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

Machine learning and reasoning for exploratory data analysis

Scientific Students' Association Report

Author:

András Földvári

Advisor:

Prof. Dr. András Pataricza

2019

Contents

Kivonat

A dolgozat célja a komplex rendszerek ellenőrzésének és diagnosztikájának bemutatása tudásfúzió és az induktív logika segítségével. A módszer a konzisztencia ellenőrzése és a hibadiagnosztika céljából a rendszerre vonatkozó előzetes tudást egyesíti a megfigyelésekkel. Továbbá, induktív logikán alapuló következtetést használ a megfigyelések általánosításához és az automatikus modellépítéshez.

Kritikus kiber-fizikai rendszerek megbízhatóságának és ellenállóképességének biztosítása tapasztalati ellenőrzést igényel. A rendszerek teljesítménymérése és a működési naplók elemzése jelenti az elsődleges eszközt a kritikus és a megbízható viselkedés biztosításához. A feltáró adatelemzés (EDA) támogatja az összegyűjtött adatok jobb megértését a szakértők és az előzetesen rendelkezésre álló tudás segítségével. Az így meghatározott modell később alkalmazható a rendszertervezés és az rendszerüzemeltetés során.

A vizuális elemzés segíti a szakértőket a rendszer viselkedésének megértéséhez és meghatározza a tüneti modelljének alapját. Az ok-okozati viszony meghatározása a begyűjtött adatokban igen nehéz lehet (önmagában az adatok közti korreláció nem jelent kauzalitást). Előzetes ismeretek a rendszerről (pl. architektúra, működési részletek) segíthetnek egy pontosabb kép meghatározásában.

A cél, az EDA-t gépi tanulással és következtetési technikákkal való támogatása az absztrakt modell lépésről lépésre történő konzisztenciájának és teljességének vizsgálatával. A kvalitatív modellezés és a gráf alapú tudásreprezentáció egy absztrakt szemantikus keretet adnak. A deduktív következtetés ellenőrzi a modellek megfelelőségét és teljességét, míg az induktív következtetés automatizált mechanizmusként szolgál a megfigyelt rendszer kompakt és absztrakt modellezéséhez.

A kvalitatív modellek átláthatóbb képet adnak a folytonos rendszerekről szimbolikus, könnyen érthető módon. A vizuális módszereken és gépi tanuláson alapuló klaszterezési technikák (pl. döntési fa) a folytonos adatokat egységes működési módoknak megfelelően csoportosítják. Ezzel a kvalitatív modellek tetszőlegesen skálázhatók, és a diszkrét modellben támogatják a kauzalitás vizsgálatát.

A diszkrét modell és az előzetes tudás a rendszer felépítéséről ábrázolhatók egy ontológia stílusú tudásreprezentációs gráfban. A modell alapján ellenőrizhető az adatok helyessége. Az így meghatározott inkonzisztenciák megmutathatják az adatsorból kilógó értékeket.

Az Answer Set Programming (ASP) egy deklaratív megközelítést nyújt a rendszerben lévő kauzális összefüggések és a rendszer architektúrájának leírására. Optimalizálási technikákkal (gyenge kényszerek) szűrhetővé válnak a zajos adatok. Végül, induktív következtetéssel az ASP-n meghatározható a rendszer egy általános modellje a megfigyelések alapján.

Ezekkel az eszközökkel támogatható a (vizuális) EDA azáltal, hogy a kiértékelés során ellenőrizhető a meghatározott magas-szintű modell teljessége és konzisztenciája. A megfigyelésekből épített modell lehetővé tesz további formális ellenőrzéseket a rendszer követelményeknek való megfelelésről. Továbbá, a rendszer hibás működése esetén a módszer kiterjeszthető a hibát kiváltó okok elemzésére is.

Abstract

The report presents the combination of discretization, knowledge fusion, inductive logic-based automated system model identification for validating and diagnosing complex systems. It uses a sequence of steps of quantization, knowledge base construction merging priory knowledge and observations and inductive logic-based reasoning for generalization of the observed results for automated model extraction, consistency checking, and fault diagnosis.

Assurance of dependability and resilience of critical cyber-physical systems necessitates empirical validation. Benchmarking and operational log analysis are the primary means to ensure critical and reliable behavior. Exploratory Data Analysis (EDA) supports a better understanding of the collected data by integrating the priory knowledge of the domain expert with the observations. It extracts a model of the phenomena used later in system engineering and operational supervision.

Visual exploration helps the domain experts to validate the behavior of the system and to formulate the root causes in the form of phenomenological models. Finding the causality driving the observed associations in the collected data is frequently tricky. It is hard accurately reason about from effects to their potential causes (correlation does not imply causation). However, prior knowledge (e.g., architecture, dynamics) about the system could provide a more precise view.

Visual technics are efficient as they can rely on the expertise of the analyst. However, the diagrammatic technics represent the observations only by low dimensional projections, like histograms or scatterplots. These perform the “visual pattern analysis” process performed by the analyst to a sequential one with how any guarantees on the consistency of the individual steps.

The goal was to support EDA with machine learning and reasoning technics performing step-by-step a consistency and completeness check on the abstracted model. Qualitative modeling and knowledge graph-based management of the model provides an abstract semantic framework for model extraction. Deductive reasoning checks the compliance and completeness of the observations and their respective evolving model. Inductive reasoning serves as an automated mechanism delivering a compact abstract model of the observed system.

Qualitative models provide a clear representation and reasoning about continuous systems in a symbolic, human-like manner. Visual and machine learning-based clustering technics (e.g., decision tree) collect the continuous data into groups corresponding to uniform modes of operation. This way, qualitative models are scale independent and support root cause analysis well over a discrete model.

Extending the discrete model with prior knowledge about the system architecture determines the knowledge base. Implementation as an ontology-styled knowledge graph checks the conformance of the data instances with this model. Instance-model inconsistencies highlight outliers.

Answer Set Programming (ASP) provides a declarative approach to represent the causal and architectural aspects of the system. The underlying optimization functionality handles noisy data. In the final process, inductive reasoning infers a qualitative model out of the ASP corresponding to an abstract model of the system and its observed behavior.

This way, the toolchain supports (visual) EDA by checking the completeness and consistency of the evaluation process and identifies a high-level abstract system model based on the observations. The end-to-end solution for the observation to model building process prepares further formal checks on the compliance with the requirements against the system. Moreover, the extension of metamodels with the notion of fault facilitates diagnostic root-cause analysis.

Chapter 1

Introduction

1.1 Context of the work

The purpose of *cyber-physical systems* (CPS) [7] is to observe and control the physical world through intelligent mechanisms. They operate over **continuous** and **discrete signals** originating in the physical world, for which they consist of physical and computational components interacting through **communication layers** [25].

The potential use cases of CPS cover a variety of application domains from manufacturing through autonomous cars to the smart ecosystem. There is an increasing need for creating and operating CPSs, as they serve a primary means of optimization and control of complex physical processes. Ensuring the proper operation of the services delivered by CPS involves fulfilling both the functional and extra-functional requirements (timeliness, throughput, reliability, availability, etc.) simultaneously.

Typically, CPSs have an extra-long lifetime originating in the physical infrastructure and the interface elements connected to them. However, functional modifications are frequent by exploiting the flexibility of the computational part, especially by the easy modification and extension of the functionalities implemented by programs. This way, the functionality required by the user is frequently subject to evaluation and occasionally involved by on-demand services.

The main paradigm of CPS construction is *system integration*. The notion of integration means here, on the one hand, integrating components and subsystems into a single entity under uniform control and on the other hand, cooperatively performing the designated functionality. Such a flexible and universal integration paradigm necessitates an easy and steady interfacing of the individual components and subsystems. This flexibility provides easy augmentative and technical maintenance and structural changes and upgrades. For instance, *publish-subscribe* is one of the popular architectural paradigms for the integration middleware, as it delegates all the deployment-related details of the middleware an exposes only the functional interdependence between sub-services.

Preparing the system for such a functionality [24] requires dynamic *system integration middleware* (e.g., DDS[34], OPC-UA[33], MQTT [5]) and a scalable model both in design and operation time that can handle these changes. The purpose of the system integration middleware is to manage the related topological and technological challenges (e.g., automatic discovery of new components, interoperability between systems [1]).

In-field resources of a moderate capacity are typically limited to performing the core functionality. As communication is usually the bottleneck in the performance domain, they perform data pre-processing in order to reduce communication. The level of intelligence of a CPS heavily depends on both the resources available for processing and the intelligence and background information of the processing algorithms.

1.2 Requirements and objectives

The breakthrough idea behind the CPS was to integrate the omnipresent resources available through the Internet (like cloud computing) and the rich intelligence available in the cyber-world as remote data and Algorithms as a Service (AaaS). This way, a typical CPS has a *hierarchical functional abstraction layer* integrating the potentially unlimited resources and intelligence performing the more demanding tasks in addition to a *local processing layer* over limited resources and intelligence .

This structural flexibility and adaptability necessitate the proper dimensioning of the components as their workload can not be fixed in design time due to this structural adaptability. This way, running the components of a CPS in various environments under different workloads could require the invocation of additional resources (*runtime resource allocation*) and scaling the service infrastructure. Both infrastructure scaling in design time and dynamic resource allocation in runtime require the building of a unified model merging requirements, system architecture, and metrics into a single framework. Assuring this need necessitates empirical exploration of the actual characteristics of the system configuration.

CPS models depend on several physical and technical parameters. In design time, the operational modes and characteristics of CPS systems cannot be described analytically with realistic faithfulness due to their highly nonlinear and multi-parameter properties. This complexity also requires a scalable system model (e.g., when applying the same algorithm to tasks of different sizes, or differently large runtime platforms the model must be adapted to the actual size of the given task).

A *discrete (qualitative) model* of the system is a highly abstract form of describing the logic of its architecture and behavior with no details on the actual values. It assures portability and technological independence of the core phenomena during operation. It allows reasoning about the system behavior by highlighting potential phenomena at a logic level. Moreover, it allows running simulations after parametrization of the qualitative model to a quantitative one. Although, discrete modeling has many advantages, due to the high level of abstraction it may also cause ambiguity.

When building the unified model by system identification, it should be usually performed by processing benchmark measurements or operational logs through empirical exploratory analysis.

The *hybrid modeling approach* (Fig. 1.1) sketched above fits well into this task. Hybrid modeling takes the advantages of both the **qualitative** and **continuous models**. It characterizes size-independent **operational ranges** with discrete values and scales the **qualitative ranges** by choosing their boundaries. The approach allows integrating all available knowledge into one united model.

The discrete model aggregates the subspace of the state space into a single abstract state (on observations, this corresponds to assigning a single element to an entire cluster). The engineering reason behind the hybrid modeling technique is that elements inside of a domain corresponding to a homogeneous operation regime posses similar behavior, which can be modeled as identical ones and map into a single state. This extremely high level

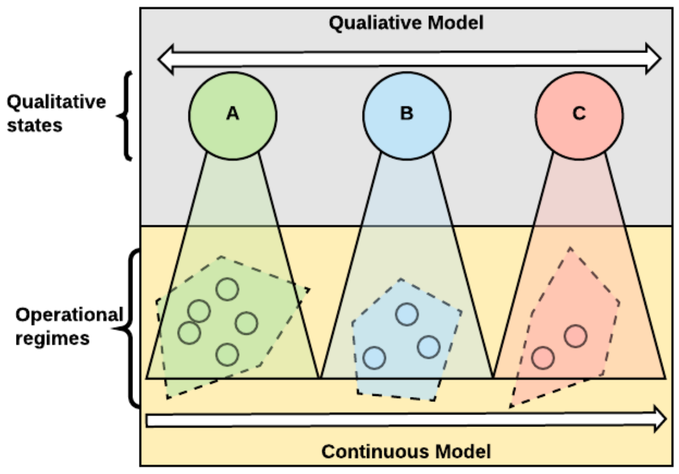


Figure 1.1: Hybrid modeling

of abstraction can reduce highly complex state spaces according to the individual state, probably of infinite cardinality into a few tens of hundreds of discrete states.

Creating a hybrid model as part of the system identification process necessitates clustering of the data into domains (**operational regimes**), which show qualitatively identical behavior of the system. This task referred to as discretization can be performed by and expert manually or by automated means.

Manual methods executed by an expert have the advantage that the discretization process can rely on the **background knowledge** of the expert is able for instance use his domain skills for variable selections, etc. On the other hand, if the number of the measured characteristics is large as required for finite granular models, a pure expert based approach become impossible.

One of the most effective methods of system identification is visual EDA, where the relationship between measurement data displayed in different graphs in space and time. These graphs are examined by a domain expert using background knowledge and requirements of the system.

EDA requires a heuristic examination of the system. For complex systems, the heuristic approach is challenging. It can be tedious to examine all the variables and aspects manually.

Machine learning techniques (e.g., neural networks) support the automatic model extraction from a continuous system, for instance by clustering. However, the explainability of these approaches is unsatisfactory. It is hard to tell which effect influenced the result.

Our goal was to support EDA with machine learning in the form of supervised learning. Reasoning techniques will perform step-by-step checks of the abstract model in order to assure consistency and completeness. The method is the combination if discretization, knowledge fusion, deductive and inductive logic based automated system model extraction.

Qualitative modeling and **knowledge database** (implemented by a knowledge graph) management of the model provides an **abstract semantic framework** for **information fusion** from different sources and **automated model extraction**. They define the operational modes of the system and makes it possible to verify the ranges by discrete value representation.

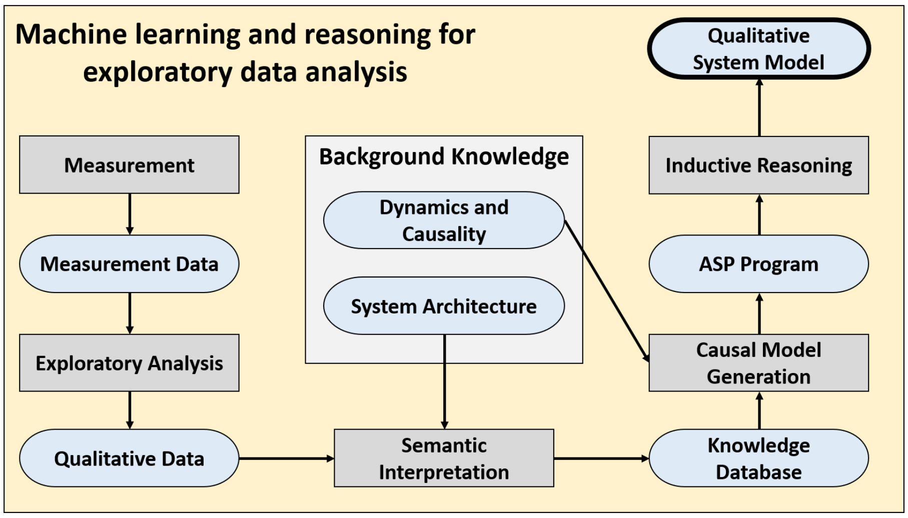


Figure 1.2: Machine learning and reasoning for EDA

Deductive reasoning checks the compliance and completeness of the observations and their respective evolving models. **Inductive reasoning** (on answer set programs (ASP)) extracts by an automated mechanism delivering a compact abstract model of the observed system.

The model extraction supports both the scalable model building in design time and the demand-driven dimensioning (e.g., dynamic resource allocation) in runtime.

This way, the toolchain in Fig. 1.2 supports (visual) EDA by checking the completeness and consistency of the evaluation process and identifies a high-level abstract system model based on the observations. The end-to-end solution for the observation to model building process prepares further automatic checks on the compliance with the requirements against the system. Moreover, the extension of metamodels with the notion of fault facilitates diagnostic root-cause analysis.

1.3 Structure of the report

The report is separated into four main parts:

- Chapter 2 gives a quick overview of the **workflow** of the method by describing the main steps with the respectively used modeling approaches.
- Chapters 3-8 present each step in detail:
  - Chapter 3 introduces **causality** as a core concept in learning observations in the form of associations into technically sound model fragments.
  - Chapter 4 defines the principles of **qualitative modeling**. It presents the extraction of such models concerning the identification of operational regimes in detail, guided by an example.
  - Chapter 5 introduces a uniform platform for **knowledge representation** and fusion of heterogeneous information based on the general-purpose notion of knowledge graphs.
  - Chapter 6 presents a specific declarative programming paradigm called **Answer Set Programming** (ASP). ASP will be used in the rest of the report

    for executing information fusion and reasoning over the abstract notions and measurement data hierarchical by the knowledge graph.

  - Chapter 7 elaborates on the utilization of the ASP for **model fusion**. The transformation of the knowledge graph and the causal model is guided by an example.
  - Chapter 8 introduces the principle of **inductive reasoning** for proposing generalizations out of the priory knowledge and observations in the form of rule proposals together with a tool for inductive learning of ASP programs.

- Chapter 9 presents the **automated model extraction** method in detail.
- Finally, Chapter 10 summarizes the work and gives a brief overview of further researches based on the evaluation of the results.

Chapter 2

Workflow

2.1 Measurement campaigns

The input information of the method is a set of *observations* that usually came from *benchmarks or operational logs.* They describe the system (output metrics) under specific workload parameters.

The first step is to analyze the *measurement campaign* on its own. Experts have to take into account the context of the measurement and outlier data.

The context of the measurement covers determines the measured parameters and boundaries of the measurement campaign. Outlier data can warn about a non-functional operation of the system or indicates that the measurement is not trustworthy.

However, different parametrizations of the same experiment (i.e., different resource allocation) may expose profoundly different phenomena. A scalable model has to merge all of this even potentially different behaviors. This way, the model building has to be adopted to the fundamental configuration settings.

2.2 Background knowledge

Processing the measurement requires having (partially) the *system architecture* and *functional model.* The information extracted from these models can be used during the evaluation phase. The design and development phase of the system provides background information on different abstraction levels.

However, it is possible to work with *partial knowledge* about the system. It is not necessary to know all the details (e.g., third-party components as a black box, only the input-output parameters are known with integration details).

Analysis of a system requires the collection of all *priory knowledge* that is available for the analyst. The background knowledge comes from different sources and covers different aspects of the system and includes the architecture, functional, resource allocation, installation, and causal model of the system.

The system architecture and the functional model provides a high abstraction about the system components and their objectives. A model like causal models can be built by extract information from them.

The resource allocation model closely connects to the functional model. It describes which component uses which resource (e.g., networking capabilities, CPU, RAM). The installation model presents the physical or logical layout of the system.

The causal model expresses the causal connections in the system based on the priory knowledge about the domain and the previous models. Furthermore, it is possible to extend the causal model by adding external (out of the measurement campaign context) causal connections to the model.

Collecting and systematizing the background knowledge is necessary for further analysis.

2.3 Discretization

A discrete (**qualitative**) model of the system is a highly *abstract* form of describing the logic of its architecture and behavior with no details on the actual values. It assures *portability* and *technological independence* of the core phenomena during operation. In the workflow, discretization means the identification of operational modes (operational regime) and the transformation of the *continuous properties into qualitative values.*

2.3.1 Clustering

The goal of the clustering is to aggregate the fundamentally similar behaviors into a uniform qualitative state. Clustering helps to identify the operational modes. There are two different approaches to achieve this goal:

- **Visual EDA**: Visual EDA uses graphs (e.g., scatter plots, time-series diagrams) to identify the boundaries of each operational mode. Because it is a manual process, it requires comprehensive domain knowledge.
- **Machine-learning based clustering**: Machine learning-based clustering techniques (e.g., decision tree, k-nearest neighbor) collect the discrete data into groups corresponding to uniform operation modes. These are algorithmic processes that should verify manually. However, they provide a great base for further analysis.

Clustering helps to build scalable models, and the representativity of the collected data is easily verifiable with the operational modes.

2.3.2 Cluster boundaries

Transforming the continuous variables into qualitative values requires the definition of thresholds. Threshold helps in the data processing and later during monitoring the system in the following ways:

- **Input data**: The input of the monitoring tool is quantitative data. The classification (into operational modes) of the continuous incoming data is based on the identified thresholds.
- **Magnitude of the actuation**: Thresholds can be used to identify the magnitude of the actuation. This way, it can provide a more precise value for actuation by transforming the qualitative values into continuous.

2.4 Modeling techniques

2.4.1 System models

- **System engineering models**: Traditional system engineering methods (ISO/IEC/IEEE42010:2011 [25]) are not always comprehensive for CPSs. Designing a CPS is a challenging task due to the constantly changing environment. Several reference architectures (e.g., OpenFog [3]) and frameworks (e.g., NIST CPS Framework [22]) were presented to ease this problem by defining a unified context. System engineering models include architecture, functional, resource allocation, installation models. These models give the foundation for the later analysis.

- **Causal model**: The causal model extracts the root causes and their manifestations in associations. With the causal model, the inputs and outputs for qualitative simulations are observable. The goal of the causal model building is to explore the (hidden) causal connections and to check the independence of the assumed independent variables. It requires the representativity of the independent factors and removing of false associations.

- **Qualitative model**: The qualitative model provides a clear representation and reasoning about continuous systems in a symbolic, human-like manner. Visual and machine learning-based clustering techniques (e.g., decision tree) collect the continuous data into groups corresponding to uniform modes of operation.

- **Answer Set Programming (ASP)**: The goal of the ASP is to provide a uniform framework for information fusion and reasoning. ASP provides a declarative approach to represent knowledge about the system. Later, it can be used for inductive reasoning, which infers a qualitative model out of the ASP corresponding to an abstract model of the system and its observed behavior.

2.4.2 Model fusion

Model fusion merges the previously constructed models into a uniform model. It is made up of the architectural, causal models, and qualitative measurement data.

The knowledge representation capabilities of ASP enables to model all the required knowledge in one framework. In the final process, inductive reasoning infers a qualitative model out of the merged ASP corresponding to an abstract model of the system and its observed behavior.

2.4.3 Objective

Automated model extraction The goal of the report is to answer questions about both the observable and unobservable causal connections to support the system diagnosis and the automated model extraction.

For instance, we will show from the observation instances that we can create **general hypotheses** that seem to be valid for all the data. Such hypotheses can appear at a higher level of abstraction (i.e., if the experiment describes the dependence between the inputs and outputs of a component, then a hypothesis can generalize the faulty behavior for all the similar component types). This way, if the validity of the hypothesis is proved, a reusable bias of knowledge can be gained.

The method uses *deductive* and *inductive* reasoning about the constructed uniform model to achieve this goal.

2.5 Running example

The running example is based on the dataset from the paper *Exploring uncertainty of delays as a factor in end-to-end cloud response time* [21] .

The paper reports about benchmarking a cloud-based web-service and the investigates instability of its performance and the delays induced by the communication channel when measured from multiple client locations.

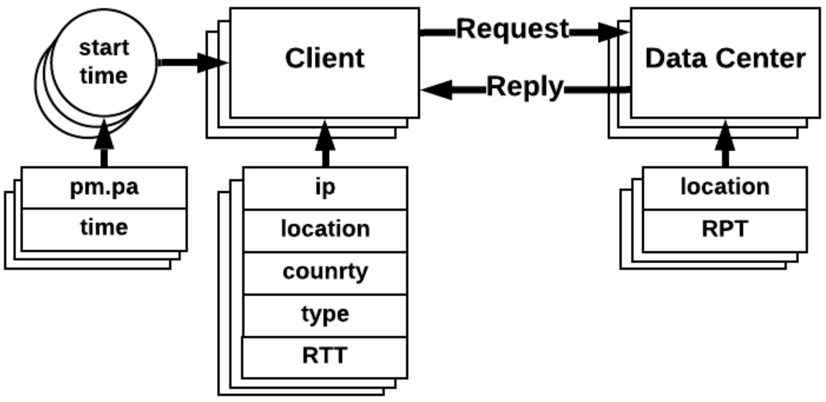


Figure 2.1: Benchmark architecture with parameters

The requests were sent from clients to data centers, which are located in the United Kingdom and South America. Each request was sent at different times of the day with different client types.

The dataset includes the following observed variables:

- ***ip***: IP address of the client.
- ***location***: Location of the client. (e.g. Newcastle, Secaucus, Chicago, Ottawa)
- ***Country***: Country of the client.
- ***client.type***: Technology being used by the client. (e.g. Java, Microsoft)

- ***RTT***: Network (the Internet delay) round trip time.
- ***DC***: Location of the data center (Redmond, Dublin)
- ***RPT***: Request processing time by a benchmark web service deployed in the cloud.
- ***RT***: End-to-end response time (RT = RTT + RPT)
- ***start.time***: Start time of the request in milliseconds since the UNIX epoch (January 1, 1970 00:00:00 UTC).
- ***pm.pa***: Day time of the request (AM/PM)
- ***Time***: Time difference between the client and data center.

2.5.1 Evaluation of the benchmark

Over-aggregation of the data suppresses interesting details like outliers. In high availability systems, however, the system has to perform with the availability of many lines after the decimal point accordingly, outliers may endanger the fulfillment of extra-functional requirements.

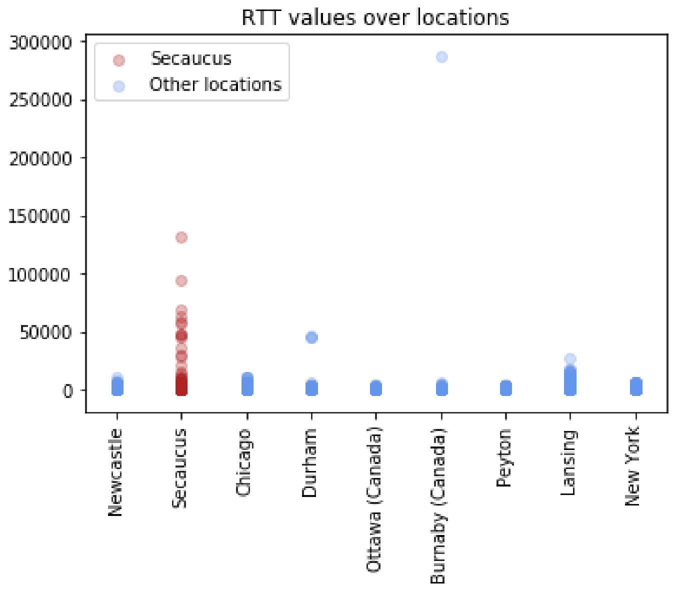


Figure 2.2: RTT values over locations indicates an anomaly in Secaucus

The assumption about the measurements is that the clients and the servers are processing the same algorithm. This way, a homogeneous result is expected. Most of the data points have equal distribution except for some outliers, that indicates potentially inhomogeneous behavior. The priory assumption and knowledge does not explain these outliers.

The authors of the original paper *did not discover an important anomaly.* Previous researches pointed out that there is some strange (dis)functionality in Secaucus (Fig. 2.2), which was ignored in the original paper.

The rest of the report uses this example to guide through the steps of the presented method.

Chapter 3

Causality

Causality is a natural, universal concept, so deeply present in our everyday life that we instinctively think in *causal relations* without pondering about their actual complexity and importance. The whole physical world around us is fueled by causality. It is the connection through which -under certain circumstances- one thing (the cause) influences another(the effect) in a deterministic way.

Causal explanations must be compatible with human notions of causality. The exploration of causality in complex technical domains has led to the development of sophisticated accounts of causality in continuous systems.

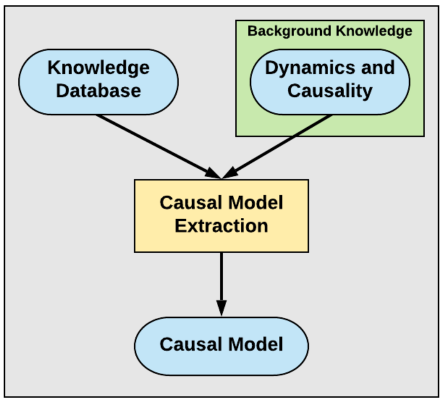


Figure 3.1: Model derivation

Causal models [35] [36] allow the exploration of the causal context of a system and the detection of independent properties and events. Two events are statistically independent if the occurrence of one does not affect the probability of the other. It is necessary to check the validity of the possible independent variables to verify the correctness of the representation.

Causal models also help the domain experts in EDA to explore a more detailed view of the system.

The presented methodology uses causal models to explore the causal behavior of the system by combining the priory knowledge with the system model (Fig. 3.1).

At first, this chapter presents the Ladder of Causation by Judea Pearl [35]. Each level of the ladder represents different cognitive abilities that a causal learner must achieve. The final goal of an inference system is to achieve the top level of the ladder.

Nowadays, most of the machine learning approaches (e.g., neural networks) are on the first level of the ladder. Traditional machine learning approaches do not provide a proper explanation of their results (i.e., what effect influenced the outcome).

Later this chapter, the directed acyclic graph (DAG) representation of the causal models will be presented with its mathematical background and techniques on how domain experts can use the graph for causal analysis.

The last part presents how the causal model build of the functional and resource allocation model of the system. By using these models, the causal connection between the functional building blocks and their resources (e.g., memory, networking, etc.) will be observable.

3.1 Ladder of Causation

Judea Pearl introduced the Ladder of Causation (Fig. 3.2) in his book Causality[35]. The three levels represent different abstractions. **Association** (seeing) expresses regularities and patterns as correlation. **Intervention** (doing) expresses special causal relationships between events. **Counterfactuals** (imagining) constructs a theory of the observed phenomenon that explains why specific actions have specific effects and what happens in the absence of such actions.

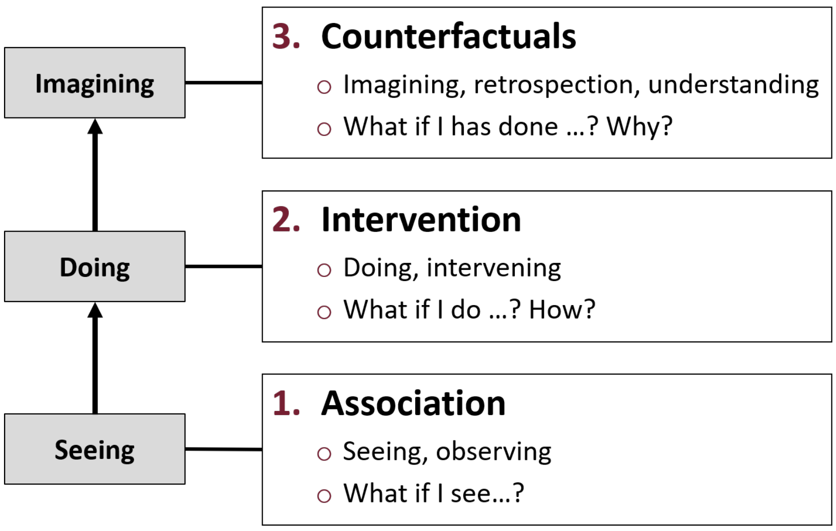


Figure 3.2: Ladder of Causation[36]

Association Two objects associate if the observed probability of one object indicates the probability of the other.

For example: *What does an observable variable tell about a disease?* High RTT could indicate a only a noisy data or it could indicates the faulty behavior of the system.

$$P(faulty\_operation|rtt(high))$$

Associations have no causal implications, as correlation does not imply causation. One event could cause the other, the reverse could also be true, or some third event could cause both events.

Intervention The main difference between Association and Intervention is, that Intervention predicts the effects of *deliberate* actions. This asserts causal relationships between the events. Causality is assessed by experimentally performing some action that affects one of the events. Causality cannot be established by examining history because some other unknown reasons (independent from history) may cause the occurrence of the observed event.

For example: *If I change the configuration, will the system operates normally?* Intervention predicts the causal relationship between changing the configuration and normal operation.

$$P(normal\_operation|do(configuration\_change))$$

In this example, *do* (do-calculus) is an operator that means experimental intervention. Without using the do-calculus, it could be other actions that cause the normal operation.

Counterfactuals The counterfactual level involves consideration of an alternate version of past events. Models that can reason about counterfactuals allow interventions. It means that the consequences of an action can be predicted.

For example, *Was it the configuration change that leads the system back to normal operation?* Answering *yes* to this question asserts the existence of a new causal relationship between the configuration changing and the normal operation.

3.2 Causal models

Causal models describe the causal mechanism of a system. Causal diagrams include *causal loop diagrams*, *directed acyclic graphs*, and *Ishikawa diagrams*. The report uses causal graphs to model causality in the benchmarking example.

A causal graph is a *Directed Acyclic Graph* (DAG), where the relations represent the causation among the variables.

3.2.1 Directed Acyclic Graph

Graph is an ordered pair $G = (V, E)$, where V is a non-empty set and $E \subseteq V \times V$. The elements of V are called vertices and the elements of E are called edges.

If G is a graph then $V(G)$, and $E(G)$ denote the sets of nodes and edges. $v(G)$ and $e(G)$ denote the number of the nodes and edges.

Directed and Undirected Graph In undirected graphs the edge (x, y) is identical to the edge (y, x). In directed graphs, the values of the edges have orientations.

Simple Graph If $e = (v_1, v_2) \in E$ then (v_1, v_2) are the end points of the edge. If $v_1 = v_2$ then e is a loop. Two or more edges that connect the same two vertices are called multiple edges. A simple graph is graph without multiple edges or loops.

Directed Acyclic Graph (DAG) DAG is a finite directed graph with no directed cycles. It consists of finitely many vertices and edges. DAG has a topological ordering, a sequence of the vertices such that every edge is directed from earlier vertex to later vertex in the sequence.

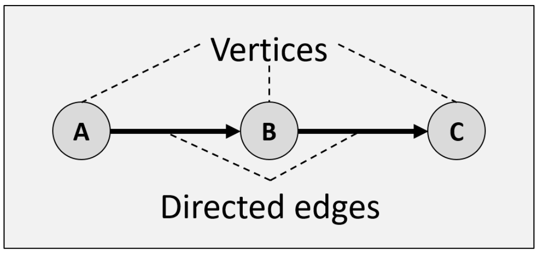


Figure 3.3: Directed Acyclic Graph (DAG)

3.2.2 Casual interpretation of DAG

A causal diagram includes a set of **variables** (vertices) and the directed **edges** between them. If two vertices are connected, that means there is a causal relationship between them. The arrow determines the direction of causality. A → B means A influences B.

The relation between vertices from the viewpoint of causality:

Ancestor An ancestor role represents a direct cause (i.e., parent) or indirect cause (e.g., grandparent) of a particular variable.

For instance on Figure 3.3, ***A*** is an ancestor of ***B*** and ***C*** . ***B*** is an ancestor of ***C*** .

Descendant A descendant role represents a direct effect (i.e., child) or indirect effect (e.g., grandchild) of a particular variable.

For instance on Figure 3.3, ***B*** and ***C*** are descendants of ***A*** . ***C*** is descendant of ***B*** .

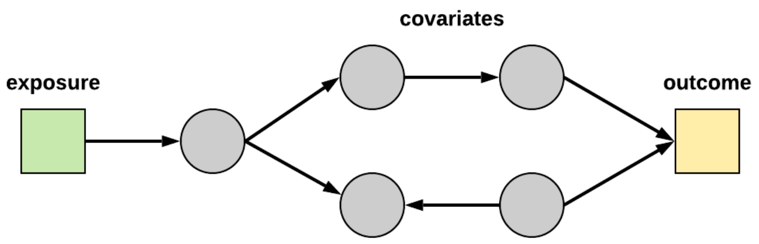
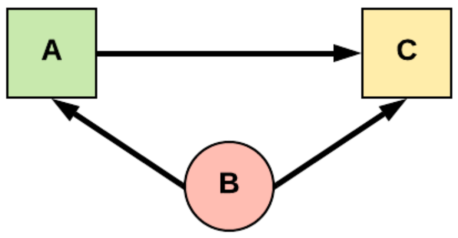


Figure 3.4: Casual Interpretation of DAG

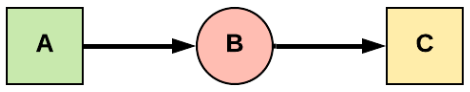
Two variables of interest are distinguishable (figure 3.4): the **exposure** (independent variable, cause) and the **outcome** (dependent variable, effect). Other variables (whether measured or not measured) are called **covariates**.

3.2.3 Covariates

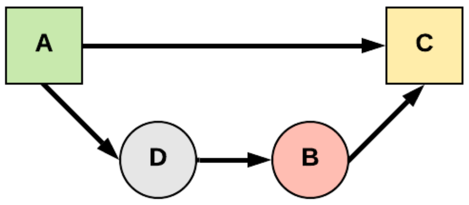
Covariates 3.5 can be categorized into several roles; only some of these roles are mutually exclusive. The report uses the terminology of DAGitty [43].



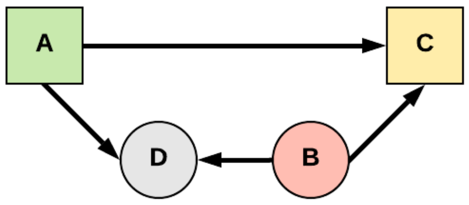
(a) Confounder



(b) Mediator



(c) Proxy Confounder



(d) Competing Exposure

Figure 3.5: Covariates

In the following examples, $\boldsymbol{A}$ is the exposure and $\boldsymbol{C}$ is the outcome.

Confounders are variables that are ancestors of both the exposure and the outcome (along a path that does not include the exposure). On Figure 3.5a, $\boldsymbol{B}$ is a confounder. For example, a resource can be a confounder if two component uses the same resource (e.g., memory, cpu).

Mediator is a variable that is a descendant of the exposure and an ancestor of the outcome. On Figure 3.5b, $\boldsymbol{B}$ is a mediator. For example, the data flow in the system can be modeled as a path of mediators. For example, a proxy confounder can be a sub-component of the system that uses the same resource with one component (A) and provides data to an other ($B \rightarrow C$).

Proxy confounders are covariates that are not themselves confounders. They are a descendant of a confounder and an ancestor of either the exposure or the outcome (but not both; else, it would be a confounder). On Figure 3.5c, $\boldsymbol{D}$ is a confounder and $\boldsymbol{B}$ is a proxy confounder.

Competing exposure is an ancestor of the outcome, and it is not related to the exposure (it is neither a confounder nor a proxy confounder, nor a mediator). On Figure 3.5d, $\boldsymbol{B}$ is a competing exposure. For example, the competing exposure can be a local resource that influences only the local components.

3.2.4 Types of causal graphs

For gaining a deeper understanding of the causal model, the DAGitty tool comes in handy, as it supports two additional views of the causal graph.

Correlation graph Correlation graphs are undirected graphs. There is an edge between all the variables that could be statistically dependent on the original causal model. The variables that are not connected by an edge must be statistically independent. Pairwise independence must be checked by hand[42].

Moral graph Moral graphs help to identify minimal sufficient adjustment sets. DAGitty transforms the original causal graph to an undirected graph. This model supports the verification of the calculation manually[40].

3.3 Building causal models

This section presents a method (Fig. 3.6) for building causal models by using *classical engineering techniques* (e.g. SySML[16]). Classical engineering models (e.g., requirement, system architecture, component, functional) helps to collect the *background knowledge* that is required *for building the causal model.* The causal model is derivable from the **functional model** of the system and its **resource allocation model**.

The *functional model* describes the continuous processes of the system, which defines the **skeleton of the causal model**. The causal model uses the described *data flow* by functional model.

Extending the functional model with the *resource allocation* model also extends the causal graph with **detailed causal relations**. This model can present the causal connections between the resources and the functions (e.g., it is observable if two components using the same resource)

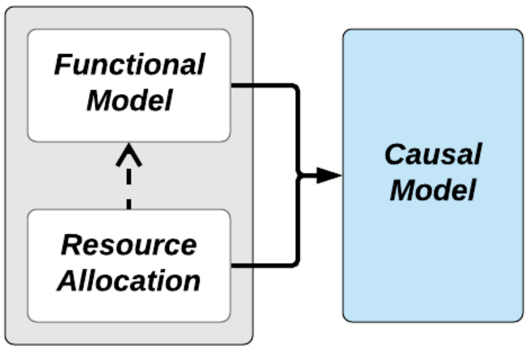


Figure 3.6: Model derivation

The causal model building process also requires background knowledge about the **dynamics and the causal details** of the system. This knowledge is usually *verified by an expert* who knows the system and its domain in great detail. This addition to the causal model makes the *hidden causal effects* observable and reasonable.

This, way the causal model building follows the steps of:

1. Building the **skeleton** of the causal model from the functional model.
2. Replace the abstract nodes with **observable variables**.
3. Extend the model with **further observable variables** that were *not part of the skeleton* model.

4. Extend the model with **unobservable** causal relations from the priory knowledge.

3.4 Example: Causal model

This subsection presents the causal model building of the benchmarking example. The model building process follows the steps described in the previous subsection.

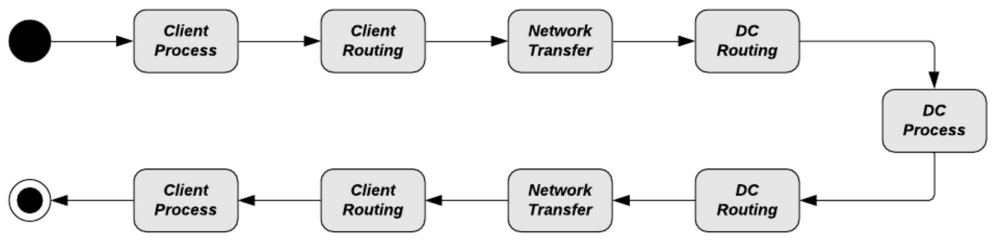


Figure 3.7: Functional model

The functional model (Fig. 3.7) is closely linked with the architecture model. The functional model describes the steps of each message is in measurement campaign:

The process starts by sending a message to the data center (DC). Both the client and DC routers are involved in the routing process. The message goes through a network that interconnects the parties. When a DC receives a message, processes it, and sends the result back to the client by involving the same steps (routing, network transfer) as in the sending phase.

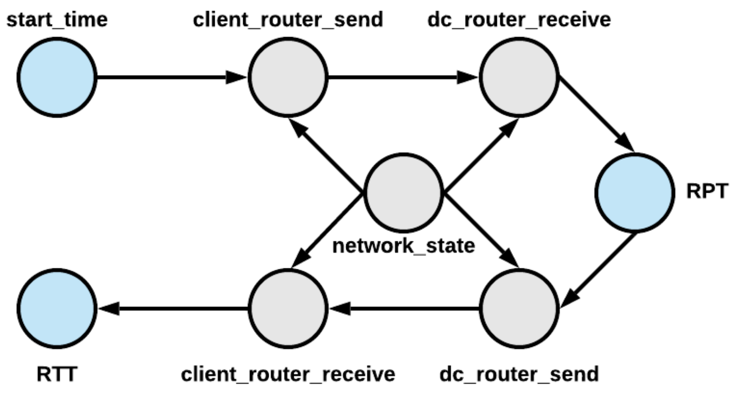


Figure 3.8: Skeleton of the causal model extended with some observable variables

This functional model is the bases of the skeleton for the causal model. The skeleton of the causal model (Fig. 3.8) contains the exact observable variables from the measurement context (e.g., the DC Process function is expressed by the RPT variable).

The next step connects all the other observable variables to the causal graph. It requires the prior knowledge of the user about the assumed causal connections.

Finally, other unobserved causal connections should be added to the model (e.g., using the fault model). This example uses the fault model of the benchmarking example to extend the causal model with unobservable properties.

The final model (Fig. 3.9) contains both observable (benchmark variables) and unobservable (priory knowledge) properties.

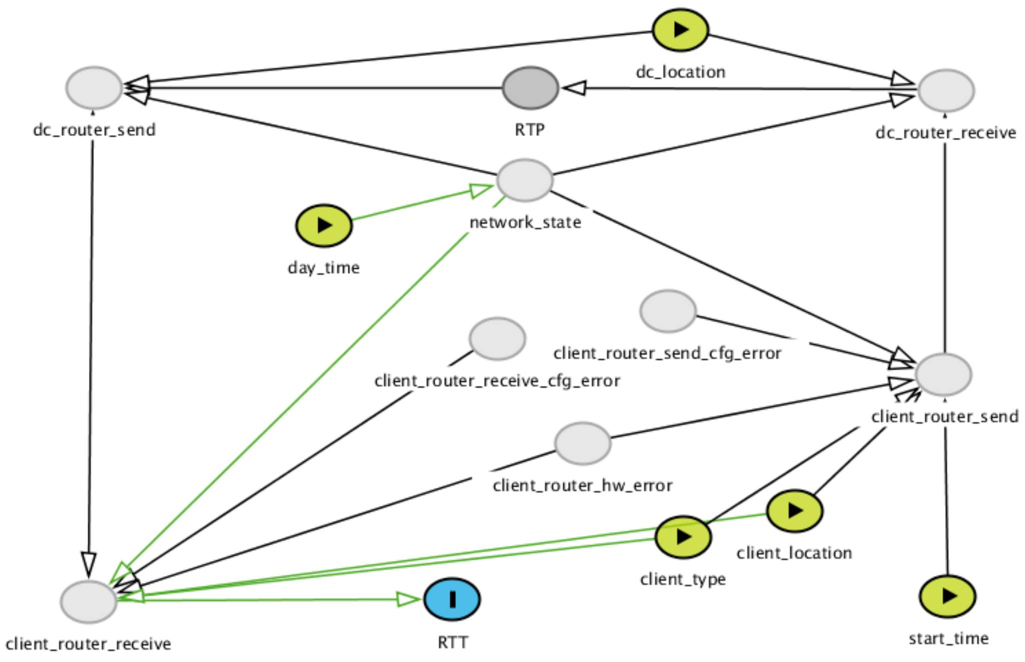


Figure 3.9: Final causal model

Chapter 4

Qualitative Modeling

Qualitative models [15] provide an abstract representation by discrete values corresponding to operational regimes of a uniform or nearly uniform behavior over the continuous domain. They are scale-independent, and the representative quality of the collected data is easily verifiable.

Qualitative modeling (reasoning) [15] has three main steps that describe its main advantages — namely, model formulation, qualitative state elaboration, and simulation.

The model formulation is about to determine the input description of the system. Input description takes into account the knowledge of the kinds of entities and phenomena that can occur (model fragment). It is also necessary to add constraints to the model about the boundaries of the system — this collected knowledge called domain theory. Knowledge bases allow storing the domain theory by providing a rich set of functionality (e.g., built-in reasoning) and representation mechanisms (relations, attributes, rules, etc.).

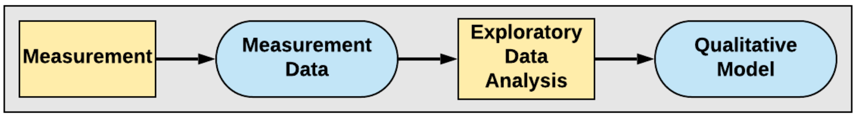


Figure 4.1: From Measurement to Qualitative Model

The qualitative state defines the active model fragments of the system. It represents the entities with their parameters. In the context of qualitative modeling, the values of the parameters are discrete.

Note that there are model fragment sets that never occur in the system. These states should be defined during the model formulation phase, but it requires a comprehensive domain and system knowledge. For example, in the case of a set of observations, it should be declared if a qualitative state is only not represented in the occurrences, or it is not part of the state space of the system.

Qualitative simulation identifies the sequence of qualitative states that can occur one after another — defining the collection of all possible behaviors of the system called envisioning. As the number of the qualitative states can be infinite (even for simple systems), the qualitative representation should be design with only the required fragments that covers the task-specific problem.

These building blocks provide portability, scalability, and technological independence.

This chapter presents the first step of the method: qualitative modeling (Fig. 4.1). The goal of this step is to create a qualitative model from the measurement data.

First, it presents the principles of qualitative modeling, and methods to identify the operational modes of a system. The end of the chapter defines the operational modes of the benchmarking example.

4.1 Qualitative models

Qualitative modeling (qualitative reasoning) is used in a wide range of disciplines. One of the earliest applications was in the field of engineering problem-solving. Later, this approach spread across different fields, from economics and decision support to ecology and bioinformatics.

The qualitative model maps the different operational domains to system states. Clustering methods identify the domains that describe fundamentally similar behaviors in the system.

The resolution of the models is goal-oriented. It usually covers all the cases (conditioned by the dimensioning) as a single integrated model.

The principles of qualitative modeling are the following:

- **Discretization** In the process of discretization, the first step is to identify the entities of continuous media. Modeling the entities is important because they can be represented and reasoned about symbolically. Then the continuous parameters are quantized, so they will only take on a finite number of possible values. It is thus providing a means of abstraction, a qualitative representation.
- **Relevance** After the discretization, constraints can be imposed on the resulting qualitative representation, both deriving from the nature of the system, and the reasoning to be done about it. The constraints cannot be formulated properly unless the qualitative values are constructed and formed in such a way that they represent the relevant information. Also, the integrity of the regions defined by the discretization must be ensured. Within each region, the behavior of the system must be the same, with respect to some task-specific criteria.
- **Ambiguity** The downside of working on such a high level of abstraction is the ambiguity. Such an abstract model often does not represent enough information to predict the behavior of the system in any given situation.

4.2 Operational regime

4.2.1 Defining operational modes

The goal of the clustering is to aggregate the fundamentally similar behaviors into a uniform qualitative state. There are two different approaches to achieve this goal: *visual EDA*, *clustering*. This subsection present a clustering in detail.

4.2.1.1 Clustering

Clustering is the identification of a grouping or categorization of the data points in such a way that data points that are in the same group have similar features, while they differ from data points belonging to another group. Clustering can be interpreted as grouping datapoints together based on their Euclidean distance in the multivariate space defined by a subset of their features. The process of clustering is an unsupervised learning method, and it is a frequently used technique in exploratory data analysis.

K-Means clustering algorithm In this paper, the k-means algorithm is presented as an example, which is one of the oldest clustering algorithms, and it is still amongst the most commonly used ones [32]. It uses an iterative algorithm to define k number of groups in the data. At first, it randomly initializes the centroid for each of the k groups. Then every datapoint is classified into one of the groups based on its distance from the centroid. In each iteration, the algorithm adjusts the position of the centroids to be in the center of the data points classified to belong in the group defined by the previous value of the given centroid.

Evaluation of clustering algorithms The evaluation of clustering algorithms is challenging [23], because unlike with supervised learning problems, there is no ground truth available for comparison. Therefore these metrics cannot measure the validity of the model's results, only give an insight on how organized the clusters seem to be, and as such can be used to compare several model's abilities of cleaner separation.

Davies-Bouldin Index [23] According to this metric, a clustering is proper if the intra-cluster distance of the data points is smaller while the inter-cluster distance is bigger.

Dunn Index [6] The idea behind the Dunn index is the same; only the approach is different. The Dunn index takes the model's weakest link to evaluate the model, as the measurement captures the most significant intra-cluster distance and the smallest inter-cluster distance.

Silhouette Coefficient [39] This metric may be the most accurate, but it is the most expensive as well in terms of computing complexity. It is computed for each individual data point, how well it fits into its own cluster. A negative coefficient means the datapoint should have been assigned to another cluster; a positive coefficient means it was well assigned. If the coefficient is zero, then the data point is right on the inflection point of clusters, and its place cannot be decided.

4.2.2 Detection of rare events

The data points that are roughly separated from the rest of the data are called outliers [37]. They typically rarely occurre in measurements. Most of the time, they can be derived from measurement errors (e.g., an obvious error may be a value outside the domain of the variable). If this is the case, the resulting outliers should be ignored after identification. However, they could also indicate non-functional behaviors. Further investigation is required in this case. The goal could be to address the need to restore proper functioning or the ability to forecast the event in the future.

Rare event detection is favorably used in several domains (e.g., in medical diagnosis, system operation, economy). Algorithmic support for detecting and categorizing outliers is essential for analyzing high dimensional data sets with many samples (obviously, there is a higher probability of the occurrence of rare events in data sets that have a large number of samples). The number of rare events is typically far from the general trend of the data, and the nature of their manifestation (size, frequency, etc.) may vary in different data sets.

The first step is to determine the characteristics of the attribute space, nature, and the distribution of the variables. After the boundaries of the normal range are determined, the out-lying data points can be separated from the others.

The two general algorithmic approaches are:

Distance-based methods Distance-based methods assume that special points are far from the center of the entire data set.

The center can often be defined by examining a single attribute since the outliers fall out of the normal value interval. In the case of multiple attributes, a finite subset of attributes should be selected. Typically, this requires first selecting the set of clusters that define the normal range.

Several algorithms were introduced in the last decades. However, they have the same foundations (Mahalanobis distance, Envelope methods, Distance-Based algorithm)

Density-based methods The basic idea of density-based methods is that the degree of the externality of a point is determined by its difference from its local environment. In the distance based methods, calculating the distance may not be trivial in many cases, for example, in a very high dimensional dataset, or if an important feature is of a sensitive data type. Most of the time, Euclidean distance cannot be applied for real-world data, especially without prior normalization. This does not pose a problem when it comes to density-based methods, as they are based on the data points' local environment by examining the local density.

Several algorithms have been introduced, for example the Local Outlier Factor, or the Nearest-Neighbor-Based Rare Category Detection.

4.3 Threshold identification

Traditional *quantitative* models use real values or floating-point approximations to describe a specific continuous variable, while *qualitative* systems identify a finite set of values as their domain. Therefore the transition from a quantitative model to a qualitative model can be viewed as a kind of abstraction of the variables.

In qualitative systems, the description of the variables can be either static or dynamic. In a static model, the value of the variables stays the same during the qualitative reasoning. However, in a dynamic model, it may change in time (e.g., new data points are added, thus influencing the values of the threshold).

Several qualitative representations of continuous variables are available. The right one shall be chosen based on the resolution required by the problem.

Status Abstraction Status Abstraction [4] is one of the simplest qualitative representations. It classifies the observed variable as *normal* and *abnormal*. The classification's semantics is context-dependent. Several techniques can help to identify the threshold that distinguishes the clusters and classifies the observed values (e.g., visual EDA, fuzzy logic, simple range calculation).

If all the variables in the system use Status Abstraction, the maximal state space is: 2^N, where N is the number of the observed variables.

Sign Algebra Sign Algebra [11] uses three qualitative values: positive(+), negative(-) and zero(0). It also enables the addition of integrity constraints: e.g. A value can not jump directly from - to + without taking the value 0. Sign Algebra is used in a lot of qualitative systems, however the aspect of ambiguity should be taken into consideration. Given the following equation:

$$[a] + [b] = [c]$$

If the discrete value of a and b is both +, then c must be + too. However, if a is - ,and b is + (or vice versa), then c is ambiguous in the sense of the sign of the result.

In the case of using Sign Algebra, to describe all the observed variables, the maximal state space is 3^N, where N is the number of the variables.

Finite Symbolic Value System This approach describes the continuous values with a small number of terms. One typical approach is to split the values (e.g. sensor data) in three intervals: *low*, *medium*, *high*. It is much trickier to apply a consistent algebra for these terms: e.g., Is $medium + low$ equals $medium$ or $high$?

The state-space could be very large (state-space explosion): M^N, where M is the number of the possible qualitative values, and N is the number of the variables.

4.4 Example

4.4.1 K-means clustering

The K-means clustering algorithm was used to identify the operational modes of the benchmarking example.

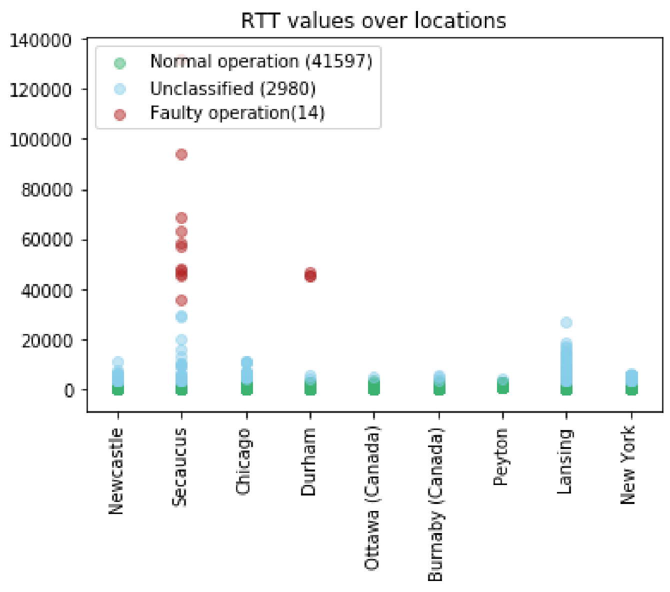
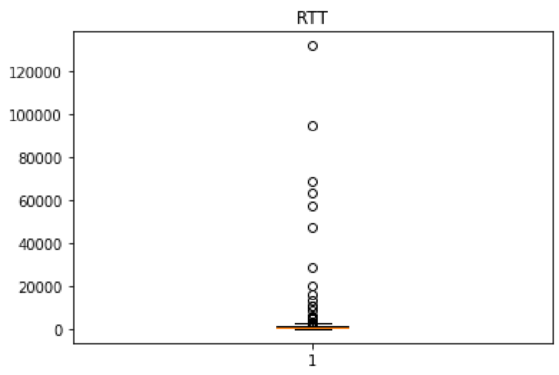


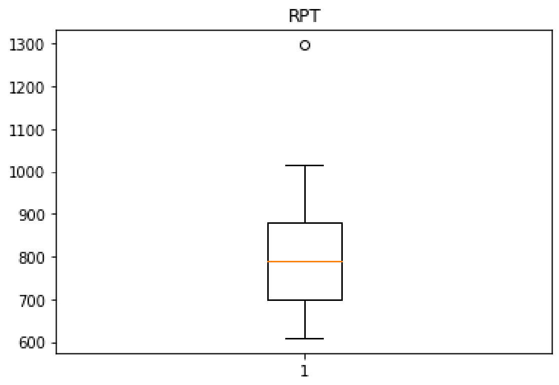
Figure 4.2: Clusters

In the benchmarking example, three operational modes are distinguishable (Fig. 4.2):

- **Normal operation**: The *Normal operation* mode descries the data points that correspond to the required operation (e.g., both RTT and RPT are low)



(a) RTT Boxplot



(b) RPT Boxplot

Figure 4.3: Continuous variables

Variable	Discrete values	Continuous interval
RTT	low	[63,3363]
	medium	[3363, 32678]
	high	[32678, 131623]
RPT	low	[609,682]
	medium	[683,788]
	high	[789,1672]

Table 4.1: Discrete values of the observed variables

- **Faulty operation**: The *Faulty operation* mode describes the data points that differ from the required operation (e.g., high RTT or RPT). These are the rest of the outliers.
- **Unclassified**: *Unclassified* data lies between the normal and faulty data. However, further classification aspects add constraints (sufficient and necessary) that could classify the remaining data.

4.4.2 Boundary identification

The benchmarking example contains two continuous variables. The example uses a static variable description. The first continuous variable describes the latency in the communication channel (RTT), and the other represents the processing time of the datacenter (RPT). The example uses Finite Symbolic Value System for representing the continuous variables (RTT, RPT)(Table 4.1) (Fig. 4.3a, 4.3b). The defined set of qualitative values is $\{low, medium, high\}$.

Chapter 5

Knowledge Representation

Knowledge representation plays a central role in qualitative reasoning. The goal is to collect all priory knowledge into a single qualitative model for further analysis (with ASP) and for verification purposes.

Ontologies [15] are one way to represent knowledge about a system from different views and aspects. Traditional mathematics-based modeling systems are informal about ontologies. Qualitative modeling makes this informal knowledge into explicit form. Ontologies in qualitative models define what entities should be included in a situation, what phenomena are relevant, and what simplifications are sensible. Three type ontologies are commonly used in qualitative modeling: components, processes, and fields.

One way to represent ontologies is the knowledge graph building. Knowledge graphs are mathematical construction to model relationships between entities. It enables the composition of the entities and defines them on different hierarchical levels (Fig. 5.1a).

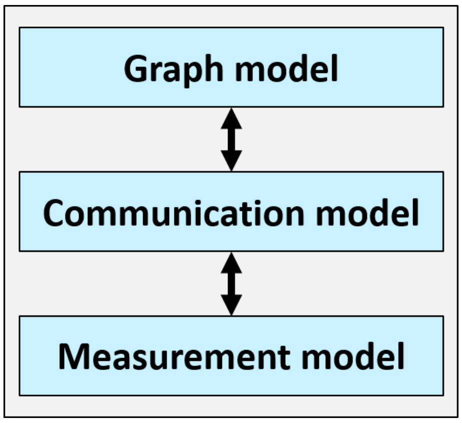
This chapter (Fig. 5.1b) presents a knowledge (graph) representation tool called GRAKN.AI and a traditional system engineering-based method for graph building on the benchmarking example.

5.1 Knowledge graph

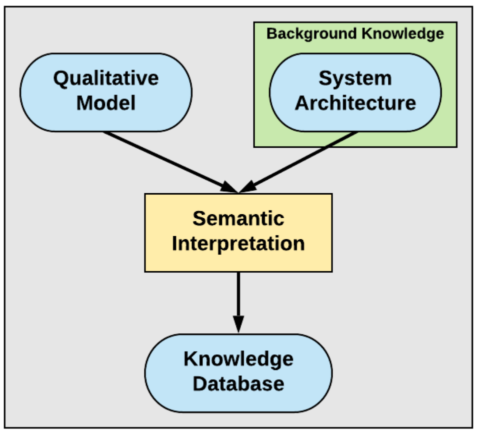
The knowledge graph can be considered as a database to organize complex networks of data in a queryable fashion. It provides a complex schema to capture entities, real-world objects, events, situations, and abstract concepts and relationships. With this kind of representation of knowledge, entities and their relations can be expressed in an explainable and reusable way, while also providing strong, deductive reasoning capabilities on graph instances. That is why knowledge graphs are used in a wide range of use cases, like recommendation systems, information storing, and supply chain management.

One of the major problems in machine learning is the explainability of the predicted model. The knowledge graph can help to map the explanation to the graph. This way, it can summarize and describe the prediction and make it traceable. Knowledge graph models support machine learning in various aspects, e.g., classification, and generation.

In this research, GRAKN.AI was used to store the qualitative benchmarking data. The next sections present the GRAKN.AI tool and its usage in the research by building the knowledge graph and reasoning about the information stored in it.



(a) Abstraction layers



(b) Workflow

Figure 5.1: Knowledge representation

5.1.1 Grakn

Grakn [2] provides an enhanced entity-relationship schema to model complex datasets. The schema allows users to model type hierarchies, along with hyper-entities and hyper-relationships, that can be extended anytime in the database lifecycle.[4] Hyper-entities are defined to be entities with multiple instances for a given attribute, and hyper-relationships are defined to be N-ary relationships, nested relationships, cardinality-restricted relationships, or between relations and entities. This enables the creation of complex knowledge models that can evolve flexibly.

GRAKN.AI is an open-source, distributed knowledge graph for knowledge-oriented systems.[2][3] It is an evolution of the relational database for highly interconnected data as it provides a concept-level schema that fully implements the Entity-Relationship (ER) model.

Grakn's schema is a type system that implements the principles of knowledge representation and reasoning. This enables Grakn's declarative query language, Graql (Grakn's reasoning and analytics query language), to provide a more expressive modeling language and the ability to perform deductive reasoning over large amounts of complex data.

5.1.2 Knowledge graph building with Grakn

Building a knowledge graph means the description of the problem to be solved by defining the schema.

The schema of the GRAKN.AI knowledge graph is based on ER (Entity-Relationship) modeling. ER models include *entities*, *relations* and *attributes*. An ER model defines the objects of the examined world and the relationship between the objects. The objects could have attributes that describe their properties. GRAKN allows the definition of type hierarchies, hyper-entities, hyper-relations, and rules.

This way, it is possible to define the knowledge graph on different association levels (Fig. 5.1a). The traceability between the abstraction levels is performed by the built-in reasoning mechanism.

5.2 Example: Knowledge graph

This section presents the knowledge graph of the benchmarking example.

Each different measurement (see Classification and MDD) was stored in the knowledge graph. Figure 5.2 presents the schema of the benchmarking example on the Measurement abstraction level. Yellow boxes indicate the entities, green rhombuses, the relationships, and blue ovals the attributes of the entities. The labels are the roles that entities play in a relationship.

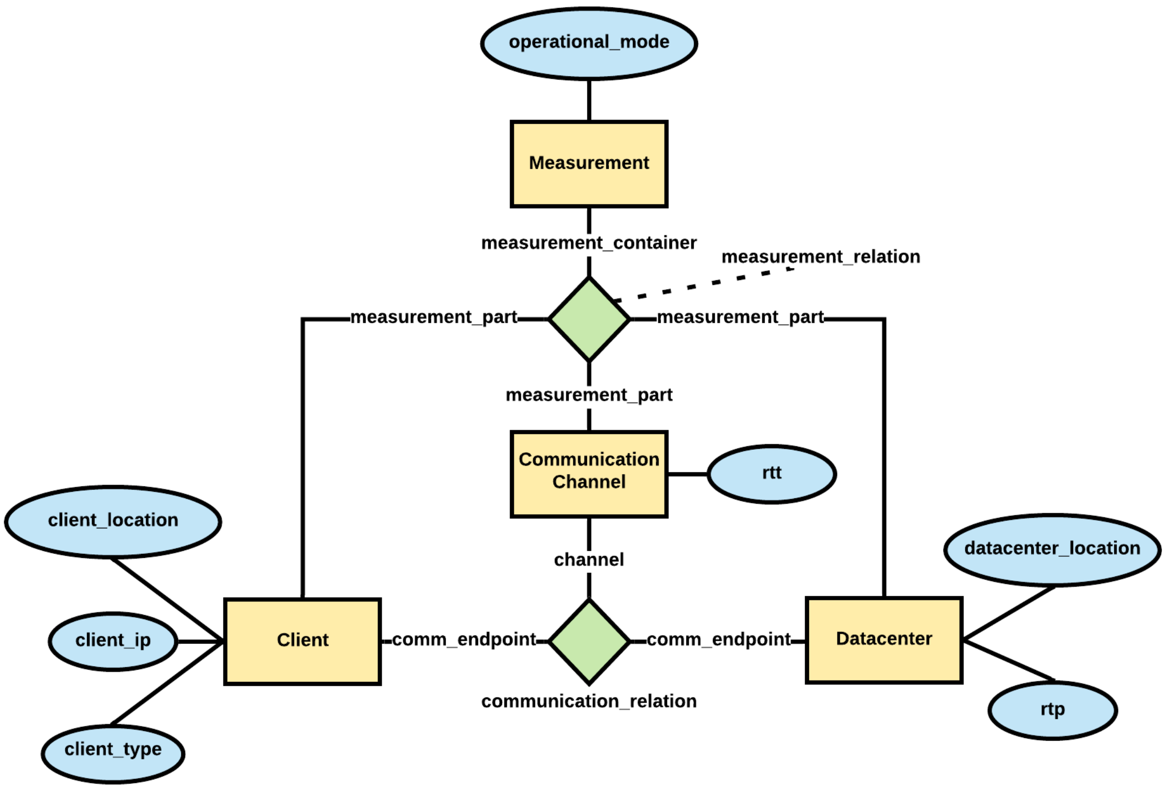


Figure 5.2: Grakn - Measurement Graph

The *Measurement* entity stores the information about the system state (faulty/normal operation, unclassified). The *MeasurementRelation* contains the communication hierarchy between the *Client* and the *Datacenter* through a *Communication Channel.*

The *Client* entity has three attributes: *client_location, client_ip, client_type. Datacenter* entity stores its specific information: *datacenter_location* and *rpt.* The *Communication Channel* entity represents the connection between the *Client* and the *Datacenter* and it contains the information about the *rtt.* The *CommunicationRealation* connects the *Client*, *Communcation Channel* and *Datacenter* entities.

Chapter 6

Answer Set Programming

The goal of the Answer Set Programming (ASP) is to provide a uniform framework for information fusion and reasoning. Standard ASP supports deductive reasoning techniques and running diagnoses (i.e., Abductive, Reiter's diagnosis).

Answer Set Programming (ASP) [8] is a declarative approach to modeling and solving search and optimization problems. It combines an expressive representation language, a model-based problem specification methodology, and efficient solving tools.

The ASP language represents domain and problem-specific knowledge, including incomplete knowledge, defaults, and preferences. Expressing these ideas in ASP often comes quite intuitively and naturally.

Answer Set Programming supports rapid prototyping and development of software, because of its strong declarative aspect. These advantages make it possible to use ASP as an information fusion framework (Chapter 7).

There are extensions for ASP-like (DLV) [31] implementations that enable logic-based diagnosis (Abductive-diagnosis, Reiter's diagnosis) [12] over DLV.

ASP Standardization Working Group at the University of Calabria partly standardizes the ASP language under the name of ASP-Core-2 [9].

6.1 Propositional setting

ASP is a special kind of declarative logic programming that directly supports advanced modeling constructs. The ASP notation used in this report follows the stable model semantics. Stable model semantics means here that the models generated out of the inputs represent the observations and the structure of the problem in the form of logic rules. The generated model must exactly fit the observations. However, several constructs serve to lose this strict requirement of exact fitting in order to deal with typical problems in processing practical observations. Two example are the following:

- Frequently, the observations are incomplete, which is why the logic has to deal with incomplete knowledge. Two of the usual engineering assumptions are using default values and the negation as failure. Negation as failure implements this principle by assuming the validity of some values unless a counterexample is found.
- This way, a model still cannot incorporate external knowledge: regularities that the model has to comply with in order to be sound in the engineering sense (i.e.,

natural law). However, observations are frequently noisy, virtually violating the requirements of exact fitting to a particular rule. Weak constraints serve to overcome this difficulty by substituting the requirements of exact fitting by the best fitting model. Weak constraints similarly optimize the objective function as a linear colleration does require the best fitting data points in terms of some of the Euclidean distance or in more general terms, like the loss function is minimizing in the case of neural networks. Several metrics are depending on the problem domain, which can be implemented in this form, practically every metric measuring the fitting can be encoded in the ASP.

In the following chapter, we will illustrate the core concept of ASP, and we address the implementation of different typical modeling fragments like those for causality. The underling solution mechanism this way can deliver a single solution compliant with all the hard constraints (including fitting to the observations), or the optimal one in the terms of weak constraints, or the entire solution space. Note that, the structure of a knowledge graph, including all the inherited relations and mutual dependencies of the different instances and concepts, can be transformed directly to ASP. This way, it is one of the powerful solution engines to solve large scale problems.

The underling solution engine is typically a sophisticated form of a constraint solver: the individual instances and concepts are merged into facts, variables and relations. This way they can be mapped to implication rules by inheritance.

Finally, a question can be formulated, which expresses the purpose of the model to be derived.

6.1.1 Building blocks of ASP

The building blocks for ASP programs are *atoms*, *literals*, and *rules*. Atoms are elementary propositions (factual statements) that may be true or false; literals are atoms a and their negations $not\ b$. Rules are expressions in the form:

$$r \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_n .$$

where r and a_i and b_i are atoms.

Intuitively, the rule above is a justification to "establish" or "derive" that the head (r) is true, if all literals of the body (a_i, b_i) are true in the following sense: a non-negated literal a is true if the atom a has a derivation, a negated one, $not\ b_i$, is true if the atom b does not have one. Facts are special rules without a body. The derivation of facts is always true.

$$fact \leftarrow .$$

ASP is based on the stable model semantics of logic programming [19]. The concept of a stable model is used to define declarative semantics for logic programs with negation as failure. Negation as failure is used to derive $not\ p$ (that p is assumed not to hold) from failure to derive p. In ASP $not\ p$ differs from the classical logical negation [20] of p $(\neg p)$.

For instance, Rule 6.1 means, we do not know about a fault in the system, we can assume that the system works properly. While, Rule 6.2 means, we know that the there is no fault in the system. The representation of incomplete knowledge is possible with this construction.

$$normal\_operation \leftarrow not\ fault\ . \tag{6.1}$$
$$normal\_operation \leftarrow \neg\ fault\ . \tag{6.2}$$

6.2 ASP solving process

The ASP solving process [17] (Fig. 6.1) consists of three main steps. First, **modeling** transforms a problem into a logic program. Then the **solving** process in which the ASP grounder and solver determine the stable models (answer sets) of the logic program. Finally, the **interpretation** of the stable models gives the solution.

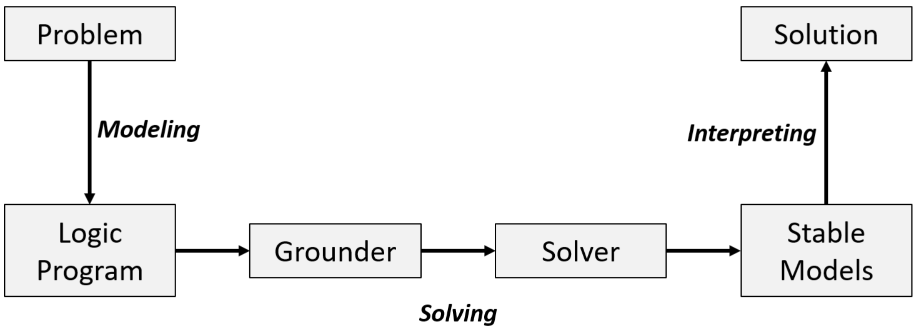


Figure 6.1: ASP solving process

6.2.1 Modeling in ASP

The ASP modeling process [17] uses the *Generate and Test* approach common to declarative and logic programs. The Generator generates all the feasible stable model candidates while the Tester eliminates the invalid ones. This way, the logic program is made up of ***facts and rules***, ***generator***, ***tester***, and ***optimizer*** modules.

The next section presents each building block with a short description and an example ASP program with its answer sets (under the lines).

Facts and rules (definite rules) Facts represent the propositional Extensional Database (EDB) knowledge, rules are used to deduct facts as an Intensional Database (IDB). The simplest example of a rule is if the truth of some statement depends on the truth of some other statements (e.g., The system operating properly if the network status is '*ok*' and we assume no fault in the system.). Note that the Generator, Tester, and Optimizer are also part of the IDB.

$$network\_ok \leftarrow\ . \tag{6.3}$$
$$normal\_operation \leftarrow network\_ok\ , not\ fault\ . \tag{6.4}$$

$$\{network\_ok, normal\_operation\}$$

Generator (disjunctive rules) Disjunctive rules describe the nondeterministic behavior of the system. A simple case is where we know for sure that at least one of some (two or more) conditions are true, but it is impossible to determine which exactly.

$$network\_ok \leftarrow . \tag{6.5}$$
$$normal\_operation \leftarrow network\_ok \ , not \ fault \ . \tag{6.6}$$
$$1 \ \{server\_load\_low; server\_load\_medium\} \ 1 \leftarrow normal\_operation \ . \tag{6.7}$$

$$\frac{}{\{network\_ok, normal\_operation, server\_load\_low\}, \\ \{network\_ok, normal\_operation, server\_load\_medium\}}$$

The numbers in the beginning and the ending of a disjunctive rule (Rule 6.7) indicate the minimum and the maximum number of the literals that can be justified as true.

Tester (integrity constraints) Integrity constraints are used to determine the system boundaries and eliminate the undesired results that the Generator processes. Constraints specify conditions that must not become true. They are formulations of possible inconsistencies.

$$1 \ \{server\_load\_low; server\_load\_medium; server\_load\_high\} \ 1. \tag{6.8}$$
$$\leftarrow not \ server\_load\_high \tag{6.9}$$

$$\frac{}{\{red\_signal\}, \{white\_signal\}}$$

The integrity constraint (Rule 6.9) eliminates the *server_load_high* from the answer sets.

Optimizer (weak constraints) Requirements that *should* be satisfied can be expressed with weak constraints. While the violation of integrity constraints eliminates the models, this does not apply to weak constraints. With the help of weak constraints optimization problems can be formulated. If an ASP model has weak constraints, not only the feasible answer sets are calculated, but The Best Models can also be chosen based on the number of violated weak constraints. A program can have several best models, where the sets of satisfied weak constraints does not have to match. Therefore, it is also possible to express an order of importance between the weak constraints by assigning coefficients to the individual constraints and then optimizing the sum of the weighted weak constraints. Another slightly different option is to prioritize the individual constraints. This way during execution, the constraints are optimized from top priority in descending order, and only one at a time is being considered.

$$\leftarrow\!\!-\!- \ rule. \ [level@weight] \tag{6.10}$$

Chapter 7

Information fusion

Information fusion is a crucial part of the workflow. It merges all the background knowledge from the engineering models and the causal model into a uniform qualitative model. This representation allows reasoning about the system-specific questions.

The input model of the step is the causal model and the information derived from the knowledge graph. The output model of the step is an ASP program that includes all the previous information.

This chapter presents the model transformation of the knowledge graph and the causal model into a uniform ASP representation.

7.1 Knowledge graph representation with ASP

Representing the Knowledge Graph with ASP requires the declaration of the knowledge schema and the definition of the knowledge instances as ASP rules [10]. The schema provides the structure and consistency for the instances. All the instances should comply with the schema.

Knowledge Schema *Entities*, their *Attributes* and the *Relations* between them are the main building blocks of an ontology-styled knowledge graph. Before adding the instances to the ASP logic program, the ontology schema has to be transformed into logic program definitions. This model transformation provides inconsistency checking and verifies if the hierarchical decomposition is reasonable and suitable for deductive reasoning.

Entities Definite ASP rules define the *Entities*, the *roles* that the given entity plays, and the *argument types* that the entity has.

The following example presents the ASP definition of the **Client** entity from the knowledge graph. It plays the *measurement_part* and *communication_endpoint* roles, and has three attributes: *client_ip*, *client_location*, *client_type*.

```
% Client
entity(client).

plays(client, measurement_part).
plays(client, communication_endpoint).

type_has(client, client_ip).
type_has(client, client_location).
type_has(client, client_type).
```

It is also necessary to add integrity constraints to the ASP schema. The integrity constraint ensures that the assigned values must be of attributes associated with the given entity.

```
% Entity integrity constraint
:- isa(I, EType), has(I, A, _), not type_has(EType, A).
```

Relations Defining a relation requires two types of definite rule. One is to define the relation itself and one that describes the associations. The example presents the Communication Channel relation between the endpoints (Client, Datacenter).

```
% Measurement Result relation
relation(communication_relation).

relates(communication_relation, communication_endpoint).
relates(communication_relation, channel)
```

Relations also require integrity constraints to restrict the possible associations to the related to certain entities (entity roles).

```
% Relation integrity constraints
:- role(RelInstace, _, Role), isa(RelInstace, RelType), not relates(RelType, Role).
:- role(RelInstace, I, Role), isa(I, EType), not plays(EType, Role).
```

Instances The instances in the knowledge graph can be represented as ASP expressions. The type of the instance and its attributes should be declared.

```
isa(secaucus_client, client).

has(secaucus_client, client_location, "Secaucus").
has(secaucus_client, client_ip, "64.20.37.202").
has(secaucus_client, client_type, "Java Client").
```

ASP representation of a relation contains the definition of the relation itself and the connections with the corresponding entities.

```
isa(comm_relation_12, communication_relation).

role(m_relation_12, secaucus_client, communication_endpoint).
role(m_relation_12, redmond_dc, communication_relation).
role(m_relation_12, channel_12, channel)
```

7.2 Representing causality with ASP

The ASP representation of causal models requires the identification of definite and disjunctive rules and the causal chain between the variables. This step requires background knowledge about the system because it is not always straightforward how to interpret paths or the branches. For example, a branch could be interpreted as mutual exclusion, or the source could affect both variables.

Example 1. *The ASP interpretation of the causality graph in Figure 7.1 deals with a single fault assumption. Dashed arrows represent additional external causal influences, while the dotted arrow represents a causal chain between the* ***client_send*** *and* ***client_receive*** *variables. This chain is not part of this example, but the original model contains it in detail.*

$$\{client\_router\_cfg\_error, client\_router\_hw\_error\} \leftarrow . \tag{7.1}$$

$$\{sending\_cfg\_error, receiving\_cfg\_error\} \leftarrow client\_router\_cfg\_error \ . \tag{7.2}$$

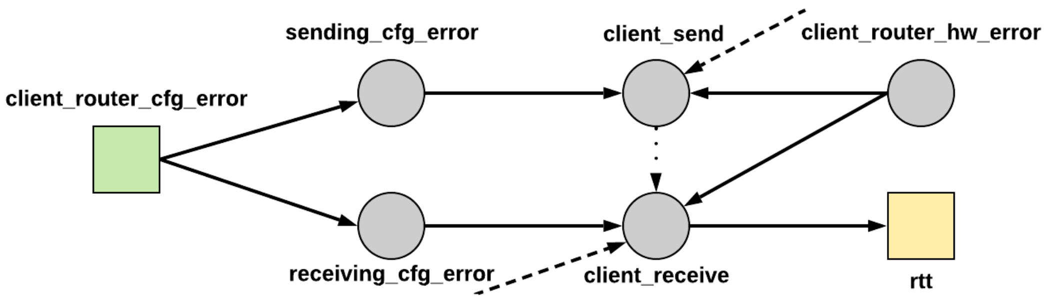


Figure 7.1: Causal Model of Client Router Error Modes

$$client\_send \ \leftarrow \ client\_router\_hw\_error \ . \tag{7.3}$$
$$client\_send \ \leftarrow \ sending\_cfg\_error \ . \tag{7.4}$$
$$client\_send \ \leftarrow \ \ldots \ . \tag{7.5}$$

$$client\_receive \ \leftarrow \ client\_router\_hw\_error \ . \tag{7.6}$$
$$client\_receive \ \leftarrow \ receiving\_cfg\_error \ . \tag{7.7}$$
$$client\_receive \ \leftarrow \ \ldots \ . \tag{7.8}$$

$$rtt \ \leftarrow \ client\_receive \ . \tag{7.9}$$

The examination of the rules in detail:

Rule 7.1 and 7.2 are disjunctive rules. The first rule (7.1) expresses that only one main fault source can be justified as true. The second (7.2) creates a hierarchy over the configuration error.

Rule 7.3 and 7.6 present that both *client_send* and *client_receive* are affected by the error of the client router.

Rule 7.4, 7.7 and 7.9 represent simple causal paths.

Chapter 8

Inductive Reasoning

Inductive and deductive reasoning are two distinguished reasoning a methods. Logical reasoning is based on the notion, that given a precondition (called premise), a conclusion (called logical consequence) can be reached through logical rules. While deductive reasoning comes to a certain conclusion by only examining the truth value of the premises, inductive reasoning results in a probability, thus expressing the uncertainty of the conclusion. The probability is based on the defined logical expressions and the given premises, which are viewed as evidence.[13] Inductive reasoning derivates general principles from specific observations.

Causal inference is a type of inductive reasoning. Causal inference draws a conclusion about a causal connection based on the conditions of the occurrence of an effect. Premises about the correlation of two things can indicate a causal relationship between them, but additional factors must be confirmed to establish the exact form of the causal relationship.

8.1 Inductive logic

In inductive logic, conclusion follows not with certainty, but only with some probability [41] [13].

For example,

```
(Premise 1) 90% of engineers drink coffee
(Premise 2) Susan is an engineer

(Conclusion) Susan drinks coffee
```

```
(Premise 1) Roughly 50% of the mangos are ripe in the market
(Premise 2) Today I bought a mango

(Conclusion) My mango is ripe
```

The acceptability of the conclusion infers mostly two questions: (i) How strong is the inference?; (ii) How high does the probability have to be before it is rational to accept the conclusion?

It is usually rather challenging to answer these questions accurately.

8.2 Inductive Learning of Answer Sets

The main idea behind the inductive learning of answer sets [26] is to find a hypothesis extended with the background knowledge that covers every positive example and does not cover any negative examples.

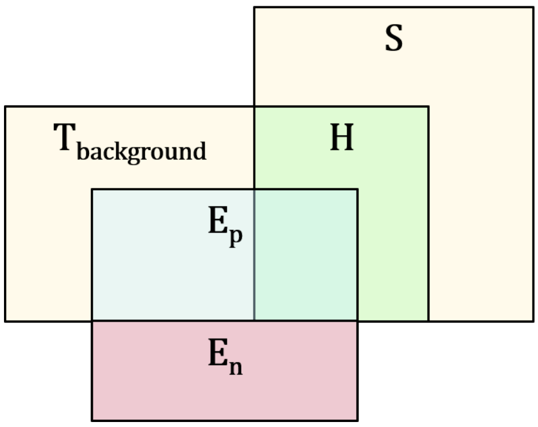


Figure 8.1: Sets

Formally (Fig. 8.1),

Given the background knowledge $T_{background}$, the positive and negative examples $E_{positive}, E_{negative}$ and the search space S, the goal is to find a hypothesis $H \subset S$, where

$$\forall p \in E_{positive} : T_{background} \cup H \vDash p \quad (8.1)$$

$$\forall n \in E_{negative} \cup T_{background} \nvDash n \quad (8.2)$$

8.2.1 ILASP

ILASP (Inductive Learning of Answer Set Programs) [27] is a machine learning system for learning ASP programs from examples. ILASP supports many ASP rules (normal rules, choice rules, hard constraints, weak constraints). ILASP uses Clingo [18] as the underling ASP solver.

Besides the ASP rules, ILASP also supports the learning of weak constraints [28] and both the brave and cautious reasoning in the following ways:

Brave induction The task of the brave induction is to find a hypothesis (augmented with $T_{background}$) such that it has at least one answer set, which satisfies the examples.

$$\exists\ s \in T_{background} \cup H$$

Cautious induction Cautious induction requires that the hypothesis (augmented with $T_{background}$) has at least one answer set and that every answer set satisfies the examples.

$$\forall\ s \in T_{background} \cup H$$

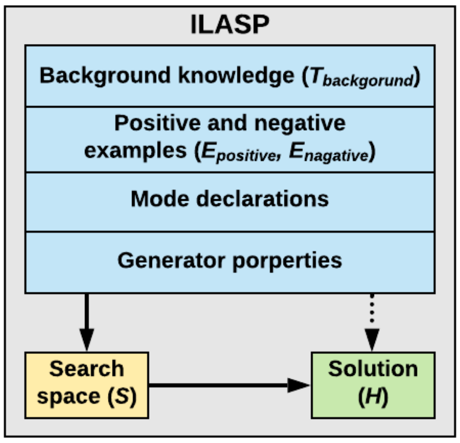

ILASP
Background knowledge ($T_{backgorund}$)
Positive and negative examples ($E_{positive}$, $E_{nagative}$)
Mode declarations
Generator porperties
Search space (S)
Solution (H)

Figure 8.2: ILASP

Defining an ILASP program (Fig. 8.2) requires the **background knowledge** that is represented by an ASP program; **positive and negative examples** in form of answer sets; **mode declarations** that are templates for generating the search space; and **generator properties** that restricts the search space generator.

8.2.1.1 Building blocks of the search space

ILASP searches the inductive hypothesis in the search space, so the search space should be defined in each ILASP program. The conventional way of specifying the search space is to give a language bias specified by mode declarations.

A mode declaration are rules. These rules will be generated by the search space generator in different permutations. Three main mode declarations are distinguishable:

Placeholders A placeholder is a term that can be replaced by any variable ($\#var(t)$) or constant ($\#constant(t, c)$.) of type t.

Head declarations Head declaration defines head ASP rules. ILASP supports normal head declarations ($\#modeh(var(t))$.) and disjunctive head declarations ($\#modeha(var(t))$.). Disjunctive mode declarations can have optional values that determines the possible minimum an maximum number of the justification of the variable.

Body declarations Body declarations defines body ASP rules ($\#modeb(var(t))$), that can be derived as facts, constraints or the actual body of a rule. ILASP also supports optimization (weak constraint) declarations ($\#modeo(var(t))$).

Generator properties The is possible to make restrictions about the search space. For example, the restriction of the maximum number of variables in any rule ($\#maxv(int)$) or the upper bound of the penalties by weak constraints ($\#max_penalty(int)$). Lower values for penalties are likely to increase the speed of computation, but in some cases larger bounds are needed.

8.2.1.2 Example

The example [29] present a simple learning task that present each building blocks of an ILASP program.

Example 2. *Simple learning example*

```
% Background knowledge
p :- not q.
q :- not p.

% Positive examples
#pos(p1, {q},{r}).
#pos(p2, {q,r},{}).
#pos(p3, {p},{}).

% Negative example
#neg(n1, {p,r},{}).

% Head declarations
#modeh(r).
#modeh(s).

% Body declarations
#medeb(r).
#modeb(s).
#modeb(p).
#modeb(q).
```

For example, the search space contains the following derived rules:

```
1 ~ r.
1 ~ s.
2 ~ s :- not r.
2 ~ s :- r.
2 ~ r :- not s.
2 ~ r :- s.
2 ~ r :- not p.
2 ~ r :- p.
2 ~ s :- not p.
2 ~ s :- p.
...
```

Running the ILASP learning algorithm on the program above results as the following hypothesis H:

```
s :- not r.
r :- not s, not p.
```

This way, the answer sets of the solution ($T_{background} \cup H$) will be:

`{p,s},{q,s},{q,r}`

Chapter 9

Automated Model Extraction

This chapter introduces how ILASP can be applied to reason about different aspects of the measurement. It uses the unified ASP model extended with the ILASP specific parameters. Each subsection describes how to define the background knowledge, positive and negative examples and the search space to answer specific questions.

The first section of the chapter introduces QCA for comparison purposes. Later in this chapter three questions will be answered about the benchmarking example.

9.1 Qualitative Comparative Analysis (QCA)

Qualitative Comparative Analysis (QCA) [38] is a data analysis technique for determining which logical conclusions a data set supports. The analysis has three main steps:

1. The analysis begins with listing and counting all the combinations of variables observed in the data set. All distinguishable observation occurs once in the analysis. These observation can extracted from BDDs or MDDs.
2. It follows the applying the rules of logical inference (inferential logic or bool algebra) to determine which descriptive inferences or implications the data supports.
3. The third step reduces the number of inferences. It finds the minimum set of inferences supported by the data.

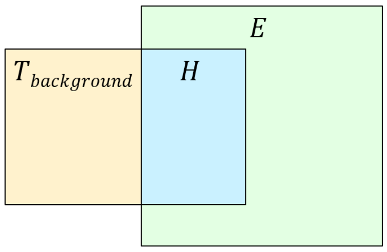
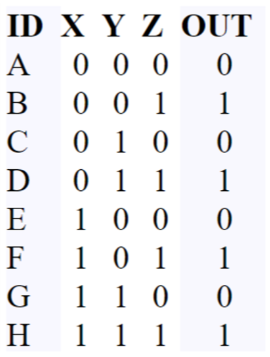


Figure 9.1: QCA

The goal (Fig 9.1) is to find a hypothesis H that explains the selected outcome variable ($E = \{E_{positive}, E_{negative}, E_{contradictions}\}$) by reducing the included condition variables ($T_{background}$). Note that, here the background knowledge is restricted only to the observable variables.

$$T_{background} \cap H \Rightarrow E$$



ID	X	Y	Z	OUT
A	0	0	0	0
B	0	0	1	1
C	0	1	0	0
D	0	1	1	1
E	1	0	0	0
F	1	0	1	1
G	1	1	0	0
H	1	1	1	1

Figure 9.2: Observations

9.1.1 Example

This example present a simple analysis about logical variables (X, Y, Z, OUT). The observations are labeled with an ID (Fig. 9.2).

Running the analysis indicates (Fig. 9.3 - striped rectangle), that the Z variable with the value 1 influences the outcome. This way, the result can be formulated as,

$$T_{background} \cap Z\{1\} \Rightarrow \{B, D, F, H\}$$

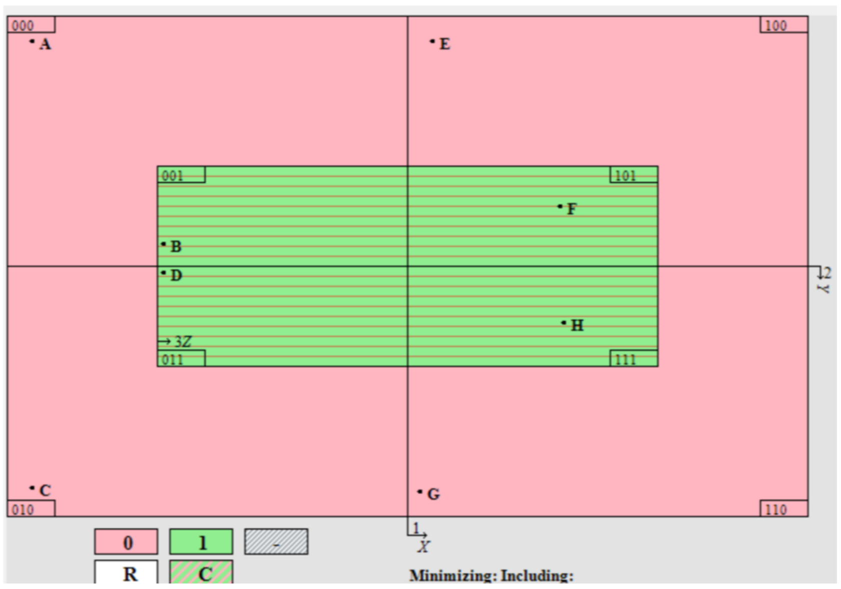


Figure 9.3: Visualization of the hypothesis

9.2 Inductive reasoning on the benchmarking example

This section present the inductive reasoning on the benchmarking example. Each presented example covers different cases (answers different questions) about the benchmarking example.

It requires to consider some aspects:

- One aspect is the granularity of the diagnosis. It defines the abstraction level of the answer. In case of an error detection, it refers to the replacement unit (e.g., if the diagnosis indicates that the client router causes the fault then it requires the replacement of the router or further analysis should be involved to examine the details).
- In case of a fault model, it is also required to define the possible number of faults. There is two possible assumptions to make: (i) Single fault assumption is the assumption that failures are only rarely the result of the simultaneous occurrence of two (or more) faults; (ii) However, multiple fault assumption deals with multiple fault occurrences in the same time.
- For the causal models, it is possible to attach probabilities to the causal connections. ASP allows weights on weak constraints (like a loss function in ML techniques) to express the probability of the covariates.

9.3 Reasoning: Observed Variables

Objective The objective of this task is to identify, how a subset of observed variables affects a selected variable or the faulty behavior of the system.

Result The result indicates the relations between the variables (i.e., which exact variable or value affects the outcome).

Note that, this specific example uses instance abstraction to reason about the outcome. The other way, is a more abstract reasoning that determines only the the type of the variable (*location*) not the specific instance (*location*(*secaucus*)).

9.3.1 ILASP program

Backgorund knowledge The background knowledge for the program is made of the observations and the causal model.

Each **observation** is an ASP literal (Rule 9.1) with five observed parameters. The parameters are: *client_location*, *client_type*, *dc_location*, *rtt*, and *rpt*.

$$obs(client\_location, client\_type, dc\_location, rtt, rpt)\ . \tag{9.1}$$

The **observations** are represented as a disjunctive rule (Rule 9.2). This construction enables to examine each observation at once.

$$1\ \{obs_1(\dots), obs_2(\dots), \dots, obs_N(\dots)\}\ 1. \tag{9.2}$$

This approach requires the examination of each observed variable individually. In this case, additional rules (e.g. Rule 9.3 and 9.4) splits the observation literal into individual variables.

$$client\_location(LOCATION) : -obs(LOCATION, \_, \_, \_, \_). \tag{9.3}$$

$$dc\_location(DC) : -obs(\_, \_, DC, \_, \_). \tag{9.4}$$

Examples In the first place, positive examples made up of observations with the operational mode labels (Rule 9.5). As the goal is the analysis of the faulty behavior, the cluster labels should added to the normal behavioral observations as an exception (rule 9.6). For example:

$$\#pos(f1, \{obs(secaucus, java, redmond, high, low), faulty\_operation\}, \{\}). \tag{9.5}$$
$$\#pos(n1, \{obs(peyton, java, redmond, low, low)\}, \{faulty\_operation\}). \tag{9.6}$$

Search space The search space declaration forms the question which the reasoning is about. In this specific case: The search space contains all the observed variables with their values (e.g., for locations: Rule 9.7 - 9.9) as body declarations.

$$\#modeb(client\_location(lansing)). \tag{9.7}$$
$$\#modeb(client\_location(chicago)). \tag{9.8}$$
$$\#modeb(client\_location(secaucus)). \tag{9.9}$$
$$\dots \tag{9.10}$$

The head declaration contains the outcome variable (e.g., Rule 9.11).

$$\#modeh(faulty\_operation). \tag{9.11}$$

9.3.2 Results

In case of multiple solutions the sequence of the search space determines a priority over the variables. In this example, two different variable ordering results in two different solutions. In the first case (Rule 9.12) the prioritized variable was the *client_location*. While, in the second case (Rule 9.13), *rtt* was ahead of the line.

$$faulty\_operation : -client\_location(secaucus). \tag{9.12}$$

and

$$faulty\_operation : -not\ rtt(low). \tag{9.13}$$

Rule 9.12 indicates, that the observations from *Secaucus* infers faulty operation. Rule 9.13 indicated, that faulty operation occurs when the *rtt* is not low.

9.3.3 Validation

It is possible to check the validity of the result by using the GRAKN query language or extend the ASP program with the generated rule(s) and perform deductive reasoning about the solution.

This example results in a cautious reasoning. The validation of the result shows that the result covers cases that are outside of the faulty operational domain. For example, Rule 9.13 covers the cases where the *rtt* variable is normal. These data points are categorized into the "Unclassified" class, it means they require further analysis.

9.4 Reasoning: Hidden Causal Effects

Objective The objective of the hidden causal effect analysis is to determine which non-observed causal effects could lead to the faulty behavior of the system.

Result The result of the analysis is an ASP rule (or multiple rules) that determine a causal connection between the non-observed causal effects and the faulty observations.

9.4.1 ILASP program

The ILASP program for the hidden causal effect analysis follows the structure of traditional ILASP programs. The program should contain both the observations and the causal model of the system.

Background knowledge The background knowledge for the program is made of the observations and the causal model.

Each **observation** is an ASP literal (Rule 9.1) with five observed parameters.The parameters are: *client_location*, *client_type*, *dc_location*, *rtt*, and *rpt*.

The **observations** are represented as a disjunctive rule (Rule 9.2). This construction enables to examine each observation at once.

The **causal model** follows the single fault assumption approach. It means that only one fault can be evaluate true in an answer set.

$$\{client\_router\_cfg\_err, client\_router\_hw\_err\} \leftarrow \ . \tag{9.14}$$
$$\{sending\_cfg\_err, receiving\_cfg\_err\} \leftarrow client\_router\_cfg\_err \ . \tag{9.15}$$

Weak constraints make it is possible to assign **probabilities** to the specific fault assumptions. The higher the weight of the weak constraint the lower the probability of the fault. The following example presents that *sending_cfg_err* and *receiving_cfg_err* have the same probability while *client_router_hw_err* has a lower probability.

$$\leftarrow\!\sim sending\_cfg\_err. \ [1@5] \tag{9.16}$$
$$\leftarrow\!\sim receiving\_cfg\_err. \ [1@5] \tag{9.17}$$
$$\leftarrow\!\sim client\_router\_hw\_err. \ [1@10] \tag{9.18}$$

Examples In the first place, positive examples made up of observations with the operational mode labels (rule 9.5). As the goal is the analysis of the faulty behavior, the labels should added to the normal behavioral observations as an exception (Rule 9.6).

Search space declaration As mentioned before, the search space declaration forms the question which the reasoning is about. Reasoning about the hidden causal effects requires head and body declarations. The head declaration is a literal that is the object of the question. The body declarations are literals that can affect the object. In the following example, the object is the *faulty_operation* rule and the body declarations are the fault modes:

$$\#modeh(faulty\_operation). \tag{9.19}$$

$$\#modeb(sending\_cfg\_err). \tag{9.20}$$
$$\#modeb(receiving\_cfg\_err). \tag{9.21}$$
$$\#modeb(client\_router\_hw\_err). \tag{9.22}$$

9.4.2 Results

The result of the defined ASP program infers a new ASP rule:

$$faulty\_operation \leftarrow receiving\_cfg\_err \; . \tag{9.23}$$

It means, that likely the *receiving_cfg_err* causes the faulty operation in the system.

Note that, in this example the *receiving_cfg_err* and *sending_cfg_err* have the same probability. In this case the sequence of the body declaration in the search space defines a priority over the variables. The restructuring of the body declarations is required to get the other (also valid) solution.

9.4.3 Validation

As the result is an unobservable variable the validation process is not that simple then is the previous example. The validation process of this task requires a further manual analysis of the result (e.g., checking the configuration for the client router). Furthermore, it can require a new measurement campaign with the new configuration.

9.5 Reasoning: Fault Indication

Objective The goal is to identify the phenomenon of a specific event. This approach uses the observable variables to reason about the unobservable fault candidates.

Result The result of the program identifies the phenomenon that can be observed (e.g., during system runtime).

This example uses the result of the Hidden Causal Effect analysis. The goal is to determine the variables with exact values that indicate the fault.

The main difference between the *Observable Variable* and this method is the direction of the reasoning. While the first method defines the faulty operation by the observable variables, this method determines which observable phenomena occurs in case of the fault.

Note that, because the state space of the observable variables are large, a restriction was made to the phenomena candidates which requires a sequential diagnosis of the problem.

9.5.1 ILASP program

The structure of the ILASP program for fault detection is similar to the program for hidden causal effect analysis. The main differences are observable in the state space declaration, hence the goal of the reasoning is to identify the phenomenon of the faulty behavior.

Backgound knowledge The representation of the observations (Rule 9.1 and 9.2) and the causal model (Rule 9.14 and 9.15) is the same as in the previous examples.

Some other constraints should be added to the background knowledge about the mutual exclusion in the values of the observed variables (e.g., Rule 9.24)

$$\leftarrow rtt(low), rtt(high). \tag{9.24}$$

Examples The examples covers the observations with the labels as in the previous examples, but they are extended with observable variable specific rules.

$$\#pos(f1, \{obs(\dots), rtt(high), \dots, faulty\_operation\}, \{\}). \tag{9.25}$$
$$\#pos(n1, \{obs(\dots), rtt(low), \dots\}, \{faulty\_operation\}). \tag{9.26}$$

Search space The search space contains the previously extracted result (Rule 9.27) as a head declaration and the observable variables (Rule 9.7 - 9.9) as body declaration.

$$\#modeh(receiving\_cfg\_error). \tag{9.27}$$

9.5.2 Results

$$rtt(low) \leftarrow \text{ not } receiving\_cfg\_error\ . \tag{9.28}$$
$$rtt(medium) \leftarrow \text{ not } receiving\_cfg\_error\ . \tag{9.29}$$
$$rtt(high) \leftarrow receiving\_cfg\_error\ . \tag{9.30}$$

The result indicates the phenomenon of the $receiving\_cfg\_error$ is the high rtt value.

9.5.3 Validation

This method also can be checked by the instance based reasoning method (GRAKN query, ASP reasoning).

9.6 Variable ordering and probabilities

There is two main approaches to prioritize over the fault modes:

- **Variable ordering**: ILASP takes in account the sequence of the search space. If it finds a solution at the beginning of the search space then it stops and returns with re result. This allows the prioritization over the variables. It requires systematic search to find all the possible outcomes.
- **Weak constraints**: Weak constraints help to overcome the sequence restrictions of ASP. It is possible to attach a weight to each variable and this way prioritize them. This mechanism of ASP enables the construction of loss function like functionalities.

9.7 Efficiency considerations

Building an efficient ILASP program requires a *careful considerations*, as its core is *search based*. This way an unfortunate formulation of the model can result in a practically infeasible run.

The main factors to be taken into account are the following ones:

- The **size of the state space** influences the performance of the solver. For a large state space (i.e., a state space sepnd 20 variables and their combinations in the state space) cause a very long solving time. The proper setting of the state space declarations can reduce its size and results in an adequate solving time. However, it is possible to reduce the state space by the iterative process of a problem (sequential diagnosis) from higher abstraction level to a more specific level.
- The other efficiency constraint can be derived from the ASP itself. Using **ASP variables** (Rule 9.32) **instead of string literals** (Rule 9.31) decreasing the solving time. Using string literals result in a very slow solving process either in case of small problems. The comparison of the string literals takes much longer then using ASP variables.

$$client\_location(secaucus). \tag{9.31}$$

$$client\_location("Secaucus"). \tag{9.32}$$

In summary, the exploitation of the offered generator and search space declaration optimizations by ILASP is required to achieve a proper solving time in case of large problem. Over and above, the string to variable should be performed in the previously presented model transformation phase.

Chapter 10

Summary

10.1 Evaluation of the work

System identification becomes one of the mainstream technologies for *dimensioning flexible and scalable applications* like CPS, cloud computing, etc. Due to the complexity of the system, faithful models necessitate a *solid empirical background*. Traditionally, EDA is used to *extract a model out of observation* data. However, the growing complexity of target systems causes EDA to become a bottleneck. In this research report, the objective was to explore the opportunities offered by *machine learning* technologies. While, it is in an initial phase, some results are very promising.

The goal was to support EDA with *machine learning* and *reasoning techniques*. Reasoning techniques perform step-by-step checks of the abstract model in order to assure consistency and completeness. The method is the combination if discretization, knowledge fusion, deductive, and inductive logic-based automated system model extraction.

Qualitative modeling and *knowledge graph* management of the model provides an abstract semantic framework for **information fusion**. The information fusion is supported by *ASP*, as it has rich knowledge representation capabilities.

Deductive reasoning checks the compliance and completeness of the observations and their respective evolving models. The *inductive learning* of ASP programs serves as an automated mechanism delivering a compact abstract model of the observed system.

This way, the toolchain supports (visual) EDA by checking the completeness and consistency of the evaluation process and identifies a high-level abstract system model based on the observations.

10.1.1 Automated model extraction

The goal of the report is to answer questions about both the observable and unobservable causal relations on a benchmarking example to support the system diagnosis.

For instance, we will show from the observation instances that we can create **general hypotheses** that seem to be valid for all the data. Such hypotheses can appear at a higher level of abstraction (i.e., if the experiment describes the dependence between the inputs and outputs of a component, then a hypothesis can generalize the faulty behavior for all the similar component types). This way, if the validity of the hypothesis is proved, a reusable bias of knowledge can be gained.

Three examples were presented:

- One that identifies the way of observing the effect of a subset of variables on a selected variable or the faulty behavior of the system. This mechanism is similar to QCA.
- The next example differs from QCA in the way that it works with unobservable variables. The objective of the hidden causal effect analysis is to determine which non-observed causal effects could lead to the faulty behavior of the system.
- The last example presented an inductive inference about the identification of the phenomenon of a specific event.

This report focuses on a specific aspect of the usage (measurements). However, there are further potential use-cases (e.g., time-series data).

10.2 Further research

10.2.1 Traditional machine learning techniques

The presented approach shows similarities (e.g., loss function) to traditional machine learning (ML) techniques. There are uncovered aspects that require further research on how to apply them in our context.

Now, the verification of the metrics relies on manual checks. The examination of how the verification metrics of ML fit into this approach is required to provide a more precise result.

Another possibility is the verification of neural networks. With additional causal inferences, the output of the networks will be explainable in the qualitative domain.

10.2.2 ILASP functionality

Beyond the presented functions of ILASP, there are many more exciting approaches. For example, ILASP can learn from noisy examples [30], which can ease the pre-processing of the data by automatically filtering the noisy data.

10.2.3 Design patterns

The report presented three possible design patterns that help in measurement data analysis by answering questions about causal inferences.

The generalization of problems and the automation of the process could help the user to use it in several contexts.

More problems could be solved with this approach. Applying design patterns to specific problems can help to speed up the analysis process.

Acknowledgements

First and foremost, I would like to thank my advisor Prof. Dr. András Pataricza, for the valuable assistance and continuous support. Furthermore, I would like to thank Csenge Kilián for her excellent guidance in machine-learning and the document reviews.

The results presented in the report were established in the framework of the professional community of Balatonfüred Student Research Group of BME-VIK to promote the economic development of the region. During the development of the achievements, we took into consideration the goals set by the Balatonfüred System Science Innovation Cluster and the plans of the "BME Balatonfüred Knowledge Center", supported by EFOP 4.2.1-16-2017-00021.

The project was funded by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013).

List of Figures

Bibliography

[1] OPC UA/DDS Gateway. `https://www.rti.com/blog/announcing-the-opc-ua-dds-gateway-standard`.

[2] Grakn Labs Ltd: GRAKN.AI. `https://grakn.ai`.

[3] OpenFog Consortium. `https://www.openfogconsortium.org`.

[4] Kathy H Abbott, Paul C Schutte, Michael T Palmer, and Wendell R Ricks. Faultfinder: A diagnostic expert system with graceful degradation for onboard aircraft applications. 1988.

[5] Andrew Banks and Rahul Gupta. MQTT version 3.1. 1. *OASIS standard*, 29, 2014.

[6] James C Bezdek and Nikhil R Pal. Cluster validation with generalized dunn's indices. In *Proceedings 1995 Second New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, pages 190–193. IEEE, 1995.

[7] Andrea Bondavalli, Sara Bouchenak, and Hermann Kopetz. *Cyber-Physical Systems of Systems: Foundations–A Conceptual Model and Some Derivations: the AMADEOS Legacy*, volume 10099. Springer, 2016.

[8] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[9] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. Asp-core-2: Input language format. *ASP Standardization Working Group*, 2012.

[10] Domenico Corapi. Is Graql a logic program . `https://blog.grakn.ai/isa-graql-logic-program-8af1258054a4`.

[11] Johan De Kleer and John Seely Brown. A qualitative physics based on confluences. *Artificial intelligence*, 24(1-3):7–83, 1984.

[12] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. The diagnosis frontend of the dlv system. *AI Communications*, 12(1-2):99–111, 1999.

[13] Aidan Feeney and Evan Heit. *Inductive reasoning: Experimental, developmental, and computational approaches.* Cambridge University Press, 2007.

[14] Kenneth D Forbus. Qualitative process theory. *Artificial intelligence*, 24(1-3):85–168, 1984.

[15] Kenneth D Forbus. Qualitative modeling. *Foundations of Artificial Intelligence*, 3: 361–393, 2008.

[16] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: Systems Modeling Language*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008. ISBN 9780080558363, 9780123743794.

[17] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Answer set solving in practice. *Synthesis lectures on artificial intelligence and machine learning*, 6(3):1–238, 2012.

[18] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo= asp+ control: Preliminary report. *arXiv preprint arXiv:1405.3694*, 2014.

[19] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080, 1988.

[20] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New generation computing*, 9(3-4):365–385, 1991.

[21] Anatoliy Gorbenko, Vyacheslav Kharchenko, Seyran Mamutov, Olga Tarasyuk, and Alexander Romanovsky. Exploring uncertainty of delays as a factor in end-to-end cloud response time. In *2012 Ninth European Dependable Computing Conference*, pages 185–190. IEEE, 2012.

[22] Edward Griffor, David Wollman, and Christopher Greer. Framework for cyber-physical systems. , National Institute of Standards and Technology - Cyber Physical Systems Public Working Group, 2016.

[23] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *Journal of intelligent information systems*, 17(2-3):107–145, 2001.

[24] ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. Standard, International Organization for Standardization, March 2011.

[25] ISO/IEC/IEEE 42010:2011. Systems and software engineering – Architecture description. Standard, International Organization for Standardization, 2011.

[26] Mark Law. Inductive learning of answer set programs. 2018.

[27] Mark Law, Alessandra Russo, and Krysia Broda. The ILASP system for learning answer set programs. `https://www.doc.ic.ac.uk/~ml1909/ILASP`, 2015.

[28] Mark Law, Alessandra Russo, and Krysia Broda. Learning weak constraints in answer set programming. *Theory and Practice of Logic Programming*, 15(4-5):511–525, 2015.

[29] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs v3. 1.0. 2017.

[30] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs from noisy examples. *arXiv preprint arXiv:1808.08441*, 2018.

[31] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)*, 7(3):499–562, 2006.

[32] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[33] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC unified architecture.* Springer Science & Business Media, 2009.

[34] Gerardo Pardo-Castellote. Omg data-distribution service: Architectural overview. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, pages 200–206. IEEE, 2003.

[35] Judea Pearl. *Causality: models, reasoning and inference*, volume 29. Springer, 2000.

[36] Judea Pearl and Dana Mackenzie. *The book of why: the new science of cause and effect.* Basic Books, 2018.

[37] Antal Péter, Antos András, Horváth Gábor, Hullám Gábor, Kocsis Imre, Marx Péter, Millinghoffer András, Pataricza András, and Salánki Ágnes. Intelligens adatelemzés. 2014.

[38] Charles C Ragin. *The comparative method: Moving beyond qualitative and quantitative strategies.* Univ of California Press, 2014.

[39] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[40] Ian Shrier and Robert W Platt. Reducing bias through directed acyclic graphs. *BMC medical research methodology*, 8(1):70, 2008.

[41] Brian Skyrms. Choice and chance: An introduction to inductive logic. 2000.

[42] Johannes Textor, Alexander Idelberger, and Maciej Liśkiewicz. Learning from pairwise marginal independencies. *arXiv preprint arXiv:1508.00280*, 2015.

[43] Johannes Textor, Benito van der Zander, Mark S Gilthorpe, Maciej Liśkiewicz, and George TH Ellison. Robust causal inference using directed acyclic graphs: the r package 'dagitty'. *International journal of epidemiology*, 45(6):1887–1894, 2016.