorAxisLength, MinorAxisLength, AspectRatio, Eccentricity, ConvexArea, EquivDiameter, Extent, Solidity, Roundness, Compactness, ShapeFactor1, ShapeFactor2, ShapeFactor3, and ShapeFactor4. A detailed explanation how the features were calculated is presented in [1].

**Table 1.**
Correlation between beans features

| | Area | Peri-meter | Major Axis Length | Minor Axis Length | Aspect Ratio | Eccentri-city | Convex Area | Equiv Diameter | Extent | Solidity | Round-ness | Compact-ness | Shape Factor 1 | Shape Factor 2 | Shape Factor 3 | Shape Factor 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Area | 1.000 | 0.967 | 0.932 | 0.952 | 0.242 | 0.267 | 1.000 | 0.985 | 0.054 | -0.197 | -0.358 | -0.268 | -0.848 | -0.639 | -0.272 | -0.356 |
| Perimeter | 0.967 | 1.000 | 0.977 | 0.913 | 0.385 | 0.391 | 0.968 | 0.991 | -0.021 | -0.304 | -0.548 | -0.407 | -0.865 | -0.768 | -0.408 | -0.429 |
| MajorAxisLength | 0.932 | 0.977 | 1.000 | 0.826 | 0.550 | 0.542 | 0.933 | 0.962 | -0.078 | -0.284 | -0.596 | -0.568 | -0.774 | -0.859 | -0.568 | -0.483 |
| MinorAxisLength | 0.952 | 0.913 | 0.826 | 1.000 | -0.009 | 0.020 | 0.951 | 0.949 | 0.146 | -0.156 | -0.210 | -0.015 | -0.947 | -0.471 | -0.019 | -0.264 |
| AspectRatio | 0.242 | 0.385 | 0.550 | -0.009 | 1.000 | 0.924 | 0.243 | 0.304 | -0.370 | -0.268 | -0.767 | -0.988 | 0.025 | -0.838 | -0.979 | -0.449 |
| Eccentricity | 0.267 | 0.391 | 0.542 | 0.020 | 0.924 | 1.000 | 0.269 | 0.319 | -0.319 | -0.298 | -0.722 | -0.970 | 0.020 | -0.860 | -0.981 | -0.449 |
| ConvexArea | 1.000 | 0.968 | 0.933 | 0.951 | 0.243 | 0.269 | 1.000 | 0.985 | 0.053 | -0.206 | -0.362 | -0.270 | -0.848 | -0.641 | -0.274 | -0.362 |
| EquivDiameter | 0.985 | 0.991 | 0.962 | 0.949 | 0.304 | 0.319 | 0.985 | 1.000 | 0.028 | -0.232 | -0.436 | -0.328 | -0.893 | -0.713 | -0.330 | -0.393 |
| Extent | 0.054 | -0.021 | -0.078 | 0.146 | -0.370 | -0.319 | 0.053 | 0.028 | 1.000 | 0.191 | 0.344 | 0.354 | -0.142 | 0.238 | 0.348 | 0.149 |
| Solidity | -0.197 | -0.304 | -0.284 | -0.156 | -0.268 | -0.298 | -0.206 | -0.232 | 0.191 | 1.000 | 0.607 | 0.304 | 0.153 | 0.344 | 0.308 | 0.702 |
| Roundness | -0.358 | -0.548 | -0.596 | -0.210 | -0.767 | -0.722 | -0.362 | -0.436 | 0.344 | 0.607 | 1.000 | 0.768 | 0.230 | 0.783 | 0.763 | 0.472 |
| Compactness | -0.268 | -0.407 | -0.568 | -0.015 | -0.988 | -0.970 | -0.270 | -0.328 | 0.354 | 0.304 | 0.768 | 1.000 | -0.009 | 0.869 | 0.999 | 0.484 |
| ShapeFactor1 | -0.848 | -0.865 | -0.774 | -0.947 | 0.025 | 0.020 | -0.848 | -0.893 | -0.142 | 0.153 | 0.230 | -0.009 | 1.000 | 0.469 | -0.008 | 0.249 |
| ShapeFactor2 | -0.639 | -0.768 | -0.859 | -0.471 | -0.838 | -0.860 | -0.641 | -0.713 | 0.238 | 0.344 | 0.783 | 0.869 | 0.469 | 1.000 | 0.873 | 0.530 |
| ShapeFactor3 | -0.272 | -0.408 | -0.568 | -0.019 | -0.979 | -0.981 | -0.274 | -0.330 | 0.348 | 0.308 | 0.763 | 0.999 | -0.008 | 0.873 | 1.000 | 0.484 |
| ShapeFactor4 | -0.356 | -0.429 | -0.483 | -0.264 | -0.449 | -0.449 | -0.362 | -0.393 | 0.149 | 0.702 | 0.472 | 0.484 | 0.249 | 0.530 | 0.484 | 1.000 |

Correlation analysis (see table 1) has shown that several of the features are strongly (positively or negatively) correlated. This is due to the fact that basically all of them are kind of geometric measures. In the original work [1] the issue of strong correlation between features has not been addressed. Generally strongly (over 0,9) features bring little extra information, so its elimination should reduce computational complexity (speed up training) with little if any loss in classification accuracy.

It is also sometimes suggested that feature scaling (often called normalization) can improve performance [4], pages 72-73. This is also investigated. To give a brief visualisation of beans dataset, the pair-plot with selected features (less correlated) has been done, see figure 1.

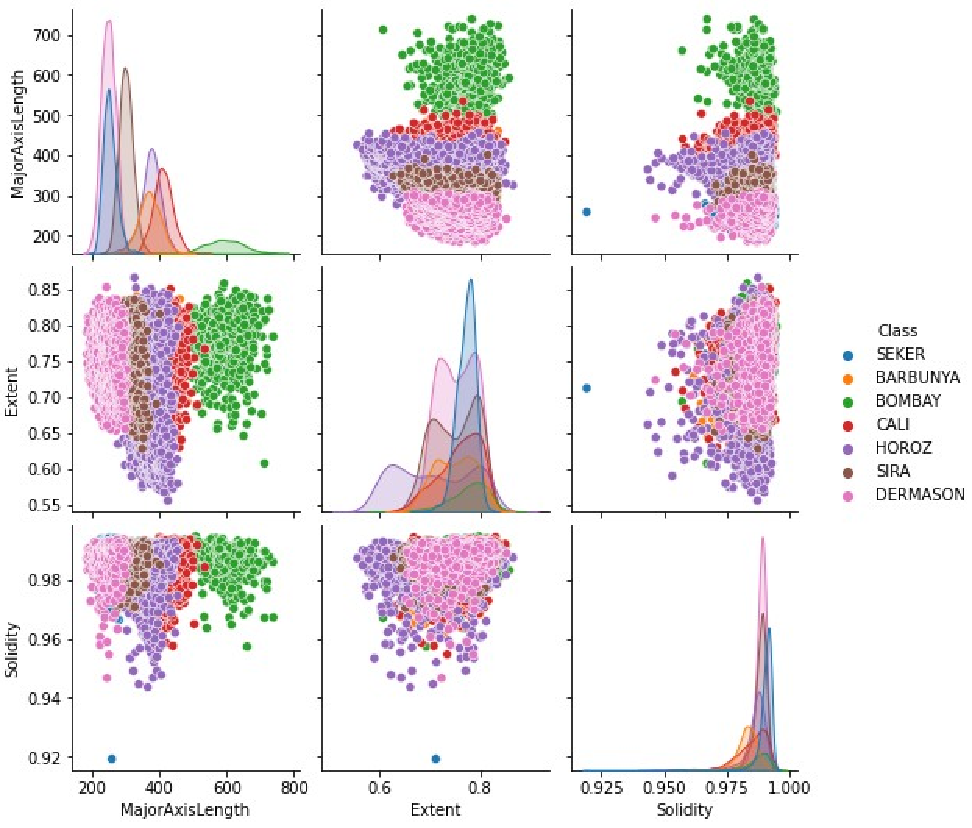


**Figure 1:** Pair-plot of selected (low corelated) bean features.

## 4. Shallow learning results

The methods tried were: Naive Bayes Classifier, Decision Tree, Random Forest, Support Vector Classifier.

### 4.1. Naive Bayes Classifier

Results for Gaussian naive Bayes classifier are shown in table 2. One can see that DDR or FS has small effect on training time. Using DDR or FS (or both) significantly increased accuracy from 77.23% to 89.83-91.00%.

**Table 2.**
Gaussian naive Bayes classifier performance

| Data | Accuracy | Approx. training time |
|---|---|---|
| Full, not scaled | 77.23% | 18.2 ms |
| Dimension reduced, not scaled | 91.00% | 16.3 ms |
| Full, scaled | 89.83% | 17.0 ms |
| Dimension reduced, scaled | 90.78% | 15.8 ms |

## 4.2. Decision tree

Results for decision tree are shown in table 3. Decision tree applied was limited to 16 leaf nodes and maximum depth of 5. One can see that FS has no effect on accuracy and little effect on training time. This probably connected with the fact that DT analyses one feature at the time, so it not cares what is the ratio of specific feature range to other features. DDR shorten training time with limited accuracy decrease.

**Table 3.**
Decision tree classifier performance

| Data | Accuracy | Approx. training time |
|---|---|---|
| Full, not scaled | 88.87% | 128 ms |
| Dimension reduced, not scaled | 88.24% | 71 ms |
| Full, scaled | 88.87% | 129 ms |
| Dimension reduced, scaled | 88.24% | 70 ms |

Decision tree is known to be sensitive for data “rotation”, see [4] p 188. DT analyses only one feature at the time. Strongly correlated features gives little extra information, but can present information in a slightly different manner, suitable for decision tree.

## 4.3. Random Forest Classifier

Results for the random forest (RF) are shown in table 4. The random forest consisted of 150 trees. No limits (max leaves, max depth and etc.) were put on trees. One can observe that training times are longer that for single decision tree (which is reasonable as here we have a set of decision trees). The accuracies are high. DDR shortened training time and allowed for slightly higher accuracy (0,14-0,18 % point). This is quite interesting that although DDR slightly reduced accuracy on single tree it improved accuracy on RF. Similarly to decision tree, SF practically has little effect on training time.

**Table 4.**
Random forest classifier performance

| Data | Accuracy | Approx. training time |
|---|---|---|
| Full, not scaled | 93.06% | 4.69 s |
| Dimension reduced, not scaled | 93.24% | 2.69 s |
| Full, scaled | 93.10% | 4.79 s |
| Dimension reduced, scaled | 93.24% | 2.59 s |

## 4.4. Support Vector Classifier

Results for support vector classifier (SVC) is shown in table 5. Polynomial kernel has been used. Generally SVC is much more “heavier” model than gaussian classifier, decision tree or random forest. Training times much longer. One can see that DDR or FS has small effect on SVC accuracy. DDR on not scaled features reduced training time. Feature scaling significantly increased training time and increased accuracy a little (about 1% point). The longest training time was observed for DDR and SF data. The training time was 9 times longer than for DDR and not SF data. The author cannot explained this effect.

**Table 5.**
Support vector classifier performance

| Data | Accuracy | Approx. training time |
|---|---|---|
| Full, not scaled | 91.81% | 42 s |
| Dimension reduced, not scaled | 91.81% | 29 s |
| Full, scaled | 93.24% | 88 s |
| Dimension reduced, scaled | 92.95% | 266 s |

## 5. Artificial neural network

For an artificial neural network (ANN) the data needs additional treatment. First, the names of bean species were labelled with numbers and then these numbers 0-6 were codded as so called ”one-hot”. The reason of using ”one-hot” encoding is well explained for example in [5] p. 376 or [6] pp. 190-194.

Three experiments has been performed to analyse: 1) influence of data dimension reduction, 2) influence of features scaling and 3) influence of activation function (sigmoid vs. ReLU). The ANN architecture was kept similar (as much as possible) to described in [1]. All ANNs had 3 hidden layers with 17, 12, 3, neurons respectively. However here ReLU function has be used as “default” option. Output layer consisted of 7 neurons with softmax activation function – one for each class. Generally training lasted for 16 epochs. However, as it was obvious that ANN with sigmoid activation is undertrained, this net was trained for 48 epochs. The training process is presented in figure 2. The performance summary is presented in table 6.

**Table 6.**
ANNs performance, 17-12-3 architecture, Adam optimiser, 16 epochs

| Data | Activation function in hidden layers | Epochs of training | Approx. training time | Accuracy |
|---|---|---|---|---|
| 16 features, scaled | ReLU | 16 | 14 s | 92.66% |
| 8 features, scaled | ReLU | 16 | 8 s | 93.24% |
| 8 features, not scaled | ReLU | 16 | 9 s | 26.74% |
| 8 features, scaled | sigmoid | 48 | 41 s | 88.14% |

It can be visible that:

1. feature scaling (or data normalisation) is very important for ANN’s. An attempt to train without prior data scaling failed. Only 55,82% accuracy has been obtained. Perhaps bigger network can manage this issue by rescaling data in a few first layers, but it will influence training time and accuracy.
2. ReLU works significantly better than sigmoid function as an activation function. ReLU network trains faster and reaches better accuracy.
3. Data dimension reduction shortens training time nearly by half and increases accuracy by about 0,58 % point.

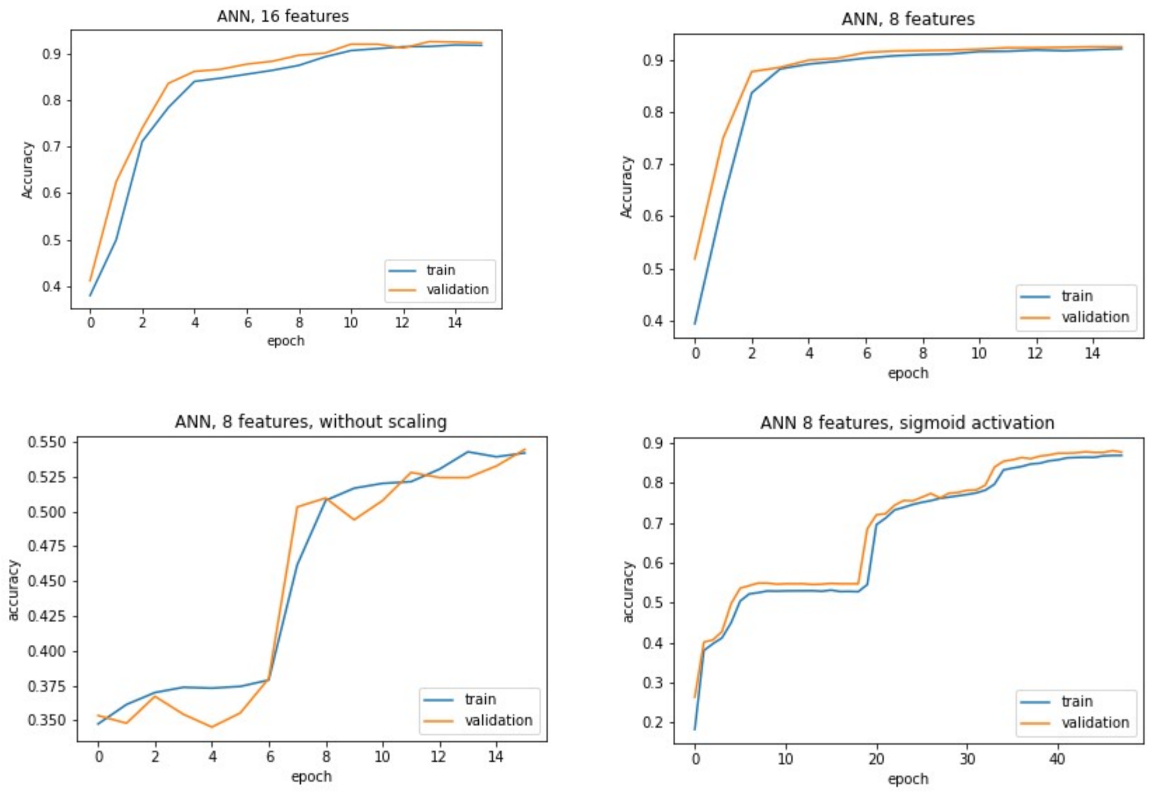


**Figure 2:** Training of different ANNs

## 6. Conclusions

Influence of data dimension reduction, data scaling (or normalisation) and activation function has been investigated. The influence depends on machine learning technique used.

Generally data dimension reduction reduces training time with rather limited influence on accuracy Data scaling is a must in case of artificial neural network. Omitting data scaling decreased accuracy from about 93% to about 56%. In case of shallow learning techniques its influence is smaller, it sometimes help a little with accuracy, sometimes not.

Generally scaling had no effect on decision tree and random forest performance. In case of support vector classifier scaling resulted in huge training time increase. Author cannot explain this effect.

The highest accuracy observed was 93,24%. It was obtained 3 times with: 1) random forest with 8 features (scaled and not scaled), 2) ANN, 8 features, scaled and 3) SVC, 16 features, scaled. It is quite intriguing that exactly the same, maximum result repeated 3 time.

## 7. References

[1] Murat Koklu, Ilker Ali Ozkan, Multiclass classification of dry beans using computer vision and machine learning techniques, Computers and Electronics in Agriculture 174 (2020) 105507

[2] Dry beans dataset at UCI repository: https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset, access 23.06.2021

[3] Colab notebook containing computation scripts for this work: https://colab.research.google.com/drive/1l5lH1QgesDX8CbbkqcnmlbqwcXfksGQB?usp=sharing

[4] Aurelien Geron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, O'Reilly, 2019

[5] Jake VanderPlass, Python Data Science Handbook, O'Reilly, 2017

[6] Francois Chollet, Deep Learning with Python, Manning Publications, 2018

# Sorting by Decision Trees with Hypotheses (extended abstract)

Mohammad Azad[1], Igor Chikalov[2], Shahid Hussain[3] and Mikhail Moshkov[4]

[1]*Jouf University, Sakaka 72441, Saudi Arabia*
[2]*Intel Corporation, 5000 W Chandler Blvd, Chandler, AZ 85226, USA*
[3]*Institute of Business Administration, University Road, Karachi 75270, Pakistan*
[4]*King Abdullah University of Science and Technology (KAUST), Thuwal 23955-6900, Saudi Arabia*

**Abstract**
In this paper, we consider decision trees that use both queries based on one attribute each and queries based on hypotheses about values of all attributes. Such decision trees are similar to ones studied in exact learning, where not only membership but also equivalence queries are allowed. For $n = 3, \dots, 6$, we compare decision trees based on various combinations of attributes and hypotheses for sorting $n$ pairwise different elements from linearly ordered set.

**Keywords**
decision tree, hypothesis, dynamic programming, sorting

## 1. Introduction

Decision trees are widely used in many areas of computer science, for example, test theory (initiated by Chegis and Yablonskii [1]), rough set theory (initiated by Pawlak [2, 3, 4]), and exact learning (initiated by Angluin [5, 6]). These theories are closely related: attributes from rough set theory and test theory correspond to membership queries from exact learning. Exact learning also studies equivalence queries. The notion of "minimally adequate teacher" using both membership and equivalence queries was discussed by Angluin in [7]. Relations between exact learning and PAC learning proposed by Valiant [8] were considered in [5].

In [9, 10, 11], we added the notion of a hypothesis (an analog of equivalence queries) to the model considered in both rough set theory and test theory and proposed dynamic programming algorithms for the optimization of the decision trees with hypotheses. Note that the dynamic programming algorithms for the optimization of the conventional decision trees that do not use hypotheses were proposed earlier [12].

In the present paper, we consider an application of the dynamic programming algorithms from [9, 10, 11] to the study of the problem of sorting. We compare the complexity of five types of optimal (relative to the depth and relative to the number of realizable nodes) decision trees

*29th International Workshop on Concurrency, Specification and Programming (CS&P'21)*
mmazad@ju.edu.sa (M. Azad); igor.chikalov@gmail.com (I. Chikalov); shahidhussain@iba.edu.pk (S. Hussain); mikhail.moshkov@kaust.edu.sa (M. Moshkov)
0000-0001-9851-1420 (M. Azad); 0000-0002-1010-6605 (I. Chikalov); 0000-0002-1698-2809 (S. Hussain); 0000-0003-0085-9483 (M. Moshkov)

based on various combinations of attributes and hypotheses for sorting $n$ pairwise different elements from linearly ordered set, $n = 3, \ldots, 6$. Results obtained for the conventional decision trees are known – see book [12]. Results obtained for the decision trees with hypotheses are completely new.

Note that in the present paper we follow [11] when discuss the notions related to the decision trees with hypotheses. Complete definitions of these notions can be found in the same paper.

## 2. Five Types of Decision Trees and Their Optimization

Let $T$ be a decision table with $n$ conditional attributes $f_1, \ldots, f_n$ that have values from the set $\omega = \{0, 1, 2, \ldots\}$. Rows of this table are pairwise different and each row is labeled with a decision. For a given row of $T$, we should recognize the decision attached to it. To this end, we will use decision trees based on two types of queries. We can ask about the value of a conditional attribute $f_i \in \{f_1, \ldots, f_n\}$ on the given row. As a result, we obtain an answer of the kind $f_i = \delta$, where $\delta$ is the number in the intersection of the given row and the column $f_i$. We can also ask if a hypothesis $\{f_1 = \delta_1, \ldots, f_n = \delta_n\}$ is true, where the numbers $\delta_1, \ldots, \delta_n$ belong to the columns $f_1, \ldots, f_n$, respectively. Either this hypothesis is confirmed or we obtain a counterexample of the kind $f_i = \sigma$, where $f_i \in \{f_1, \ldots, f_n\}$ and $\sigma$ is a number from the column $f_i$ that is different from $\delta_i$. We will say that this hypothesis is proper if $(\delta_1, \ldots, \delta_n)$ is a row of the table $T$.

We study the following five types of decision trees:

1. Decision trees based on attributes only.
2. Decision trees based on hypotheses only.
3. Decision trees based on both attributes and hypotheses.
4. Decision trees based on proper hypotheses only.
5. Decision trees based on both attributes and proper hypotheses.

As time complexity of a decision tree we consider its depth, which is equal to the maximum number of queries in a path from the root to a terminal node of the tree. We consider the number of realizable relative to $T$ nodes in a decision tree as its space complexity. A node is called realizable relative to $T$ if the computation in the tree will pass through this node for some row and some choice of counterexamples. We use the following notation:

- $h^{(k)}(T)$ denotes the minimum depth of a decision tree of the type $k$ for $T$, $k = 1, \ldots, 5$.
- $L^{(k)}(T)$ denotes the minimum number of nodes realizable relative to $T$ in a decision tree of the type $k$ for $T$, $k = 1, \ldots, 5$.

In [9] and [10], dynamic programming algorithms for the optimization of decision trees of all five types relative to the depth and the number of realizable nodes were proposed (see also journal extension [11] of these papers that considers additionally two cost functions: the number of realizable terminal nodes and the number of nonterminal nodes). Note that algorithms for the minimization of the depth and number of nodes for decision trees of the type 1 were considered in [12] for decision tables with one-valued decisions and in [13] for decision tables with many-valued decisions.

**Table 1**
Experimental results for the depth

| $n$ | $h^{(1)}(T_n)$ | $h^{(2)}(T_n)$ | $h^{(3)}(T_n)$ | $h^{(4)}(T_n)$ | $h^{(5)}(T_n)$ |
|---|---|---|---|---|---|
| 3 | 3 | 2 | 2 | 2 | 2 |
| 4 | 5 | 4 | 4 | 4 | 4 |
| 5 | 7 | 6 | 6 | 6 | 6 |
| 6 | 10 | 9 | 9 | 9 | 9 |

**Table 2**
Experimental results for the number of realizable nodes

| $n$ | $L^{(1)}(T_n)$ | $L^{(2)}(T_n)$ | $L^{(3)}(T_n)$ | $L^{(4)}(T_n)$ | $L^{(5)}(T_n)$ |
|---|---|---|---|---|---|
| 3 | 11 | 13 | 9 | 14 | 9 |
| 4 | 47 | 253 | 39 | 254 | 39 |
| 5 | 239 | 15,071 | 199 | 15,142 | 199 |
| 6 | 1,439 | 2,885,086 | 1,199 | 2,886,752 | 1,199 |

Dynamic programming optimization algorithms are applicable to medium-sized decision tables. These algorithms first construct a directed acyclic graph (DAG) whose nodes are some subtables of the original decision table given by conditions of the type “attribute = value”. Then they pass through all the nodes of the DAG, starting with the simplest subtables, and for each subtable they find the minimum value of the considered cost function.

In the present paper, we use algorithms proposed in [9, 10, 11] to study decision trees of all five types optimal relative to the depth and relative to the number of realizable nodes for the sorting problem. Results for decision trees of the type 1 were obtained earlier [12]. Results for decision trees of the types 2–5 are new.

## 3. Problem of Sorting

In this paper, we study the problem of sorting $n$ elements. Let $x_1, \ldots, x_n$ be pairwise different elements from a linearly ordered set. We should find a permutation $(p_1, \ldots, p_n)$ from the set $P_n$ of all permutations of the set $\{1, \ldots, n\}$ for which $x_{p_1} < \cdots < x_{p_n}$. To this end, we use attributes $x_i : x_j$ such that $i, j \in \{1, \ldots, n\}$, $i < j$, $x_i : x_j = 1$ if $x_i < x_j$, and $x_i : x_j = 0$ if $x_i > x_j$.

The problem of sorting $n$ elements can be represented as a decision table $T_n$ with $n(n-1)/2$ conditional attributes $x_i : x_j$, $i, j \in \{1, \ldots, n\}$, $i < j$, and $n!$ rows corresponding to permutations from $P_n$. For each permutation $(p_1, \ldots, p_n)$, the corresponding row of $T_n$ is labeled with this permutation as the decision. This row is filled with values of attributes $x_i : x_j$ such that $x_i : x_j = 1$ if and only if $i$ stays before $j$ in the tuple $(p_1, \ldots, p_n)$.

For $n = 3, \ldots, 6$ and $k = 1, \ldots, 5$, we find values of $h^{(k)}(T_n)$ and $L^{(k)}(T_n)$ using dynamic programming algorithms described in [9, 10, 11] – see results in Tables 1 and 2.

From the obtained experimental results it follows that the decision trees of the types 2–5 can have less depth than the decision trees of the type 1. Decision trees of the types 3 and 5 can have less number of realizable nodes than the decision trees of the type 1. Decision trees of the types 2 and 4 have too many nodes.

## 4. Conclusions

In this paper, we found the minimum depth and the minimum number of realizable nodes of five types of decision trees for sorting $n$ elements, $n = 3, \ldots, 6$.

In the future, we are planning to study joint behavior of the depth and the number of nodes in such decision trees. It would be also interesting to compare the complexity of optimal decision trees of the considered five types constructed by dynamic programming algorithms and the complexity of decision trees constructed by entropy-based greedy algorithm proposed in [14].

## Acknowledgments

Research reported in this publication was supported by King Abdullah University of Science and Technology (KAUST). The authors are indebted to the anonymous reviewers for interesting comments.

## References

[1] I. A. Chegis, S. V. Yablonskii, Logical methods of control of work of electric schemes, Trudy Mat. Inst. Steklov (in Russian) 51 (1958) 270–360.

[2] Z. Pawlak, Rough sets, Int. J. Parallel Program. 11 (1982) 341–356.

[3] Z. Pawlak, Rough Sets - Theoretical Aspects of Reasoning about Data, volume 9 of *Theory and Decision Library: Series D*, Kluwer, 1991.

[4] Z. Pawlak, A. Skowron, Rudiments of rough sets, Inf. Sci. 177 (2007) 3–27.

[5] D. Angluin, Queries and concept learning, Mach. Learn. 2 (1988) 319–342.

[6] D. Angluin, Queries revisited, Theor. Comput. Sci. 313 (2004) 175–194.

[7] D. Angluin, Learning regular sets from queries and counterexamples, Inf. Comput. 75 (1987) 87–106.

[8] L. G. Valiant, A theory of the learnable, Commun. ACM 27 (1984) 1134–1142.

[9] M. Azad, I. Chikalov, S. Hussain, M. Moshkov, Minimizing depth of decision trees with hypotheses (to appear), in: International Joint Conference on Rough Sets (IJCRS 2021), 19–24 September 2021, Bratislava, Slovakia, 2021.

[10] M. Azad, I. Chikalov, S. Hussain, M. Moshkov, Minimizing number of nodes in decision trees with hypotheses (to appear), in: 25th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2021), 8–10 September 2021, Szczecin, Poland, 2021.

[11] M. Azad, I. Chikalov, S. Hussain, M. Moshkov, Optimization of decision trees with hypotheses for knowledge representation, Electronics 10 (2021) 1580. URL: https://doi.org/10.3390/electronics10131580.

[12] H. AbouEisha, T. Amin, I. Chikalov, S. Hussain, M. Moshkov, Extensions of Dynamic Programming for Combinatorial Optimization and Data Mining, volume 146 of *Intelligent Systems Reference Library*, Springer, 2019.

[13] F. Alsolami, M. Azad, I. Chikalov, M. Moshkov, Decision and Inhibitory Trees and Rules for Decision Tables with Many-valued Decisions, volume 156 of *Intelligent Systems Reference Library*, Springer, 2020.

[14] M. Azad, I. Chikalov, S. Hussain, M. Moshkov, Entropy-based greedy algorithm for decision trees using hypotheses, Entropy 23 (2021) 808. URL: https://doi.org/10.3390/e23070808.

# On Reliable Wireless Streaming of Real-time Sensor Data

Agnieszka Boruta[1], Pawel Gburzynski[2] and Ewa Kuznicka[1]

[1]*Warsaw University of Live Sciences, ul. Nowoursynowska 166, 02-787 Warsaw, Poland*
[2]*Vistula University, ul. Stokłosy 3, 02-787 Warsaw, Poland*

**Abstract**
We discuss a practical problem related to wireless transmission of a continuous stream of readings from a sensor. The problem arose in the context of a contraption devised for monitoring the behavior patterns of working canines with the intention of spotting signs of their stress, exhaustion, or any indication of the animal's fatigue or discomfort that would call for the attention of its human companion. The device has been (is being) designed primarily as a vehicle for research in collaboration between the University of Live Sciences and Vistula University. The point we are trying to make is this paper is that tiny embedded systems aimed at specific applications within the realm of the so-called Internet of Things (IoT) come out best when built following a "holistic" approach. By "holistic" we mean taking into account, from the very bottom, the idiosyncratic aspect of the application instead of blindly relying on ready, layered, standardized, library solutions. In addition to reducing the footprint of the application, and enabling it to run in a cheaper and resource-frugal device, such an approach also translates into better performance. With the right selection of tools, it may in fact speed up the development process while resulting in a better quality of the product.

**Keywords**
wireless communication, remote sensing, wireless telemetry, streaming

## 1. The context

This paper deals with a subset of the technical aspects of a wider project aiming at researching simple and reliable automated methods for assessing the well-being of working dogs, including service dogs (military, police, disaster-response) as well as assistance dogs (guide, therapy). While the goals of our research probably need no long arguments in defense of their compassionate motivation, there are also solid remunerative reasons why an effective assessment of the animal's "quality" in providing its service matters. The dog training process is lengthy, complex, and expensive [1, 2] and good quality service/assistance dogs are extremely valuable [3, 4]. This stimulates studies along two lines: (1) to establish reliable criteria for early assessment of a dog's suitability for a particular kind of work/service [5, 6, 7, 8]; (2) to make sure that the animal is well taken care of and, in particular, any problems related to its work stress and generally health are quickly detected, diagnosed, and addressed [9, 10]. The latter issue can be reformulated as

*29th International Workshop on Concurrency, Specification and Programming (CS&P'21)*
agnieszka_boruta@sggw.edu.pl (A. Boruta); p.gburzynski@vistula.edu.pl (P. Gburzynski); ewa_kuznicka@sggw.edu.pl (E. Kuznicka)
0000-0002-3471-025X (A. Boruta); 0000-0002-1844-6110 (P. Gburzynski); 0000-0002-3760-1622 (E. Kuznicka)

that of an effective communication in the animal-to-human direction [11], as opposed to the more popular direction inherent in dog training.

While anomalies in a dog's behavior indicative of deficiencies in its well-being can be spotted by an expert human (veterinarian, behaviorist) through direct observation, our primary interest is in harnessing to this task sensing devices that, ideally, should be able to detect such problems automatically and signal them to the human supervisor. The issue can be viewed as falling under the more general domain of sensor-based diagnostics, e.g., similar to taking and interpreting ECG readings [12]. While it is obviously possible to subject the animal to extensive and authoritative veterinary assessments, including tests and sensor data collection interpreted by a human expert, we are interested in completely automated monitoring carried out by an inconspicuous and (basically) maintenance-free device being (basically) unnoticeable by the animal. Such a device, a wearable *Tag*, would be permanently carried by the dog, e.g., attached to a collar, and would convey wirelessly sensor data to a nearby access point for automated interpretation [13, 14].

The unobtrusiveness premise of the sensing/monitoring device trades off against the overt information content of the data that it can possibly collect. For example, it may seem worthwhile to try to obtain an EKG/ECG chart of the animal, which is a valuable source of information about the heart activity. One can expect such a chart to reasonably easily translate into a representation of the dog's tiredness or stress. While there exist experimental techniques for collecting this kind of data through "wearable" sensors [15], they overtax our premise by requiring direct access to the animal's skin. Even the less ambitious task of taking reliable heart rate without skin contact proves challenging [16].

Our long-term goal is to investigate how much one can accomplish with a completely unobtrusive device, requiring no skin contact and, preferably, no rigid attachment to the animal (in a specific position or place). The most natural sensor to try in this context is the Inertial Measurement Unit (IMU) being a combination of an accelerometer, a gyro, and a compass. Previous studies have reported various degrees of success in using the sensor (most notably the accelerometer component) for diagnosing various behavioral anomalies/problems in dogs [10, 17, 18]. We want to establish criteria for the classification of IMU data into simple signals indicative of some threshold levels of the animal's well-being in relation to its level of fatigue or stress that can be easily communicated to the human companion. The next step will be to built an actual practical device and application based on the outcome of our studies.

## 2. The setup

Our end goal being to fabricate a practically useful device, it makes sense to start with a view of the target application in mind. Ideally, we should use the same hardware for the experiments and for the final application. As the classification of the animal's activity patterns will be carried out based on the indications of an IMU, the embodiment of the sensor (the weight of the device and the mode of its attachment to the animal) is likely to matter because of its own (inherent) inertia component which will tend to influence the readings. This aspect of the project makes it similar to one of our earlier endeavors [19] where a series of research experiments carried out with an IMU-based sensor provided data to drive the design of a classification algorithm that could be subsequently implanted into the same device to a more practical end.

Having agreed that the experimental device should be identical to the target one, we still have to understand that the experimental version of the application (the software run by the device) is going to be drastically different from its target version. The role of the experiments is basically raw data collection. We want to amass a large amount of sensor readings, taken at the maximum rate that we can afford, from various representative animals acting under controllable conditions where experts can annotate the collected data with authoritative labels indicative of the dog's state. That data will be later used off-line to search for patterns that can guide the classification algorithms to be applied on the data collected by the target incarnation of the device. That part of the research methodology is beyond the scope of the present paper; we merely want to clarify the technical requirements for the experimental guise of the application.

The nature of the experiments, demanding that the animals act within environments appearing as close to their natural work conditions as possible, is an additional argument for the experimental devices being identical to the target ones, and it obviously precludes wired connectivity of the devices to external equipment. The footprint of the target device makes it impossible to store large volumes of data directly on it. Besides, the data must be annotated in real time, which makes it natural to use an external computer (a laptop or a tablet) for the actual collection and simultaneous annotation. Therefore, the sensor device is going to *stream* its readings wirelessly to the external computer. The streaming protocol is the aspect of the application discussed in the remainder of this paper.

The device used for data collection (and envisioned for the target application) is the CC1350 SensorTag[1] manufactured by Texas Instruments and featuring an ARM-based CC1350 microcontroller [20]. The SensorTag comes equipped with a number of sensors including the MPU9250 IMU by TDK InvenSense.[2] The complete device weights ca. 20 g (including a CR 2032 battery) and its dimensions are 44×32×6 mm. When attached to a dog's collar, it is not more obtrusive than a (slightly oversized) name tag.

The experimental setup consists of a pair of CC1350-based devices, one of them being the *Tag* worn by the dog, the other a CC1350 LaunchPad,[3] dubbed the *Peg*, connected over USB to the computer and acting as the RF access point (data sink) for the Tag. While the RF module of CC1350 can be configured to operate in Bluetooth mode, thus eliminating the need for a special access point, we opt for the so-called proprietary mode of the radio which gives us access to the raw channel. We prefer to circumvent the Bluetooth standard to: 1) increase the range of communication (to provide for a larger separation between the animal and the access point), 2) implement our private reliability scheme to increase the delivery fraction of the collected data. The proprietary mode operates within the 915 MHz ISM band offering parameterizable transmission power up to 14 dBm and (raw) transmission rates up to 500 kbps. Data exchanged over the RF channel is organized into packets with the maximum packet length (practically) limited to 60 bytes. The Bluetooth capability of the device will become handy in the target application where the device will be able to communicate directly with a smartphone. The essential classification will then be carried out in the Tag [19], so there will be no need for reliable streaming of large volumes of readings to the access point.

[1]https://dev.ti.com/cc1350stk

[2]https://invensense.tdk.com/products/motion-tracking/9-axis/mpu-9250/

[3]http://dev.ti.com/launchxl-CC1350

## 3. The problem

In its most general formulation, independent of the hardware and application context, the problem is that of implementing reliable communication across an unreliable channel. We deal with an asymmetric link where the Tag continuously sends a stream of data to the Peg. Ideally, we would like *all* the data generated by the Tag to (eventually) reach the Peg, in the proper order, such that the complete, timed, and annotated stream of sensor readings produced by the Tag is stored at the collection computer.

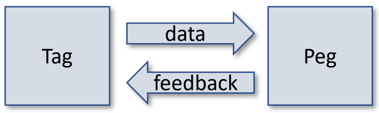


**Figure 1:** The channel model

Standard solutions to this problem involve a feedback channel operating in the Peg-to-Tag direction, as shown in Figure 1. The simplest of them has been known as the Alternating Bit Protocol [21, 22] and consists in explicitly acknowledging every single packet received by the Peg from the Tag. Various generalizations and improvements upon this simple scheme, including the one underlying TCP, are known under the name of ARQ protocols [23, 24]. Their primary objectives are: 1) to reduce the amount of feedback traffic, 2) to improve the continuity of the forward traffic when the losses are low and/or the bandwidth-delay product of the link is large [25].

The problem of *absolutely* reliable data transmission primarily concerns information whose utmost integrity is essential, i.e., the transmission of files. With streaming, the data is assumed to be created continuously and the problem of its reliable delivery receives a different flavor. In many cases the information is not stored at the source in the form of a complete file whose missing fragments could be requested at random by the recipient and retransmitted later. But even if this happens to be the case, the issue of its reliable delivery is conditioned by the real-time character of the reception where the data often become obsolete and useless if not delivered within a certain time window (like in streaming of voice and/or video). Consequently, while standard streaming applications may insist on high-quality of delivery, they typically are (and usually must be) prepared to deal with acceptable (innate) data loss [26, 27].

Our problem is specific in that: 1) the information collected by the Tag cannot be entirely stored at the source, so the availability of its undelivered fragments is restricted; 2) there is no issue of real-time playback at the recipient (unlike in streaming audio or video); 3) we can be prepared to deal with (acceptable) losses (and seemingly have to because of 1). The problem is reminiscent of one we dealt with before [28] (the hardware context was similar), and it may be worthwhile to emphasize the difference. In the case of [28] the entire collected data set was stored at the source Tag, and the problem was formulated as minimizing the time of its *complete* delivery to the Peg (the completeness of delivery was essential, so we were basically interested in reliable and fast file transfers). In the present case, we are inclined to deal with occasional data loss (the stream is infinite for all practical purposes) and the issue is to maximize the delivery fraction thus maximizing the feasible data collection rate.

If losses are unavoidable, one can often simplify the solution by eliminating the feedback channel altogether focusing instead on enhancing the reliability of communication in the physical layer, e.g., through forward error correction (FEC) techniques [29]. The proprietary mode of the RF module of CC1350 comes with a number of options that can be applied to this end. They effectively trade the transmission range and bit rate for reliability and amount to an important set of parameters that have to be tuned for best performance regardless of any other tools. However, depending solely on them would be too restrictive from our point of view. The dynamic nature of the propagation environment, including the variable distance between the Tag and the Peg, would lock the channel into a conservative setting, offering an acceptable (or passable) loss rate for the worst case scenario, while unnecessarily reducing the opportunities for collecting more data in a friendlier environment. Our hope is to achieve a much better flexibility with a properly designed feedback scheme: the bandwidth to be sacrificed for reliability will only be sacrificed when needed.

In a high-level discussion of ARQ schemes (and in many implementations of such schemes in the wired world) it is assumed that the feedback channel (Figure 1) is separate from the forward (data) channel. This is to say that the feedback messages do not disrupt the data stream and, in particular, they can be sent at any rate up to some maximum with their impact being solely positive. This is seldom true in wireless communication. Even if the two channels are in fact separate, which they mostly aren't, they will tend to interfere. Generally, setting aside a sizable portion of the RF bandwidth for a "frivolous" feedback channel makes that bandwidth unavailable for the *proper* use which is transmitting data. For example, in Bluetooth, e.g., within the framework of an ACL link [30], the essentially single channel must be partitioned (time-divided) into two parts to provide for two-way communication. Realizing that the feedback channel is going to directly coexist with the forward data channel, we would like to make it flexible and, in particular, avoid a rigid, time-division-based pre-allocation of bandwidth for the two channels. Our goal is to try to reduce the impact of the feedback channel on data bandwidth to the minimum needed by the application and demanded dynamically by the temperamental RF medium.

## 4. The solution

We propose a protocol for improving the reliability of conveying sensor data from the Tag to the Peg. The improvement will be evaluated in reference to the reliability achievable with purely hardware means, by assuming a unidirectional data channel (no feedback). The solution illustrates how application constraints can influence the design of low-level communication schemes stimulating a holistic approach to programming the application.

Owing to the lack of real-time requirements, the possibility of unrecoverable losses results solely from the finite buffers at the Tag. Whatever buffer space is available will be allocated to a shifting window of the collected data. The Peg will be able to request retransmission of those lost packets that are still present within the window.

The **Tag side** of the protocol is described by two threads: the generator of blocks of sensor readings (dubbed the generator thread) and the transmitter of those blocks on the RF channel. The blocks are stored in a singly-linked queue, denoted by $Q$ and depicted in Figure 2, whose

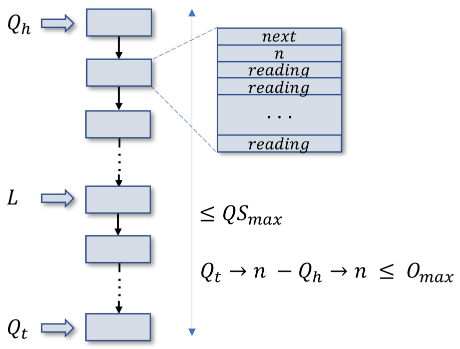


**Figure 2:** The block queue

size is limited. The queue is represented by two pointers: $Q_h$ (the head), and $Q_t$ (the tail). When $Q$ is empty, we have $Q_h = Q_t = null$. One block contains readings to be expedited in a single RF packet. In addition to the readings (whose number is the same for all blocks), a block $b$ contains a link to the next block in $Q$ (or *null* if the block is the tail one) and the sequence number of the block in the stream, which we shall denote by $b{\rightarrow}n$. This number starts with 1 (for the first block generated in a session) and is incremented by 1 for every new block issued by the thread, as explained below. The maximum size of the queue (i.e., the maximum number of blocks that it can contain at a time) is determined by the amount of storage available at the Tag and denoted by $QS_{max}$. $QS_{max}$ can be assumed to be (roughly) equal to $M/s$) where $M$ is total amount of RAM available at the Tag for storing $Q$ and $s$ is the (fixed) block size. When the streaming operation starts, $Q$ is initialized to empty and $N$ (the current block number) is initialized to 1.

## 4.1. The generator thread

Every $1/f_s$ seconds, where $f_s$ is the sampling frequency, a sensor reading is taken. The reading is stored in the *current buffer* denoted by *CB*. *CB* is filled by consecutive readings ($f_s$ times per second) until it becomes complete (its capacity is reached).

When *CB* becomes complete, the thread executes a function named *add_CB* which sets $CB{\rightarrow}n$ to $N$, increments $N$ by 1, and appends *CB* at the end of $Q$ (updating $Q_t$ and possibly $Q_h$ as needed). Before adding the new block to the queue the function makes sure that, after the addition of *CB*, $Q$ will not exceed its two limitations (of which both are necessary). **Limitation 1** says that the total number of blocks in the queue is never bigger than $QS_{max}$. **Limitation 2** requires that $Q_t{\rightarrow}n - Q_h{\rightarrow}n$, i.e., the difference between the first and the last block number in the queue, be less than or equal to $O_{max}$ which we call the *maximum block offset*. The first limitation simply makes sure that $Q$ never exceeds its allotted storage. The second limitation constrains the age difference of the blocks kept in the queue. This is needed for two reasons. The first (informal) reason is that, in the face of the storage limitations, it makes little sense to keep around old blocks that (for one reason or another) have not made it to the collection point. The second reason is that we want to be able to reference past blocks relative to the current place (block number) within the session, for which we want to restrict the range of requisite offsets.

Note the *CB* is appended at the tail of $Q$ becoming new $Q_t$. To enforce the limitations, *add_CB* examines the block at the front of $Q$ (pointed to by $Q_h$) and discards it for as long as any of the two limitations is violated when *CB* is included the queue. One invariant of the queue is that the numbers of blocks stored in it are strictly increasing (they need not be consecutive as we shall shortly see). Thus, looking at $Q_h$ and the total number of blocks stored in $Q$ is enough to verify the limitations. Having added *CB* to $Q$, the function opens a new empty version of *CB* which the thread will now be filling from scratch.

The blocks stored in $Q$ will be transmitted to the Peg by the second thread (as explained below), but not immediately discarded, unless new blocks, containing most recent readings, cannot be accommodated into $Q$ because of the limitations. When that happens, the sampling thread will be removing blocks from the front of the queue thus giving preference to fresh readings.

## 4.2. The transmitter thread

The thread operates in rounds where it transmits a *train* of blocks to the Peg and then *reverses the channel* for a short while [28] to allow the Peg to acknowledge the train. Once the transmission of a train is started, it is carried out back-to-back with a minimum inter-packet spacing, just to enable the recipient to accept the individual packets from the channel.

A train always consists of the same number of packets (blocks) which we shall denote by $T$. When commencing a train, the thread starts by scanning consecutive packets from $Q$ (beginning from $Q_h$) and transmitting them in sequence. When the last packet (the one pointed to by $Q_t$) has been handled and the train is still incomplete, the transmitter thread will simply wait until the generator thread delivers the next block, i.e., the remaining packets in the train will be sent as new blocks materialize in $Q$. Note that the blocks are *not* removed from $Q$ as they are being transmitted.

A train packet contains the block number $b{\rightarrow}n$ of the block carried by the packet, and the packaged sensor readings copied from the block. The block number always allows the Peg to authoritatively place the contents of a received packet within the complete stream of readings, regardless of how many packets have been lost and how the received packets have been misordered with respect to the original stream.

Having completed the current train, the transmitter thread enters a loop in which it expects to receive a response (an acknowledgment packet) from the Peg. Within that loop, the thread periodically sends a short EOT (end of train) packet (to make sure that the Peg has recognized that the train has ended and a response is expected) and waits for a short while for the ACK (which should normally arrive after a minimum delay). The EOT packet carries three items of information: the train number modulo 256 (a single byte) used to match trains to acknowledgments, the number of the last block transmitted in the train (denoted by $L$), and the back offset ($O_b$) from $L$ to the oldest packet still held in $Q$ ($O_b = L - Q_h{\rightarrow}n + 1$).

The idea is that having received an EOT packet, the Peg can assess which blocks have been missing (it knows the number of the last block sent by the Tag) and it also knows the minimum number of the block that it can still ask the Tag to retransmit. The reason why the latter is specified as an offset with respect to $L$ is technical: the offset information can be conveyed in two bytes instead of four (needed for a full block number). It illustrates one practical and seemingly mundane aspect of real life in the embedded world where it always makes sense to try to save on individual bytes. Note that the generator thread enforces a limitation reducing the maximum difference between the numbers of blocks stored in $Q$.

The number of the oldest block still available in $Q$ conveyed by the Tag in the EOT packet reflects the state of $Q$ at the moment the EOT packet is constructed and scheduled for transmission. This information may become outdated by the time the Peg builds and transmits its response and, more importantly, by the time that response arrives and is interpreted by the Tag, because $Q$ can be trimmed by the generator thread independently of the transmitter thread. This is OK. The possibility that a block may be irretrievably lost is factored into the scheme. It can happen that the Peg asks for the retransmission of a block that is no longer available. At the end of the next train the Peg will learn (from the new EOT packet) that the block is no more, so it will know that it makes no sense to keep asking for it.

The transmitter thread will keep retransmitting the EOT packet (possibly updating its parameters) until an acknowledgment arrives from the Peg. Normally this should happen right away, but in a pathological scenario, if the connectivity has been broken for a long time, the contents of $Q$ may evolve to the point where $L$ is no longer available. This is why the minimum value of $O_b$ for the situation when $Q$ still contains the block number $L$ is 1. When $O_b$ in an EOT packet is 0, it means that none of the blocks up to (and including) the end of the last train is available any more, so the Peg need not ask for any retransmissions.

## 4.3. The acknowledgment

The role of the acknowledgment (ACK) packet is to indicate to the Tag which blocks have been missing with respect to the end of the last train and relative to the Peg's knowledge regarding the blocks that it can still hope to receive. Again, mundane technical constraints force us to be frugal about the representation of information within the ACK. For one thing, the entire message should fit into a single packet whose useful (payload) size is limited to 60 bytes. Organizing the ACK message into multiple packets would introduce obscure reassembly problems complicating things beyond practical [28]. Consequently, the ACK format should allow the Peg to maximize the population of (independent) block numbers that it can specify in a single packet to cover worst-case scenarios and, more generally, to minimize the size of the (typical) ACK packet. As the ACK traffic *interferes* with an otherwise smooth series of back-to-back transmissions of useful data (causing channel reversals [28]), its impact (and the incurred reduction of bandwidth) should be minimized.

One mandatory item carried in an ACK packet is the train number (modulo 256) to which the acknowledgment applies. Any other data included in the packet pertain to the blocks that the transmitter thread of the Tag should *retain* in $Q$ before commencing the next train. In other words, these are the blocks that the Peg wants retransmitted. If the ACK contains no data beyond the single mandatory byte (which is the ideal case), the message to the Tag is simple: erase $Q$ up to and including $L$, i.e., the end of the last train.

The structure of an ACK packet is best explained by the way the information is interpreted by the Tag upon its arrival. The interpretation is carried out by function *handle_ACK* invoked by the transmitter thread. Following the train number stored as the first byte of the packet, the function interprets consecutive bytes as descriptors of the blocks that have been missing by the Peg and, if possible, should be retransmitted. Those blocks are specified within the ACK packet

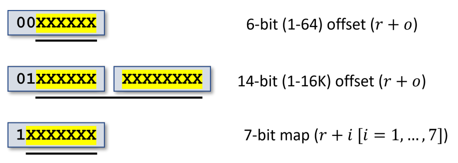

00XXXXXX
6-bit (1-64) offset ($r + o$)
01XXXXXX
XXXXXXXX
14-bit (1-16K) offset ($r + o$)
1XXXXXXX
7-bit map ($r + i$ [$i = 1, \ldots, 7$])

**Figure 3:** Descriptors of missing blocks

in the increasing order of their numbers.

While interpreting the descriptors, the function stores in $r$ the last block number mentioned by a previous descriptor, to be used as a reference. The value of $r$ is initialized to $L - O_{max} - 1$ where $O_{max}$ is the maximum legal difference between block numbers in $Q$ (see above). The ACK bytes are interpreted in the following way (see Figure 3):

1. If the two most significant bits of the byte are zero, then its remaining six bits are interpreted as a forward offset from $r$ minus 1, i.e., $r$ is incremented by the nonnegative integer value stored on those bits plus 1, and block number $r$ is marked to be retained in $Q$. Note that the minimum sensible value of an offset is 1.
2. If the two most significant bits of the byte are 01, then the remaining six bits of the byte are prepended (on the left) to the next byte and the two bytes form together a 14-bit (forward) offset from $r$ minus 1.
3. If the most significant bit of the byte is 1, then the remaining seven bits are treated as a bit map, each bit indicating an individual block relative to the value of $r$. For this purpose, the bits are numbered from 1 to 7 (right to left), and when bit number $i$ is set, the block number $r + i$ is marked to be retained. At the end of processing the byte, $r$ is set to $r + 7$, i.e., the last block number covered by the bit map, regardless of whether the block was marked as retained or not.

The important point is that the interpretation of the consecutive bytes, starting from the front of the packet's payload, produces increasing values of $r$ which eases the operation of updating $Q$. The queue is scanned in place, in the most natural manner, starting from $Q_h$, and any blocks whose numbers are not mentioned in the ACK are discarded. Before interpreting the first byte of the ACK, *handle_ACK* initializes $r$ to $L - O_{max} - 1$ to provide a sensible, default, initial value (so the first byte of the ACK can be a forward offset). It might seem natural to initialize $r$ to $L - O_b$ (based on the value of $O_b$ passed in the EOT packet), which would make the range of the initial offset better contained. However, while the value of $L$ is nailed to the train, $O_b$ may change in the different (retransmitted) versions of the EOT packet for the same train. As the Tag cannot know which particular copy of EOT served the Peg as the basis for its ACK, $O_b$ is not a well-known value that both parties can always agree on.

Following the reception of an ACK packet from the Peg, the transmitter thread will start the next train with a trimmed-down version of $Q$. The queue will have been emptied of all blocks that 1) have numbers less than or equal to $L$ from the previous train, and 2) have *not* been mentioned in the ACK packet.[4]

[4]Strictly speaking, the packet carries a *negative* acknowledgement.

**Table 1**
A tentative setting of protocol parameters

| Parameter | Value | Units |
|---|---|---|
| Channel transmission rate | 50 | kbps |
| Samples per block | 12 | 3-vectors |
| Max. $Q$ size: $QS_{max}$ | 128 | blocks |
| Train length: $T$ | 64 | packets |
| Max. block offset: $O_{max}$ | 2047 | blocks |
| Packet space (within train) | 5 | ms |
| End of train space (for the ACK) | 20 | ms |

### 4.4. The Peg

The device passes the received blocks to the computer, locally keeping track of the holes in the received sequence, down to the maximum negative offset from the end of the last train. The blocks are tallied in a bit map whose fixed size covers the interval $O_{max}$. For the ease of calculations, the actual momentary coverage of the map is described by the current *base* block number $B$ which is shifted to the newest value of $L - O_b$ learned by the Peg. As $B$ is updated, the (logical) beginning of the bit map shifts automatically (in a circular fashion), so the bit map itself is not shifted (no copying is involved), except for clearing the obsolete entries at the tail.

Having received an EOT packet, the Peg updates the base of its bit map to $L - O_b$, sets its reference block number $r$ to $L - O_{max} - 1$, and begins constructing the consecutive bytes of the ACK packet. Given the next missing block to be accounted for, there are these possibilities which are greedily examined in this order: 1) the last entry in the ACK packet is a bit map byte and the block number falls under its coverage, 2) the block number is within a bit-map range from the current value of $r$, 3) the block number can be represented by a short offset from $r$, 4) a long (two byte) offset is needed. In the first case, the block is simply added to the bit map byte without extending the ACK packet. In the second case, if the difference between the block number and $r$ is less than 7, a new bit map byte is added to the packet. Note that a bit map byte comes at the same storage expense as a short offset from $r$, so there is no point in looking ahead (a greedy approach works fine). Preference is given to the bit map when, based on the current block number alone, the map stands a chance of accommodating at least one more block.

In the unlikely case when the ACK packet becomes filled up to the limit of its size, the receiving Tag will assume that all the blocks falling behind the last block number represented in the ACK are implicitly marked as missing. Note that this will only happen in highly abnormal conditions where pessimistic assumptions regarding the unknown are probably warranted.

## 5. Performance

The first implementation of our scheme addresses a planned series of experiments with animals aimed at collecting 128 acceleration samples per second. Our intention was to tune the parameters of the protocol until we get a satisfying performance for the task at hand. Table 1 lists the numerical values of the parameters assumed for the initial tests.

While the values in Table 1 must be treated as tentative, they were produced by confronting our expectations with the parameters and capabilities of hardware. One sample of acceleration amounts to three scalars which we pack into 30 bits (10 bits per value). With some modest creativity, a 50-byte packet encodes 12 samples plus the 32-bit block number. The collection rate of 128 samples per second translates into about 11 packets per second, the total (transmitted) length of every packet being 60 bytes. This implies the (continuous) rate of 5280 bits per second and clearly suggests that the raw RF channel bandwidth of 50 kbps is more than sufficient to accommodate the transfers.

Our experiments have demonstrated, at first sight somewhat surprisingly, that it is virtually impossible to lose data in a streaming session for as long as the session operates within the framework of raw technical feasibility. We can easily cater to sampling frequencies up to 512 samples per second (which is just one notch below the maximum capacity of the the sensor) without increasing the channel rate, and up to the maximum of 1024 samples per second at a slight increase of the channel rate, practically without losing *any* samples!

This can be argued quite formally. Let $R$ denote the raw rate at which blocks can be transmitted, back-to-back, assuming smooth operation and no errors (we shall ignore the ACKs for a while). Let $r_a$ denote the target effective rate corresponding to the frequency at which we would like to reliably collect samples of sensor readings. Let $r_e$ be the block error rate of the channel. The system can be modeled as a server shown in Figure 4.

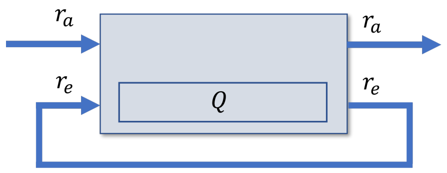


**Figure 4:** The performance model

The bottom path represents the traffic incurred by errors and the consequent retransmissions requested by the Peg in its acknowledgments. This is what the queue $Q$ is really for: to accommodate blocks that have to be retransmitted because of errors.

Consider the system at equilibrium and note that the upper path is stable and deterministic: blocks arrive at a steady rate $r_a$, their processing time is fixed, and they leave the server at the same rate. Consequently, we can ignore the dynamics of the upper part assuming that its impact consists in removing from $R$ a fixed portion amounting to $r_a$. Whatever is left, i.e., $R - r_a$ can be treated as the bandwidth available for the bottom part of the traffic, i.e., for retransmissions.

Suppose that the errors are independent and they occur at the same probability $P_e$ for every transmitted block. Then we have:

$$r_e = r_a \times \sum_{i=1}^{\infty} P_e^{\,i} = \frac{r_a \times P_e}{1 - P_e} \tag{1}$$

The bottom part of the service can be approximated as an M/D/1 queue where $\rho = r_e/(R - r_a)$ (the utilization parameter) indicates the fraction of the spare bandwidth (whatever remains after accounting for $r_a$) taken by the retransmissions. The expected occupancy of the queue is then:

$$C = \rho + \frac{1}{2}\left(\frac{\rho^2}{1-\rho}\right) \tag{2}$$

The graph of $C$ versus $\rho$ (Figure 5) is quite illuminating. It shows that unless the utilization factor becomes very close to unity, i.e., the retransmissions fill all the bandwidth left to them, the demand for queue space is extremely modest. Consequently, to see the queue overflow (for any size above the train length $T$), and actual losses start to materialize, we have to bring the system to the very edge of its equilibrium. Then, of course, there is no surprise that the system refuses to cooperate: it could not possibly do any better under the best possible scheme.

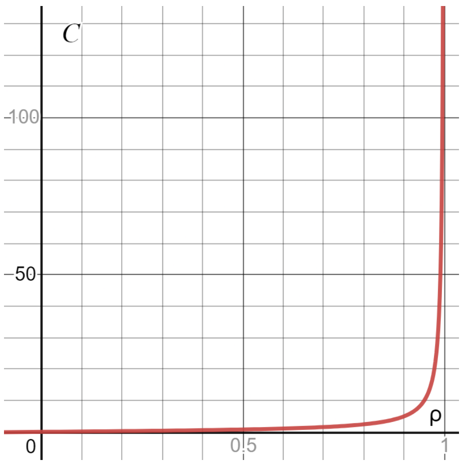


**Figure 5:** $C$ = the average occupancy of $Q$ versus the utilization factor $\rho$

The above model is oversimplified a bit by its abstraction from the acknowledgments. Their main impact is in stealing a fraction of $R$ proportional to the total portion of the bandwidth used by the trains. To factor them in, we should express the utilization parameter as:

$$\rho = \frac{r_e}{R - r_a - f_A \times (r_e + r_a)} \tag{3}$$

where $f_A$ is the amount of bandwidth (expressed in blocks) used up by the exchange of one acknowledgment.

## 6. Summary

We have presented a protocol for reliable streaming of telemetric data over an unreliable wireless channel. Our scheme seems to make a good use of the available bandwidth, especially in the specific context of its inspiring application. On the sender's side, this is accomplished by an efficient organization of storage for the outstanding (unacknowledged) packets. The feedback sent by the recipient is minimized to reduce the impact of channel reversals on the bandwidth available for the forward traffic. The proposed scheme is intended for small-footprint wireless sensing devices where the amount of memory for packet buffers is drastically limited.

## References

[1] B. J. Cooke, L. B. Hill, D. P. Farrington, W. D. Bales, A beastly bargain: A cost-benefit analysis of prison-based dog-training programs in Florida, The Prison Journal 101 (2021) 239–261.

[2] R. A. Yount, M. D. Olmert, M. R. Lee, Service dog training program for treatment of posttraumatic stress in service members., US Army Medical Department Journal (2012).

[3] G. Lippi, G. Cervellin, M. Dondi, G. Targher, Hypoglycemia alert dogs: a novel, costeffective approach for diabetes monitoring?, Alternative therapies in health and medicine 22 (2016) 14.

[4] R. Schoenfeld-Tacher, P. Hellyer, L. Cheung, L. Kogan, Public perceptions of service dogs, emotional support dogs, and therapy dogs, International journal of environmental research and public health 14 (2017) 642.

[5] G. S. Berns, A. M. Brooks, M. Spivak, K. Levy, Functional MRI in awake dogs predicts suitability for assistance work, Scientific reports 7 (2017) 43704.

[6] E. E. Bray, K. M. Levy, B. S. Kennedy, D. L. Duffy, J. A. Serpell, E. L. MacLean, Predictive models of assistance dog training outcomes using the canine behavioral assessment and research questionnaire and a standardized temperament evaluation, Frontiers in veterinary science 6 (2019) 49.

[7] C. Byrne, J. Zuerndorfer, L. Freil, X. Han, A. Sirolly, S. Cilliland, T. Starner, M. Jackson, Predicting the suitability of service animals using instrumented dog toys, Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 1 (2018) 1–20.

[8] J. M. Slabbert, J. S. Odendaal, Early prediction of adult police dog efficiency—a longitudinal study, Applied Animal Behaviour Science 64 (1999) 269–288.

[9] M. Brložnik, V. Avbelj, A case report of long-term wireless electrocardiographic monitoring in a dog with dilated cardiomyopathy, in: 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), IEEE, 2017, pp. 303–307.

[10] G. J. Jenkins, C. H. Hakim, N. N. Yang, G. Yao, D. Duan, Automatic characterization of stride parameters in canines with a single wearable inertial sensor, PloS one 13 (2018) e0198893.

[11] J. M. Alcaidinho, The internet of living things: enabling increased information flow in dog-human interactions, Ph.D. thesis, Georgia Institute of Technology, 2017.

[12] M. Brložnik, Š. Likar, A. Krvavica, V. Avbelj, A. Domanjko Petrič, Wireless body sensor for electrocardiographic monitoring in dogs and cats, Journal of Small Animal Practice 60 (2019) 223–230.

[13] Pet Pace Pets Remote Monitoring System, User Manual, PetPace Ltd., 2020. URL: https://petpace.com/.

[14] Animo Quick Start Guide, SureFlap Ltd., 2019. URL: https://www.surepetcare.com/en-au/animo.

[15] M. Foster, R. Brugarolas, K. Walker, S. Mealin, Z. Cleghern, S. Yuschak, J. Condit, D. Adin, J. Russenberger, M. Gruen, et al., Preliminary evaluation of a wearable sensor system for heart rate assessment in guide dog puppies, IEEE Sensors Journal (2020).

[16] M. Foster, S. Mealin, M. Gruen, D. L. Roberts, A. Bozkurt, Preliminary evaluation of a

wearable sensor system for assessment of heart rate, heart rate variability, and activity level in working dogs, in: 2019 IEEE SENSORS, IEEE, 2019, pp. 1–4.

[17] F. M. Duerr, A. Pauls, C. Kawcak, K. K. Haussler, G. Bertocci, V. Moorman, M. King, Evaluation of inertial measurement units as a novel method for kinematic gait evaluation in dogs, Veterinary and Comparative Orthopaedics and Traumatology 29 (2016) 475–483.

[18] M. Foster, J. Wang, E. Williams, D. L. Roberts, A. Bozkurt, Inertial measurement based heart and respiration rate estimation of dogs during sleep for welfare monitoring, in: Proceedings of the Seventh International Conference on Animal-Computer Interaction, 2020, pp. 1–6.

[19] E. Kuźnicka, P. Gburzyński, Automatic detection of suckling events in lamb through accelerometer data classification, Computers and Electronics in Agriculture 138 (2017) 137–147.

[20] Texas Instruments, CC1350 SimpleLink Ultra-Low-Power Dual-Band Wireless MCU, 2019. URL: http://www.ti.com/lit/ds/symlink/cc1350.pdf, technical document SWRS183B.

[21] K. A. Bartlett, R. A. Scantlebury, P. T. Wilkinson, A note on reliable full-duplex transmission over half-duplex links, Communications of the ACM 12 (1969) 260–261.

[22] W. Lynch, Reliable full-duplex transmission over half-duplex telephone lines, Communications of the ACM 11 (1968) 407–410.

[23] J. F. Kurose, K. W. Ross, Computer Networking: A Top-Down Approach Featuring the Internet, Addison-Wesley, 2004.

[24] S. Lin, D. Costello, M. Miller, Automatic-repeat-request error-control schemes, IEEE Communications Magazine 22 (1984) 5–17.

[25] T. Lakshman, U. Madhow, The performance of TCP/IP for networks with high bandwidth-delay products and random loss, IEEE/ACM transactions on networking 5 (1997) 336–350.

[26] C. Baransel, W. Dobosiewicz, P. Gburzyński, Routing in multi-hop switching networks: Gbps challenge, IEEE Network Magazine (1995) 38–61.

[27] R. Pereira, E. G. Pereira, Video streaming considerations for internet of things, in: 2014 International Conference on Future Internet of Things and Cloud, IEEE, 2014, pp. 48–52.

[28] P. Gburzynski, B. Kaminska, A. Rahman, On reliable transmission of data over simple wireless channels, Journal of Computer Systems, Networks, and Communications 2009 (2009).

[29] A. Nafaa, T. Taleb, L. Murphy, Forward error correction strategies for media streaming over wireless networks, IEEE Communications Magazine 46 (2008) 72–79.

[30] Bluetooth SIG, Bluetooth technology, 2020.

# Graph-based Sparse Neural Networks for Traffic Signal Optimization

Lukasz Skowronek[2], Pawel Gora[1,2], Marcin Mozejko[2] and Arkadiusz Klemenko[2]

[1]*Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland*
[2]*TensorCell*

**Abstract**
We investigate the performance of sparsely connected neural networks, with connectivity determined by road network graphs, for solving the Traffic Signal Setting optimization problem. We conducted experiments on three realistic road network topologies and found these types of graph neural networks superior to fully connected ones, both in terms of generalization properties on fixed test sets and - more importantly - near target function minima obtained in the gradient descent optimization process. We additionally confirm the soundness of our method by showing that random perturbations of the actual graph lead to consistent deterioration of model performance.

**Keywords**
traffic optimization, graph neural networks, Traffic Signal Setting problem, surrogate modelling

## 1. Introduction

Traffic optimization problems have a natural underlying graph structure, determined by the topology of the corresponding road network. In this paper, we introduce a neural network architecture based on a road network graph adjacency matrix to solve the so-called Traffic Signal Setting (TSS) problem, in which the goal is to find the optimal traffic signal settings for given traffic conditions (as defined in [1]). Some variants of this problem were proven to be NP-hard even for very simple traffic models ([2]), and therefore, heuristics and approximations have been used to solve it ([1]), but the existing approaches still have some drawbacks. For example, evaluating the quality of traffic signal settings using accurate traffic simulations (which is a standard evaluation method) can be too time-consuming, especially for large-scale road networks and/or online evaluation ([3, 4]). Also, the size of the space of possible solutions is so large that it turns out infeasible, in any reasonable time, to obtain global minima (or even a relatively good signal settings) of the simulator output by checking all the possible solutions or doing a random search ([1]), as most points in the input space are far from the optimal solutions.

A strategy used for solving these two difficulties was presented in [1] and consists of generating a reasonably sized training set using a traffic simulator and then fitting a machine learning

*29th International Workshop on Concurrency, Specification and Programming (CS&P'21)*
p.gora@mimuw.edu.pl (P. Gora)
https://www.mimuw.edu.pl/~pawelg (P. Gora)
0000-0002-8037-5704 (P. Gora)

model to approximate the outcomes of traffic simulations very fast and accurately. The output of these models can be then minimized using an optimization algorithm, such as gradient descent, in hope to obtain close to optimal traffic signal settings. This strategy turned out to be quite successful ([1, 5, 4]), yet the models' accuracy degraded close to points considered as minima by the optimization algorithm and the model, making further optimization far more difficult ([3, 4]).

In this paper, we show that the graph-based neural networks (GNN) that we use, built based on road network graphs, can outperform feed-forward fully connected neural networks (FCNN) on this task. Comparing to the standard FCNNs, the introduced GNNs have most of the connections deleted, keeping only the crucial connections between neurons (making the information flow corresponding to the traffic flow in the road network), which makes this architecture relatively sparse, easier to train and generalize. As a consequence, GNNs have better accuracy on the test set, as well as close to the local optima found using gradient descent optimization applied to the TSS problem. We also prove that GNN architectures work better than analogous ones built based on perturbed adjacency matrices.

The rest of the paper is organized as follows. Section 2 puts our work in the context of a research in the domain of building surrogate models for complex processes, solving TSS problem and using graph neural networks. Section 3 presents the two types of graph neural network architectures we used. In Section 4, we describe the setup of our main experiments including a description of the used datasets and their generation. Section 5 summarizes our main experiment results showing that neural network architectures introduced in this paper outperform other models used in such a task. In Section 6, we summarize the results of several 'sanity checks' we performed in order to confirm that our results were not obtained by chance. Moreover, the results in Section 6 can be especially interesting for researchers in graph neural networks, e.g. we show that out-of-sample performance of graph-based sparse neural nets decreases (almost) monotonously as a function of the distance of the adjacency matrix we use for constructing the network from the true adjacency matrix. We summarize the presented research in Section 7, outlining some possible future research directions.

## 2. Related works

Complex processes, such as road traffic in cities, are difficult to study due to large number of interacting components (e.g., vehicles), nondeterminism or sensitive dependence on initial conditions. Very often, the only reasonable method to accurately predict the behaviour of such systems is to apply computer simulations which can be time-consuming and usually can't be simplified due to computational irreducibility. However, in many tasks related to complex processes it is not necessary to obtain very accurate predictions, it can be sufficient to get approximate outcomes but as fast as possible (due to stochasticity or sensitive dependence on initial conditions, it may be impossible to predict the exact value anyway). Therefore, in such cases it is natural to build the so-called surrogate models (metamodels) approximating outcomes of simulations very fast and with a good accuracy ([6]). Such applications are especially common in the case of optimization tasks, in which quite often it is necessary to run multiple simulations in order to evaluate many different input settings ([7, 8, 9]). This is the case of traffic optimization problems [10], such as the TSS problem. Many such surrogate models are based on machine learning methods, such as neural networks [7, 8], and also some previous works on solving TSS [1, 5, 3, 4] use various machine learning techniques (e.g., based on neural networks or gradient boosted decision trees) to build metamodels of traffic simulations, which were used to evaluate quality of traffic signal settings. Such metamodels were able to approximate outputs of simulations (the total times of waiting on red signals) with a very good accuracy (e.g., values of the MAPE metrics were at the level $1 - 2\%$) and a few orders of magnitude faster than by running microscopic simulations [1, 5]. Thanks to that, it was possible to use optimization algorithms such as genetic algorithms or gradient descent, to find heuristically optimal signal settings without performing extensive parameter space searches that would take weeks to complete [1, 5, 3, 4]. However, information about the road network structure has never been used in those experiments, even though it should naturally be relevant when optimizing traffic.

Introducing the direct connection between the network architecture and the graph structure can help to leverage additional information represented by a graph. Similarly to our work, [11] introduces a graph NN layer in which each vertex has specific parameters assigned to combine information from its neighbors. However, differently to our method, this layer uses only an original graph matrix and skips the dual graph structure when performing computations. The notable usage of a dual structure can be found in [12], where it is compressed to a PPMI matrix using aggregated statistics from a random graph walk procedure. This aggregation is used to introduce a vertex neighborhood context similarly to a popular T-SNE method [13]. [14] provides an extensive overview of different graph neural networks architectures and applications.

Due to their capability to capture a road-network structure, GNNs were used in multiple traffic applications. In [15, 16, 17] authors used spatio-temporal GNNs for a traffic situation prediction, whereas [18] used the same technique in order to predict the TAXI demand. However, our application of graph neural networks in the traffic optimization domain and the Traffic Signal Setting problem seems to be the first such approach.

## 3. Network architecture

The key idea in defining our sparse graph-based neural network architecture is an intuitively compelling rule that information/signal should propagate locally between the net layers. By locality, we mean the presence of only those neuron connections that have a corresponding non-zero entry in the adjacency matrix of the corresponding graph. In the case of the road network, in order to implement such a rule, the neurons in the successive layers of the neural network should be linked to the neurons corresponding to vertices and/or edges of the corresponding graph. Thus, we propose the following general ways to build a graph neural network (see Section 1 of Supplementary materials ([19]) for mathematical formulas):

1. Neurons in the even numbered layers, starting with the input layer as layer 0, correspond to graph vertices (in our case - road crossings). Neurons in the odd numbered layers correspond to graph edges (in our case - road segments). An exception should be the final layer with just one neuron. Connections from a vertex-localized layer to an edge-localized layer should only be present if a given vertex is an end of a given edge in the corresponding road network graph. There are exactly two such connections for every edge

neuron. Connections from an edge-localized layer to a vertex-localized layer should only be present if the edge has the vertex as its end in the corresponding road network graph. The number of such connections is equal to the number of particular vertex neighbors.

or

2. Neurons in all layers, with the exception of the output layer, correspond to road network graph vertices. Connections from a neuron in one layer to a neuron in the next one should only be present if the corresponding vertices are neighbors in the road network graph. The number of connections for the vertex node is equal to the number of the vertex neighbors.

Although architecture 2 might seem to be more basic, architecture 1 appears to naturally model a traffic flow through the road networks (see Supplementary materials ([19]), Section 2, for a detailed explanation). In the rest of this paper, we focus solely on GNNs of the architecture type 1.

It should also be pointed out that GNNs can have multiple channels at each edge/vertex. The number of channels in each layer is a hyperparameter of the network. In the following part, we always assume the number of channels to be constant across the hidden layers of the network.

One may also notice a similarity between our GNN architecture 2 and the graph neural networks proposed by Thomas Kipf [20]. However, we do not share any weights in our model, as we aim to focus on local patterns connected to roads / crossroads. Theoretically, we could introduce some weight sharing in the 'edge' layers of GNNs of type 1, but our first experiments using this approach led to highly disappointing results.

In typical ML literature terminology, our GNNs should likely be called 'NNs with a fixed sparse connectivity mask'. In case of multi-channel networks, sparsity is applied in the 'spacial', but not in the 'channel' dimension.

## 4. Experiment setup

In order to train the surrogate models, it was necessary to generate datasets first. For this task, we simulated vehicular traffic on 3 realistic road networks, corresponding to selected districts in Warsaw: Centrum, Ochota and Mokotów, including 11, 21 and 42 intersections with traffic signals, respectively. The simulations were run using a microscopic traffic simulator, Traffic Simulation Framework [21], for which a road network description for Warsaw was obtained from the OpenStreetMap service [22]. The inputs to the simulator were vectors of lengths 11, 21 and 42, respectively. Each position in a vector represented an offset of a traffic signal on a corresponding intersection. The offsets are shifts with respect to a global two minute traffic signal cycle start - times from the beginning of the simulation to the first switch from the green signal state to the red signal state (it was assumed for simplicity that the duration of a green signal phase is always equal to 58 seconds, while duration of a red signal phase is equal to 62, constituting a 120-second cycle). The offsets were provided as integers, measured in seconds, hence they ranged from 0 to 119 (note the periodicity of these variables). The simulator output in each case was the total waiting time on red signals, summed for all the cars participating in the simulation on a considered area (finding the inputs minimizing this output value was the optimization goal of the considered TSS problem instance).

Each simulation lasted 10 minutes and consisted of 42000 cars on the whole road network of Warsaw. The datasets for Ochota, Mokotów and Centrum were generated using approximately 100000 randomly selected inputs for the TSF simulator (the input offset values from the set $\{0, 1, \dots, 119\}$ were sampled from the uniform distribution independently). These datasets are publicly available to enable further research ([23]).

After preparing the datasets, we trained GNN and FCNN networks as metamodels to solve TSS using gradient descent. Before training, we scaled the inputs to $[-1, 1]$ using the following mappring $x \mapsto \sin(2\pi x/120)$ and $x \mapsto \cos(2\pi x/120)$, thus doubling the input size (actually, increasing the number of input channels). This is motivated by the periodicity of the problem - the neural network may learn that the offsets are periodic and values 0 and 120 correspond to the same setting and this can improve the training [3, 4]. For the output, we used a standard scaler that divides the data by its standard deviation and shifts the mean to zero.

For each of the 3 considered road networks, we tested 9 different GNN architectures and 9 FCNN architectures. The 9 selected GNN architectures corresponded to all combinations of values from the following parameter sets:

- number of hidden GNN layers: 2, 3, 4 (not counting input and output layers);
- number of channels per layer: 3, 4, 5.

The activation function we used was `tanh`, indicated as superior to ReLU in preliminary experiments and in previous works [4].

For comparison, we also tested 9 FCNN architectures with `tanh` activation function, covering all combinations of parameter values from the following sets:

- number of hidden layers: 2, 3, 4
- number of neurons per layer: 20, 40, 100

For each of the 3 datasets, we used the same 90/10 train/test split for each of the considered 18 hyperparameter settings (9 GNN architectures and 9 FCNN architecture). For each of the architectures, we ran the following procedure:

1. Train a model on the training set for about 1100 epochs (concretely, minimize on $10^5$ random mini-batches of size 997 (997 is the closest prime to 1000 - a prime number was chosen to assure better randomization, although it was not expected to have any real effect) using Adam optimizer ([24]) and a learning rate of 0.0035).
2. Evaluate the trained model on the test set using the mean relative error with respect to the original outputs as the core metrics.
3. Generate 100 gradient descent trajectories of the trained model output with respect to its inputs (in the original input space, backpropagating through the sin/cos transformation). Gradients were evaluated at inputs rounded in the original parameter space (our traffic simulator (TSF) accepts only integer inputs). Nesterov updater ([25]) with a learning rate of 0.01 and momentum of 0.9 is used. Each trajectory had 3000 steps. This is similar to approach used in [4].
4. Every 30 steps, transform the current trajectory point to the original parameter space, round and send to the TSF simulator. Save the inputs and the simulator outputs to a new 'simulation' test set.

5. Evaluate the trained model on the 'simulation' test set using various metrics (cf. the discussion in Section 5).

All the experiments were run on virtual machines in the Azure cloud (`NC6` with `NVIDIA Tesla K80` ([26])). The code used in the experiments can be found at ([27]). All of the models trained for the main experiment and all the out-of-sample simulation datasets can be found at [28]. The core dataset can be obtained at [23].

## 5. Experiment results

**Table 1**
Core results for the three best GNN and the three best FCNN architectures according to the accuracy (MAPE) on the test set (i.e. gradient descent results did not affect the selection of these models).

| Measure | Model | Ochota | Mokotów | Centrum |
|---|---|---|---|---|
| Min. MAPE | GNN | 1.33% | 0.76% | 0.80% |
| on the test set | FCNN | 1.71% | 0.94% | 0.87% |
| Min. simulation output | GNN | 32,205 | 265,129 | 63,606 |
| | FCNN | 32,587 | 266,237 | 63,553 |
| Avg MAPE on the lowest | GNN | 1.26% | 0.53% | 0.76% |
| 5% sim. outputs | FCNN | 5.35% | 3.04% | 2.49% |
| Avg MAPE on the lowest | GNN | 1.75% | 0.84% | 1.22% |
| 10% sim. outputs | FCNN | 4.53% | 2.74% | 2.25% |
| Avg MAPE on the lowest | GNN | 1.51% | 1.00% | 1.11% |
| 15% sim. outputs | FCNN | 4.65% | 2.56% | 2.04% |

The key results of our experiments with GNNs are shown in Table 1, as well as in Figure 1, complemented by the tables and figures in Section 3 of Supplementary Materials [19]. Table 1 shows a summary of core performance measures, calculated for three top GNN and three FCNN, ranked based on the average accuracy on the test set (MAPE). The core measures presented are:

- Minimum MAPE (mean absolute percentage error) on the test set. This number can be obtained *before* doing gradient descent. The minimum is taken among the 3 top ranked GNNs or FCNNs (according to the row description). Because of the model selection criterion we use for Table 1, this minimum is global within the respective 9-element model universe (GNN or FCNN).
- Minimum simulation output obtained when doing gradient descent (note that while being interesting from a traffic optimization perspective, this measure lacks robustness, as it can be distorted by a single data point).
- Average MAPE on $x\%$ (for $x = 5, 10, 15$) gradient descent trajectory ends, selected according to the corresponding simulator output value (sorted lowest first). An average is taken over the three models selected, GNN or FCNN, according to the row description.

First, let us note that the results of Table 1 show a better performance of GNNs comparing to FCNNs, particularly in terms of minimum MAPE of the test set and average MAPE on the lowest points from the gradient descent trajectory according to the simulation. The improvement is visible for all the 3 road maps (Ochota, Mokotów, Centrum) and all the 5 core measures (with the exception of the minimum simulation output value obtained for Centrum, where one FCNN turned out to yield slightly (less than $0.1\%$) lower result than all the GNNs).

To summarize, the core improvement areas are:

- Much lower error on the test set.
- Lower minimum simulator output value obtained when doing the gradient descent (except for Centrum, for which we can count a draw).
- Much lower approximation error obtained on the trajectory ends corresponding to $5\%$, $10\%$ and $15\%$ lowest simulator output values.

Figure 1 as well as similar figures for Ochota and Mokotów (see Section 3 of Supplementary Materials [19]) show the density of gradient descent trajectory points as heatmap plots. The horizontal axis corresponds to gradient descent trajectory point number (recorded every 30 steps), and the vertical axis corresponds to the simulator output. Each trajectory had 3000 steps, but we recorded points every 30 steps. The plots show a heatmap of these points on the (point number, simulator output) plane. Thus, the more points in some area, the brighter the color. Also, if one architecture reaches a lower minimum than another, the resulting heatmap is taller.

Besides confirming some of the quantitative conclusions from Table 1, the heatmaps also show that in many cases, the gradient descent is less 'noisy' for GNN, suggesting a smoother function surface, less prone to overfit noise (this is best visible in the plots in Section 3 of [19]).

## 6. Consistency checks

The findings of the previous section call for some careful consistency checks before reaching final conclusions. In particular, it is not fully clear that the actual adjacency matrix gives any value. Perhaps, any similar graph, even not related to the problem at hand, can do equally well.

To address that question, we decided to fix the number of layers to 3 and the number of channels to $4$ per layer (for GNN of type 1), and built our nets using random graphs with various degrees of resemblance to the true problem graph (we repeated this for all the three road networks we considered). As a measure of graph similarity, we used the symmetric difference between the sets of graph edges. The random graphs were generated in two ways. The first method (referred later as **'Edge/Non-edge switching'**) used random edge insertions and deletions, with the desired value of the symmetric difference kept fixed. The second method (referred later as **'Vertex label permutation'**) used random permutations of the vertex labels while keeping the connection graph structure exactly the same. Graphs generated by this method were isomorphic, but not identical to the original one.

It is worth mentioning that although the first method generates truly random graphs similar to the original one, the new graph might not represent a plausible road network. The second method, on the contrary, always keeps the same, realistic road network graph structure, but it provides spurious insights to the training algorithm as crossroads are switched.

Results obtained by the two methods are shown in Figure 2, including Ochota, Mokotów and Centrum. The plots show that the mean relative error achieved on the test sets by neural nets

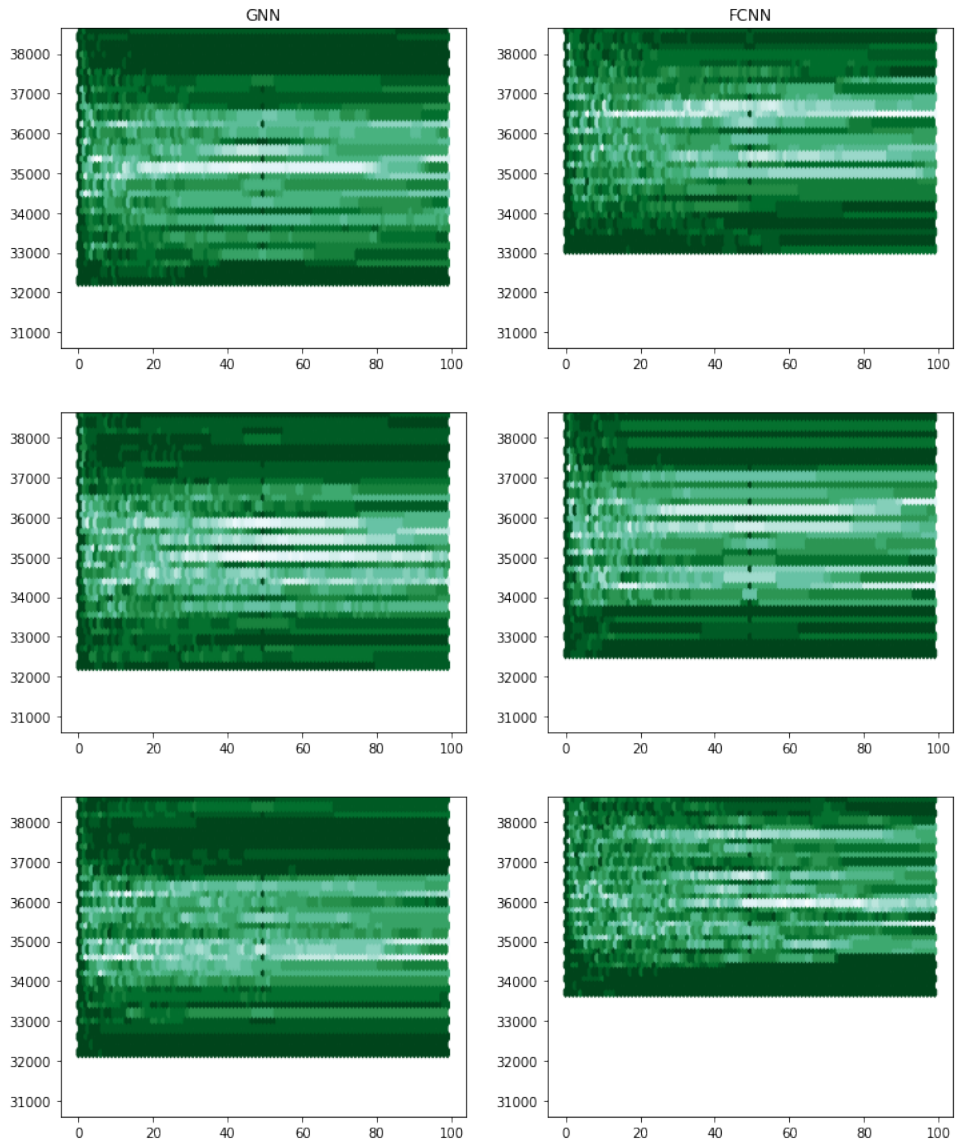
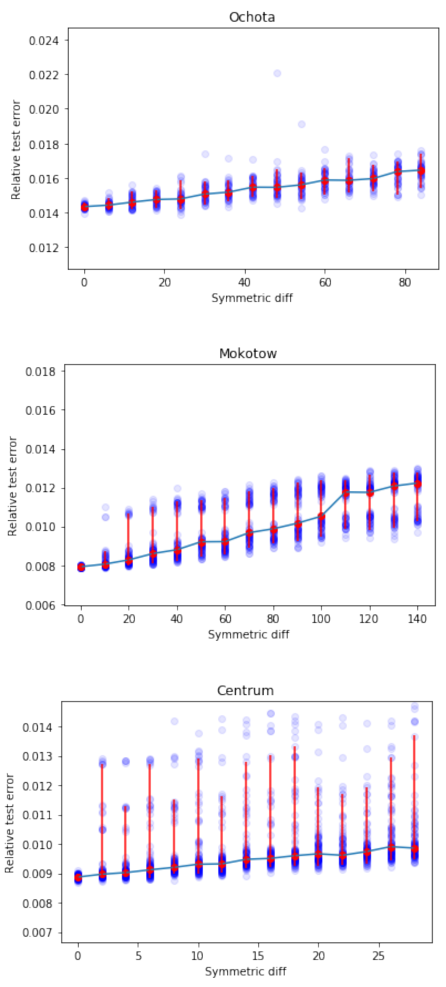
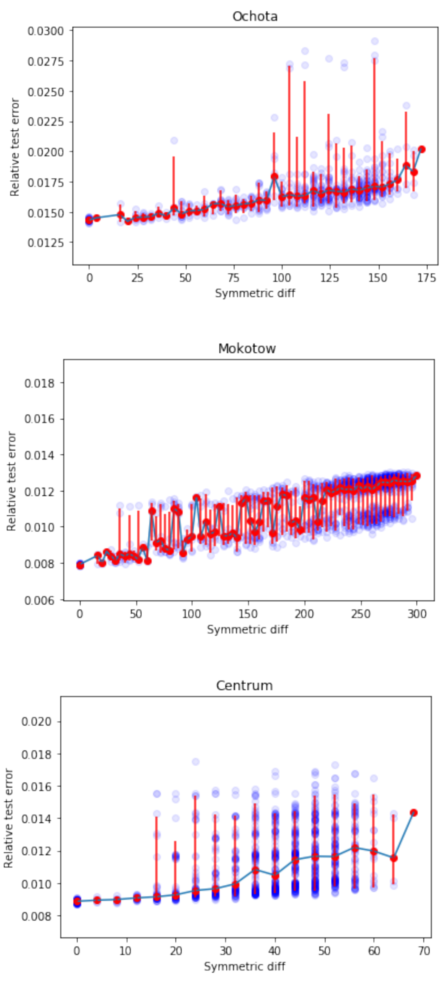


**Figure 1:** Gradient descent trajectory density plots for Warsaw Ochota for the 3 best GNN and FCNN models. Horizontal axis corresponds to trajectory point number (recorded every 30 steps), vertical axis to simulator output value.

based on random graphs, after roughly 330 epochs of training, as a function of the distance of the graph used for constructing the net to the true graph. The distance was measured using the symmetric difference between the respective edge sets.

As we can see, the median of the mean relative error, denoted with a red dot, grows almost monotonously as a function of the distance of the graph we use to the actual problem graph. This is visible for both graph sampling methods. The minimum average relative error attained for a particular value of the distance also grows, perhaps with a bit more of noise.





(a) Edge/Non-edge switching

(b) Vertex label permutation

**Figure 2:** Mean relative error achieved by a GNN on test set after roughly 330 epochs of training, shown as a function of the distance of a random graph to the true one. In subfigure 2(a), edge/non-edge switching (described in the text) was used for generating random graphs. In subfigure 2(b), vertex label permutation was used. Red dots denote median result. Errorbars correspond to 5% quantiles.

## 7. Conclusions

We demonstrated the usefulness of sparsely connected neural networks, with sparsity based on an adjacency graph, in a problem from the traffic optimization domain. GNN consistently outperformed FCNN on fixed test sets for the three realistic road networks we considered (Ochota, Mokotów and Centrum districts in Warsaw). More importantly, GNN achieved approximation quality far superior to FCNN near unseen simulator output value minima. By using randomly perturbed graphs, we also showed that the choice of the proper graph when constructing a GNN is important for achieving good results on a test set.

The kind of NN sparsity considered in this paper, where only some of the connections are allowed, may be regarded as a kind of a regularizer based on the problem graph. It is similar to L1 regularization of a fully connected neural network in that it keeps only some weights non-zero in the trained model. The resulting networks have much fewer parameters than analogous fully connected networks and turn out to generalize significantly better than any architecture we considered so far for solving the TSS problem.

## Acknowledgments

The presented research was supported by Microsoft's "AI for Earth" computational grant.

## References

[1] P. Gora, K. Kurach, Approximating traffic simulation using neural networks and its application in traffic optimization, in: NIPS 2016 Workshop on Nonconvex Optimization for Machine Learning: Theory and Practice, 2016.

[2] C. Yang, Y. Yeh, The model and properties of the traffic light problem, in: Proc. of International Conference on Algorithms, 1996, pp. 19–26.

[3] P. Gora, M. Brzeski, M. Możejko, A. Klemenko, A. Kochański, Investigating performance of neural networks and gradient boosting models approximating microscopic traffic simulations in traffic optimization tasks, in: "NeurIPS 2018 Workshop "Machine Learning for Intelligent Transportation Systems", 2018.

[4] M. Możejko, M. Brzeski, L. Madry, L. Skowronek, P. Gora, Traffic signal settings optimization using gradient descent, Schedae Informaticae 27 (2018).

[5] P. Gora, M. Bardoński, Training neural networks to approximate traffic simulation outcomes, in: 2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), IEEE, 2017, pp. 889––894.

[6] J. Zhang, S. Chowdhury, J. Zhang, A. Messac, L. Castillo, Adaptive hybrid surrogate modeling for complex systems, AIAA J (2013) 643–656.

[7] D. Rijnen, J. Rhuggenaath, P. R. d. O. d. Costa, Y. Zhang, Machine learning based simulation optimisation for trailer management, in: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), 2019, pp. 3687–3692. doi:10.1109/SMC.2019.8914329.

[8] R. D. Hurrion, A sequential method for the development of visual interactive meta-simulation models using neural networks, The Journal of the Operational Research Society 51 (2000) 712–719.

[9] R. R. Barton, M. Meckesheimer, Chapter 18 metamodel-based simulation optimization, in: S. G. Henderson, B. L. Nelson (Eds.), Simulation, volume 13 of *Handbooks in Operations Research and Management Science*, Elsevier, 2006, pp. 535 – 574.

[10] C. Osorio, M. Bierlaire, A surrogate model for traffic optimization of congested networks: an analytic queueing network approach, in: EPFL-REPORT-152480, 2009.

[11] A. Micheli, Neural network for graphs: A contextual constructive approach, IEEE Transactions on Neural Networks 20 (2009) 498–511.

[12] C. Zhuang, Q. Ma, Dual graph convolutional networks for graph-based semi-supervised

classification, in: Proceedings of the 2018 World Wide Web Conference, WWW '18, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2018, p. 499–508. URL: https://doi.org/10.1145/3178876.3186116. doi:10.1145/3178876.3186116.

[13] L. van der Maaten, G. Hinton, Visualizing data using t-SNE, Journal of Machine Learning Research 9 (2008) 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[14] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P. S. Yu, A comprehensive survey on graph neural networks, CoRR abs/1901.00596 (2019). URL: http://arxiv.org/abs/1901.00596. arXiv:1901.00596.

[15] Y. Li, R. Yu, C. Shahabi, Y. Liu, Graph convolutional recurrent neural network: Data-driven traffic forecasting, CoRR abs/1707.01926 (2017). URL: http://arxiv.org/abs/1707.01926. arXiv:1707.01926.

[16] B. Yu, H. Yin, Z. Zhu, Spatio-temporal graph convolutional neural network: A deep learning framework for traffic forecasting, CoRR abs/1709.04875 (2017). URL: http://arxiv.org/abs/1709.04875. arXiv:1709.04875.

[17] S. Guo, Y. Lin, N. Feng, C. Song, H. Wan, Attention based spatial-temporal graph convolutional networks for traffic flow forecasting, in: AAAI, 2019.

[18] H. Yao, F. Wu, J. Ke, X. Tang, Y. Jia, S. Lu, P. Gong, J. Ye, Z. Li, Deep multi-view spatial-temporal network for taxi demand prediction, CoRR abs/1802.08714 (2018). URL: http://arxiv.org/abs/1802.08714. arXiv:1802.08714.

[19] Supplementary, Supplementary materials, 2021. URL: https://drive.google.com/file/d/1sba_cunGhao4z4-loIfQYV7u4cXCdnWk/view?usp=sharing.

[20] T. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: 5th International Conference on Learning Representations, ICLR 2017, Conference Track Proceedings, 2017.

[21] P. Gora, Traffic simulation framework - a cellular automaton based tool for simulating and investigating real city traffic, in: Recent Advances in Intelligent Information Systems, 2009, pp. 641–653.

[22] OSM, Openstreetmap, 2021. URL: https://www.openstreetmap.org.

[23] Dataset, Dataset used for experiments, 2021. URL: https://drive.google.com/file/d/1aLUL3QPxGxeUVmqds6HWeGnVnQ5O4Mxr/view?usp=sharing.

[24] D. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Proceedings of the 3rd International Conference on Learning Representations (ICLR), 2015.

[25] Y. Nesterov, A method for unconstrained convex minimization problem with the rate of convergence o $(1/k^2)$, Doklady AN USSR 269 (1983) 543—-547.

[26] VMs, Description of virtual machines used in experiments, 2021. URL: https://docs.microsoft.com/en-us/azure/virtual-machines/sizes-gpu.

[27] Code, Zipped repository of the code used in our experiments, 2021. URL: https://drive.google.com/file/d/1FF6q8GTJljkYjKSNMYsL5neoXbOqPIcm/view?usp=sharing.

[28] Models, Models trained for the main experiment and all the out-of-sample simulation datasets, 2021. URL: https://drive.google.com/file/d/1mPnFt1Y1wGLGE-ha2_JiYsuJEebnfBYx/view?usp=sharing.

# Prediction of Football Games Results

Roman Nestoruk [1] and Grzegorz Słowiński [2]

[1] *Sollers Consulting Sp. z o.o., ul. Koszykowa 54, Warsaw 00-675, Poland*
[2] *University of Technology and Economics, Engineering Department, ul. Jagiellońska 82f, Warsaw 03-301, Poland*

**Abstract**

For creation of 3 machine learning models, dataset of 50, 100 and 200 games are being used. All the models are built, using deep learning (DL) and machine technology (ML) technique with the goal to prove, that even ML algorithms can be used to predict football games result. The data set consists of different real games results, collected from the most recognizable tournaments, such as: English Premier League, Italian Seria A, German Bundesliga, Spanish La Liga and French League 1. The target values of the work are prediction of exact game score (Average accuracy obtained after the last wave of testing – 11.6%) and prediction of game result (Average accuracy obtained after the last wave of testing – 39%).

**Keywords**

machine learning, football games prediction, deep learning

## 1. Introduction

Mainly, the regular person thinking that football is unpredictable and sometimes, analogical game, but we are living in the 21st century, where technologies have become one of the biggest parts of our lives.

We are using virtual assistance, image and voice recognition, autopilots, we almost meet the era of self-driving cars. The brain of all these discoveries is Artificial Intelligence, with neural networks inside. We think these technologies are very helpful for achieving the main target of this work – proving that even football, where every match consists of thousands of different moments, can be predicted by Artificial Intelligence better than by benchmark.

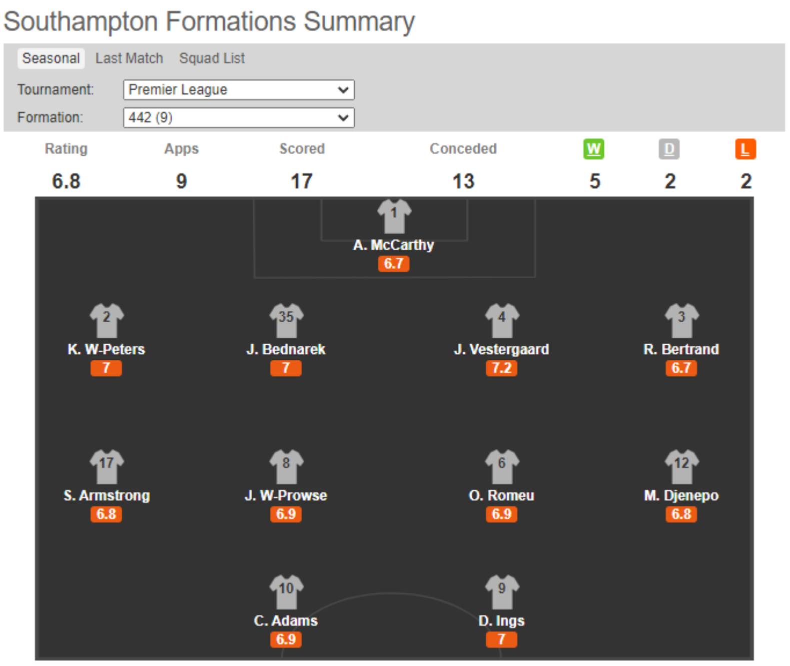
## 2. Used Tools and technology

As football statistic is not available in the format of data files, or API communication response, scraping algorithm is needed. To not enhance existing stack with extra languages, scrapping algorithm was written in Python and with use of Selenium Web Driver framework & BeautifulSoup4 library. For machine learning processes TensorFlow and keras frameworks has been used and CSV library for storing data.

## 3. Data for training and validation

One of the most recognized kinds of statistics in football games are possession and shots, but for this algorithm, some more data are also useful:

- Average game mark: Shows the performance of the team, during the season.
- The average amount of goals, per game: Result of dividing the number of goals, scored by the look at team, by the number of played games.
- Average possession: Average percentage of possession of the ball during the games.
- Pass accuracy: Counting by diving number of all successfully completed passes, by the number of all passes of the team.
- Shots per game: Anyone, who is connected to football knows, that goals are mainly the result of shots.
- Average players mark from most possible starting line up: Shows the performance of every single player, during the season.


Southampton Formations Summary

Seasonal | Last Match | Squad List

Tournament: Premier League

Formation: 442 (9)

| Rating | Apps | Scored | Conceded | W | D | L |
|---|---|---|---|---|---|---|
| 6.8 | 9 | 17 | 13 | 5 | 2 | 2 |

**Figure 1:** Table with player’s mark

## 4. Model creation

For this experiment, model with 3 dense layers is being used. As shown on figure 2, model is consisting of:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_9 (Dense)              (None, 56)                3192      
_________________________________________________________________
dense_10 (Dense)             (None, 28)                1596      
_________________________________________________________________
dense_11 (Dense)             (None, 2)                 58        
=================================================================
Total params: 4,846
Trainable params: 4,846
Non-trainable params: 0
```

**Figure 2:** Model summary

- Hidden layer 1
  Consist of 56 units, with RELU activation.
- Hidden layer 2
  Consist of 28 units, with RELU activation.
- Output layer
  Consist just of 2 units, with Linear activation.

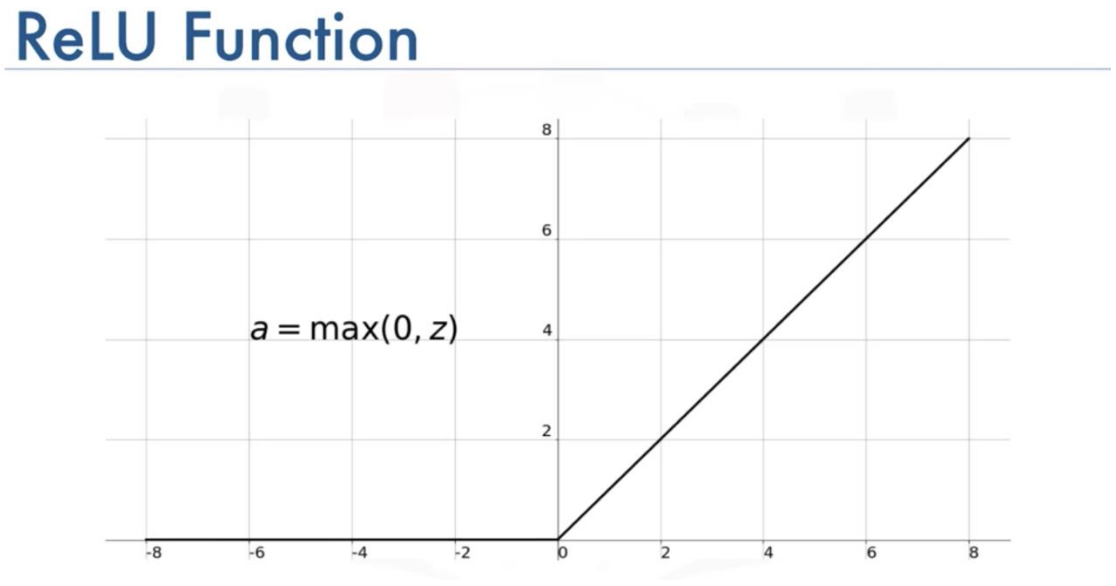


**Figure 3:** ReLU activation function graph [Source: 8]

For the first two layers, RELU activation helps to decrease all negative values, as team can not score -1 goals.

## 4.1. Data preparation

Considering that almost never in football one team is scoring more than 10 goals, expected result was transformed to the format of 0-1 value by dividing it by 10. For example: Actual score: 1:3, score after transformation: 0.1:0.3.

To be able, to better validate the result, extra 10% of the data was used for testing and validation of the model.

## 5. Models structure and usage

As a result of experiment, 3 models where created:

- Model 1: Trained on 50 examples of games with no unexpected result and validated on 5 extra examples.
- Model 2: Trained on 100 examples of games with no unexpected result and validated on 10 extra examples.
- Model 3: Trained on 200 examples and validated on 20 extra examples.

To make a summary, how effective are models in daily games prediction. Upcoming days games statistic were taken as input data for model.

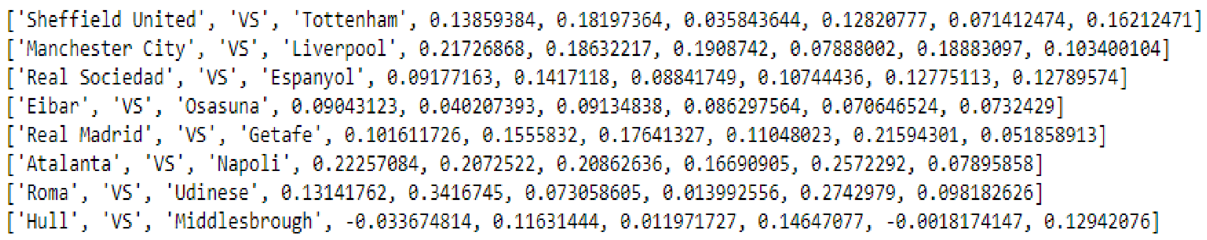
```
'[[6.72, 1.0, 0.0, 4.0, 0.0, 0.0, 0.0, 0.0, 3.0, 0.0, 3.0, 1.0, 2.0, 1.0, 45.1, 75.0, 9.7, 6.7, 6.7, 6.8, 6.9, 6.8, 6.9, 6.8,
6.9, 6.6, 7.1, 6.8, 6.78, 2.0, 3.0, 1.0, 1.0, 1.0, 1.0, 0.0, 3.0, 1.0, 1.0, 2.0, 0.0, 1.6, 51.8, 81.2, 12.2, 6.8, 7.2, 6.8, 6.
6, 6.6, 6.9, 7.3, 6.9, 7.1, 7.2, 7.5]]'

'[[7.01, 1.0, 0.0, 0.0, 2.0, 3.0, 0.0, 5.0, 0.0, 1.0, 2.0, 2.0, 0.0, 2.5, 62.0, 89.2, 19.1, 6.7, 6.9, 6.8, 7.4, 6.9, 6.9, 8.0,
7.5, 7.7, 8.2, 7.2, 7.0, 0.0, 3.0, 0.0, 2.0, 2.0, 1.0, 2.0, 3.0, 0.0, 0.0, 4.0, 0.0, 2.3, 59.5, 84.2, 15.6, 6.7, 7.3, 7.1, 7.2,
7.0, 7.4, 7.1, 6.8, 7.2, 7.5, 7.5]]'
```

**Figure 4:** Example of information, used for the result prediction

To simplify the process of validation, result of models prediction is stored table with following format:

- Team playing home as t1
- Team playing away as t1
- Number of goals predicted by the first model, for the home teams as m1t1
- Number of goals predicted by the first model, for the away teams as m1t2
- Number of goals predicted by the second model, for the home teams as m2t1
- Number of goals predicted by the second model, for the away teams as m2t2

- Number of goals predicted by the last model, for the home teams as m3t1
- Number of goals predicted by the last model, for the away teams as m3t2



```
['Sheffield United', 'VS', 'Tottenham', 0.13859384, 0.18197364, 0.035843644, 0.12820777, 0.071412474, 0.16212471]
['Manchester City', 'VS', 'Liverpool', 0.21726868, 0.18632217, 0.1908742, 0.07888002, 0.18883097, 0.103400104]
['Real Sociedad', 'VS', 'Espanyol', 0.09177163, 0.1417118, 0.08841749, 0.10744436, 0.12775113, 0.12789574]
['Eibar', 'VS', 'Osasuna', 0.09043123, 0.040207393, 0.09134838, 0.086297564, 0.070646524, 0.0732429]
['Real Madrid', 'VS', 'Getafe', 0.101611726, 0.1555832, 0.17641327, 0.11048023, 0.21594301, 0.051858913]
['Atalanta', 'VS', 'Napoli', 0.22257084, 0.2072522, 0.20862636, 0.16690905, 0.2572292, 0.07895858]
['Roma', 'VS', 'Udinese', 0.13141762, 0.3416745, 0.073058605, 0.013992556, 0.2742979, 0.098182626]
['Hull', 'VS', 'Middlesbrough', -0.033674814, 0.11631444, 0.011971727, 0.14647077, -0.0018174147, 0.12942076]
```

**Figure 5:** Data stored into the “results” variable.

## 6. Models evaluation

After all matches, we were interested in, have been finished. We can start comparing our predictions with the actual result. To simplify the result verification, we should transfer output of DL model to the integer, for that purpose, values where multiplied by. As a standard for this transformation, regular rules of rounding where used:

- Values are less than integer and half will be rounded to closes lower integer. For example: 1.5252->2, 2.9842->3, 0.5->1
- Values are less then integer and half, will be rounded to closes lower integer. For example: 0.4999->0, 1.1->1, 2.332->2.

Following these rules, a result like 0.5 vs 0.49 will be considered as 1 vs 0, but a result of 1.49 vs 0.5 will be considered as 1 vs 1.

The most known kind of prediction is a white guessing. Considering that probability of randomly guessing the result of any football game is 1 by the amount of possible – 3 (Winning of the home team, draw, or winning of away team), technically it is 33.3%.

The possibility of predicting the exact score of the game is more complicated because all possible combinations of the score should be considered. The chance of scoring more than 4 goals is too small, to be considered. So, to calculate the chance of predicting the exact score of the game, we should calculate the combination of 5 elements (score from 0 to 4) into 2 places (for 2 playing teams, home and away). We can calculate it by using the formula 6 – 1. The result of this calculation gives as 15 and the chance of prediction of the exact score of the game is 1 by 15 or 6.7%.

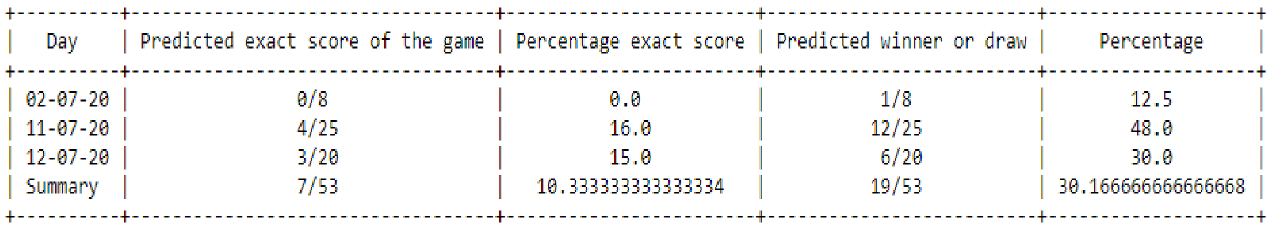
$$C(n+r-1,r) = \frac{(n+r-1)!}{r!\,(n-1)!}$$

**Formula 6:** Combination with repetition

### 6.1. Model 1 evaluation

As mentioned in Chapter 6, model 1 was trained on the smallest amount of real data – 50 games.

- The average percentage of the predicted exact score of the games is on the level 10.3%. It's around 1.5 times more than mathematical chances to predict it. Of course, 0 predicted games for the 02-07-20 is looking not promising, but we had a very small amount of data to predict. Next days, this amount increased and was more than 2 times greater than the mathematical probability.
- The average percentage of the predicted winner or draw category is much higher. Of course, in Picture 8.2 we can see that the first day was failed again. Next days we can see results 4 and 3 times higher than on the 02-07-20, but this time average is below mathematical.



```
+----------+-------------------------------+---------------------+----------------------+-------------------+
|   Day    | Predicted exact score of the game | Percentage exact score | Predicted winner or draw |     Percentage    |
+----------+-------------------------------+---------------------+----------------------+-------------------+
| 02-07-20 |              0/8              |         0.0         |         1/8          |        12.5       |
| 11-07-20 |              4/25             |         16.0        |         12/25        |        48.0       |
| 12-07-20 |              3/20             |         15.0        |         6/20         |        30.0       |
| Summary  |              7/53             |  10.333333333333334 |         19/53        | 30.166666666666668 |
+----------+-------------------------------+---------------------+----------------------+-------------------+
```

**Figure 7:** Statistic of prediction from model 1

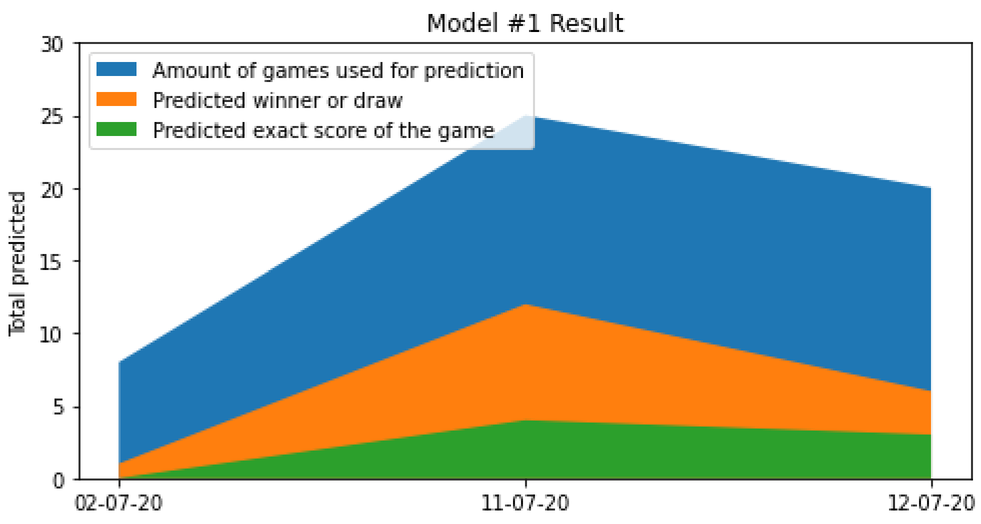
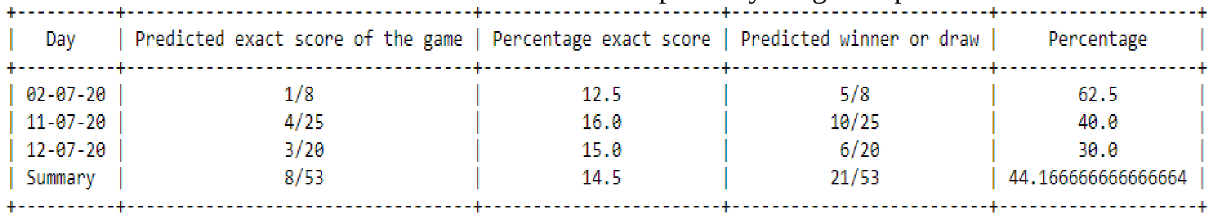


**Figure 8:** Diagram for the statistic of prediction from model 1

## 6.2. Model 2 evaluation

Model 2 was trained on the higher amount of real data – 100 games.

- This time we can see good progress on the average percent of predicted games, mainly because of the predicted games from day 02-07-20. The average percentage for the exact score category was much more stable and stands on the level 2 times more than mathematical probability.
- For the Predicted winner or draw category we can again see a big difference on the first row, but almost similar results on the day two and three. The average prognosis stands on the level 44.2%, it's now 33% more effective than mathematical probably for games prediction.



```
+----------+-------------------------------+---------------------+----------------------+-------------------+
|   Day    | Predicted exact score of the game | Percentage exact score | Predicted winner or draw |     Percentage    |
+----------+-------------------------------+---------------------+----------------------+-------------------+
| 02-07-20 |              1/8              |         12.5        |         5/8          |        62.5       |
| 11-07-20 |              4/25             |         16.0        |         10/25        |        40.0       |
| 12-07-20 |              3/20             |         15.0        |         6/20         |        30.0       |
| Summary  |              8/53             |         14.5        |         21/53        | 44.166666666666664 |
+----------+-------------------------------+---------------------+----------------------+-------------------+
```

**Figure 9:** Statistic of prediction from model 2

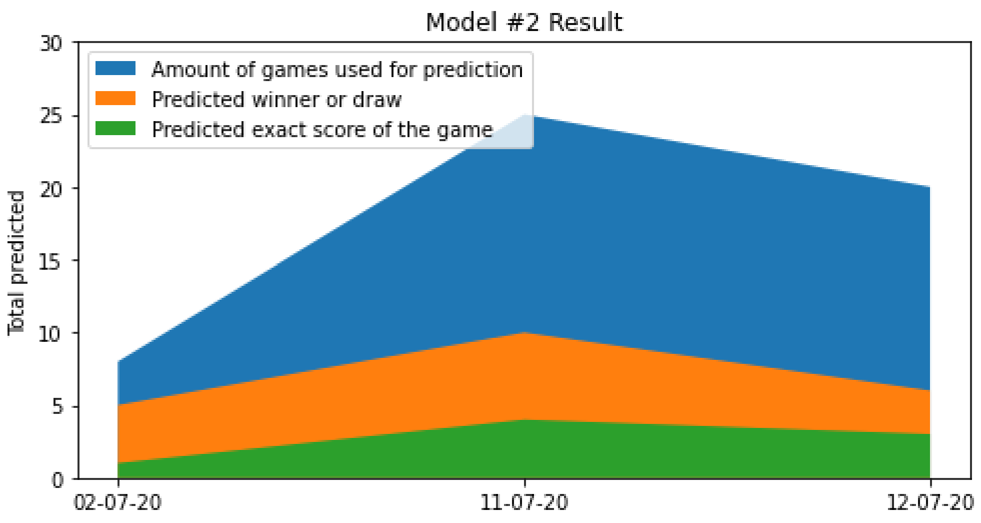


**Figure 10:** Diagram for the statistic of prediction from model 2

## 6.3. Model 3 evaluation

Model 3 was trained on the highest amount of real data – 200 games.

- The last model, we are taking into the evaluation shows a very interesting result. The average percentage of predicting the exact score is above the mathematical chances, average percentage against stands on the level of around 10%, but much more stable than the first model.
- On the Predicted winner or draw category we can see a big difference in percentages, the first day was amazingly predicted with 6 out of 8 games. This is more than two times higher than the random guess probably. But after the checking next two days we can see, that this percentage drops so much to an extremely low level.

```
+----------+-----------------------------------+-----------------------+---------------------------+-------------------+
|   Day    | Predicted exact score of the game | Percentage exact score | Predicted winner or draw |    Percentage     |
+----------+-----------------------------------+-----------------------+---------------------------+-------------------+
| 02-07-20 |                1/8                |         12.5          |            6/8            |       75.0        |
| 11-07-20 |                2/25               |          8.0          |            7/25           | 28.000000000000004 |
| 12-07-20 |                2/20               |         10.0          |            5/20           |       25.0        |
| Summary  |                5/53               |   10.166666666666666  |           18/53           | 42.666666666666664 |
+----------+-----------------------------------+-----------------------+---------------------------+-------------------+
```

**Figure 11:** Statistic of prediction from model 3

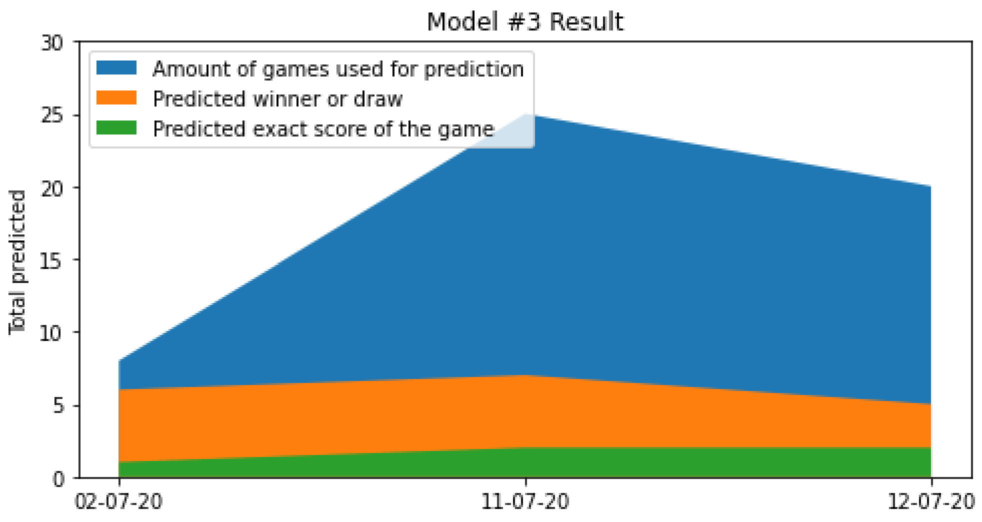


**Figure 12:** Diagram for statistic of prediction from model 3

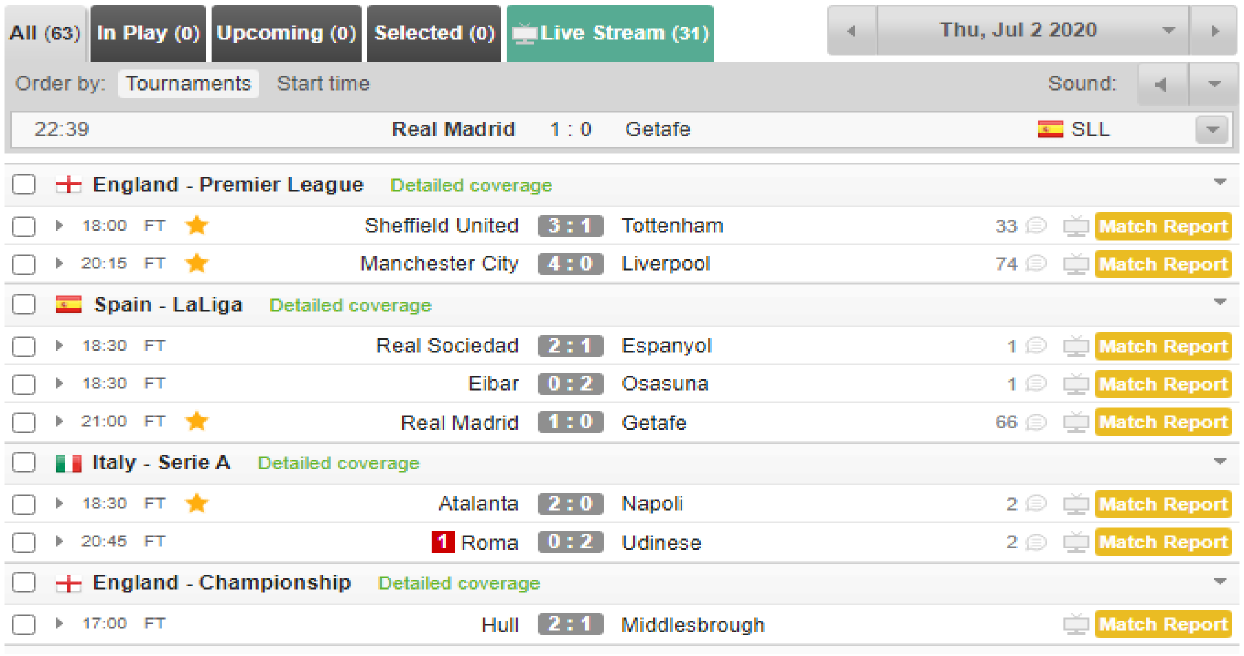
## 6.4. Models comparison

For the model's comparison, we are using predictions, they made for real games. To discuss mainly the advantages and disadvantages of the model, we will be concentrated only on those 3 dates: 02-07-20, 11-07-20 and 12-07-20

All of them are very useful, in terms of finding the problems, which can be improved during the training of the future model.

### 6.4.1. First day of experiment

After the first look at the model's result, we can be somehow disappointed about their prediction for the first day. Two of them were doing very well and predicted more than 60% of the games, but one completely failed the experiment. To find the reason for this situation we should look at the games, we were trying to predict Figure 15.



| | | | | | | |
|---|---|---|---|---|---|---|
| **England - Premier League** | | | | | | |
| 18:00 | FT | Sheffield United | 3 : 1 | Tottenham | 33 | Match Report |
| 20:15 | FT | Manchester City | 4 : 0 | Liverpool | 74 | Match Report |
| **Spain - LaLiga** | | | | | | |
| 18:30 | FT | Real Sociedad | 2 : 1 | Espanyol | 1 | Match Report |
| 18:30 | FT | Eibar | 0 : 2 | Osasuna | 1 | Match Report |
| 21:00 | FT | Real Madrid | 1 : 0 | Getafe | 66 | Match Report |
| **Italy - Serie A** | | | | | | |
| 18:30 | FT | Atalanta | 2 : 0 | Napoli | 2 | Match Report |
| 20:45 | FT | 1 Roma | 0 : 2 | Udinese | 2 | Match Report |
| **England - Championship** | | | | | | |
| 17:00 | FT | Hull | 2 : 1 | Middlesbrough | | Match Report |

**Figure 13:** Games, took place at the date of the experiment (02-07-20)

Most of these games were very important because it was games between the table "place mates". The difference between the data was very small, but for almost all the games, except the game Roma vs Udinese, we have seen the success of a team, having a little bit better statistic.

After this analysis, we can assume that our model 1 is having a very small amount of data with different quality, which impacts a bigger range of possible results. For example:

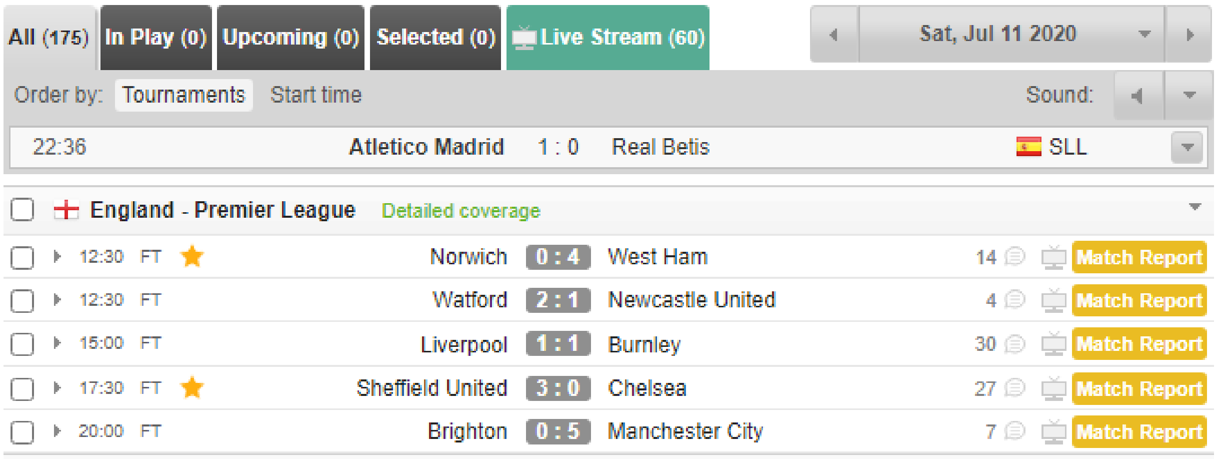
Let's imagine the ball falls to the ground, we need to predict the height at which the ball will rise after falling. If we have seen too few examples, with different results we will think like this: The ball might have small pressure and will raise only for 40% of the initial height or this ball might have different materials and jump for 60% or even higher. Neural networks trying to consider as most data, as it can, so some of the examples from the data set can make only mistakes in prediction.

Now, we can investigate model 3, For the first day we have an amazing result, but why we have such a big drop for the next days? The problem is very similar to the one above. Because we have a big amount of data, saying for example: "Ball almost every time jumps to the 50% of the initial height", the model will ignore extra data and always trying to make a prediction without caring about extra data. This problem is usually called overtraining.

Let us finally discuss model 2. There we have an average amount of data, so after the training model "thinks", usually the ball jumps for 50% of the initial height, but let us consider materials, this ball was created from, etc. This is the reason, why this model is not failing very much during all the games.

### 6.4.2. Second day of experiment

After reviewing day 2, we can see, how the overtraining problem having an impact on forecasting. According to our statistics, the best prediction for this day was made by the first model. To make the right conclusion about this day, we should investigate the games and their results (Figure 15) from football analytics way. Some of them finished with the surprising and hardly predictable result even for the specialists in analytics, like draw in the game between the English champion and the team, from the second part of the table.



All (175) | In Play (0) | Upcoming (0) | Selected (0) | Live Stream (60) | Sat, Jul 11 2020

Order by: Tournaments Start time Sound:

22:36 Atletico Madrid 1 : 0 Real Betis SLL

England - Premier League Detailed coverage

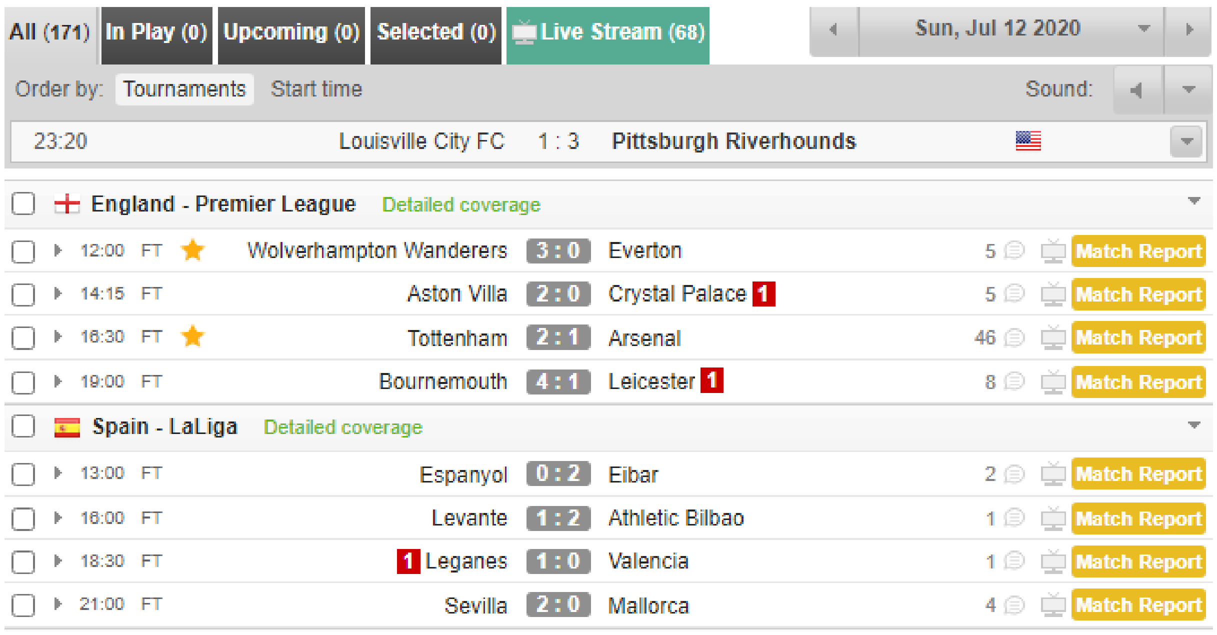
| | | | | | | |
|---|---|---|---|---|---|---|
| 12:30 | FT | Norwich | 0 : 4 | West Ham | 14 | Match Report |
| 12:30 | FT | Watford | 2 : 1 | Newcastle United | 4 | Match Report |
| 15:00 | FT | Liverpool | 1 : 1 | Burnley | 30 | Match Report |
| 17:30 | FT | Sheffield United | 3 : 0 | Chelsea | 27 | Match Report |
| 20:00 | FT | Brighton | 0 : 5 | Manchester City | 7 | Match Report |

**Figure 14:** Games, took place at the date of the experiment (11-07-20)

The first model showed us the best result 12 out of 25 or 48% of correct predictions for the "Predicted winner or draw" category, while the second model was working also well, with 40% of predicted games. The worst result was shown by the third model. 7 out of 25 and 28%, which is also fine if it happens rarely.

### 6.4.3. Third day of experiment

All of the models having similar results, but they have predicted different games. Because of 20 games with different kinds of teams, we can see a very good percent of exact prediction on the 10-15%. But as well this variety of teams is having a big impact on the amount of predicted winner/draw. All of them are on the 25-30%. For this day, as well as the previous one, we can see the worst result is made by model 3.


All (171) In Play (0) Upcoming (0) Selected (0) Live Stream (68) Sun, Jul 12 2020

Order by: Tournaments Start time Sound:

23:20 Louisville City FC 1 : 3 Pittsburgh Riverhounds

England - Premier League Detailed coverage

| | | | | | | |
|---|---|---|---|---|---|---|
| 12:00 | FT | Wolverhampton Wanderers | 3 : 0 | Everton | 5 | Match Report |
| 14:15 | FT | Aston Villa | 2 : 0 | Crystal Palace 1 | 5 | Match Report |
| 16:30 | FT | Tottenham | 2 : 1 | Arsenal | 46 | Match Report |
| 19:00 | FT | Bournemouth | 4 : 1 | Leicester 1 | 8 | Match Report |

Spain - LaLiga Detailed coverage

| | | | | | | |
|---|---|---|---|---|---|---|
| 13:00 | FT | Espanyol | 0 : 2 | Eibar | 2 | Match Report |
| 16:00 | FT | Levante | 1 : 2 | Athletic Bilbao | 1 | Match Report |
| 18:30 | FT | 1 Leganes | 1 : 0 | Valencia | 1 | Match Report |
| 21:00 | FT | Sevilla | 2 : 0 | Mallorca | 4 | Match Report |

**Figure 15:** Games, took place at the date of the experiment (12-07-20)

## 7. Summary

The main idea of this work – is to create a new kind of working model, for football result prediction. While other peoples are trying to predict the winner of the game, we decided to look more for trying to predict the number of goals each team should score.

After the following experiments, we can conclude that amount of data, taken for the training is not having the main impact on successful predictions. Mainly the quality of the data and a little bit of luck are making success in this field. For getting these 3 different neural networks, we have done more than 500 tries to fit them, because, for such kind of sport like football, the models should have some understanding about the usability of every component and not every time we can receive the same result even with the same statistic.

## 8. References

[1] Maureen Caudill, Neural Network Primer: Part I", February 1989

[2] Francois Chollet, Deep Learning with Python Paperback, 10 January 2018

[3] Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python: A Guide for Data Scientists, 2015

[4] Sebastian Raschka, Vahid Mirjalili, Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow, 2 December 2019

[5] Yann LeCun, Gökhan BakIr, Thomas Hofmann, Bernhard ,Alexander J. Smola, Predicting Structured Data (Neural Information Processing series), July 2007

[6] David J. Livingstone, Artificial Neural Networks, 2009

[7] https://medium.com/@toprak.mhmt/activation-functions-for-deep-learning-13d8b9b20e Access date: 11.07.2021

# Dry Beans Classification Using Machine Learning

Grzegorz Słowiński

*University of Technology and Economics, ul. Jagiellońska 82f, 03-301 Warsaw, Poland*

**Abstract**

A dataset containing over 13k samples of dry beans geometric features is being analysed using machine learning (ML) and deep learning (DL) techniques with the goal to automatically classify the bean species. First the original dataset was reduced to eliminate redundant features (too strongly correlated and echoing others). Then the dataset was visualised and analysed with machine learning techniques: Multinomial Bayes, Support Vector Machines, Decision Tree, Random Forest, Voting Classifier and Artificial Neural Network. The overall accuracies obtained were in range: 88.35 – 93.61%.

**Keywords**

machine learning, deep learning, classification of dry beans.

## 1. Introduction

Classification of dry beans is of some economic importance. Manual classification is labour intensive, etc. Over 13 k samples of dry beans of 7 various species were photographed and their geometry was measured via computer vision techniques in [1]. Then the set was analysed via several machine learning (or data science) and deep learning (or artificial neural network) techniques. The overall accuracy obtained was 87.92-93.13%, depending on technique used.

The dataset used in [1] has been published in the UCI machine learning repository [2]. This work analyses the same dataset using slightly different techniques. Data dimensionality has been reduced. Slightly better accuracies has been achieved. Discussion and comparison to [1] has been carried out.

## 2. Tools

The entire analysis was done using Python and its ML frameworks: numpy, pandas, matplotlib, seaborn, scikit-learn and keras. Google Colab a free cloud version of jupyter notebook was used. The reader can find the Python scripts under link [3].

## 3. Preliminary analysis and visualisation of the dataset

The dataset under study consists of 13611 samples. A sample amounts to 16 geometrical features and a label identifying the species of the bean. The species are as follows: Barbunya, Bombay, Cali, Dermason, Horoz, Seker, and Sira. The features are: Area, Perimeter, MajorAxisLength, MinorAxisLength, AspectRatio, Eccentricity, ConvexArea, EquivDiameter, Extent, Solidity, Roundness, Compactness, ShapeFactor1, ShapeFactor2, ShapeFactor3, and ShapeFactor4. A detailed explanation how the features were calculated is presented in [1].

The geometrical data carry no information about the bean colour. From the practical point of view it is unfortunate, as different dry bean species tend to vary in colour. On the other hand, it makes little difference if we just want to treat the dry beans classification problem as an exercise in building and comparing machine learning models.

[1]29th International Workshop on Concurrency, Specification and Programming (CS&P'21)
EMAIL: grzegorz.slowinski@uth.edu.pl
ORCID: 0000-0001-9770-5063

## 3.1. Correlation analysis and feature reduction

Correlation analysis has shown that several of the features are strongly (positively or negatively) correlated. This is due to the fact that basically all of them are kind of geometric measures. The decision was taken to drop some features to avoid correlations over 0.9 (or negative correlation below -0.9) between them. The benefits of such a decision should be: 1) a significant reduction of the computational complexity 2) a lower risk of overfitting 3) ease of visualisation. The disadvantage is a limited risk of loosing some valuable information and, as a result, a decrease in accuracy.

**Table 1**
Correlation between selected beans features

| | MajorAxis Length | MinorAxis Length | Aspect-Ratio | Extent | Solidity | Roundness | Shape Factor2 | Shape Factor4 |
|---|---|---|---|---|---|---|---|---|
| MajorAxis Length | 1.0000 | 0.8261 | 0.5503 | -0.0781 | -0.2843 | -0.5964 | -0.8592 | -0.4825 |
| MinorAxis Length | 0.8261 | 1.0000 | -0.0092 | 0.1460 | -0.1558 | -0.2103 | -0.4713 | -0.2637 |
| AspectRatio | 0.5503 | -0.0092 | 1.0000 | -0.3702 | -0.2678 | -0.7670 | -0.8378 | -0.4493 |
| Extent | -0.0781 | 0.1460 | -0.3702 | 1.0000 | 0.1914 | 0.3444 | 0.2380 | 0.1485 |
| Solidity | -0.2843 | -0.1558 | -0.2678 | 0.1914 | 1.0000 | 0.6072 | 0.3436 | 0.7022 |
| Roundness | -0.5964 | -0.2103 | -0.7670 | 0.3444 | 0.6072 | 1.0000 | 0.7828 | 0.4721 |
| Shape-Factor2 | -0.8592 | -0.4713 | -0.8378 | 0.2380 | 0.3436 | 0.7828 | 1.0000 | 0.5299 |
| Shape-Factor4 | -0.4825 | -0.2637 | -0.4493 | 0.1485 | 0.7022 | 0.4721 | 0.5299 | 1.0000 |

Thus, in this work it was decided to limit the set of features list to these 8 members: MajorAxisLength, MinorAxisLength, AspectRatio, Extent, Solidity, Roundness, ShapeFactor2, ShapeFactor4, and to exclude: Area, Perimeter, Eccentricity, ConvexArea, EquivDiameter, Compactness, ShapeFactor1, ShapeFactor3. The issue of high correlations among some features was not addressed in [1]. The visualisation of the data was done by pair-plot, and is presented in figure 1.

It shows that the Bombay species is trivial to classify as its beans are significantly bigger than others. the classification of other species seems to be much more difficult, and we can expect more errors. The correlations between pairs of the selected features are listed in Table 1.

## 4. Machine Learning techniques used and results

In this work the following techniques were used: Multinomial Gaussian Classifier, Support Vector Classifier, Decision Tree, Random Forest, Voting Classifier, Artificial Neural Network (Multilayer Perceptron or MLP).

The full dataset was divided into the training and test subsets. 80% of samples were used for training and 20% for testing. Division of all available samples into the training and test subsets is crucial for a correct methodology. The aim of all ML or DL methods is to achieve a ”generalization” ability. Thus it is important to check the accuracy of classifying new samples, ones that have not been used during training. Otherwise, there is a very serious risk that the model will suffer form overfitting. Overfitted models perform very well on the training data but much worse on new data. Overfitting (as one of the most important issues in ML) is widely discussed in ML handbooks [4-6].

## 4.1. Multinomial Naive Bayes classifier

Naive Bayes models are based on Bayes's theorem. They are extremely fast and simple, but on the other hand, their performance is usually limited. They can be used as a baseline for classification problems (see [4], p. 382).

The overall accuracy obtained with Multinomial Bayes Classifier was 64,30 %. The problem was to classify into 7 different classes. Thus the blind (random) classification should result in about 1/7 =

14,29% accuracy. As one can see, classification is more difficult if there are more classes. Random classification should give accuracy equal to about 1/(number of classes). Thus we can see that even this simple model perform about 50 percent points better than the random approach.

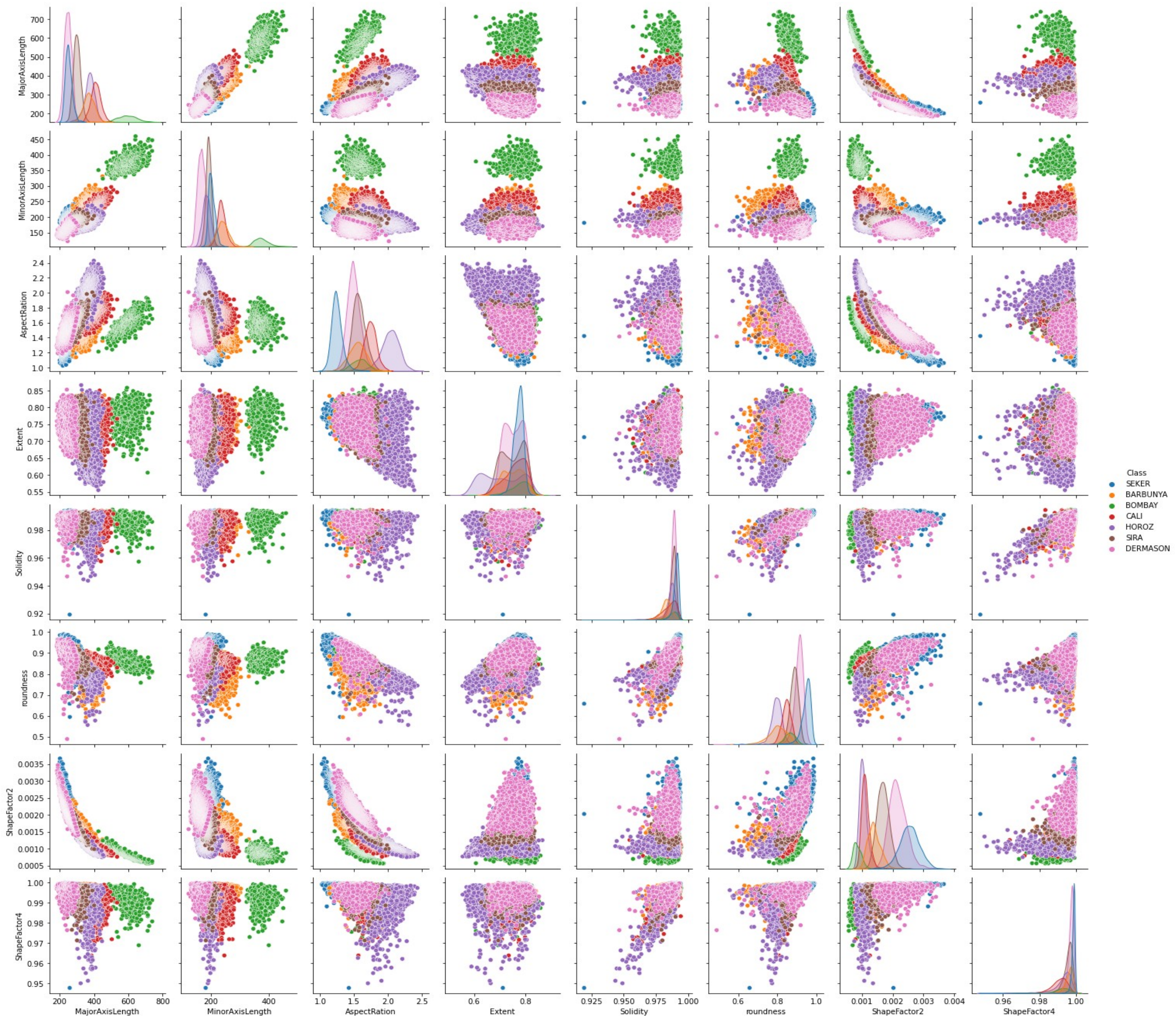


**Figure 1:** The selected features of dry beans (pairplot)

## 4.2. Support Vector Classifier

Support vector machines (SVM), which can be used as regressors or classifiers, are considered a very powerful and flexible algorithms. On the other hand. they may need a lot of computing power (see [4] p.405). The SVM principle is to partition the classes by ”drawing a line” (or plane) in a way that maximises the margin between classes. As straight lines (or planes) do not usually produce the best solution, SVC can apply different kernels (polynomial, radial and others). SVC is wider explained in [4,5]. SVCs with different kernels were tried. Table 2 presents the parameters used and the accuracy obtained.

The results are quite similar for all kernels. The accuracy can be further improved to some extent (tenths of %, maybe 1%) by increasing C, but this will also significantly increase the training time.

**Table 2**
The parameters for SVM (other parameters have default values).

| Kernel type | C parameter | Approx. computing time* | Overall accuracy |
|---|---|---|---|
| Linear function | $10^5$ | 41 s | 91.55% |
| Polynomial, degree=3 | $10^5$ | 21 s | 91.26% |
| Radial basis | $10^7$ | 34 s | 92.18% |

*- computation was done on colab: Intel(R) Xeon(R) CPU @ 2.20GHz and 12,69GB RAM

## 4.3. Decision Tree

A decision tree (DT) belongs to the class of so called non-parametric algorithms. The term non-parametric can be misleading. In fact, a decision tree has parameters, but their number is not constant.

During the learning phase, a decision tree tries to find the best questions partitioning the dataset in order to reduce information impurity (the measure is the Gini index or information entropy). The great advantage of decision trees is that they are extremely intuitive. On the other hand, a decision tree has no limited degrees of freedom, so it is easy to overfit (if the user is not aware of that). The splits made by a decision tree are always orthogonal (made on one feature at a time), so the decision tree is very sensitive to data rotation (see [5], p.188).

In [1] the authors created a decision tree with the depth of 4 (4 questions max) and 9 leaves. We decided to limit the depth of our decision tree to 5 and to 16 leaves max in order to get a decision tree that has size similar to DT obtained in [1].

Figure 2 shows the decision tree obtained under the above limits. The overall accuracy is 88.35%. Preliminary tests showed that a better accuracy of about 92,3% could be obtained with a bigger decision tree; however, the bigger the decision tree, the less intuitive it becomes, and the more difficult it is to visualise.

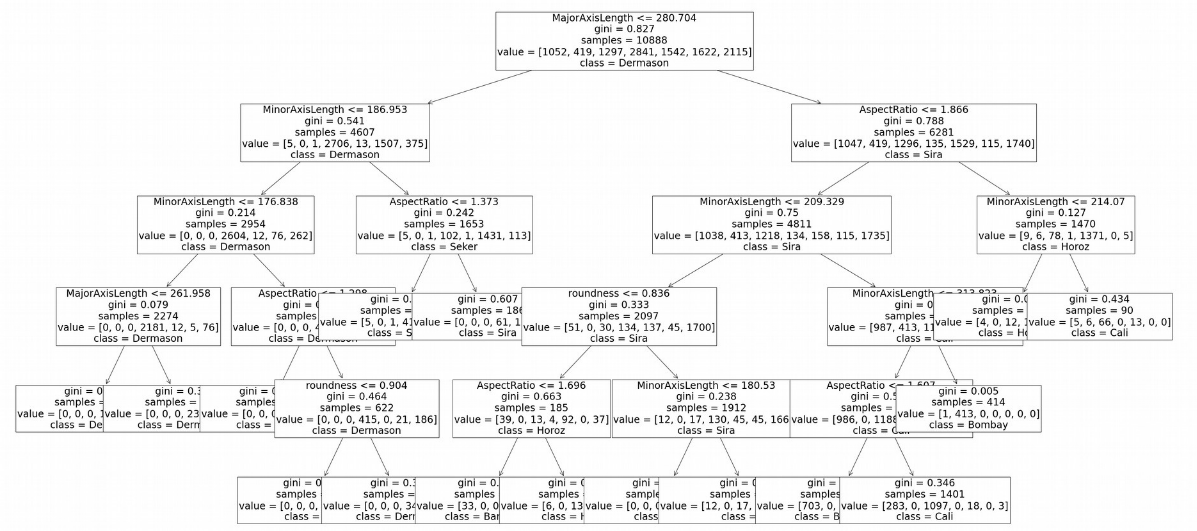


**Figure 2:** Visualisation of the obtained decision tree produced with the plot_tree method from sci-kit learn.

In another experiment with a big decision tree we set max depth =10 and max leaf nodes =30. The accuracy improved and reached 91.59% (Table 3).

**Table 3**
Decision trees parameters and performance

| Decision tree | Max depth | Max leaf nodes | Overall accuracy |
|---|---|---|---|
| small | 5 | 16 | 88.35% |
| big | 10 | 30 | 91.59% |

It can be seen that to obtain an accuracy similar to that reported in [1] with a decision tree, the tree would have to be much bigger (losing the main advantage of decision trees, i.e., the intuitive interpretation). One should keep in mind that in this work the amount of features has been reduced from 16 to 8. The excluded features were highly correlated to the retained features (being other geometrical measures of the same beans), so they accounted for little additional information. However, they present this information in a slightly different manner (”rotated”), making the task easier for the decision tree.

To see this better, assume that in some dataset we have two parameters A and B, and the classification is obvious, but it depends on the A/B ratio that is not explicitly present in the dataset. This can be hard for a decision tree to solve. The addition of an extra column, A/B, will add no new information to the dataset, but it will help the decision tree quite a bit. It can be supposed that in the dry bean case, the 8 removed categories contained little extra information, but they presented essentially the same information in a way more appropriate for the decision tree.

## 4.4. Random Forest

The random forest idea is as follows: take many decision trees (employing some randomness, so the trees differ) and let them vote. So the classification decision taken by a random forest is a decision taken by the most numerous group of decision trees in a random set of trees.

Usually a random forest performs better than a single decision tree. However, a random forest is considered a ”black-box” model being very hard to interpret.

A random forest of 150 decision trees was created. No restrictions on trees were applied. The accuracy obtained was 93.61%, the best so-far, better than the best accuracy reported in [1]. In addition, the training process was fast and took about 2 s, which 10-20 times faster than for SVC.

## 4.5. Voting Classifier

The idea of ”voting”, which by default is used in random forests, can be applied to any classifiers. There are 2 main ways of voting: ”hard” (straightforward, direct voting) and ”soft” (the votes are weighted depending on how confident the classifier is with its choice). Like in the case of a random forest, there is a good chance that the voting result will be more accurate than for any particular classifier.

The hard voting classifier was implemented using 3 classifiers described above: the radial kernel SVC, the ”big” decision tree, and the random forest.

The obtained accuracy was 92.80%. Thus in this case it is worse than for the random forest. This gives us a clue that voting should be used carefully and preferably with models exhibiting similar performance; otherwise ”stupid” models can outvote ”smart” models. It seems that this flaw of democracy does not only apply to human societies, but is more universal in nature.

## 4.6. Artificial Neural Network

Besides the (shallow) machine learning/data science methods presented above, a deep learning technique, the so-called dense artificial neural network, has also been tried.

For an artificial neural network the data needs additional treatment. First, the names of bean species were labelled with numbers and then these numbers 0-6 were codded as so called ”one-hot”.

The reason of using ”one-hot” encoding is well explained for example in [4] p. 376 or [6] pp. 190-194. The other operation is scaling, a standardisation or normalisation of the training data. The data (each feature) is centred around zero (by subtracting the average) and normalised (by dividing by the standard deviation). Standardisation is said to ease the training process and tends to bring in improvement in performance [5] p. 72.

Two architectures of ANN were tried. The first one is similar to the one described in [1], except that the input layer size in our case is 8 not 16. The network has 3 hidden layers with 17, 12, 3, neurons, respectively. Rectified Linear Unit was used as an activation function in hidden layers. The output layer has 7 neurons, one for each bean species. Sigmoid is the activation function for the output layer. The optimiser used was: RMSprop, and the loss function was the categorical cross entropy. The validation set was 20% of training set. The network was trained for 24 epochs. The architecture of this network and the training process are presented in figure 3.

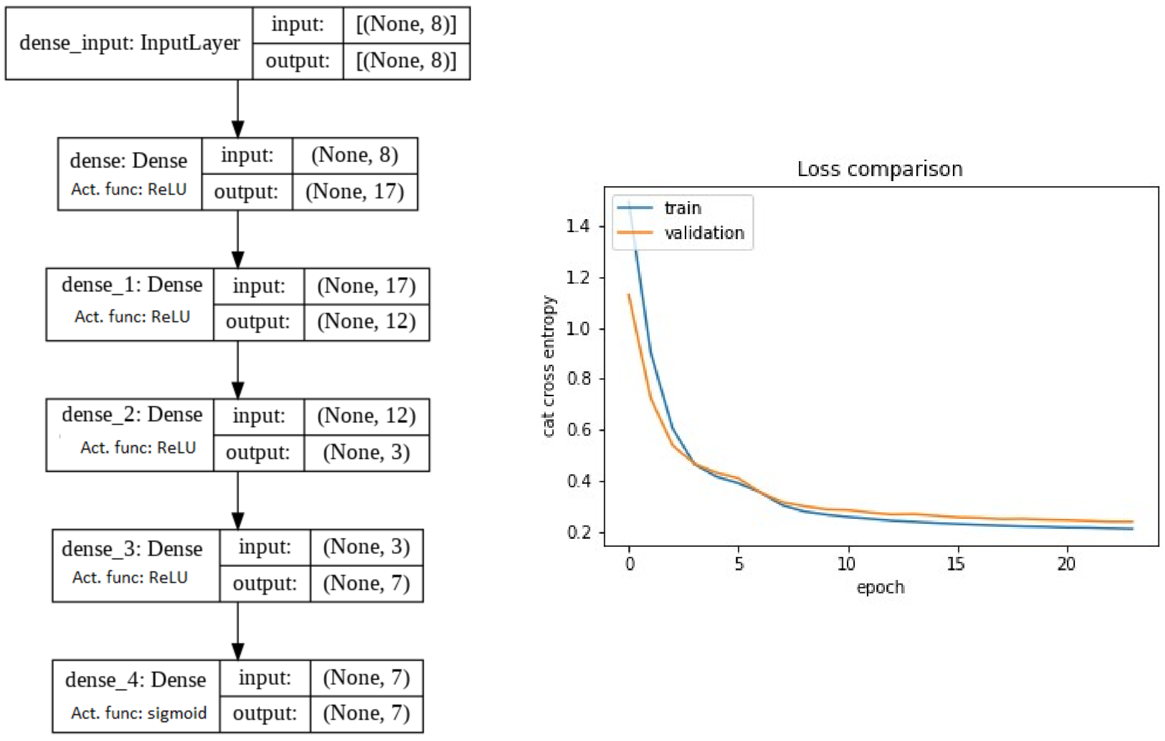


**Figure 3.** The architecture (left) and the training process (right) of the first ANN.

The overall accuracy was 92.58%. In an attempt to improve it, another ”bigger” ANN was tried. Besides the normal layers, a dropout layer was added. A dropout layer only works during the training and randomly ”cuts off” (sets to zero) some inputs. It is expected that dropout layers reduce the risk of overfitting [6] p.109. The architecture of the network and its training are shown in figure 4. The same optimiser and loss function were used: RMSprop and the categorical cross entropy, respectively. The network was also trained for 24 epochs. The overall accuracy was 92.77%, thus the improvement was not much.

## 5. Results and Conclusions

The dry beans dataset has been analysed by different machine learning and deep learning techniques. Table 4 shows the summary results.

It can be seen that in general the task of beans classification is a relatively simple task in terms of the necessary computing power. All training times were shorter than 1 minute using a free google-colab computer.

The accuracy of Naive Bayes is much worse than for the other methods. This is not surprising, as this method is known to be fast but not very accurate, and is suggested as a preliminary method to check if there is „something" in the data, rather than to do a full analysis. The other methods give accuracy in a relatively close range of 88.3-93.6%. This is comparable to [1] where the accuracy was in range 87.9-93.13%.

The authors in [1] trained their models with all 16 features. Here, some strongly correlated features were eliminated. The results show that this elimination has not decreased the accuracy.

The only technique, that suffered from this elimination to some extend seems to be decision tree. This is due to the fact that decision tree operates on one feature at a time and cannot "combine" features. See also the discussion in p. 4.3 devoted to the decision tree section.

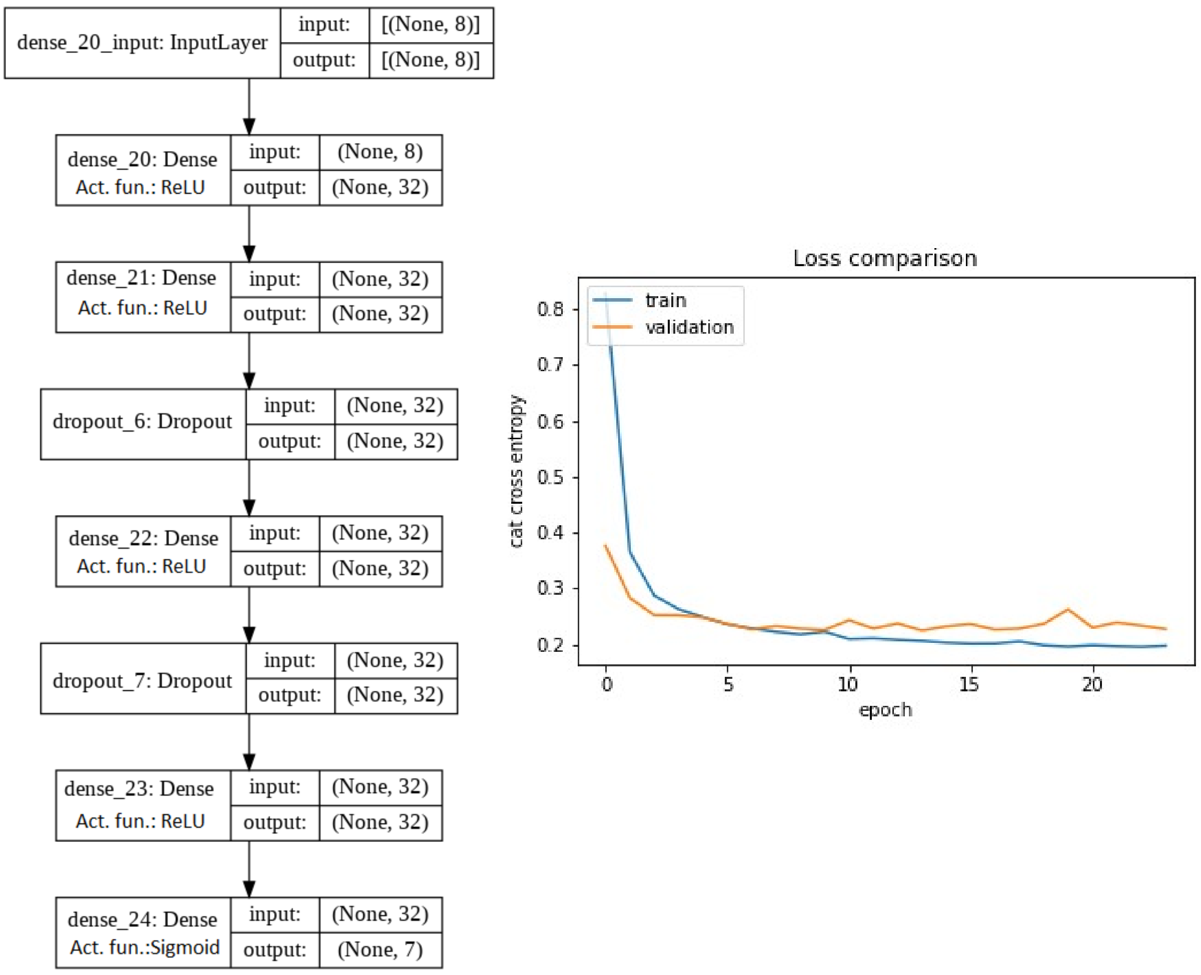

dense_20_input: InputLayer | input: [(None, 8)] | output: [(None, 8)]
dense_20: Dense Act. fun.: ReLU | input: (None, 8) | output: (None, 32)
dense_21: Dense Act. fun.: ReLU | input: (None, 32) | output: (None, 32)
dropout_6: Dropout | input: (None, 32) | output: (None, 32)
dense_22: Dense Act. fun.: ReLU | input: (None, 32) | output: (None, 32)
dropout_7: Dropout | input: (None, 32) | output: (None, 32)
dense_23: Dense Act. fun.: ReLU | input: (None, 32) | output: (None, 32)
dense_24: Dense Act. fun.:Sigmoid | input: (None, 32) | output: (None, 7)
Loss comparison
train
validation
cat cross entropy
epoch
0.2 0.3 0.4 0.5 0.6 0.7 0.8
0 5 10 15 20

**Figure 4.** The architecture (left) and the training process (right) of the second ANN.

One can see that the models vary strongly in the terms of the computing time. SVC and ANN (and also Voting Classifier, because it includes SVC) are the slowest learners. Random forest seems to be the best method in this case, as it perform best and its training time is also reasonable.

Confusion matrices provide a comfortable way to visualise results in more details and compare actual values with predicted ones. The confusion matrix for the random forest classifier (the best performer) will be discussed further. It is presented in figure 5. The most frequent mistakes were between Dermason and Sira (38 + 44). On the other hand, Bombay was classified perfectly which is not surprising. It is easy to notice that Bombay beans are significantly bigger than other species.

The dry beans dataset appeared to be an interesting dataset to demonstrate and compare ML techniques. Two ideas for further research:

- Deeper insight how and if the elimination of correlated features influences the ML training process. This study shows that there is little, if any, performance decrease. One may try to investigate if the elimination reduces the training time and how much.
- Despite ”manual” feature reduction, as done in this work, on may try to use PCA (the primary component analysis) to reduce the dimensionality of data and also analyse its influence on model performance (accuracy and training time)

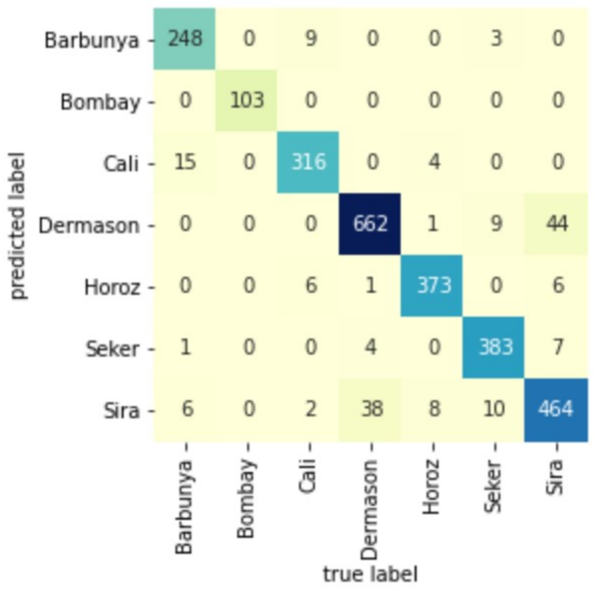


**Figure 5**. Test subset confusion matrix for the random forest classifier.

## 6. References

[1] Murat Koklu, Ilker Ali Ozkan, Multiclass classification of dry beans using computer vision and machine learning techniques, Computers and Electronics in Agriculture 174 (2020) 105507

[2] Dry beans dataset at UCI repository: https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset, access 23.06.2021

[3] Colab notebook containing computation scripts for this work: https://colab.research.google.com/drive/11X6VevSMybenGkRqK1Xj_1EJmU3vomFB?usp=sharing

[4] Jake VanderPlass, Python Data Science Handbook, O’Reilly, 2017

[5] Aurelien Geron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, O’Reilly, 2019

[6] Francois Chollet, Deep Learning with Python, Manning Publications, 2018

# Author Index