Proceedings of the Federated Conference on Computer Science and Information Systems pp. 145–154
DOI: 10.15439/2020F168
ISSN 2300-5963 ACSIS, Vol. 21

Introducing LogDL – Log Description Language for Insights from Complex Data

Maciej Świechowski
QED Software, Warsaw, Poland
Email: maciej.swiechowski@qed.pl

Dominik Ślęzak
Institute of Informatics
University of Warsaw, Poland

***Abstract*—We propose a new logic-based language called *Log Description Language* (LogDL), designed to be a medium for the knowledge discovery workflows over complex data sets. It makes it possible to operate with the original data along with machine-learning-driven insights expressed as facts and rules, regarded as so-called *descriptive logs* characterizing the observed processes in real or virtual environments. LogDL is inspired by the research at the border of AI and games, precisely by *Game Description Language* (GDL) that was developed for *General Game Playing* (GGP). We emphasize that such formal frameworks for analyzing the gameplay data are a good prerequisite for the case of real, "not digital" processes. We also refer to *Fogs of War* (FoW) – our upcoming project related to AI in video games with limited information – whereby LogDL will be used as well.**

I. Introduction

COMPUTER languages have played crucial role in the way how people use computers and interact with them. The most common types of languages are general-purpose programming languages such as Java or C++, query (data manipulation) languages such as SQL, which are often domain specific, and description (markup) languages such as XML.

In this paper, we present a new language – *Log Description Language* (LogDL). The term "Log" was chosen deliberately as it refers to both *logic*, because LogDL is logic-based, and *logs*, i.e. information obtained from and about some process (a network activity, a video game, etc.). This ambiguity accentuates the fact that LogDL is perfect for representing both the static knowledge stored in a form of database and dynamic knowledge, i.e. new insights, inferred dynamically by reasoning mechanisms based on AI, logic, machine learning (ML) as well as computational intelligence (CI).

LogDL allows us not only for richer data representation – in terms of *descriptive logs* (d-logs in short) – but also for formal logical reasoning, spatio-temporal analysis and interactions. Because we keep LogDL as human-friendly as possible, it may be used to guide the discovery algorithms as well as for preferences specification, complex querying, data labelling and augmentation. It is also designed to be evolvable to make it a good fit for large-scale evolutionary algorithms (EA).

Modern knowledge discovery approaches need to deal with large multimodal process-related and spatio-temporal data sources such as games, sensors, monitoring, UI controllers, etc. Though the original data shall remain unstructured or multi-structured, the layer of insights can take a form of collections of well-established facts, rules and formulas expressed in LogDL – the aforementioned d-logs describing the observed processes and activities in real or virtual environments. Figure I illustrates the usage of LogDL and how it can be involved in various data-related operations and activities.

LogDL provides building blocks (e.g. facts, operators, rules) which algorithms may use, constraints (expressed by means of e.g. domains and rules) within which they operate and some built-in concepts such as time that can be interpreted automatically. Any language that is not text-based, e.g. with dynamic elements, needs a dedicated interpreter. LogDL has a few dynamic elements – already-mentioned rules and operators, interactive querying and logical reasoning based on a current knowledge base. In automated scenario, an interpreter is used by algorithms that can operate with a much higher rate than humans. Therefore, we take into consideration yet another aspect – any constructs having negative impact on performance of LogDL interpreters must be avoided.

In summary, LogDL shall enable us to: 1) be both human-friendly and computer-friendly (like scripting languages); 2) represent knowledge, e.g. from diagnostic logs, in a structured way (like SQL or *Game Description Language* – GDL); 3) perform analytical queries directly in LogDL (like in dedicated software for analytics) and represent results of those queries; 4) perform logical reasoning (like in Prolog); 5) enrich the data by custom rules and facts that can be formulated directly in LogDL; 6) provide a framework for AI/ML/CI-based knowledge discovery algorithms (like numpy in Python).

The paper is organized as follows. Section II outlines the related works. Sections III-IV are devoted to GDL and its limitations. Sections V-VII introduce some of fundamental notions related to LogDL. Sections VIII-IX are devoted to potential applications of LogDL in video games and "real world". As this is the first article about LogDL, we let these sections occupy its significant portion in order to motivate a new language. Section X concludes our work with some open questions and comments. More in-depth specifications and properties will be published in future papers.

II. Related Work

The most closely related work concerns GDL that has inspired us to develop LogDL. GDL was proposed as a way to represent

This work was co-financed by EU Smart Growth Operational Programme 2014-2020 under GameINN project POIR.01.02.00-00-0184/17-00.

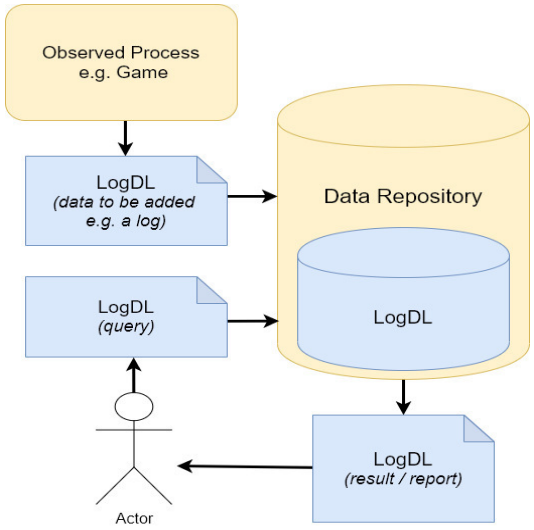

Fig. 1. The usage of LogDL from the data and knowledge processing perspectives. It stands as a medium to represent insights (in form of d-logs) derived from the original data, store them in a way which is efficiently integrated with that data and provide the means for human-computer interaction.

game rules [1]. We devote Section III entirely to it. GDL has been used to this day in the aforementioned GGP research. It is a first-order logic language that is heavily inspired by Datalog which is a logical database language [2]. GDL and Datalog are similar to each other, although not equivalent because GDL has constructions that are not a part of Datalog syntax.

One can emulate Datalog in Prolog [3], although these languages use different semantic conventions. A conversion to Prolog requires three things. Firstly, the notation is different, so each GDL element must be mapped to a Prolog counterpart. Secondly, additional code has to be written in Prolog to handle game-specific logic. Thirdly, there are special cases of negation that are handled differently in Prolog and GDL, so they have to be rewritten for Prolog. There have been numerous extensions to Datalog proposed, such as e.g. Datafun [4] – a functional oriented version and Dedalus [5] which is aimed at rich distributed services and tests for correctness.

Such languages as e.g. Ludemic-GDL [6] and Answer Set Programming [7] were proposed as well. Generally, GDL is a significant step forward as it allows to deal with the aforementioned game rules in an abstracted way decoupled from any particular game. It made it possible to create universal game-playing programs that accept games as input parameters. Nevertheless, GDL was used only in research so far.

We designed LogDL with the aim of taking the best from GDL and optimizing the rest to make it useful for the game industry and "real-world" applications. The usefulness of a language is often reflected in how efficient interpreters or compilers can be developed. Works such as [8] focus on the process of creating such interpreters and provide a good source of knowledge about GDL-style languages too. In [9], the performance of a few interpreters is compared.

Outside of the game research, there are commercial data analytics solutions such as e.g. Splunk [10]. These are services designed for a different purpose than LogDL, although it is certainly worth combining those two conceptual layers of data processing and reasoning to create efficient AI pipelines. Additionally, it is worth comparing some ideas of LogDL to those behind Complex Event Processing (CEP) [11].

LogDL is a logic-based language with a built-in reasoning mechanism. There exist commercial logical languages such as the already-discussed Prolog or 4QL [12]. Most of our comparison between GDL and LogDL translates to those languages as they are symbolic-based. From the perspective of *Fogs of War* (FoW) – our upcoming project related to AI in video games with limited information – it is also useful to look at some formal frameworks utilizing e.g. 4QL to reason about unknown and inconsistent situations [13].

There are numerous formal representations of insights derivable from the complex data. Let us mention about GVGDL – one more extension of GDL aimed at dealing with video games [14]. One may think about it as a step toward modelling "real world", though it still refers to "digital reality". There are also attempts to adapt spatio-temporal logics to model machine-generated processes, e.g. network events [15].

Last but not least, we shall refer to a collection of inspiring use-cases and potential applications of LogDL that revolve around learning new concepts and extracting new knowledge through logical reasoning, association rule mining as well as applying search-based and learning-based methods to construct new LogDL-based rules and facts. Such rules and facts constitute new knowledge that can be used for prediction, approximation and explanation [16]. This kind of strategy fits well into some of hot trends in AI, such as e.g. metaconcept learning and neuro-symbolic machine learning [17].

III. Game Description Language

GDL includes predefined keywords: *role*, *init*, *true*, *next*, *legal*, *does*, *terminal*, *goal*, *distinct*. Most of them can be treated as domain-specific extensions required to build a forward-model, i.e. simulate a game. When a GDL program is interpreted at run-time, facts can appear in three ways [18]:

- *Constant facts* are defined directly in the GDL code and they are considered *true* for the whole game. They can be regarded as "the laws of physics".
- *State facts* are parts of dynamic game states. They are initialized by *init* rules. They are cleared in each game's step, their new set is derived by *next* rules.
- *Temporary facts* are produced by non-keyworded rules. The set of such facts is derived dynamically. They are needed only temporarily in the logical resolution process initiated by one of keyworded rules.

In GGP, GDL is written using prefix Knowledge Interchange Format (KIF). It can also be represented by infix KIF or Lisp S-expressions. They are all syntactically equivalent.

A. Facts

On the top-most level, any GDL transcript consists only of *facts* and *rules*. For instance, the fact that a soldier a with

rocket launcher is present in region denoted by coordinates 2 and 3 could be defined using the following form:

```
(region 2 3 soldier rocket_launcher)
```

This is a proposition with symbols. The first symbol denotes the proposition's name. The remaining ones are arguments (also called attributes). Each proposition with the same name must have the same number of arguments. A proposition can be viewed as relation, i.e. all symbols together are in relation. There cannot be multiple facts with exactly the same symbols. Such facts would be interpreted as a single one.

GDL allows for nested facts, e.g "(weapon silver sword)" below describes a weapon of a soldier:

```
(soldier1 2 3 (weapon silver sword))
```

Symbols have no type – they are all plain text. Apart from predefined keywords, symbols have no meaning – their interpretation is purely due to humans. In particular, game rules in GGP are often obfuscated in order to avoid any game-specific reasoning based on the choice of words, e.g. *board*. Game mechanics do not change if each unique symbol that is not a keyword is consistently changed to another one.

B. Rules

A rule in GDL can be specified as follows:

```
(<= (empty_region ?x ?y)
(true(region ?x ?y soldier ?weapon))
```

Rules are defined using the $<=$ operator. The first proposition after it is the consequence. A rule is *true* or *false*. In addition it may produce results in form of facts that become *true*. The consequence defines a structure of such facts.

The remaining propositions are conditions that have to be satisfied in order for the rule to hold. Conditions, in contrast to facts defined directly in GDL description, can contain variables denoted by a symbol starting with ?. The variables are substituted by constant symbols in the resolution process. For example, $?x$ and $?y$ variables will be substituted by symbols 2 and 3, if the following fact holds:

```
(region 2 3 soldier <anything>)
```

If many facts of type *region* satisfy the query, then there will be many variable bindings produced. GDL realizes the *variable unification property*: variables with the same name receive the same bindings in the rule's scope. This rule would only consider regions with coordinates equal to each other:

```
(<= (empty_region ?x ?x)
(true(region ?x ?x soldier ?weapon))
```

Conditions may refer to other rules. Facts may be *true* either through explicit specification or through rules, e.g.:

```
(cat lion)
(<= (cat ?y)  (true(mammal ?y)
              (true(domestic ?y)
              (true(not(dog ?y))
```

The complete game world is defined by propositions which are *true*. There exists the completeness property which means that everything what cannot be derived as *true* from the available rules and facts at particular moment is *false*. Thus, GDL follows so-called *closed world assumption*.

IV. Limitations of GDL

Although GDL became useful in the game AI research, it has several drawbacks that hamper its wider usage. We point out the major limitations of GDL that have inspired us to develop LogDL in order to make it more applicable.

A. Poor Interpretation Performance

This is one of crucial limitations of GDL to make it more applicable in the AI workflows. The fastest GDL interpreters are based on Prolog and propositional networks [19] which instantiate all possible variables and lead to massive structures. There are cases (e.g. bigger games in GGP) in which the propositional network representation is totally infeasible.

Simulations of games in GDL are usually significantly slower than those performed by dedicated implementations. One of the reasons is that GDL is purely symbolic language. There are no built-in types that allow for taking advantage of CPU optimizations. Every piece of logic has to be written as GDL rules, whereas some aspects could be implemented more efficiently using a lower level language. Many well-established algorithms cannot be implemented efficiently because of lack of data structures such as heap, priority queue, etc.

A good example is lack of simple integer comparison operator. In GDL, it is usually implemented as follows:

```
(succ 0 1) (succ 1 2) (succ 2 3)
(<= (greater ?a ?b) (succ ?b ?a))
(<= (greater ?a ?b) (distinct ?a ?b)
                    (succ ?c ?a)
                    (greater ?c ?b))
```

B. Lack of Continuous Domains and Infinity

Another consequence of GDL being a symbolic language is that it cannot deal with continuous domains such as real numbers. For instance, it is not possible to define multiplication on real numbers, as it would require to define all possible results. There is no way of emitting new symbols that would represent the results of such operations and, even if the was, the algorithm of multiplication would have to implemented from scratch purely based on symbolic logic.

C. Lack of Stochasticity

GDL suits *finite*, *deterministic* and *synchronous* games. Accordingly, each rule in GDL is deterministic. It is either *true* or *false* given the current state. There are is no concept of randomness and no random number generators. It is debatable whether the world is deterministic or not, nevertheless, stochasticity and fuzziness are useful in modelling many real world phenomena. Moreover, there are many video games with randomness and incomplete information.

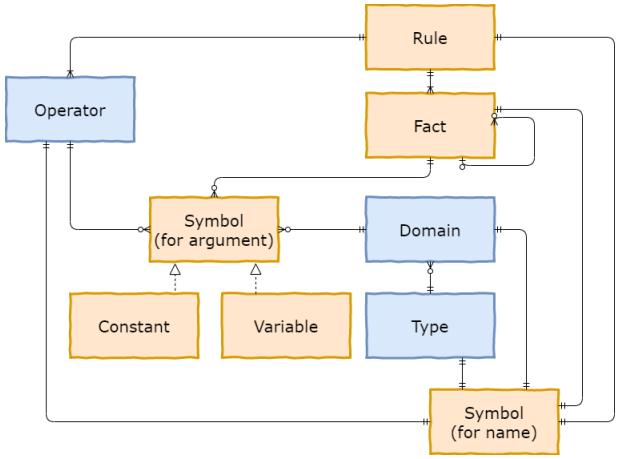

Fig. 2. Basic elements of LogDL.

D. Lack of Time-based Reasoning

In GDL, the term "synchronous" means that all players submit their actions simultaneously and then the game state is updated. Such updates are performed in consecutive frames. There is no notion of continuous time flow. We can tell that some fact appeared later than the other, but we cannot tell when exactly it happened and how much time have elapsed. There is no concept of time interval between frames.

E. Issues with Advanced Algorithms

Symbolic description without reflection, types and metadata is not well-suited for more sophisticated algorithms. For instance, let us consider the problem of rule evolution – quite important in the game industry and e.g. process mining – which could be achieved using evolutionary algorithms (EA) [20]. Consider a specific case of mutation operator that randomly perturbs a value of a certain argument. Such operator would greatly benefit from having a domain to choose values from. GDL does not support numerical domains.

As another example, crossover operator could replace a rule's condition to a different one that fits it. However, in GDL it is hard to determine whether a condition "fits" – there is nothing that could be tested against the existing rules for potential variable unifications. Moreover, such replacement could result in unexpected behavior such as a long computational time or even an infinite loop because of recursion without a proper stop condition. In GDL, there are no control statements such as *IF* and recursion caused by the interplay of rules and conditions terminates only when all its branches are evaluated as *false* at some point in the resolution process.

That said, programs written in generic programming languages such as C++ can be even more difficult to manipulate by EA, for different reasons. Although there are classes and types, there are too many degrees of freedom in how the code can be constructed. We believe that the GDL structure with rules, facts and conditions would be suitable for EA-style manipulation if only the language was extended by additional metadata. This is one of inspirations for LogDL.

F. Bloated Description

Due to lack of domain-specific operators the reasoning performance can be higher compared to programs in general purpose languages. GDL descriptions can be also extensive if they rely on concepts that are not easy to map to logical rules.

V. Log Description Language

In this section, we outline some selected ideas behind LogDL. Whenever useful, we do it in comparison to GDL. Figure 2 depicts the components of our new language. Operators, domains and types are distinguished using colors because they are not present in GDL. Symbols are similar to thosw in GDL. Symbols for arguments can either be constants (fixed literals) or variables (starting with "?"). Names can only be fixed literals and they are subject to additional uniqueness constraints, e.g. in order to avoid rules and operators with the same names or other kinds of ambiguities.

A. Facts

Facts in LogDL are similar to the GDL ones. However, we simplified the notation to make it more similar to the JSON notation and we also introduced *metadata* for arguments. The metadata consists of the argument's *name* and *domain* over a *type*, so type is indirectly part of metadata too. Domains are discussed in the next subsection. Names can be declared in a few ways, e.g. globally with simple configuration.

In GDL, arguments are specified in order of appearance, both in queries (conditions) and in implicit definitions. In LogDL, arguments can be addressed by name. The name is mapped onto the corresponding argument's index. Unnamed arguments are mapped onto those that have not yet been mapped. Here are equivalent fact definitions in LogDL:

```
1: region{2 3 soldier sword}
2: region{x: 2 y: 3 unit: soldier
       weapon: sword}
3: region{x: 2 3 soldier sword}
4: region{y: 3 2 soldier sword}
5: region{weapon: sword 2 3 soldier}
6: region{weapon: sword y: 3 x: 2 soldier}
7: region{x: 2 3 weapon: sword soldier}
```

To avoid ambiguity, i.e. which fragment is part of name or value of a specific argument, there are lows regarding a use of quotation marks and how interpreter reads characters.

The names of arguments are part of metadata. Figure 3 shows a simplified way how the data can be attributed to a particular type of fact. It also puts forward the idea of decoupling names from the data. In the implementation of LogDL, there will be optimizations for data storage such as using hash functions for logical reasoning purposes and indices that increase the performance of the expected queries.

B. Domains

Domains are associated with arguments of facts, i.e. the allowed range of their possible instantiations and the whole

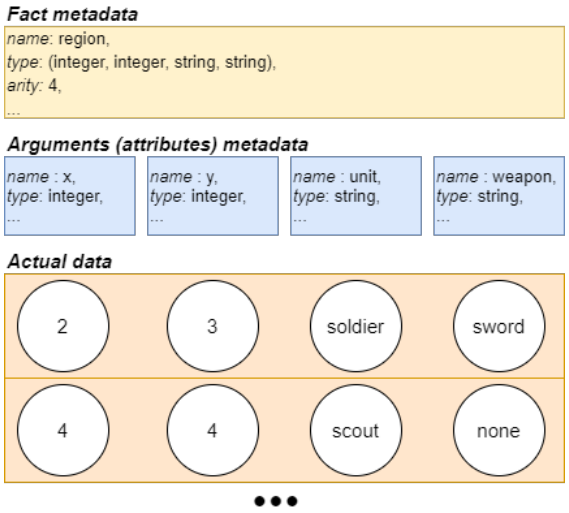

Fig. 3. Data and metadata stored for a given type of facts.

facts (complex domains). A domain is defined over a specific type. The following types are possible in LogDL:

1) Basic types – *integer*, *double*, *boolean*, *char*, *string*.
2) Complex types – tuples of other types, e.g. (*integer*, *integer*). They can be implicitly created by rules producing facts with the respective types of arguments used.

Types have to be defined as part of metadata description. Domain specifications are optional. If domains are not specified, then LogDL interpreters will make educated guess based on the data and arguments' usage. *Boolean* domain comprises always of two values: *true*, *false*. For numerical types such as *integer*, *double*, *char* (0-255), domains are defined by:

- The minimum and maximum values.
- Whether min/max values are included or excluded.
- Stride, i.e. distance between consecutive values (for *integer* and *char* it must be an integer number).

Domains can be also defined by the sets of allowed and disallowed values. If a symbol appears in both above sets, then it is considered disallowed. For numerical types, such set specifications can be combined with the range definitions. For example, we can consider a domain that has integer values from the $[0, 10]$ interval, but excluding 2 and 3.

Types and domains are part of LogDL for two main reasons:

1) They enable to introduce operators that work with specific types, which in turn allows for optimized implementation. Low performance, especially in case of math operations, is one of the main limitations of GDL.
2) They mimic what is called reflection in programming languages. A formalized structure enables us to use the AI/CI techniques that operate on the LogDL description, in particular search-/population-based EA methods.

C. Operators

Operators in LogDL can be treated as predefined functions. They are not present in GDL and we have already discussed why we believe they are important. The idea behind them is to perform certain operations more efficiently than by generic symbol manipulation and to be able to extend our language with a specialized application-specific logic.

LogDL is designed to be easily extendable by functions that can be used as conditions for rules. We are not saying that the language schema is extendable but rather a way the things are computed. This is a practical approach that also plays along with using LogDL in automated AI pipelines. Operators (just like rules) are building blocks to be used by the AI/CI algorithms, in particular EA approaches. Due to the introduction of domains and types, the algorithms can be aware of the context in which a particular operator is used.

Operators may use variables, facts or results of other operators as input parameters. The user specifies types for inputs and outputs. From LogDL point of view, the implementation is treated as a black-box. There are two categories:

1) *Built-in operators* have reserved names, i.e. keywords, in the LogDL language. By the standardization, their implementation has to be already provided by a LogDL interpreter. They are ready to be used.
2) *Custom operators* are not part of the standard. They can be declared and implemented as third-party extension to the interpreter for specific application.

Examples of built-in operators are: *logical* (conjunction, alternative, negation, etc.), *fact-related* (count, top N, getArity, etc.) and *mathematical* ($+$, $-$, square root, mean, etc.).

LogDL is still in its R&D phase, so the list of operators will continue to grow. Custom operators are provided to the LogDL interpreter by code that is compatible with the particular interpreter implementation. For example, if an already compiled interpreter is used e.g. as .NET Assembly or C++ library on Windows, then a dynamic link library (DLL) with the custom operators implementation should be provided.

D. Time and Data Organization

In contrast to GDL, LogDL includes the notion of time. It is defined by the reserved fact with the name "time", which has one argument of type *double*. For example:

```
time{5.0}
```

This is a time-stamp. It makes sense only if certain assumptions are made about the process that LogDL describes.

As GDL followed some constraints required to build a simulator of a game written in it, LogDL uses its specific conventions as well. First, it is assumed that LogDL is used to express the knowledge from and about processes. This suits use-cases that will be outlined in further sections.

Second, it is assumed that the process states are grouped within so-called *activities*. The concept of time is valid within the scope of the activity. Examples of activities can be: a game session, logs from a networking device, changing prices of specific stocks, health data of a specific patient. Any activity is described by a changing state in time. Another way to interpret activity is the correlated data observed chronologically. It enables to perform causal inference between events that

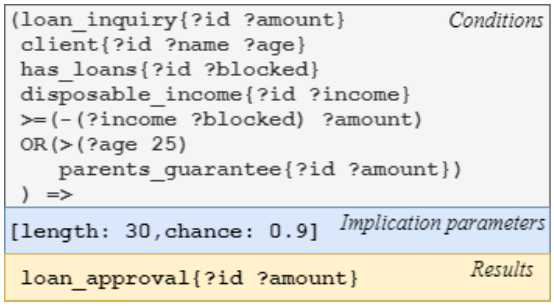

Fig. 4. Simple example of a rule in LogDL.

happen within the activity. There is a strong analogy between an activity and a Markov Decision Process [21].

Activities are grouped together within so-called *worlds*. With worlds, general rules (i.e. law of physics) of certain process are associated. For example, a world can denote a particular game, map in the game or a specific type of a networking device (e.g. router). The idea is that the data within the same *world* but distinct *activities* can be used to find patterns and extract knowledge about the process.

VI. Rules in LogDL

Rules are valid in the scope of *worlds*. Each rule consists of the following three parts: *conditions*, *implication parameters* and *results*. LogDL rule's structure is as follows:

```
(conditions) =>
[implication_parameters]{results}
```

The respective parts are distinguished with different colors in Figure 4. Implication parameters are optional to specify – there are default values in our LogDL language.

A. Conditions

In LogDL, conditions are either *fact propositions*, like in GDL, or *operators* that specifically return a boolean value. This is new compared to GDL. Facts and operators used as conditions may contain variables. Each condition is evaluated as *true* or *false*. If it contains variables, it provides instantiations (bindings) of those variables just like in GDL.

There are certain constraints imposed on rule conditions, which makes reasoning simpler and potentially faster. Conditions are checked in the order they are defined. Each condition may introduce new variables and use the already introduced ones. Naturally, a condition may be just a check, i.e. without any variables at all. The introduced variables are those with names that have not been used so far by conditions checked earlier. We refer to Table I to see how introduction of variables and their usage in conditions is defined. Variables in boldface are the ones that are introduced by a condition. The results of checking conditions in LogDL are:

- A boolean value indicating whether conditions as a whole are evaluated as *true* or *false*.
- If *true*: a list of records satisfying all variables.

Let us now list constraints that allow us for efficient implementation of LogDL interpreters. They are imposed on conditions starting from the second one in the order of appearance:

1) Operators cannot introduce new variables. However, they can use the already existing ones.
2) Conditions defined by facts should either: a) use at least one already introduced variable; b) do not have variables at all; c) have only one valid instantiation.

B. Results

This part of our language is analogous to GDL, i.e. it consists of specification of facts that become *true* if the rule is satisfied. The produced facts can include constants and any variables that have been introduced in the previous subsection. Implication parameters that are discussed below define the probability and time constraints, i.e. "from when" and "for how long" the results are considered *true*.

C. Implication Parameters

This aspect is new compared to GDL. It is a construction that describes additional context how results are created. All such parameters have default values in order to allow for concise expressions. They consist of the following elements:

1) *resultStartTime* is the expected delay time between the moment when conditions are met and results are produced. It is measured in the same units as LogDL's *time*. The default value is equal to 0.
2) *length* is the time that results are expected to hold, i.e. from *resultStartTime* to *resultStartTime* + *length*. This can be set to a real value or two special values *UPKEEP* and *PERSISTENT*. *UPKEEP* denotes that the rule effects will hold as long as its conditions do. The *PERSISTENT* option denotes that the rule effects will be persistent in the scope of the context the rule is launched.
3) *chance* is conditional probability. If conditions are met, then results will be produced with probability equal to this value. The default is 1, i.e. if not specified otherwise, rules will always produce results immediately and results will be valid as long as conditions are valid.

We may also consider a fuzzy component, e.g. by reserving the first argument of each fact as its satisfaction degree $\in [0, 1]$. In contrast to purely symbolic GDL, LogDL could handle fuzzy membership functions, fuzzy literals and the overall *computing with words* paradigm [22]. It might be worthwhile to make it possible to configure the logical reasoning mechanism, so it uses fuzzy norms to determine whether a rule is satisfied and to what degree. Rules and operators could be then used to perform the fuzzification and defuzzification processes.

VII. LogDL Compilation

Figure 5 shows typical environment for the LogDL usage. *LogDL description* is a code written in LogDL that is based on facts, rules and operators. *LogDL metadata* are definitions of global aspects such as domains and types. Both the LogDL program and metadata together form *data repository*. One of unique aspects of languages such as LogDL or GDL is that

TABLE I
AN EXAMPLE OF CONDITIONS OF A RULE. EACH ROW IS A TOP-LEVEL CONDITION. THE LAST TWO ROWS CONTAIN NESTED CONDITION.THE THIRD COLUMN DENOTES THE TRACKING OF THE INTRODUCED VARIABLES AFTER EACH CONDITION IS TAKEN INTO ACCOUNT.

Condition	Condition type	No. of variables new / total
loan_inquiry{**?id ?amount**}	fact	2 / 2
client{?id **?name ?age**}	fact	2 / 4
has_loans{?id **?blocked**}	fact	1 / 5
disposable_income{?id **?income**}	fact	1 / 6
>=(-(?income ?blocked) ?amount)	operator using nested operator	0 / 6
OR(>(?age 25) parents_guarantee?id ?amount))	operator nested operator nested fact	0 / 6

the code and the data are essentially inseparable, e.g. a standalone definition of a fact means that it is *true*, therefore it is a part of data.

In our proposal, which is still in development, we designed various ways to provide *LogDL description* to a repository, e.g. using a file (*.logdl), interactively as in e.g. Python console, or programmatically through a dedicated API. The last case can be useful e.g. when a game, or another data provider, is able to log the structured data in LogDL format.

LogDL metadata can be either provided through files or in a GUI-based *administration suite* which is the preferred approach The idea behind it is to have an easy to use, graphical tool to define the structure of particular problem to be modelled using LogDL. It can allow for defining all kinds of metadata, organizing the data by *worlds* and *activities* (see Section V-D), viewing / updating / deleting all kinds of the data as well as providing LogDL compiler with implementation of custom operators and making them visible for LogDL.

We assume dedicated LogDL compilers written in a few programming languages. Having a compiler for specific language allows for two features: (1) extending LogDL with custom operators with a native implementation as well as (2) having access to the interpreter from a code in the host language. (1) requires some metadata to bind LogDL expression with functions exported from a library that contains implementation. This includes providing a way to call the function, reserve its corresponding operator name as well as provide metadata for its arguments. All of this will be possible to set up via the GUI-based *administration suite*.

When a LogDL program is available, the user can utilize the interpreter either as interactive program (like in Python console) or directly from the user's application through API. The latter is possible if there exists an interpreter for a programming language of the host application. We will provide bindings to the most popular languages. First and foremost, the interpreter allows for interactive queries based on rules, facts and operators. The user may e.g. wish to perform a logical resolution, check a hypothesis or just fetch or count the specific data. The program can also be compiled without the interactive interpreter. In such case, queries must be provided beforehand, e.g. via a file. Then the compiler will compile the file to a program returning results of specified queries.

VIII. MOTIVATION – VIDEO GAMES

The subsequent sections are devoted to potential LogDL use-cases. First, we motivate why logic, structured logs and LogDL can be useful for the game industry domain. In particular, we intend to apply it in our upcoming projects. We have already carried out a prototype implementation aimed at verification of the expressive power of LogDL, its integrity and ease of use. Selected aspects have also been implemented in an optimized way, together with the tests measuring what kind of performance can be expected when the whole ecosystem outlined in Section VII is developed and integrated.

A. AI Development

The traditional approach to AI in commercial video games is extensively based on heuristics. A heuristic can be part of search algorithms, functionally realized by finite state machines or in form of behavior trees [23]. Heuristics require the expert knowledge expressed by means of some important aspects of the game environment, preferences, weights, probabilities, threshold values and utility values. For example – *it is worth shooting the opponent with weight X if the distance to it is less than Y and otherwise it is worth running away*. All components and parameters are usually chosen by a repetitive trial-and-error method or chosen arbitrarily.

LogDL promotes a different, evidence-based approach. Human game testers are usually the biggest group of people involved in the game production process. Logs from such test runs can provide a valuable data source, based on which game creators may build and tune heuristics. LogDL is particularly useful to represent facts from game replays and new insights that can be discovered from them. For example it may find choke-points on maps, usefulness of in-game items and how different strategies work against each other.

LogDL was in part inspired by our experience with the *Grail* library aimed at developing AI in video games [24]. Grail supports algorithms such as Utility AI and Monte Carlo Tree Search (MCTS) [25] which can be used as action-selection mechanism for AI players. Utility AI is based on curves that define relationship between an action's utility and a given consideration. Identification of considerations can be extracted from logs thanks to LogDL. Similarly, in video games MCTS is typically optimized with heuristics that provide early cut-off (i.e. scores in non-terminal states), limit the number of considered actions or guide the search process.

B. Game Testing and QA

When logs from games are available, LogDL can be a valuable tool for Quality Assurance (QA) [26]. Firstly, it can be used for balancing, i.e. identifying too strong aspects of the game, e.g. a weapon that inflicts too much damage or an enemy that cannot be reliably defeated. This is a similar case to developing the AI, but this time we are interested in other types of insights from the data. Secondly, it can be used to verify hypotheses about the game. For example a hypothesis may state that *60%*

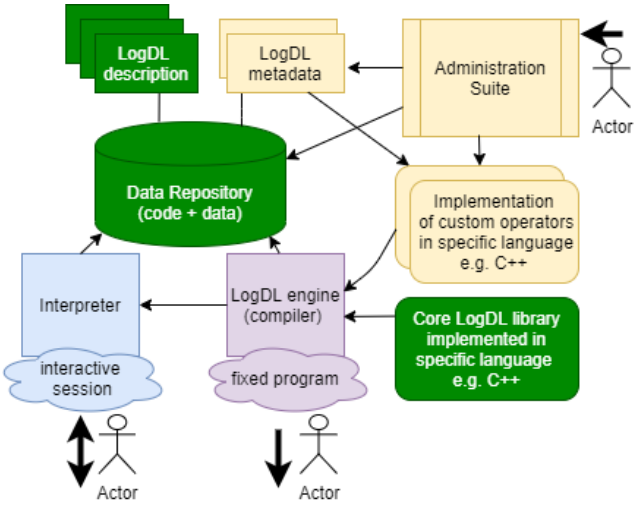

Fig. 5. Technical overview of the usage of LogDL.

of the time a player is able to finish a particular level without losing life. LogDL is particularly suitable to query the game data with its structured form and the notion of time and space. Thirdly, the QA requirements can be expressed as queries and rules directly in LogDL. This enables to build an automated or semi-automated QA pipeline similar to continuous software integration systems. Lastly, LogDL can aid automated QA provided that the testing agents (bots) are available. The data in LogDL can be analyzed on the fly thanks to the reasoning mechanisms and custom rules provided once for the testing process. It makes it possible to guide agents in real-time in their testing behavior. For example, it can be revealed that certain game areas or interactions need to be tested more thoroughly. Although the automated testing is not common in the video game industry yet, it will be more popular in future with the AI becoming smarter.

C. Game Analytics

Game analytics and e-sport are one of the hottest topics not only in games but in entertainment, in general. LogDL allows for transformation of raw information collected from games into useful information that can be presented to players, teams, sponsors or game development companies. On a technical level, this use-case is related to the previous ones, i.e. developing AI, testing and QA. LogDL is designed to be queried interactively, used in the knowledge discovery processes and to provide new insights from the data. Such insights can be utilized in game analytics and coaching to increase human players' skills [27] or to comment e-sport games.

Fact-based and rule-based description can integrate the gameplay data with metadata about players as well as maps and concepts that are related to the given game but not actually present within it. It also includes the data obtained as input from players, game developers and data scientists. The rules can be continuously refined in an incremental self-improving process with human feedback in the loop. For example, logical analysis could find inconsistencies, potential candidates for anomalies, unexpected correlations, new strategies or just new concepts that human experts could use and name.

D. Explainability in Games

AI behavior that is understood and trusted is not only a requirement for high-risk applications such as e.g. those in medical field. Game studios extensively test the AI in their games to minimize the risk of players encountering unexpected or unnatural behavior in the shipped game [28].

The other facet of explainability refers to "cheating AI". In multi-player games with bots, players often do not believe that their AI opponents played with the same rules. Cheating AI is so prevalent in games without perfect information (especially in real-time strategy games) to make up for poor strategic skills, that if a bot in a particular game is competent, then it is often accused of cheating. Hidden information is important, because players cannot verify during the game whether the bot plays along the same game rules and limitations.

FoW – our already-mentioned new R&D project – will concentrate on believable AI bots in games with hidden information and on explaining their limitations and reasoning processes behind actions. LogDL is good for such applications, particularly when it serves for representing game logs too. It can be used to express the knowledge of bots and explanatory rules of their behaviors in an accurate or approximate way depending on what kind of AI algorithms are applied.

E. Multi-Agent Communication

Let us refer to FoW once again, now from the viewpoint of creating tools for game developers, which are aimed at reasoning under uncertainty (hidden and stochastic information), managing and updating beliefs of computer agents as well as their coordination and communication in a multi-agent environment [29]. One of our goals is to simulate human-like behavior in games with imperfect information with human-plausible simulation of perception. Formal languages, e.g. the aforementioned 4QL, can be used for modelling multi-agent interactions. However, they are often too difficult to apply in the industry. LogDL shares similarities to 4QL and other logic-based languages, however, it is simpler and reasoning is faster what fits better into game production pipelines.

F. Mechanics, Prototyping, Narrative Design

Tools aiding designers to create a plot in games have gained popularity in recent years [30]. They are typically based on graphs that contain events, branches, triggers and milestones that define progression in the game. They often allow for specifying the "keys and doors" systems, i.e. the goals that have to be achieved to unlock a certain game's aspect. LogDL can help in two ways in such use-case. All elements of the narrative can be expressed in it in a form of facts and rules. However, more importantly, due to the Prolog-style logical reasoning, it can provide immediate feedback to the designers. For example, LogDL queries can check whether the game can be completed, in how many ways, what is the most probable

or efficient way, etc. Thanks to the extensive nature and ability to add custom operators, a narrative system build on top of LogDL can be fine tuned for a particular game.

In GGP, game rules are described in GDL and game-playing algorithms use such descriptions to conduct simulations. We believe that LogDL could be employed in a similar fashion to express the core mechanics and rules in video games. Due to complexity of such games, their descriptions would have to be high-level game approximations. Nevertheless, it could be useful for rapid prototyping aimed at testing the soundness of ideas in the creative design process [31].

IX. Motivation – Real World

Below we present some real-world use-cases that we intend to investigate. By a use-case we mean an area, wherein LogDL can bring extra value. The data and ML-related challenges in video games and "real world" are similar to each other [32]. This is why LogDL can be useful in both cases.

A. Process Mining

In general, LogDL is suitable for applications where the data needs to be stored, analyzed and new – most likely difficult to predict – insights from the existing data can be discovered. Good examples are innovative R&D applications, where new knowledge can emerge and provide technological advantage. For instance, it can help to gather information about any given process in the form of rules that describe it [33]. Imagine using process mining to discover weather patterns, laws of physics, proving hypotheses, making scientific discoveries, analyzing art, texts, making reverse-engineered models. Works such as [34] illustrate that the analytics of processes requires significant effort at the level of concept formation. Once appropriate concepts are specified, one can reason about their occurrences in the data in a logical fashion.

B. Business Intelligence

This is a field wherein the state-of-the-art data processing approaches are often applied. LogDL can be used as a glue that binds together logs, domain-specific concepts which subject matter experts understand and the automated data science that they usually are not deeply familiar with. A system that incorporates LogDL can be developed in such a way that technical AI/ML details are hidden and friendly interfaces are exposed. Actually, our aforementioned system that advises players how to improve their skills can be treated as an example of business intelligence development in the game industry [27]. Similarly, the analytics can be conducted in "real world" over complex multimodal data sources [35]. In both scenarios, the original data needs to be first digested/enriched – in the already-discussed form of d-logs – and then the main business intelligence layers can be employed.

C. Hierarchical Learning

In a typical ML scenario, languages such as JSON are used only in the first step of the data processing pipeline – to store the input data. However, the phase of reasoning about structured information expressed by LogDL-based d-logs can be still a part of a discovery process. Such an approach, i.e. to combine numerical and symbolic ML techniques (in our case: deriving d-logs from the raw data and reasoning about them) has long stayed under the radar, but recently it has been attracting researchers' attention. This is first, to provide the end-users with the explainable ML models and second, to utilize the layer of d-logs for other purposes, such as the above-discussed process mining or business intelligence.

The logical layer can also reinforce ML-based algorithms with reasoning and rule mining performed on a higher level, e.g. on definitions, features and concepts discovered by the ML-based algorithm such as neural networks [17]. This way, a natural hierarchical system can be achieved [36].

Good examples can be found in applications, where the data from videos or pictures is analyzed by convolutional neural networks (CNN) that are suitable for extracting low level and local features [37]. Methods that manipulate a LogDL description could take it from there and use those local features to induce/infer higher-level concepts. Works such as [38] demonstrate efficiency of such hybrid approach with respect to multimodal spatio-temporal data, whereby LogDL could be additionally used to express the domain knowledge of subject matter experts at that higher level of abstraction.

D. Interactive Analytics

LogDL enables to be interactively queried and manipulated by the AI/CI/ML algorithms which typically run in a continuous loop. It provides advantages of logic-based languages in terms of reasoning as well as robustness and flexibility of database languages. We believe that there is a need for such a medium that can be used interactively by humans algorithms.

In particular, one may consider utilizing LogDL rules for *real-time annotations* or *feedback* that consists of comments of the data. Examples of the corresponding applications refer to augmented reality, virtual reality, streaming platforms, self-driving cars, etc. Methods operating with logic and rules could help human investigators analyze large chunks of machine-generated or sensory data [39]. Such data usually consists mostly of the cases that are not particularly interesting and only at certain points there are photos or video frames that need human attention [40]. Rule-based systems combined with feature extraction may help to narrow down the cases and show only the most interesting ones to human operators.

X. Conclusions

We introduced a new logic-based language – LogDL – for complex data and knowledge discovery workflows. Examples of usefulness have been shown by motivations from both, video games and "real world" applications outside of the entertainment industry. The GDL language from the game research domain was the protoplast for LogDL. However, there are significant differences between them, e.g. types, domains, custom operators, time, probabilistic elements, etc.

Our goal was to combine advantages of data representation languages such as JSON and query languages such as SQL and

introduce a necessary formalism to take advantage of the AI-driven knowledge discovery. In future, we plan to create highly efficient LogDL-based system integrating manual management and the AI/CI methods in the process of discovering new concepts/insights from the complex data. We believe that at foundations of such system there should be a language that can allow for logical reasoning, can be interactively queried and manipulated by intelligent (e.g. EA) algorithms.

A feature that could be beneficial to such manipulations is built-in granularity (level of detail) of rules [41]. It is already possible to define rules that operate with various granularities, but it is up to human interpretation and transparent from the LogDL's viewpoint. We could introduce a convention, e.g. that rules with the same name and a dedicated argument denoting the level of granularity describe the same concept.

Another aspect refers to our aforementioned project FoW, whereby LogDL will be used for reasoning (under uncertainty) by AI players and explanations for human players. Therein, we will need to decide whether to follow the GDL-style closed world assumption or rather make the world "open", like e.g. in case of Action Description Language (ADL) [42].

We also plan to integrate modern ML techniques (e.g. deep learning) which are powerful but difficult to explain with a logical/symbolic top-layer. Such combination not only would make using such a system more interpretable and trustful for human operators but it could also lead to discovery of new knowledge using the concepts defined within LogDL.

REFERENCES

[1] M. R. Genesereth, N. Love, and B. Pell, "General Game Playing: Overview of the AAAI Competition," *AI Magazine*, vol. 26, no. 2, pp. 62–72, 2005.

[2] S. Greco and C. Molinaro, "Datalog and Logic Databases," *Synthesis Lectures on Data Management*, vol. 7, no. 2, pp. 1–169, 2015.

[3] V. S. Costa, R. Rocha, and L. Damas, "The YAP Prolog System," *Theory and Practice of Logic Programming*, vol. 12, pp. 5–34, 2012.

[4] M. Arntzenius and N. R. Krishnaswami, "Datafun: A Functional Datalog," in *Proc. of ICFP 2016*, pp. 214–227.

[5] P. Alvaro, W. R. Marczak, N. Conway, J. M. Hellerstein, D. Maier, and R. Sears, "Dedalus: Datalog in Time and Space," in *Proc. of Datalog 2010*, pp. 262–281.

[6] E. Piette, M. Stephenson, D. J. Soemers, and C. Browne, "An Empirical Evaluation of Two General Game Systems: Ludii and RBG," in *Proc. of CoG 2019*, 2019, pp. 1–4.

[7] M. Thielscher, "Answer Set Programming for Single-Player Games in General Game Playing," in *Proc. of ICLP 2009*, pp. 327–341.

[8] J. Kowalski and M. Szykuła, "Game Description Language Compiler Construction," in *Proc. of Australasian AI 2013*, pp. 234–245.

[9] Y. Björnsson and S. Schiffel, "Comparison of GDL Reasoners," in *Proc. of GIGA@IJCAI 2013*, pp. 55–62.

[10] M. Okumura and S. Fujimura, "Constructing a Log Collecting System using Splunk and its Application for Service Support," in *Proc. of SIGUCCS 2016*, pp. 103–106.

[11] O. Etzion and P. Niblett, *Event Processing in Action*. Manning Publications, 2010.

[12] J. Małuszyński and A. Szałas, "Logical Foundations and Complexity of 4QL, a Query Language with Unrestricted Negation," *Journal of Applied Non-Classical Logics*, vol. 21, no. 2, pp. 211–232, 2011.

[13] B. Dunin-Kęplicz and A. Strachocka, "Paraconsistent Multi-party Persuasion in TalkLOG," in *Proc. of PRIMA 2015*, pp. 265–283.

[14] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, "Towards a Video Game Description Language," in *Artificial and Computational Intelligence in Games*. Dagstuhl, 2013, pp. 85–100.

[15] I. Haghighi, A. Jones, Z. Kong, E. Bartocci, R. Grosu, and C. Belta, "SpaTeL: A Novel Spatial-Temporal Logic and Its Applications to Networked Systems," in *Proc. of HSCC 2015*, pp. 189–198.

[16] D. Pedreschi, F. Giannotti, R. Guidotti, A. Monreale, S. Ruggieri, and F. Turini, "Meaningful Explanations of Black Box AI Decision Systems," in *Proc. of AAAI 2019*, pp. 9780–9784.

[17] M. H. Segler and M. P. Waller, "Neural-Symbolic Machine Learning for Retrosynthesis and Reaction Prediction," *Chemistry – A European Journal*, vol. 23, no. 25, pp. 5966–5971, 2017.

[18] M. Świechowski and J. Mańdziuk, "Fast Interpreter for Logical Reasoning in General Game Playing," *Journal of Logic and Computation*, vol. 26, no. 5, pp. 1697–1727, 2016.

[19] C. F. Sironi and M. H. M. Winands, "Optimizing Propositional Networks," in *Proc. of CGW@IJCAI 2016*, pp. 133–151.

[20] J. C. Tay and N. B. Ho, "Evolving Dispatching Rules using Genetic Programming for Solving Multi-Objective Flexible Job-Shop Problems," *Computers & Industrial Engineering*, vol. 54, no. 3, pp. 453–473, 2008.

[21] D. J. Lizotte and E. B. Laber, "Multi-Objective Markov Decision Processes for Data-Driven Decision Support," *Journal of Machine Learning Research*, vol. 17, pp. 211:1–211:28, 2016.

[22] L. A. Zadeh, *Computing with Words – Principal Concepts and Ideas*. Springer, 2012.

[23] D. Mark, *Behavioral Mathematics for Game AI*. Cengage Learning, 2009.

[24] M. Świechowski and D. Ślęzak, "Grail: A Framework for Adaptive and Believable AI in Video Games," in *Proc. of WI 2018*, pp. 762–765.

[25] M. Świechowski, T. Tajmajer, and A. Janusz, "Improving Hearthstone AI by Combining MCTS and Supervised Learning Algorithms," in *Proc. of CIG 2018*, pp. 445–452.

[26] D. Irish, *The Game Producer's Handbook*. Cengage Learning, 2005.

[27] A. Janusz, D. Ślęzak, S. Stawicki, and K. Stencel, "SENSEI: An Intelligent Advisory System for the eSport Community and Casual Players," in *Proc. of WI 2018*, pp. 754–757.

[28] M. Świechowski, "Game AI Competitions: Motivation for the Imitation Game-Playing Competition," in *Proc. of FedCSIS 2020*, pp. 155–160.

[29] B. Dunin-Kęplicz and R. Verbrugge, *Teamwork in Multi-Agent Systems – A Formal Approach*. Wiley, 2010.

[30] G. N. Yannakakis and J. Togelius, "A Panorama of Artificial and Computational Intelligence in Games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 4, pp. 317–335, 2014.

[31] J. Ruan, W. Van Der Hoek, and M. Wooldridge, "Verification of Games in the Game Description Language," *Journal of Logic and Computation*, vol. 19, no. 6, pp. 1127–1156, 2009.

[32] J. Togelius, "AI Researchers, Video Games Are Your Friends!" in *Proc. of IJCCI 2015*, pp. 3–18.

[33] W. Van Der Aalst, "Process Mining," *Communications of the ACM*, vol. 55, no. 8, pp. 76–83, 2012.

[34] T. Kawamura, T. Kimura, and S. Tsumoto, "Estimation of Service Quality of a Hospital Information System Using a Service Log," *The Review of Socionetwork Strategies*, vol. 8, no. 2, pp. 53–68, 2014.

[35] L. Dey, I. Verma, A. Khurdiya, and S. B. H., "A Framework to Integrate Unstructured and Structured Data for Enterprise Analytics," in *Proc. of FUSION 2013*, pp. 1988–1995.

[36] P. MacAlpine, M. Depinet, and P. Stone, "UT Austin Villa 2014: RoboCup 3D Simulation League Champion via Overlapping Layered Learning," in *Proc. of AAAI 2015*, pp. 2842–2848.

[37] M. Przyborowski, T. Tajmajer, Ł. Grad, A. Janusz, P. Biczyk, and D. Ślęzak, "Toward Machine Learning on Granulated Data – A Case of Compact Autoencoder-based Representations of Satellite Images," in *Proc. of Big Data 2018*, pp. 2657–2662.

[38] J. Ludziejewski, Ł. Grad, Ł. Przebinda, and T. Tajmajer, "Integrated Human Tracking Based on Video and Smartphone Signal Processing within the Arahub System," in *Proc. of FedCSIS 2020*.

[39] G. J. Nalepa, E. Brzychczy, and S. Bobek, "On the Opportunities for Using Mobile Devices for Activity Monitoring and Understanding in Mining Applications," in *Proc. of IDEAL (2) 2018*, pp. 75–83.

[40] C. Han, J. Mao, C. Gan, J. Tenenbaum, and J. Wu, "Visual Concept-Metaconcept Learning," in *Proc. of NeurIPS 2019*, pp. 5002–5013.

[41] M. Świechowski and D. Ślęzak, "Granular Games in Real-Time Environment," in *Workshop Proc. of ICDM 2018*, pp. 462–469.

[42] R. Reiter, *Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.