Federal Office
for Information Security

Security of AI-Systems: Fundamentals

Security Considerations for Symbolic and Hybrid AI

Authors

Dr. Christian Müller, DFKI
Roland Vogt, DFKI
Dr. Andreas Nonnengart, DFKI
PD Dr. Matthias Klusch, DFKI
Dr. André Meyer-Vitali, DFKI

Publisher
Federal Office for Information Security
P.O. Box 20 03 63
53133 Bonn
E-Mail: ki-kontakt@bsi.bund.de
Internet: https://www.bsi.bund.de

Table of Contents

1 Introduction

Until recent years, cybersecurity was mostly concerned with an arms race of attackers and defence mechanisms on the level of admission control, protection of data transmission, cryptography, and so forth. On the other side, functional safety of software-based systems dealt with faults in the system behaviour, which are caused by electro-mechanical errors in one of its sub-components or systematic errors (like bugs). With AI-based systems becoming ever more wide-spread and complex, both paradigms need to be extended and, in a way, they are growing closer together. AI security and AI safety have a large overlap. Part of what AI safety tries to cope with are perturbations (or distribution shifts) that occur "naturally", for example because the environment changes (day to night, summer to winter, Europe to Asia, simulation to reality, etc.) or because the domain gradually evolves (demographic changes, generation changes, etc.). Seldom, events can occur that have never been considered in training, causing an undesired emergent behaviour (misclassification, wrong decision, etc.) at inference time. Couldn't we also say that the unexpected event caused an "evasion"? AI security, aside from assuming an adversary, deals with similar problems. Data poisoning is the attempt to smuggle-in examples to the training set that decrease the accuracy of the system (or increase test error), thereby trying to be as efficient and subtle as possible. Evasion attacks alter the inference situation, either by manipulating the environment or otherwise making sure that the system receives input that leads to misclassifications. In a sense, they are trying to create an event that was not expected during training. It is possible that poisoning and evasion attacks are combined in a sense that the poisoning attack introduces a trigger for the later evasion. The proximity between the two problem domains exists on all levels: in highly automated driving, for example, it is plausible to describe a case in which the car with ego-centric vision is tricked by the behaviour of another vehicle (agent) exhibiting a strange manoeuvre. If we had reasons to assume that the agent's "adversarial driving" was based on knowledge about the inner working of the ego car, we would call it a security breach – otherwise a safety issue. It becomes apparent that the distinction is somewhat arbitrary. Moreover, if we look at the body of literature in AI security, the game of finding new attacks, on the one side, and inventing new ways of defending them, on the other, could also be framed under the umbrella of research on robustness.

This study is on security of symbolic and hybrid AI and, therefore, we shall introduce those terms before we further elaborate on the considerations above. Symbolic AI methods are called that way, because the input they process are symbols, i.e., they are categorial ([cat, dog, cow], [Monday, Tuesday, Wednesday]) or otherwise inherently semantic (data value of 20 with inherent semantic 20° Celsius, 7-day incidence of 4000). Sometimes, the term "GOFAI" (for "good old-fashioned AI") is used, but the valuing connotation can be misleading. "Knowledge representation and reasoning" is another way to outline the set of methods that we are referring to. But altogether, in the context of this study, "symbolic AI" is most useful. In contrast, methods dealing with data that is purely numerical, can be called sub-symbolic. A deep neural network for pixel-based image segmentation is a common example. Symbolic AI systems are mostly based on logical concepts and relations, while sub-symbolic ones are predominantly statistical. The question we shed light on in Chapter 3 of this study is: what are the differences in attacking a symbolic/semantic system as opposed to a numerical/statistical one?

In this study, we focus on hybrid AI (Medsker, 1995) that is a combination of both and is now also called "neuro-symbolic" or "neuro-explicit". In contrast to works in the first "wave" of hybrid AI (see (Klusch, 1993); (Müller, 2005) for examples), many of contemporary neuro-explicit systems are more advanced than a layered conglomerate of two models. Rather, they are entangled in various ways such that the learning is either supported by reasoning or planning, or vice versa. In Chapter 4, we introduce common variants and discuss the security implications.

In the literature, one finds by far more contributions on security for sub-symbolic AI, especially (deep) neural networks, than for symbolic or even neuro-explicit AI. As a matter of fact, the latter can be described as largely unexplored. However, the proximity of AI-security to AI-safety in the sense of intended vs. natural perturbations, as described above, offers the possibility of initial hypotheses by analogy. Indeed, the safety community has long assumed that hybrid AI can make a significant contribution to overcoming the drawbacks of purely statistical AI. The issue of robustness to distribution shifts is directly transferable here. There are reasons to assume that a method that is more robust to naturally occurring perturbations will also be harder to attack with deliberately induced ones – and vice versa. However, this is an open field of research so that a robustness test cannot replace the security check. In addition, the safety feature of explainability also has potential for increasing AI security. Hence, in this study, we approach the topic from multiple perspectives. For both symbolic and hybrid AI, we line-up a selection of methods, describing them using examples, and point out cybersecurity and AI security aspects. Altogether this paints a picture of what we can call the hypotheses on the security of symbolic and hybrid AI, which in most cases require concrete and detailed empirical underpinning, formulated as research questions for the community to tackle.

Before we move on to a more detailed description of our concept of security in Chapter 2, we would like to discuss special features of this study. First, it is important to explain why there is a sub-chapter on "Learning" (3.4) in Chapter 3. The dividing line between sub-symbolic and symbolic methods is not sharp. Quite often it is a matter of interpretation or is decided only in the concrete application whether a certain method belongs to the one or the other class. Thus, there are machine learning methods that are quite clearly symbolic (such as inductive logical programming), while knowledge representation may include data whose semantic quality is debatable. In other cases, the community does not agree on where exactly a method should be located. Bayesian networks (see Section 3.4.2), for example, are sometimes subsumed under probabilistic reasoning and sometimes under machine learning. We want to emphasize that it is not the goal of this study to clarify these terms. We have made a pragmatic selection of methods and a pragmatic classification as well. In this sense, Section 3.4 serves to consider machine learning methods that are more symbolic than common (deep) neural networks. Another feature concerns the use of design patterns to describe methods and, in many cases, attack patterns. The underlying idea was that this firstly leads to a better understanding of the methods and secondly that generalisations can be found regarding the security aspects.

2 Security and Patterns

2.1 Security

Cybersecurity. In this study, we subsume under cybersecurity processes that fall under the following pattern: attackers manage to overcome the authentication mechanisms of a computer (usually a server). From there, they can manipulate or simply steal artifacts (models, data, software). In the context of the present study, the following questions are relevant with respect to this scenario: Which parts of a symbolic model (system) are likely targets? Which manipulations are conceivable or probable and which competencies on the part of the attacker would have to be assumed? What purpose could the attackers pursue and what would corresponding scenarios look like in concrete applications? The answers to such questions, often accompanied by numerical or fuzzy ratings such as "high", "medium", "low" or "20,000 €", can then be used for risk assessment, which allows a company to identify focal points regarding threat defence and the respective protection requirements. While there are certainly different answers for the individual methods to be assumed, the mitigation measures mentioned in this context essentially refer to improving access control.

AI security. The scenario within what we call AI security in the context of this study is significantly different from the above. It is commonly referred to as “adversarial machine learning”, but this term implies a restriction to ML algorithms and is therefore not correct in the context of this study. Here we assume that an "attacker" develops a method that is precisely adapted to the weaknesses of a given AI algorithm. How exactly the course of the "attack" is, plays a subordinate role in AI security. This is also the reason why we have put the terms "attacker" and "attack" in quotation marks here. The discourse on AI security is predominantly conducted on a theoretical and cooperative level. Author A finds a method for attacking method X. Author B then proposes an improvement to X that defends against the attack. Often A and B are even the same person (or group). Attack and defence are in this sense a framework in which an incremental improvement process takes place. The same process could also be framed as A exposing weaknesses and B fixing weaknesses. In this case, we would find ourselves, as already described at the beginning, in a style that also prevails in AI safety. This is not to exclude that there are or will be actual attackers in the sense of AI security. But pointing out the focus away from the actual attack to a development of an adversarial method is important for understanding our implications.

We focus on the three predominant patterns of AI security attacks: data poisoning, evasion, and privacy attacks. Moreover, we also look at model stealing. For different AI methods, the relative importance of the attack pattern changes. For example, a method without learning from data is not susceptible to data poisoning attacks. But let us first clarify what we mean with these terms.

The idea with data poisoning is to create training examples that make it difficult for the learner to generate a meaningful decision hyperplane. So, if we want to explain to a child what birds are and what fish are, then in fact penguins and flying fish are kinds of adversarial examples. We have to deal with that, we explain the exceptions, but in doing so we have made our model more complex. The trick with poisoning attacks is that they are not so obvious and therefore hard to identify. The problem of generating adversarial training examples can be formulated as a search problem: what is sought is a minimal perturbation of a natural example, but one that generates a maximal increase in test error. In principle, this is possible for all learning systems. The difference between the methods is the way the search problem has to be solved, the lower bound for the size (visibility) of the perturbation as well as the upper bound for the effect.

How would data poisoning play out as an actual cybersecurity attack? A company often has large training data sets generated off-site and, in addition, often from external parties. Under these circumstances, it is conceivable that such manipulation is taking place. However, it can be considered unlikely that non-specific poisoning takes place with the aim of increasing the test error. The customer would certainly associate this with the quality of the training data and bring about changes. However, data poisoning can also be used as preparation for an evasion attack (see below). The classic case of an attacker gaining unauthorized access to a computer infrastructure must always be considered but offers little insight that would be specific to particular methods or (as here) attack patterns.

In the case of evasion attacks, the attacker tries to create a test example that misleads the model. So, to stay with the example from above, we are in the situation where we have explained to the child the differences of birds and fish and then someone arrives with a penguin or a flying fish and asks "what is this?". In fact, in real-life applications, the environment plays a role. The examples of stickers on stop signs that made the model "think" it was a right-of-way sign have become well known (see, e.g., (Eykholt, et al., 2018). Here, the perturbed image is first determined experimentally and then printed out and placed in the environment. Combinations of data poisoning and evasion attacks are conceivable, in which the attacker influences the learning process so that the model reacts to certain adversarial cues.

Privacy attacks are attempts to “reverse engineer” the data from the model, like for example in so called membership inference attacks. Truex et al. (2019) describe an example of a cancer treatment centre that provides an AI-based online information about the likelihood of cancer given a set of symptoms. The attacker uses data of a particular person X to find out whether X is a patient of the centre (i.e., was part of the training set). Membership inference violates the privacy of both the individual participants involved in the model training (privacy of individuals) and the owner of the training dataset (leakages of business value or trade secrets). It is also possible to reconstruct datasets. Fredrikson et al. (2015) reconstructed, for example, approximate face images of a person X from a face recognition system given only a black-box access and X’s name.

Model stealing or model extraction has similarities with privacy attacks in that it is a reverse engineering process. The difference is that the attacker does not want to reconstruct the training data but the model itself. The underlying motivation could be, on the one hand, to simply steal the model or, on the other hand, to use the model for constructing so-called white-box evasion attacks (i.e., attacks with knowledge of the model specifics).

2.2 Patterns

In the history of software engineering, it appeared that architectural design choices were often repeated. With increasing expertise in designing software systems, such common approaches and patterns emerged. The documentation of design patterns (Gamma, et al., 1995) has several benefits. Firstly, the experience of lessons learned from successful approaches can be documented in a standardised way. Secondly, documented patterns can be used for communication among software developers and architects to better describe their choices. Thirdly, reuse is enabled on the level of patterns as architectural building blocks, rather than at the level of code snippets in specific programming languages or frameworks. As a result, software architecture designs become more understandable and reliable and, hence, raise the level of abstraction of software engineering – a long-standing trend in all kinds of engineering.

In the case of artificial intelligence, like any field of software engineering, many experiments are being performed with the aim to find the best approaches in certain contexts. Although the field is very dynamic, the documentation of design patterns supports the identification of similar approaches by classifying them and to improve the communication among researchers and developers. With this goal in mind, it is useful to document several patterns using a pattern language (van Bekkum, et al., 2021). The taxonomy and visual components of that language are explained below.

Design patterns. For describing and communicating design patterns it is necessary to define a common language. A taxonomy allows for a simple definition of such a language as a hierarchical list of terms. In the context of software engineering for AI systems, the following four main terms can be defined:

- *Instance*
  An instance is the basic input and output of a software system, which is either data or symbols. The difference is that data is simply a thing (such as a number or a text) without interpretation, while a symbol refers to a well-defined concept and can be composed, according to rules, to form other symbols.
- *Model*
  A model is a description of instances and their relationships. A statistical model defines relations among data, such as (un-)certainty in probabilistic models (Bayesian or neural networks, etc.). A semantic model represents the meaning of symbols by specifying their concepts, attributes, and relationships (ontologies, knowledge graphs, differential equations, etc.).
- *Process*
  A process defines the steps to perform operations on instances and models. It produces other instances or models.
- *Actor*
  An actor is an autonomous entity with intentions and goals that initiates processes. It can be a human operator or user, but also a software agent or robot. Multiple actors can cooperate according to various interaction patterns and using human-machine interaction methods.

For the visual language of the diagrams for the design patterns the following graphical representation can be used. In the above-mentioned pattern language, instances are represented with rectangles, models with hexagons, processes with ovals and actors with triangles.

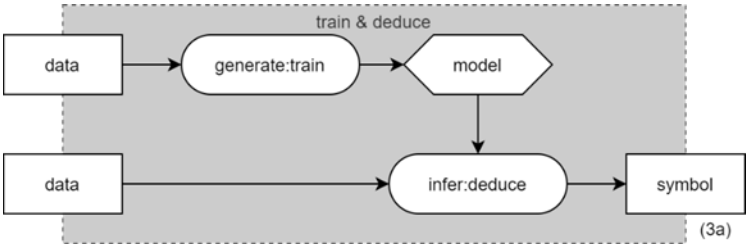


Figure 2.1: Standard machine learning pattern

Consider a simple example of the standard machine learning pattern in Figure 2.1. Here, a training data set is used to train a model and a test data set is used to deduce symbolic predictions using that model (e.g., labels of objects in a picture). This is visualized, for example, in Figure 2.1, where the process "generate:train" performs the training step and the process "infer:deduce" performs the deduction step in the operational phase. The notation indicates that *generate* and *infer* are top-level concepts of the corresponding taxonomy and, after the ":", *train* and *deduce* are the most concrete subconcepts of those higher-level concepts in this specific pattern.

One of the means to categorise potential attacks are **attack patterns**. They are used for some of the methods below, where applicable, and, in analogy to the design patterns above, they are supposed to generalise specific attacks and provide details about how attacks are performed. For general IT systems they have been defined, for instance, by the US Cybersecurity & Infrastructure Security Agency (CISA) (Barnum, 2013). In this study an emphasis is put on patterns that describe attacks on IT systems involving (symbolic) AI mechanisms rather than on general IT systems.

An attack pattern is an abstract description of how types of attacks can be executed. It also provides recommended methods of mitigating the attack. CISA proposes that an attack pattern should typically include the following information (Barnum, 2013):

- **Pattern Name and Classification**: The label given to the pattern which is commonly used to refer to the pattern in question.
- **Attack Prerequisites**: What conditions must exist or what functionality and what characteristics must the target software have for this attack to succeed?
- **Description**: A description of the attack, possibly including the chain of actions taken.
- **Method of Attack**: How is the attack carried out (e.g., data poisoning, manipulation of models)?
- **Attack Motivation-Consequences**: What is the attacker trying to achieve by using this attack?
- **Attacker Skill or Knowledge Required**: What level of skill or specific knowledge must the attacker have to execute such an attack?
- **Resources Required**: What resources (e.g., IP addresses, tools, time) are required to execute the attack?
- **Solutions and Mitigations**: What actions or approaches are recommended to mitigate this attack?

3 Symbolic AI Method Classes

3.1 Preliminaries

In the following, we consider each symbolic AI method as one implemented system module denoted as *S-module* with symbolically encoded semantic model (Model:sem), input (Symbol:In), and output (Symbol:Out). Hybrid neuro-symbolic or neuro-explicit AI methods are implemented in corresponding hybrid AI systems each with one S-module for the integrated symbolic AI method.

The *S-module model* represents knowledge used by the S-module for reasoning, planning, or learning. For example, in AI planning methods, the Model:sem includes the planning domain model, problem model and internal model for processing such as belief and policy trees. The *S-module input* is symbolic input (Symbol:In) of this method. Depending on the method type, symbolic input includes, for example, semantic labels, relations, conceptual queries, domain models, and problem models, but also numeric observation data that is semantically interpreted (as an intermediate abstraction) by the S-module for its internal processing. That implies that, if required, the appropriate transformation (semantic interpretation) of some non-symbolic encoded data received by the S-module to symbolic encoded Symbol:In happens at the S-module interface post-processing side; we do not explicitly indicate this transformation process within the design pattern of the S-module. The *S-module output* is symbolically represented output (Symbol:Out), which, as with the symbolic input, can vary depending on the type of the symbolic AI method from semantic labels to complex action plans and traces.

3.2 Reasoning

3.2.1 Reasoning in Formal Ontologies

3.2.1.1 Description and Methods

In AI, in the 1980-90s, fragments of first-level predicate logic were investigated for decidability and the existence of algorithms that can check as efficiently as possible whether a formula can be fulfilled, in order to be able to efficiently calculate relationships between conceptual descriptions in ontologies and taxonomies. For the families of sublanguages studied, the names terminological logics, concept languages, and finally description logics emerged. For overviews of research and language development in AI, we refer the reader to Brachman et al. (2004), Baader et al. (2007), and Baader et al. (2017). In the early 2000s, the idea of intelligent agents leveraging ontologies with formal model-theoretic semantics to accomplish their goals with symbolic AI methods of reasoning, learning, or planning underpinned the initial vision of the Semantic Web (Berners-Lee, et al., 2001). In particular, the web consortium issued standards for such a web ontology language, called OWL (W3C Recommendation, 2004), (W3C Recommendation, 2012). The OWL standard has become widely accepted for the representation of ontologies (Paulheim, 2016) and is also supported by tools, e.g., Protegé (Protegé Ontology Editor, 2021).

Today, there exist many ontologies in different ontology languages with different degrees of expressivity in different application domains ranging from the prominent biomedical ontology SNOWMED in OWL (SNOMED CT, 2022), to ontologies for cultural heritage applications (Vlachos, et al., 2022), recommender systems (Rahayu, et al., 2022), industrial production and machine maintenance (Mazzola, et al., 2016) and oncology (Silva, et al., 2022).[1] In addition, knowledge graphs such as for DBpedia (DBPedia, 2021) focus on representing factual (assertional) knowledge in (subject-predicate-object)-triple form with limited expressivity of RDF/S (Resource Description Framework/Schema) often extended with an argument to capture the source or time of a fact entry (Singhal, 2012). Knowledge graphs are also associated with linked open data projects in the linked data cloud of the web of data with a wide range of applications (Ji, et al.,

[1] https://www.w3.org/wiki/Lists_of_ontologies

2021), and investigated for embedding-based reasoning to deal with uncertainty and to predict plausible knowledge (Zhang, et al., 2022).

There are various standard operations on ontologies. For example, several different ontologies can be merged into one. It should be noted that inconsistencies can be introduced here. Or a questionnaire to be answered uses only a small subset of the concepts and roles used in the ontology. In this case, the operation of "forgetting" can limit the given ontology to those concepts and roles that are relevant to the current questions. No inconsistencies can be "smuggled in" through this operation. OWL 2 is based on description logics (DLs) and provides a set of expressive language operators for defining concept hierarchies, concepts, relations, and more. The formal semantics allows logical deductive reasoning, such as consistency checking, materialization of the fact base in compliance with the terminological schema of the ontology, query answering, and inductive and abductive reasoning.

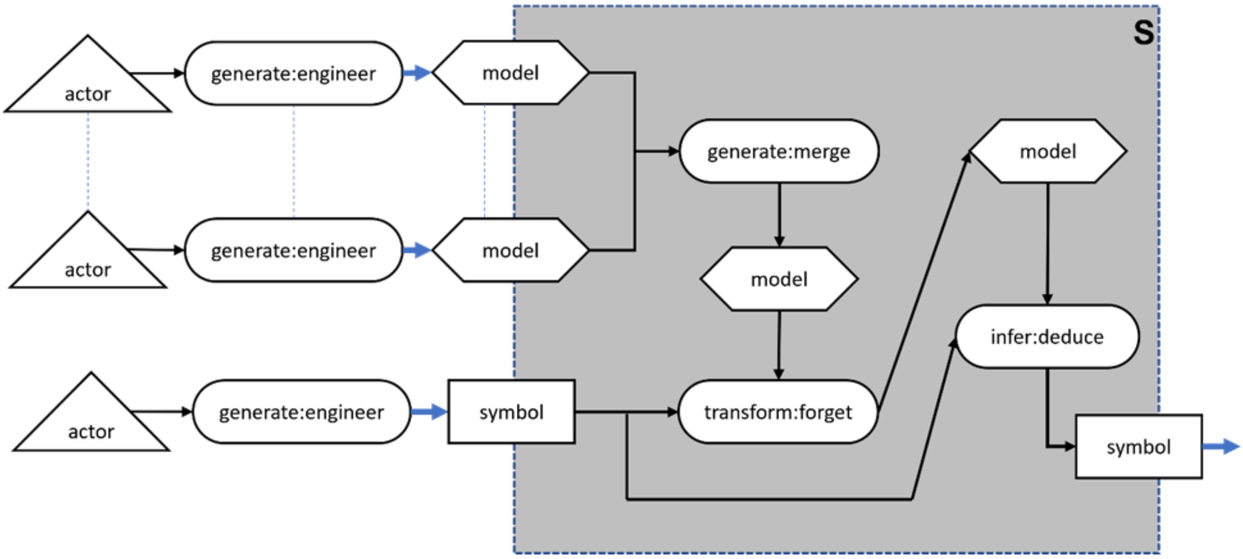


Figure 3.1: Design pattern for a typical operation or deduction scenario of methods of reasoning in formal ontologies

Figure 3.1 shows one typical scenario of ontology reasoning: Two or more ontologies are merged into one, which together with the collection of questions (lower left symbol in Figure 3.1) gets transformed by "forgetting" into a smaller ontology that finally is used for deduction. Note that, in many practical applications several ontologies are generated independently, often even from different sources. To use all of them for answering a collection of questions they first must be merged into a single ontology. The questionnaire might have a restricted set of concepts and roles (sub-language). Therefore, and to avoid redundancy during the deduction process, the merged ontology gets reduced by a "forgetting" transformation. This operation results in an ontology that only knows about the concepts and roles from the questionnaire but nevertheless reveals the same validities (of the sub-language) as the original ontology. For more information on ontologies and reasoning on them, we refer to Staab et al. (2010).

3.2.1.2 Applications and Security

In practical applications, ontologies with formal semantics include an application-dependent terminology or schema in terms of concepts and relations between them, as well as a fact base that is consistent with this terminology. As mentioned above, formal ontologies allow to logically reason on and query-answering over such knowledge representations. The results of reasoning based on symbolically represented logics are in symbolic form and considered explainable, which might help to logically detect and explain attacks to users even before unintended ontology updates are realized and used in the application system. However, any malevolent manipulation of a formal ontology jeopardises the usefulness of the result of dependent logical reasoning and query-answering processes in practice, and as one consequence, may directly or indirectly cause potentially harmful decision-making by human users or machines dependent on ontology-based reasoning processes. An attacker may also utilize the API of the ontology-based reasoning module to gain access to the semantic model, the ontology, the questionnaire, or the reasoning result as output, which, in the end, can only be avoided through effective means of authorized access.

Another type of attack focuses on “poisoning” the reasoning process by offering a manipulated ontology as required import that either contains an inconsistency by itself or reveals an inconsistency together with certain other ontologies or introduce unintended schema and fact knowledge that corrupt the reasoning result. These attacks can either be purely destructive, or an attacker could create false inferences in a goal-directed manner. The consequences are that either any question against the formal ontology will be answered positively because everything is true in an inconsistent setting, or the original purpose of the using AI system is permanently disrupted. If the discrepancy only becomes apparent after the merging and “forgetting”, it can hardly be corrected afterwards. The import of ontologies from trusted sources could avoid this problem together with tools for automated and dynamic consistency checking and validation of ontologies. But these tools are rare and not integrated into practical applications at large. On the other hand, the attacker must have enough knowledge and skills to write an inconsistent ontology that cannot be easily recognised as such. In the case where the offered ontology is not inconsistent by itself but the merging with the other ontologies is, the attacker must have detailed knowledge about the merging process as well as the content of the other ontologies.

3.2.2 Case-Based Reasoning

3.2.2.1 Description and Methods

The basic idea behind case-based reasoning (CBR) is that any form of problem solving is based on experience from previous, similar cases whose solutions are already known. The generic design pattern for this type of symbolic AI method is given in Figure 3.2.

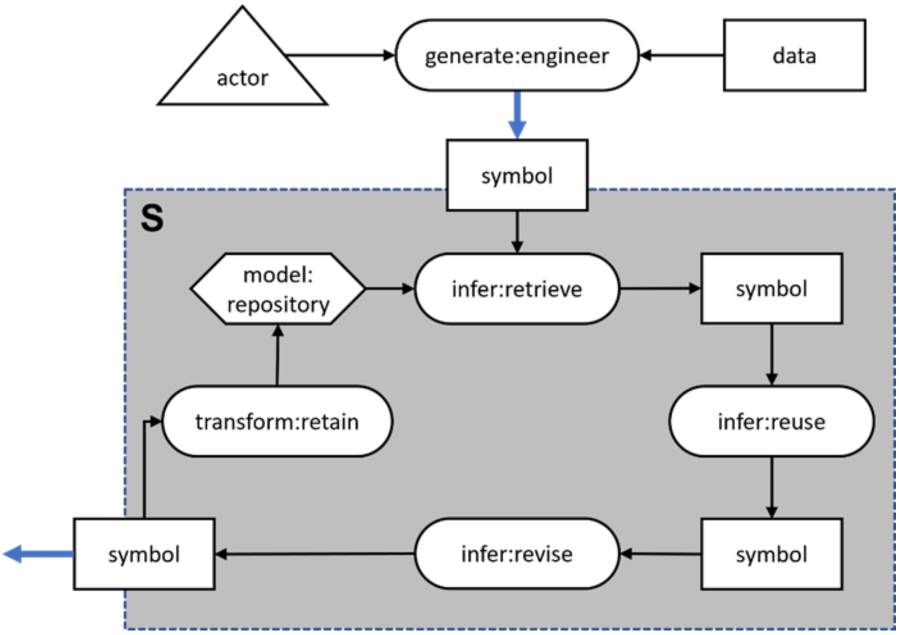


Figure 3.2: Generic design pattern of the case-based reasoning cycle

In the typical four-step process of case-based reasoning in terms of design patterns it is assumed that there is a target problem description (data in the top box of Figure 3.2) from which a syntactically appropriate (semi-)formal version is generated (top symbol in Figure 3.2). The model repository contains a collection of formerly described and solved problems together with annotations that describe how the respective solution was derived. The main four processes of CBR are as follows:

1. **Retrieve:** Given a target problem, retrieve cases that seem relevant to solving it from the model repository.
2. **Reuse:** Choose a most relevant solution and map it to the target problem. This may involve some adaptation to fit the new situation.
3. **Revise:** Test the new adapted solution and, if necessary, revise.
4. **Retain:** After the solution has been successfully adapted to the target problem, store result as a new case in the model repository.

As always when arguing with analogies, care must be taken to ensure that the proposed solutions are adequate for the problem at hand. For instance, it has to be checked whether the adaptation did not alter the preconditions on which the historical solution was based (obsolescence of knowledge).

3.2.2.2 Applications and Security

A typical example for the use of case-based reasoning is provided in Watson (1997). Here a bank system is described that minimises the risk of losing a loan to clients that are actually not able to pay the loan in an appropriate manner. In such a case, stealing the model from the model repository is highly critical, for it contains a huge amount of personal data or at least information from which personal data may be easily inferred (privacy attack).

Another application can be found in diagnosis systems for car faults. De et al. (2018) for instance propose a diagnosis system that builds upon case-based reasoning to retrieve possibly similar cases stored in a case repository. A manipulation of the cases in the model repository could lead to repair instructions that are not appropriate to the actual problem or even cause additional damage to the car.

In any case, business competitors might be interested in knowing about the model repository of others to gain some competitive advantage. Building up such a repository by themselves is time-consuming and expensive, and stealing of an output solution is also useful in case they have similar problems to solve. The target problem of case-based reasoning might be of minor importance here. However, knowing about the strength and weaknesses of a competitor's problem solver might be advantageous as well.

In some kind of "data poisoning" attack on the model repository, the attacker may try to destructively spoil the original problem solver to come up with bad, useless solutions or to enforce an unintended change of the output to the advantage of the attacker. These attacks are hard to avoid with more than securing the system API against unauthorized access with fine-grained authorization of model parts access, or some well-defined logging mechanism to at least reveal the source of attack. The same holds for manipulation attacks on the output solution, and the attack targeting the model repository or the target problem.

3.2.3 Physical Reasoning

3.2.3.1 Description and Methods

Physical reasoning systems refer to technical systems and their modular components whose behaviour is specified in terms of physical laws, in particular differential equations. Typical examples can be found in airplane dynamics or industrial plants. A common formal representation of such systems can be found in so-called "*Hybrid Automata*".[2] The term "Hybrid" stems from the fact that such automata combine discrete as well as continuous behaviour in a single description. Each modular component is then specified as a finite graph whose nodes serve as different modes of operation, the edges between nodes as (instantaneous) transitions from one mode to another, and the nodes themselves provide the dynamic behaviour (physical laws, differential equations) that take place within the corresponding mode of operation. According to our design pattern approach, the generation of such a formal description takes place in the development phase of the reasoning system's life cycle and may be represented as shown in Figure 3.3.

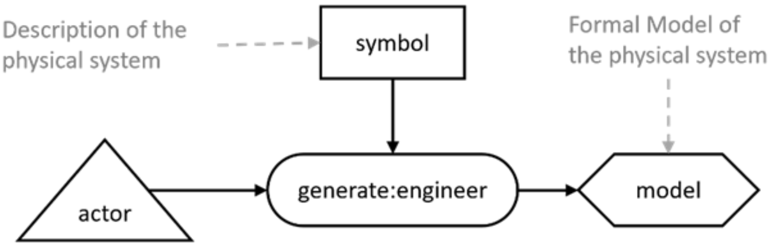

Description of the physical system
symbol
Formal Model of the physical system
actor
generate:engineer
model

Figure 3.3: Design pattern of physical reasoning methods: Formal description generation (development phase)

Here an actor transfers the description of the components of the physical system into (communicating) hybrid automata. Such a formal model may be interesting by itself. However, in addition one usually considers possible properties one is interested in for various reasons. Such properties are often given as informal or semi-formal descriptions that have to be transformed (typically) into temporal logic formulas. As for our design patterns this action is very similar to the figure above with the input symbol acting as the informal property description and the output model represents the resulting temporal logic formula.

Once both the formal model and the system property have been defined the next steps depend on the purpose of the formalization. We distinguish here three possible kinds of physical reasoning systems: *Verification, Monitoring, and Controller Synthesis.* In the following, we describe each of them and briefly discuss their security aspects thereafter.

[2] Not to be confused with "Hybrid AI Systems" or, special cases of the former, "Neuro-symbolic AI systems" within which symbolic and sub-symbolic methods are combined.

Verification. In this case one is interested in a formal proof that the system under examination (or rather its formal model) satisfies the given property (see Figure 3.4). To this end a *hybrid automaton verifier* comes into play. Hybrid automaton model (top middle) and the property (the model in the lower right) are fed into the verifier whose operation results (provided the verification process terminates) in the output *true* (in case the property holds) or *false* (otherwise). In case of true liveness or false safety properties, often a derivation trace (witness) is produced that may help to analyse the system's behaviour.

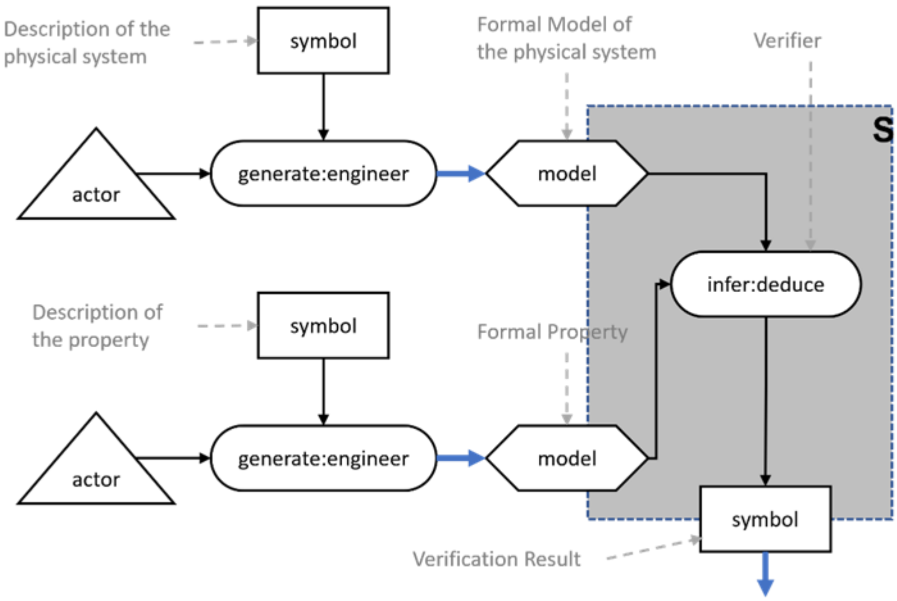


Figure 3.4: Generic design pattern for physical reasoning methods: Verification

Monitoring. Monitoring differs in several aspects from verification. In a naïve setting it could be handled similarly. This, however, would mean to frequently start verification processes and this is something that will in general fail due to verification effort. More promising is an attempt dual to verification. Whereas in verification one is faced with a single initial state and several properties to be verified during monitoring one is interested in a single monitoring property but several changing initial states. Therefore, the engineer generates an alarm model from the technical system`s formal model and the property to be monitored that reflects the result of a backward reachability analysis up to a certain time threshold, say 20 seconds. This alarm model will reveal a potential danger whenever the current state of the physical system may lead to an undesired behaviour.

Controller synthesis. This application of physical reasoning aims at synthesizing the behaviour of a controller that may influence certain input parameters of a given physical system to reach a pre-defined goal (see Figure 3.5). The general idea behind this approach is to formally model the physical system to be controlled but also to formally model an (abstract, omnipotent) system that may arbitrarily influence the control points of the first. Assume that the desired behaviour of the combined system is described as a temporal logic liveness formula. Such liveness formulas (in the sense of "eventually it will be the case that …" or "it is always inevitable that …") have the advantage that a proof will be accompanied with a trace witness that may be interpreted as a sequence diagram that specifies the correct instantiations of the abstract controller automaton such that the model of the controlled system together with the model of the instantiated control automaton satisfies the specified desired behaviour.

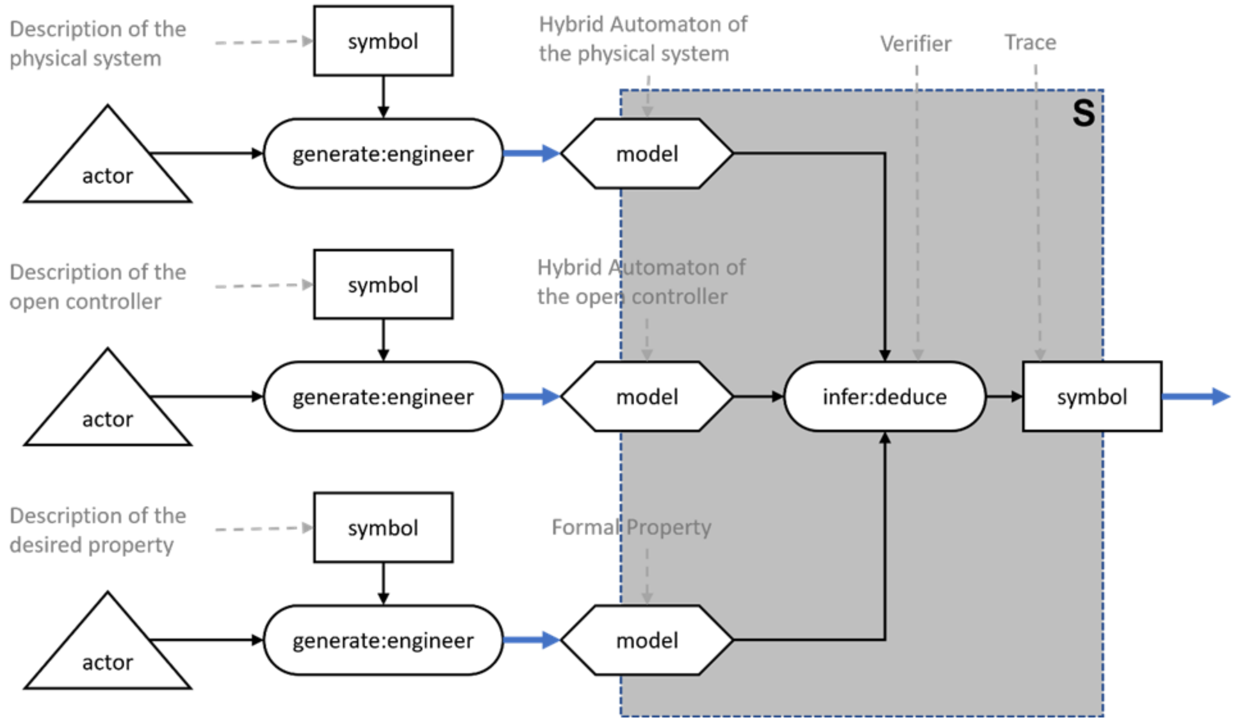

Description of the physical system
symbol
Hybrid Automaton of the physical system
Verifier
Trace
S
actor
generate:engineer
model
Description of the open controller
symbol
Hybrid Automaton of the open controller
actor
generate:engineer
model
infer:deduce
symbol
Description of the desired property
symbol
Formal Property
actor
generate:engineer
model

Figure 3.5: Generic design pattern for physical reasoning methods: Controller synthesis

3.2.3.2 Applications and Security

In verification, the attacker may attempt to manipulate the output, that is, to turn false safety properties to true and true liveness properties to false in the verifier output, either in all or only in selected cases. Another attack option is the manipulation of the model which is a formal description of the technical system under investigation. The model is usually generated during the development phase rather than the operational phase of the verification life cycle, and stored in a repository, a database, or in a cloud. The attacker then tries to gain access to this model and modifies it at will. That is, any complete or partial modification of the model after its generation yields an inadequacy of the formal description such that whatever property will be proved or disproved does not reflect the original system. In monitoring, the attacks on verification apply here as well. In addition, the alarm model of the monitoring process must not only be protected against theft (confidentiality) but also manipulation (integrity).

In all processes of physical reasoning, the stealing of models requires reading access to them via the system API and is harmful for the model owners. The development and production of technical systems is often elaborate, complex, and expensive. A manufacturer therefore has a great interest in protecting this intellectual property from competitors. An attacker may be interested in gaining access to this knowledge to save development costs. As a competitor, he could use it to offer similar products at lower prices and thus cause great damage to the original developers. In addition, any information about desired properties that are not guaranteed by the model (failed proofs) can be utilized to harm the developer's reputation.

One application of physical reasoning is the automated discovery of safety violations in air traffic control. In Pasaoglu et al. (2016) the authors present a hybrid system description of modelling the decision process of the air traffic controllers in en-route and approach operations. In their model they describe the dynamics of an airplane (Boeing 737-800) and a collision avoidance controller implemented in a corresponding flight deck simulator. Their system is scalable to scenarios with up to 100 aircrafts. This control system can safely be considered as a monitoring automaton in the sense above. It does not only check for critical situations but also considers the potential effects of its proposed actions. Its conflict resolution capabilities have been validated on real-time data which included over 7000 flights in a 96-hour period. Not all attacks mentioned in this section apply to this application. For example, manipulation of the output during operation (flight) can be safely neglected. More important in this use case are model theft and model manipulation during the development phase. The former because suitable flight models are difficult to obtain and are both time-consuming and costly. Competitors in the air traffic control manufacturing market save high costs of their own if they can use the knowledge of others for their own benefit. More dangerous and even life-threatening is the manipulation of one of the involved models. Attackers need access to the models and a good knowledge of the necessary mathematical background and the specification language that had been used.

3.2.4 Verification in Process Algebra

3.2.4.1 Description

Process algebras, such as the pi calculus (Milner, 1999) or CSP (Communicating Sequential Processes) (Hoare, 2004), are a means of describing concurrent, dynamic, and mutually communicating processes. For this purpose, atomic events (i.e., events that cannot be further split) are linked with suitable linking rules in such a way that the sequential execution, concurrency or input and output are reflected syntactically. The design pattern for verification methods in process algebras is shown in Figure 3.6.

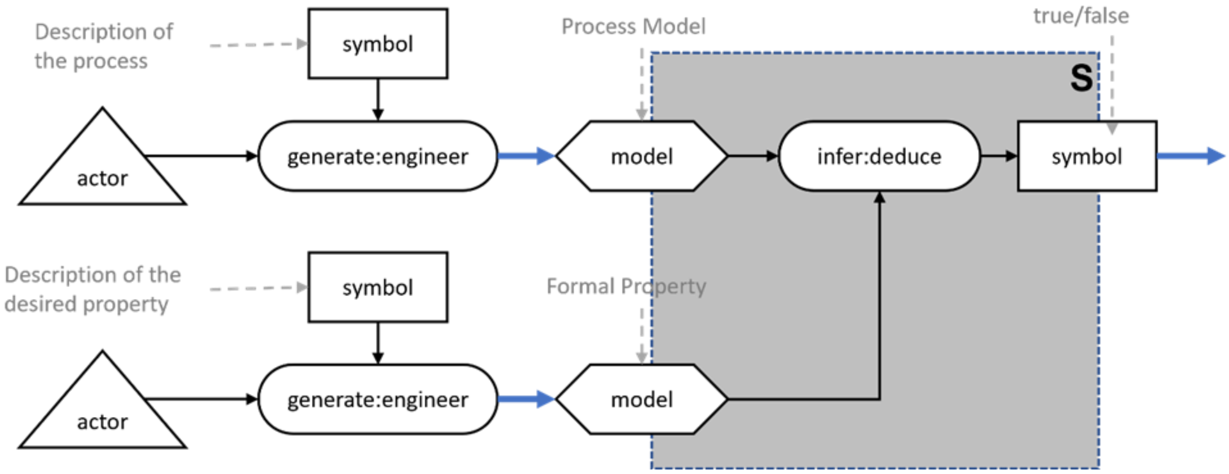


Figure 3.6: Generic design pattern for verification in process algebras

The semantics of such processes (process models) is defined with the help of state transition systems. The proof of concrete properties of processes described in this way is generally carried out in two different possible ways. In the so-called bisimulation, the property to be verified is itself described as a separate process and an attempt is made to show that two equivalent states, one belonging to the process, the other to the property to be proven, lead to equivalent states again via state transitions and thus the two processes cannot be distinguished by outside observers. Another possibility for proving properties results from the trace semantics of process descriptions. This involves a complete set of state sequences that result from the possible sequences of the process. The actual proof then takes place via induction over these state sequences.

3.2.4.2 Applications and Security

The similarity between Figure 3.4 and Figure 3.6 is no coincidence. On the abstract level of the design patterns, they can hardly be distinguished, hence similar attacks apply here. One application of the verification methods for process algebras concerns the description and verification of communication protocols. In Abadi & Gordon (1997), the authors present the spi-calculus, an extension of the pi-calculus to formally describe and analyse (cryptographic) communication protocols. It has been used in a variety of communication protocols as for example the Needham-Schroeder Protocol. The authors emphasize on two main analysis aspects, namely authenticity and secrecy. Authenticity ensures that the recipient of a message that presumably comes from a certain sender must actually originate from that sender. It is therefore not possible for an attacker to send a message with a forged sender. Secrecy on the other hand guarantees that a message cannot be read in transit from one principle (communication participant) to another. To formally prove such properties, it is first necessary to generate from the rather informal description a spi-calculus version of the protocol. Second, the property to be analysed must be specified. In general, this will also be a spi-calculus process description. Finally, a spi-calculus verifier proves the equivalence of the two descriptions. Of course, if the protocol at hand has a flaw with respect to authenticity or secrecy, then at least one of the two proof attempts will fail. The whole process follows nicely the design patterns shown in Figure 3.6.

3.2.5 Reasoning with SAT-Solvers

3.2.5.1 Description

The problem of satisfiability of a Boolean logic formula (SAT) is one of the fundamental problems of computer science and was also the first problem to be proved NP-complete (Cook, 1971). With the development of the DPLL algorithm, a correct and complete calculus was available with which the satisfiability of a propositional logic formula could be checked fully automatically (Davis, et al., 1960), (Davis, et al., 1962). This laid the foundation for so-called SAT solvers. Local search algorithms allow SAT solvers to scale to very large instances (Selman, et al., 1992). An overview of the algorithmic techniques of SAT solvers can be found in Biere et al. (2009) and Gomes et al. (2008). In the following, we restrict our security analysis of SAT-solving to its application for product configuration.

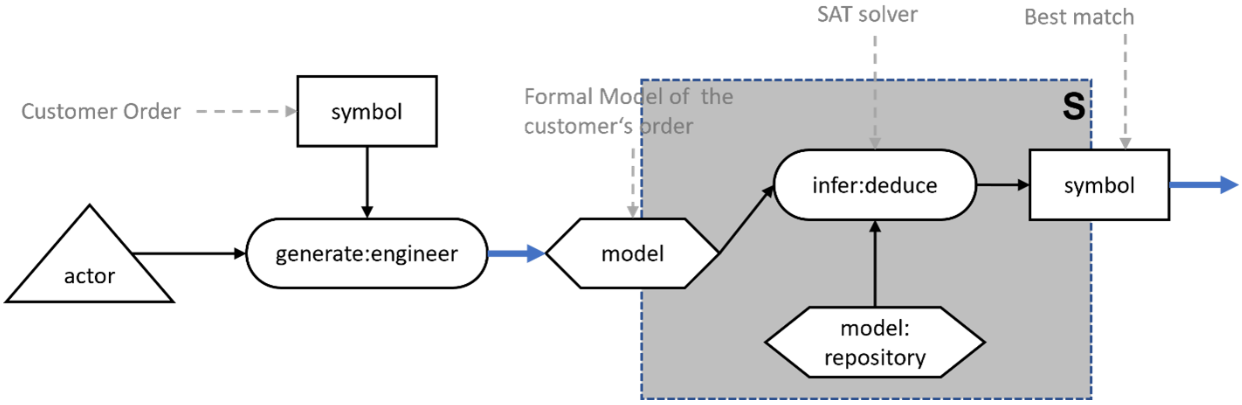


Figure 3.7: Generic design pattern for SAT-solving for product configuration

Product configuration is a very common problem that can be formulated and solved with the help of SAT-Solvers (cf. Figure 3.7). Assume a collection of items that might become part of a certain product. Each of these parts is given a name (propositional variable). Not all of these items can be combined at will. Some require the absence of other items, others the combination with certain fixed configurations. Such restrictions have their natural representation in propositional logic. The goal of SAT-solving then is to find out whether a customer's desires can be matched with these restrictions and, if not, to find a "best" match.

3.2.5.2 Applications and Security

The application areas of SAT solving are widespread. According to Stackexchange[3], practical applications of SAT-solving include not only product configuration as described and analysed above but range from Product Configuration over Hardware Configuration, Bounded Model Checking, (Hardware) Equivalence Checking, Asynchronous circuit synthesis, Software-Verification, Expert system verification, Planning (air traffic control, telegraph routing), Scheduling (sport tournaments), up to Crypto-analysis. Attacks on specific customer orders, customer models and the model repository of the reasoning system require attackers to utilize the system API. A sophisticated change to the order can result in the car manufacturer not being able to make an offer to the customer. Another motivation could be the loss of the customer's trust in the car manufacturer by rigidly shortening or extending the order. However, as with previous reasoning methods, the system API can be secured against illegal access with authentication means.

3.2.6 Reasoning with Constraint Programming

3.2.6.1 Description

In constraint programming, application problems are described by a set of variables that are assigned a possible range of values and constraints that the assignment of values to the variables must satisfy (Apt, 2003), (Tsang, 2014), (Dechter, et al., 2003). A constraint solver (CLP-Solver in Figure 3.8) searches for a value assignment of all variables that satisfies all constraints. In constraint optimisation, an objective function is also defined, and a satisfying assignment is sought that minimizes or maximizes the objective function. In recent years, the development of constraint modelling languages has led to expressive languages and scalable algorithms for a wide variety of language constructs, so-called global constraints (Rossi, et al., 2006). The development of the constraint language MiniZinc (Nethercote, et al., 2007), which is supported by many constraint solvers, is an attempt to standardise language development and provide a unified modelling language that is both practical and reflects the state of research (Wallace, 2020).

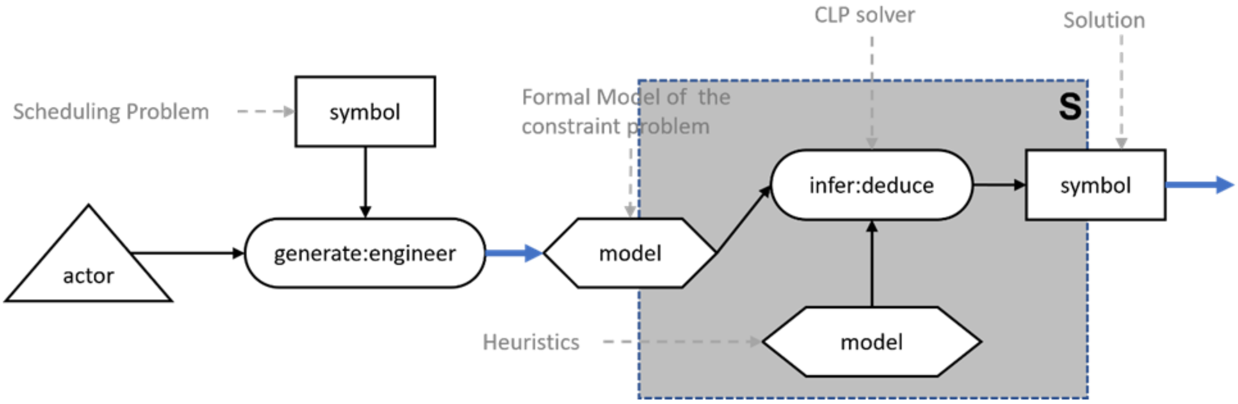


Figure 3.8: Generic design pattern for methods of scheduling with constraint programming

In the scientific field, constraint-based programming is used, for example, in natural language processing, program analysis and molecular biology. In industrial practice, typical applications are optimisation problems and scheduling tasks, circuit design and verification.

[3] https://ai.stackexchange.com/questions/67/what-are-the-real-world-uses-for-sat-solvers

3.2.6.2 Application and Security

ENSTA Paris[4] provides various application examples for Constraint-logic Programming as, for instance timetabling and car sequencing. Typically, timetabling is used to produce school timetables that respect room specifications, teacher skills, as well as additional requirements that come up for educational reasons. Car sequencing is concerned with solving operational research problems. It is supposed to find optimal solutions to sequencing process steps that are constrained according to certain demands, capacities, further constraints, and data restrictions.

In both applications cases, the theft or manipulation of the solver heuristics can be particularly attractive to business competitors, apart from targeting the constraint model, specific problems or the output solution directly. In industrial applications, solutions to scheduling problems generally cannot be achieved by using a CLP solver alone. The search for variable assignments that satisfy the given constraints usually must be supported by suitable heuristics that support the search for possible values in a goal-directed manner. A typical attack intends to lead such a search down the wrong path and thus make it difficult or even impossible to find suitable solutions. Manipulation of solver heuristics will cause either a non-optimal solution or even prevent finding a solution at all. Like with attacks on previous reasoning methods, an attacker could utilize the CLP-solver system API with appropriate, available CRUD (create, read, update, delete) operations to manipulate the CLP-solver heuristics, against which strong authentication mechanisms can help.

[4] https://perso.ensta-paris.fr/~diam/ro/online/cplex/cplex1271/OPL_Studio/opllanguser/topics/opl_languser_app_areas_CP.html

3.2.7 Other Logical Reasoning Methods

3.2.7.1 Description

Modal logics and agents. Prominent types of non-classical logics for reasoning are modal logics which extend classical propositional or predicate logic by additional logical operators. The interpretation of these additional operators and the associated calculus offers a wide range of possible specifications. Typical examples are:

- *Temporal logics* (e.g., "from now on ...", "sometime in the future it will hold that ...").
- *Epistemic logic* (e.g., "person X knows that ...", "person Y considers it possible that ...").
- *Deontic logic* (e.g., "it is forbidden that ...", "it is required that ...").

Typical AI applications of such (non-classical) logical reasoning can be found in multi-agent systems but also in the formal representation of narratives and dialogues. In its simplest form, a logical reasoning agent (see Figure 3.9) perceives its environment through its sensory system and adapts its worldview to these perceptions. A rule-based approach is then typically used to determine one or more actions to be performed next. These are then executed via suitable actuators. The reasoning process for adapting its own world view depends on the properties of the modalities used. Depending on the axiomatisation, very different conclusions can be drawn. For example, by adding a single axiom (axiom of linearity), the meaning of "possibly in the future ..." can be changed to "it will inevitably happen that ...". The axiomatisation of epistemic operators can be supplemented by positive introspection ("the agent knows what it knows") or also negative introspection ("the agent knows what it does not know").[5] With these additional axioms, one can draw additional conclusions that change the world view in a different way than without these axioms.

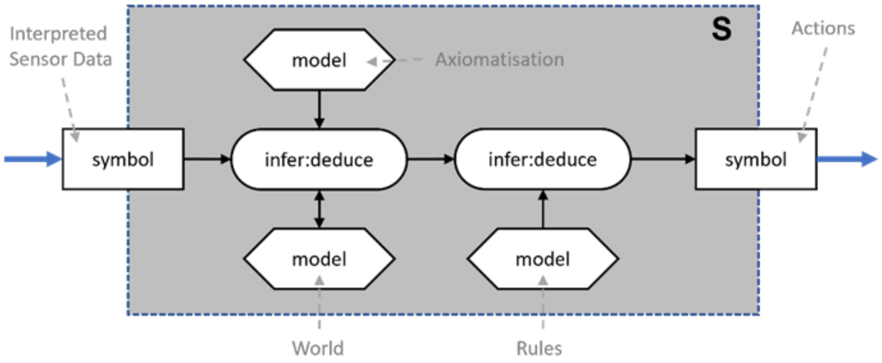


Figure 3.9: Generic design pattern of basic logical reasoning agent

[5] Note that logical reasoning is performed either under the assumption of complete information, that is a closed world (what is not known to be true must be false) or the opposite, that is an open world (what is not known to be true is unknown).

Non-monotonic reasoning. Among all examined approaches to deal with non-monotonicity in logical reasoning, default reasoning (see Figure 3.10) has gained quite some maturity in the meantime. Default assumptions may be added to current knowledge as long as newly acquired knowledge does not contradict them. For example, Prakken (2013), suppose there is a rule that states that 'an offer and an acceptance create a binding contract'. A default logic interpretation would "silently" add 'unless the offeree is exceptional'. Somebody might be declared "exceptional" if s/he is not legally sane or if s/he is a child. The added default assumption is that, whenever an offeree cannot be proved "exceptional", it can safely be assumed that s/he is not and therefore the rule usually applies in its original form. Adding the fact 'the offeree is a child' then would create an exception, though, and therefore the previous conclusion cannot be drawn anymore.

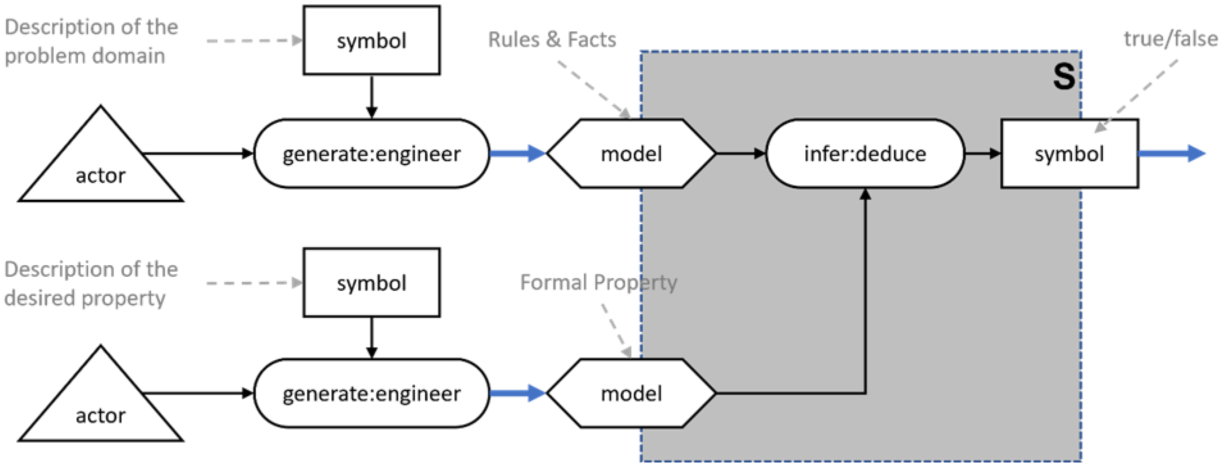


Figure 3.10: Generic design pattern for default reasoning

Additional knowledge can lead to the fact that earlier inferences lose their validity. Thus, default reasoning is a form of non-monotonic reasoning. Inference procedures use either a calculus that explicitly describes the different possibilities of inference, or a translation into other, more well understood, logics such as modal logic or temporal logic. In this case, however, the explicitness of single inference steps is often lost, which makes the comprehensibility and the explicability of derivations more difficult.

3.2.7.2 Security Aspects

In the above design pattern (Figure 3.10) of default reasoning, it is assumed that the "Rules & Facts" model is a "living" object. That is, a legal argumentation may continuously produce new facts (and sometimes even new rules) and, whenever this happens, formerly answered questions can be asked again, which maybe leads to a different answer. In addition, rules and facts may be imported from not trustworthy sources or provided by attackers in a "data poisoning" attack on the default reasoning system. This shows one critical point in default reasoning regarding security, namely the *extension of the fact model by fake facts.* In a legal argumentation, fake facts can easily corrupt the entire reasoning process. The final verdict will differ to the unimpaired case. However, the attacker has to have sufficient knowledge about default reasoning to modify the fact base in a goal-directed manner, which might require a steep learning curve, and access to it, which can be avoided with authentication means.

3.3 Planning

This section is concerned with a brief study of security aspects of selected symbolic AI methods for automated planning of autonomous agents. For this purpose, we differentiate between online and offline planning each with certainty or under uncertainty. Offline planning is decoupled from subsequent execution of produced plans and gets no feedback about it during planning, while dynamic or online planning is interleaved with controlled action execution in a closed-loop manner. From a security perspective, this is a key distinction with respect to potential attacks of the planning system even during runtime. We limit our study to the description of a small set of selected basic methods for automated planning and relevant attack patterns for them. For a more comprehensive introduction to automated planning, we refer the interested reader to, for example, Ghallab et al. (2004), Ghallab et al. (2016).

3.3.1 Offline Action Planning

3.3.1.1 Description and Methods

The abstract design pattern for systems of offline action planning is shown in Figure 3.11. An actor generates a planning *domain model* which formally specifies planning relevant information on the considered application domain and environment. Such domain knowledge representation can take the form of, for example, an appropriate first-order logic-based ontology scheme complemented with a set of available primitive or complex action operators, and, in case of uncertainty, together with some action probability distribution. The domain model serves the same or other actor to define the concrete planning problems to be solved in terms of problem models. Each *problem model* refers to its underlying domain model and includes the compliant representation of an initial concrete situation and goal of planning such as an initial and desired state of a static or stochastic environment seen as a state transition system.

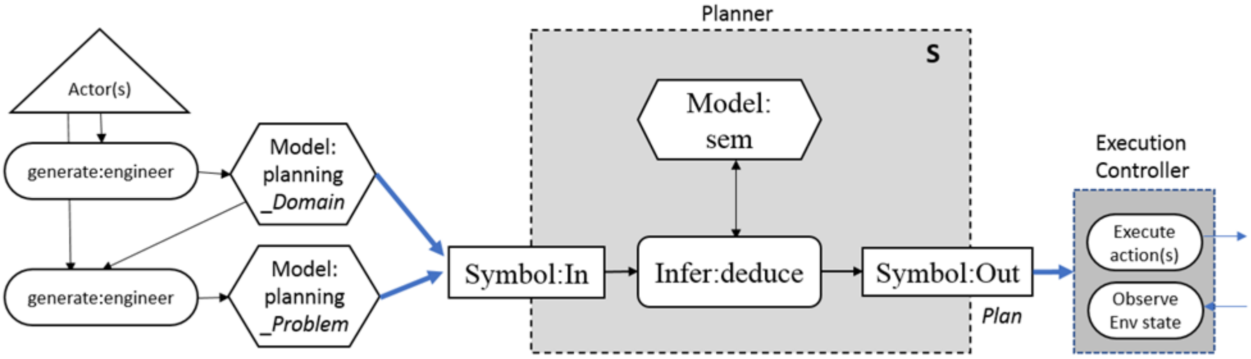


Figure 3.11: Generic design pattern of offline planning systems

Both planning domain and problem models are fed into the planner which, if sound and complete, outputs as a solution, if it exists, a *plan* through which subsequent execution by the controller the given goal can be accomplished in the environment starting from the specified initial situation. The *internal planning model* represents all information the planner internally uses, creates, and updates during its planning process; therefore, it is the complete *semantic model* Model:sem of the planner at any time. This model includes the domain and problem models, as well as instances of additional data structures used for planning such as planning graphs, intermediate states, methods, events, and the plan. After planning ends, the created plan is passed to and gets executed by the plan execution controller in the environment.

Unlike online action planning, the planning process itself is not interleaved with the real action execution. Offline planning may only simulate the execution of available, applicable actions during search for a solution to the given problem. Consequently, any changes of the original domain or problem model during planning may render the produced output of the planner invalid, hence require a full restart of planning.

Planning under uncertainty can be addressed, in general, along the dimensions of non-determinism in terms of probabilistic action outcomes, goal specifications that account for it, and incomplete knowledge or partial observability of the states of an environment. Thus, depending on the chosen technique for offline planning with certainty or under uncertainty, a *plan* can take the form of, for example, a finite sequence of primitive actions, a conditional action plan, or an action policy on belief states with maximal expected utility.

In the following, we briefly describe a few prominent basic techniques for offline (classical and pattern-based) planning with certainty and planning under uncertainty with reference to their generic design pattern in Figure 3.11 restricted to their inputs and outputs.

Methods for (neo-)classical planning such as state-space and planning-graph based planners assume complete knowledge about a deterministic, static, finite domain environment seen as a restricted state-transition system. Domain-compliant, set-theoretic planning problems are specified in terms of explicit representation of an initial state and goal state or set of goal states to reach through executing a plan as sequence of actions or sets of actions. Standard language for representing the planning domain and problem models is PDDL (Planning Domain Definition Language) with current version PDDL3 and a few validation tools for plans or functional equivalence of planning domain models in PDDL such as for example VAL (Howey, et al., 2004), respectively, D-VAL (Shrinah, et al., 2021). We limit our view on basic classical and pattern-based or hierarchical task network (HTN) planning methods.

Classical state-space planners like FastDownward (Helmert, 2006), (Helmert, 2009), FastForward (Hoffmann, et al., 2001) generate an action plan, that is, a totally ordered sequence of actions, based on heuristic search, and intermediate planning states are explicit.

> *Domain model*. A classical planning domain includes (a) an ontology scheme (e.g., predicates, classes) in some first-order logic language covering the set of possible environment system states as its grounding, and (b) a set of deterministic planning operators each with definition of preconditions and effects of their execution on states based on the ontological definitions. Actions are ground instances of these planning operators. Domain models are referenced by unique identifiers or names.
>
> *Problem model*. A planning problem statement or model includes the explicit definition of the initial state and the goal state in terms of sets of facts (propositions) with groundings of predicates and classes defined in the given domain model.
>
> The expressivity of both domain and compliant problem models depends on the chosen PDDL version and planner that can act on them. For example, PDDL3 in contrast to PDDL2.1 allows to specify so-called avoid conditions, that are temporal-logic state constraints which should be avoided to become true in plan execution, hence characterizing dangerous or highly undesirable behaviour.
>
> *Plan*. A classical plan is a sequence of actions, which execution enables the application environment to transition from the given initial state to the goal state. Apart from enabling state reachability analysis for heuristic search it allows to express and solve planning problems by means of constraint satisfaction problem (CSP) solving (Barták, et al., 2010) or planning as satisfiability (SAT) (Rintanen, 2012). The same holds for neoclassical planning such as GraphPlan (Lopez, et al., 2003).

Neoclassical planning-graph based planners such as GraphPlan (Blum, et al., 1997) and XPlan1 (Klusch, et al., 2006) differ from classical planning in that the output is a sequence of sets of actions rather than a single plan, and with a planning graph structure for search. The planning graph allows for a levelled analysis of the reachability of union of sets of propositions in several states with which applicable actions in the search space are reachable from a given initial state. GraphPlan first performs a state reachability analysis by constructing a plan graph, and then performs logic-based goal regression within this graph to find a plan as a sequence of subsets of actions, not a sequence of actions, that is extracted from the final planning graph that satisfies the goal (Klusch, 2008). While the domain and problem model are as in classical planning, the generated solution plan slightly differs from a classical plan as described above.

In contrast to classical declarative state-based planning, the idea of pattern-based action planning is to utilize given actionable domain knowledge in terms of parametrized, structured action patterns that are to be instantiated and decomposed into atomic actions for their execution in the considered environment state. Most prominent type of pattern-based planning is hierarchical task network (HTN) planning. It is used in many applications such as (semantic) web service composition planning (Tang, et al., 2011), (Klusch, et al., 2006), (Nau, et al., 2005).

HTN planning focuses on performing tasks based on an initial state description, a task network as an objective to be achieved, and domain knowledge consisting of networks of primitive and compound tasks (methods). A task network represents a hierarchy of tasks each of which can be executed, if the task is primitive, or decomposed into refined subtasks and satisfies given constraints of task execution such as on precedence, preconditions and postconditions. The planning process starts by decomposing the given initial task network and recursively continues until all compound tasks are decomposed and given constraints are satisfied, that is, a solution is found. The solution is a plan which equates to a totally (or partially ordered) set of primitive tasks applicable to the initial world state (Georgievski, et al., 2015), (Georgievski, 2015).

HTN planning is more expressive than classical planning (Ghallab, et al., 2004). Available HTN planners include for example SHOP3 (Goldman, et al., 2019), SHOP2 (Nau, et al., 2005), SIPE-2 (Wilkins, 1999), O-Plan (Currie, et al., 1991), or extensions of HTN planning like T-HTN for timeline-based HTN planning in multiagent systems (Parimi, 2021) and Pyhop-m that utilizes Monte-Carlo planning tree search (MCTS) to guide the method decompositions by the HTN planner (Shao, et al., 2021).

> *Domain model.* A HTN planning domain model includes a set of operators and a set of methods. Methods are compound tasks defined as hierarchic networks of primitive or other compound tasks. In other words, a method specifies generic, hierarchically structured actionable knowledge such as that the construction of a roof on top of a house has only to come after that of its basement and floors. Each method is uniquely named and defines one or multiple preconditions for its applicability to an environment state together with the respective set of totally (in SHOP2 partially) ordered sub-tasks it shall be decomposed to.
>
> *Problem model.* A domain compliant HTN planning problem model comprises an initial state and an initial task network as a kind of “goal” to achieve for the given state with reference to domain model.
>
> Representation of planning domain and problem models for HTN planning can be rewritten as an equivalent classical planning problem, with appropriate translation to PDDL such as in HDDL (Höller, et al., 2020), though it may become exponentially larger in size. One approach is to augment the domain with a set of methods and a way to translate each classical goal into a task list (Alford, 2009). However, in turn, not every classical planning problem can be represented as a total-order task network planning problem.
>
> *Plan.* A solution to or plan for a HTN problem is a sequence of actions extracted (left-to-right) as sequence of primitive tasks from the ground instance of the decomposed initial task network tree which satisfies all its constraints in this initial state.

Offline action planning under uncertainty allows for actions with non-deterministic effects and incomplete initial states caused by only partial observability of the environment system in the domain, respectively, problem model. A planner takes these uncertainties explicitly into account for producing conditional plans with (contingency, POMDP) or without (conformant) environmental state sensing capabilities during their execution. However, as with classical offline planners, in case of changes of the domain or problem model during the planning process such as deletion, insertion or update of operators with disjunctive effects, also this kind of planners will essentially have to recompute an appropriate solution plan from scratch.

Conditional or contingency planners like Cassandra (Pryor, et al., 1996), CNLP and DTPOP (Peot, 1998) devise a plan that accounts for each (or limited to number of) possible contingency that may arise in the planning domain. This corresponds to an optimal Markov action policy in the POMDP framework for planning under uncertainty with probabilities, costs, and rewards over a finite horizon. A contingency planner anticipates unexpected or uncertain outcomes of actions and events by means of planned sensing actions and attempts to establish the goals for each different outcome of these actions through a respectively conditional branching of the respectively tree-shaped plan in advance. Examples of decision criteria according to which contingency branches are inserted in the plan, and what the branch conditions should be at these points, are the maximum probability of failure, and the maximum expected future reward (utility) as a function of, for example, time and resource consumption. Uncertainty is often characterized by probability distributions over the possible values of planning domain predicates. The plan execution is driven by the outcome of the integrated sensing subplans for conditional plan branches and decoupled from its generation, which classifies these planners as offline (Klusch, 2008).

Offline POMDP planners like SARSOP (Kurniawati, et al., 2008), (Kurniawati, et al., 2011) represent a planning problem as an optimization problem in the presence of uncertainty about action outcomes and partial observability of the environment states. In general, this is a key characteristic of so-called Markov Decision process (MDP) based planning approaches for fully or partially observable stochastic domains. In a partially observable MDP (POMDP), the MDP-based planner can only partially observe the environment state space of current and preceding time steps via the controller to generate beliefs about states for finding an optimal action policy for all states. That is, these planners are not assumed to have full knowledge or observability of the environment state anymore. For example, an autonomous car may fully know its own state but not the state of pedestrians in its sensed surrounding who are occluded by static or moving obstacles on a sidewalk. The goal to achieve is indirectly represented by means of some utility function based on costs and rewards of executing certain actions in the environment. Therefore, MDP-based planning searches as a “plan” not for some action sequence but an action (selection) policy mapping any state to an action with maximal expected utility overall, but unlike classical planning without any explicit declaration of initial and goal state to reach. In POMDPs, the plan execution controller observes probability distributions over environment states, which transforms the latter into so-called belief states for POMDP planning. For the controller, multiple different states may result from same observation and vice versa.

> *Domain model.* A POMDP domain model defines finite sets of (environment) states, actions, and observations together with state transition and observation probability distributions. The latter is modelling partial observability of the domain. State transition defines transitioning at given time point from given state into some other state when executing a given action, while each observation probability defined for each state and action represents the probability of the execution controller to observe a (state) observation in a given state after executing a given action.
>
> *Problem model.* A domain compliant POMDP planning problem model defines as a goal a utility function for an action policy to be optimized with reference to the domain model. The utility function for evaluating the utility of a policy in a belief state is defined through a reward (and cost) function, and a discount factor for reward accumulations.
>
> As with HTN planners, the description format of these models as input to POMDP planners deviates from standard PDDL. For example, SARSOP takes its input models specified in the XML-based POMDPX file format and produces a policy file. The POMDPX format and other formats such as PPDDL (Probabilistic PDDL) (Younes, et al., 2004) and nuPDDL can be translated from the RDDL (Relational Dynamic influence Diagram Language) that is used as common description language in the POMDP track of the international probabilistic planning competition (IPPC).
>
> *Plan.* A generated plan as solution is an optimal action policy on belief states, that is, unlike in classical planning, a mapping of belief states to actions such that its expected utility is maximal. The expected utility of a policy considers the probability of belief state histories induced by the policy.

In contrast to online POMDP planning approaches (cf. Section 3.3.2.1), offline POMDP planners do not interleave construction and execution of action policy (cf. Figure 3.12). That is, the action policy constructed by the planner is passed to the controller that executes actions one by one and observes into which state each action puts the environment. Offline POMDP planners fully precompute an action policy for the whole belief space. While they have the advantage of fast plan execution because the plan is precomputed for the whole, potentially huge belief state space, this computation can take prohibitively long. POMDP planning is known to be computationally hard, in PSPACE. High dimensionality of POMDP planning through a large, continuous state space is usually addressed by means of various sampling methods such as particle filtering and Monte Carlo tree search (MCTS) in online POMDP planning.

3.3.1.2 Security Aspects

The following table (Table 3.1) comprises possible attack patterns for the various attack targets related to offline action planning. Subsequently, we describe these attack patterns in more detail with complementary remarks on the planning methods described above.

Table 3.1: Attack patterns and targets for offline action planning systems

Attack Pattern/ Attack Target	Domain Model	Problem Model	Internal Model	Plan	Planner
Model Theft	X	X	X	X	
Manipulation Model	X	X		X	
Manipulation Output				X	
Flooding					X

We assume that an *offline* planner prohibits access to its internal model during planning, and otherwise provides read-only access restricted to problem, domain, and plan files it processed or created within given period. In this respect, the internal model may be considered as an attack target for model theft but not model manipulation. The attack pattern for model manipulation is given below, the others are available in the Annex A.1.

Attack Pattern Name: *Manipulation Model*

> **Attack Prerequisites**: Data stores with API or planner API through which the planning domain and problem model files can be retrieved, read, updated, and deleted, or digital communication line of the planner through which the plan files are passed for execution by the controller. Alternatively, the planner provides API for read-only access of its internal model after planning ends.
>
> **Description**: The attacker manipulates the planning domain model and/or problem models. As a consequence, the planning may output an unintended plan or fail on purpose.
>
> A special case of this attack in this context is the ontology-specific form of *Data Poisoning* attack (cf. Section 3.2.1). In this case, the attacker offers the actors an ontology (schema and fact base) deemed relevant for its inclusion in the planning domain or problem model - but which is intendedly inconsistent or prepared to reveal inconsistency of this model after inclusion.
>
> **Method of Attack**: See attack pattern *Model Theft* as one prerequisite for performing this attack. The attacker illegally gains access to the planning domain and/or problem models, and subsequently changes their elements as s/he deems appropriate.
>
> For classical state and pattern-based planning, both models can be changed by inserting new, deleting, or modifying predicates, objects, and actions, and modifying the goal state. Such changes may either render the domain model itself inconsistent or trigger the subsequent planning process to insert and/or replace existing with unwanted actions defined by the attacker into the plan. For example, the latter can be achieved by the attacker as follows: simple replacement of an action

with one that executes malevolently, or insertion of some new action and predicates in the domain and then change of initial and goal state such that precondition, and effect of this action enforce its inclusion into the solution (plan) for the modified problem instance. For POMDP planners, the domain model in PPDDL can be additionally changed with respect to probabilistic effects of actions at the costs of actions of the attacker, and reward values for effects of actions of a plan.

The manipulation may also concern the making of an ontology imported by the actors inconsistent (cf. special case of *data poisoning* mentioned above). In this case, the attacker assumes that the import is not validated before its translation into PDDL and then inclusion in the planning domain and problem models.

Attack Motivation-Consequences: The motivations of model manipulation are manyfold: The attacker gains advantage (profit) from the integrated but otherwise unnoticed execution of its own or other third-party actions in produced plans, or from enforcing the planning of a certain plan-based service provider to fail in any case, or for given user group of problem instances, or certain goal states.

Attacker Skill or Knowledge Required: See attack pattern *Model Theft* as one prerequisite for performing this attack. The attacker must also have the skill to perform sound modifications of the models depending on the above-mentioned attack motivations.

Resources Required: See attack pattern *Model Theft*; in addition: PDDL editor for editing, maintaining of domain and problem models and checking syntax and validation of changed models and "to be enforced" solutions (plans) in PDDL such as PDDL editor[6] and PDDL4J[7], Web Planner[8], VAL and D-VAL.

Solutions and Mitigations: See attack pattern *Model Theft*. In addition: prior manual or automated verification and validation of all input models in PDDL (Fourati, et al., 2016) by the planner. Utilization of tools for checking of syntactic validity and consistency of imported formal ontologies before or after their merging and subsequent inclusion into planning domain and problem models in PDDL.

Syntactic validity of ontologies used for domain and problem models refers to the syntactically correct encoding of the ontology specification, which can be tested with appropriate validation tools such as the OWL validator[9], Apache Jena Inference API for OWL[10], and OOPS![11]. Consistency of ontology requires that the ontology specification does not include or allow for any logical contradiction. In other words, an ontology is inconsistent if it does not allow any formal model to satisfy its axioms. Checking of consistency can be done with a classical ontological reasoner such as Pellet. The above-mentioned validation tools also support consistency checking, though they differ in the extent to which they are verifying the considered ontology for common problems or pitfalls in this regard (Mazzola, et al., 2016).

[6] http://editor.planning.domains/
[7] https://sourceforge.net/projects/pdd4j/
[8] https://web-planner.herokuapp.com/
[9] http://mowl-power.cs.man.ac.uk:8080/validator/
[10] https://jena.apache.org/documentation/inference/index.html#validation
[11] http://oops.linkeddata.es/

3.3.2 Online Action Planning

3.3.2.1 Description and Methods

The abstract design pattern for systems of dynamic or online action planning is shown in Figure 3.12. It basically differs from the one for offline planning in that the planner is allowed to observe and take changes of the (application) environment into account during its re-/planning process. The initial domain and problem models are the same as for offline planning. As mentioned above, the *internal planning model* (Model:sem) includes the domain and problem models, as well as additional data structures used for online planning. In contrast to offline planning, this model is *updated by an online planner* during the interleaved planning and execution based on observed *changes of the environment* such as new facts or operators available. Such external changes can be reported to the planner through its interface in the closed-loop feedback cycle by the controller of the plan-based application environment. In particular, the change events received are further processed by the planner module "update:model" into the internal model. The planning module decides whether to proceed with or to (partially) restart its planning as appropriate.

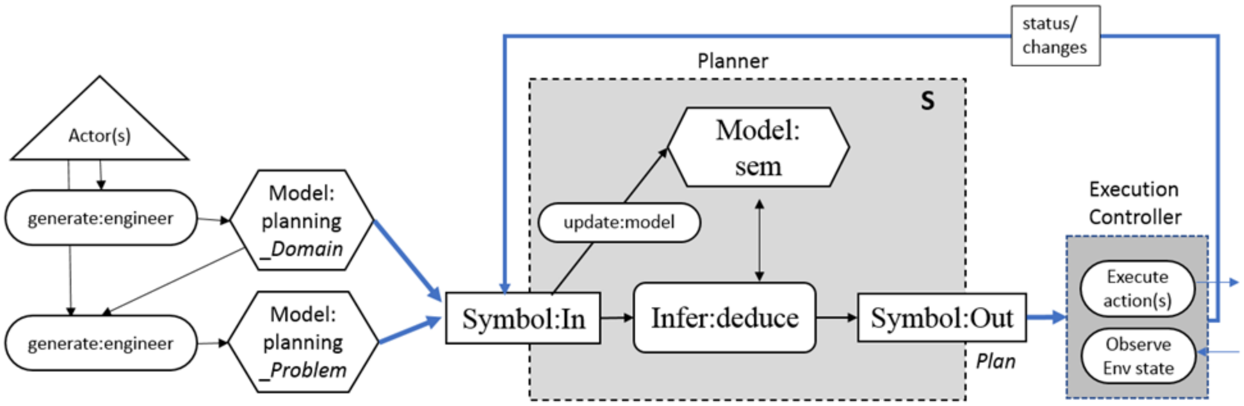


Figure 3.12: Generic design pattern of online planning systems

We consider two kinds of online planning under uncertainty: *restricted* and *reactive*. In both cases, the closed-loop execution controller is coupled with, and reports change events to the planner during its planning. Restricted dynamic planners may invoke specific information-gathering or sensing actions of the controller at each planning step in order to update its internal planning model and to decide on possible replanning. We distinguish between basic and advanced versions of restricted dynamic planning.

Basic restricted dynamic planners such as SHOP2 and OWLS-XPlan1 (Klusch, et al., 2006) make exclusive use of sensing actions, also called book-keeping services or call-backs, to detect previously unknown action outcomes adding new knowledge in form of ground facts to the planning states. That is, these observations trigger an *update of the internal planning model* of the planner during its planning. However, the internal model update is assumed to occur under the very restrictive IRP (Invocation and Reasonable Persistence) assumption to ensure conflict avoidance. The *IRP assumption* states that (a) the information gathered by invoking the call-back once cannot be changed by external or subsequent actions, and (b) remains the same for repeating the same call during planning. Therefore, the incremental execution of call-backs under IRP assumption during planning has the same effect when executing them prior to planning in order to complete the initial state for closed-world planning. Like in classical planning, however, world (environment) state altering services with physical effects (in opposite to the so-called knowledge effects of actions or service outputs) are only simulated locally by the planner but never get executed at planning time (Klusch, 2008).

Advanced versions of restricted dynamic planners such as XPlan2 (Klusch, et al., 2006) allow to react on arbitrary changes in the environment state that may affect the current plan during planning. This is in contrast to offline planning under uncertainty where sensing subplans of a plan can be planned but executed at runtime only. For example, the advanced restricted dynamic planner XPlan2 accepts as input the *planning domain and problem models* of classical planning in PDDL2.1 and outputs a sequence of actions as plan, if it exists. Like XPlan1, it is a heuristic hybrid fast-forward planner based on the FF- and GraphPlan planners combining a guided local fast-forward search with relaxed graph planning, and a simple form of HTN planning. In addition, XPlan2 performs event-driven dynamic re-planning through heuristic computation of best re-entry points for re-planning at the end of each planning step, if the current plan fragment is affected by the observed change. In other words, XPlan2 performs dynamic re-planning offline under closed-world assumption at each step, if required. The following types of changes observed by XPlan2 via its interface to the plan-based application (controller) are directly triggering an *update of the internal planning model (Model:sem)*: new operators are available (domain model), or operators are not available anymore (domain model); facts of the environment state changed (problem model, intermediate state), or new predicates (domain model) are available such that additional operators using these predicates in their precondition or effects become applicable to the new intermediate state; or the goal state changed. However, in both versions of restricted dynamic planning the interleaved execution of planning with world state altering services is prohibited to prevent planning inconsistencies and conflicts. In reactive dynamic planning the execution of *arbitrary* actions is allowed at planning time (Klusch, 2008), (Klusch, et al., 2006).

Pure reactive planners such as in RETE-based production rule planners, and the symbolic model checking based planner SyPEM (Bertoli, et al., 2004) produce a set of condition-action (if-then) or reaction rules for every possible situation that may be encountered, whether or not the circumstances that would lead to it can be envisaged or predicted. The interleaved planning and execution are driven through the evaluation of state conditions at every single plan step to select the relevant if-then reaction rule and the immediate execution of the respective, possibly environment state altering action; this cycle is repeated until the goal is reached.

Online approximate POMDP planners like IS-DESPOT (Luo, et al., 2019) and MCTS (Monte Carlo Tree Search)-based POMCP (Silver, et al., 2010), (Weichert, 2021) interleave the planning and execution phases. In particular, in each belief state of the environment, the *best action for this current belief state* is planned, then this action gets executed after which the action policy planning continues for the next belief state until time-out or a given goal is reached. While online planners do not need to precompute a plan for the whole belief space as offline POMDP planners do (cf. Section 3.3.1.1), they suffer from the fact that there is often a limited planning time available in practice.

A variant of reactive dynamic planning is dynamic contingency planning like in SAGE (Knoblock, 1995). In this case, a plan that is specified up to the information-gathering steps gets executed to that stage, and, once the information has been gathered, the rest of the plan is constructed. Interleaving planning and execution this way has the advantage that it is not necessary to plan for contingencies that do not actually arise. In contrast to the pure reactive planners, reasoning is only performed at branch points predicted to be possible or likely. In any case, such reactive dynamic planning comes at the possible cost of plan optimality, and even plan existence, that is suboptimality and dead-end action planning or failure (Klusch, 2008).

3.3.2.2 Security Aspects

The following table (Table 3.2) comprises possible attack patterns for the various attack targets related to online action planning process. Subsequently, we describe these attack patterns in more detail, together with remarks on their relation to the individual planning methods as described above. In general, the attack patterns for offline action planning systems are valid also for online action planning systems.

Table 3.2: Attack patterns for online action planning systems

Attack Pattern/ Attack Target	Domain Model	Problem Model	Internal Model	Plan	Planner
Model Theft	X	X	X	X	
Manipulation Model	X	X	**X_1**	X	**X_2**
Manipulation Output				X	
Flooding					X

However, there is a significant difference regarding security: The planner may become an attack target during its planning. In fact, the dynamic re-planning process of the planner can be influenced through means of targeted manipulation of its internal planning model used and updated online by the planner (cf. **X_2** in Table 3.2). That renders the internal model a first-class attack target for manipulation (cf. **X_1** in Table 3.2).

The attack pattern for model manipulation is given below, the others are available in the Annex A.1. Please note, that we assume online planners to be equipped with an API that particularly allows for a read-only access to the full internal planning model at runtime, which includes the current observations (events), the intermediate state reached and the current plan fragment. A model theft attack through such an extended planner API would enable an attacker to gain a more detailed view on the planning process online. This, in turn, is a prerequisite for targeted manipulations of the internal model through the issuing of appropriate but false changes of the environment during planning in time. Note that, usually, the internal model update process of online planners does neither authenticate nor semantically validate received events. Therefore, faked change events sent by the attacker may trigger the dynamic replanning process to generate an output that is more preferred by the attacker.

Attack Pattern Name: *Manipulation Model*

Attack Prerequisites: See attack pattern *Manipulation (Model)* for offline planning (cf. Section 4.1.2). In addition, the online planner API provides **read-only access to the full internal planning model during planning. The update module** does neither authenticate nor semantically validate received change events.

Description: As in offline planning, the attacker manipulates problem and domain models. In addition, the internal planning model can be manipulated during planning.

Method of Attack: See attack pattern *Manipulation Model* for offline planning. In addition, the attacker sends such manipulations as faked change events to the planner during planning which updates the module then updates the internal planning model accordingly.

Attack Motivation-Consequences: See attack pattern *Manipulation Model* for offline planning. In particular, the attacker now has an additional opportunity to influence the planning through the sending of faked change events even during the planning process which, in turn, increases the chance that a plan is being generated that the attacker considers appropriate.

Attacker Skill or Knowledge Required: See attack pattern *Manipulation Model* for offline planning. In addition, the attacker has to understand the syntax and semantics of change events the planner is listening for during its planning via the API. The attacker may also need to understand the intermediate states and current plan fragment generated during the planning process in order to be able to send appropriate faked change events in time.

Resources Required: See attack pattern *Manipulation Model* for offline planning.

Solutions and Mitigations: See attack pattern *Manipulation Model* for offline planning. In addition, **the update module also has to** authenticate and semantically validate received change events before the content of change events is processed into the internal planning model.

3.3.3 Applications

This section presents selected use cases of automated planning in the domains of health care, machinery maintenance in Industry 4.0 context and collision-free navigation of self-driving cars, each with examples of applicable attack patterns and targets. In general, the leveraging of automated planning for the detection of application and network attacks or the design of penetration testing plans such as in Bozic et al. (2020), Pozanco et al. (2021), Amos-Binks et al. (2017), Roberts et al. (2011), and Greenwald et al. (2009) appears more prominent than, in turn, to secure the planners themselves against attacks through means of, for example, plan validation (Da Cruz, et al., 2020). In the following, we only sketch visions of potential attacks for different use cases.

Planning of Medical Assistance Services. In Möller et al. (2006), the authors present the use of the planner OWLS-XPlan2 for medical transportation service planning in support of repatriation of patients in the European Union. The repatriation process is supposed to be planned by an emergency medical assistance party from either the private or the public healthcare sector with central planning site and connected medical assistants of patients during transport from hospital to their home or other hospital with closed-loop feedback to the planner. In this context, the advanced restricted dynamic planning (cf. Section 3.3.2.1) of OWLS-XPlan2 allows to react on, for example, changes of flight availabilities or change of destinations (goals) or medical states of patients to be transported during the complex planning process.

All attack patterns of offline and online planning apply here. For example, model theft attacks by competitors in this e-health sector may save them costs of development and knowledge acquisition. One step further, they can significantly harm the operational business of the attacked party by domain, problem, and internal planning model attacks such as changes of goal state with wrong flight destination and targeted hospital or change of medical state of the patient. The same holds with plan manipulation attacks such as wrong information about flight bookings or replacement of ambulance services the medical assistant relies upon for optimal patient repatriation depending on their medical state during the whole travel and respectively best services available. Needless to say, that these attacks can be life-threatening for the patient.

Planning of Machinery Maintenance and Production Services. In Mazzola et al. (2018), the authors present the semantic business service composition planner ODERU and its application to selected industrial use cases of machinery maintenance and car exhaust production in the context of Industry 4.0. It combines (a) pattern-based semantic composition of process service plans through semantic service selection for annotated process tasks of BPMN2 model and computation of possible data flows, and (b) optimization of non-functional aspects by means of QoS (Quality of Service)-based constraint optimization problem solving. The planning of optimal process service plans at design time is offline, while being basic restricted dynamic at runtime; ODERU is part of the CREMA platform for cloud-based elastic manufacturing[12]. The domain and problem models for the planning base on the public CREMA ontology (Mazzola, et al., 2016), the semantic service repository (actions), and the annotated BPMN models for machinery maintenance and exhaust production. The problem model instances have to be optimally implemented with an executable process service plan. In the first CREMA use case of ODERU, the objective is to suggest the maintenance manager the optimal combination of maintenance assistance teams and spare parts with minimal costs and time based on potentially competing assignments, hard and soft constraints provided by the client. The second process model concerns the production of car exhaust filter systems with the objective of achieving

[12] https://www.crema-project.eu/

an OEE (Overall Equipment Effectiveness) optimal assembling of a set of partially finished parts with appropriate tooling in a robot cell and operator skills and subsequent testing of the result for product conformity to the client quality requirements (Mazzola, et al., 2018).

Since the CREMA domain ontology is public, model theft attacks rather focus on private problem models with specific industrial objects and facts as well as goals to achieve for clients to gain competitive advantage. On the other hand, internal model attacks such as changing available maintenance services in the domain model or changing the problem model in terms of optimization constraints like price and deadlines for own benefit are reasonable but not addressed in ODERU.

Planning of Collision-Free Navigation. In Bai et al. (2015) and Pusse et al. (2019), the online approximate POMDP planner IS-DESPOT is used for collision-free navigation planning of self-driving cars. Apart from the fact that this reactive dynamic planning under uncertainty (cf. Section 3.3.2.1) may suffer from the computational time it takes to find the best action for the current belief state and observations, all attack patterns of online planning apply here in principle.

For example, online manipulation of the internal model by means of sending faked or noisy observations to the planner API yields a respectively wrong (root) belief state for MCTS-based action policy planning, hence outputs an action that might not be optimal for the situation at hand. This is potentially life-threatening for both passenger of the autonomous vehicle and pedestrians in critical traffic scenarios. On the other hand, model theft attacks may reveal interesting but private information such as previous or current goals and planned paths of the AV passengers, which could be sold to interested third parties such as service providers along this path or in the proximity of the goal. The same holds for gaining competitive advantage by insights of the specific car and pedestrian models used for the collision-free navigation planning.

3.4 Learning

In this section, we discuss selected machine learning methods, which fall under the category of being non-neural. Moreover, emphasized on those methods that are commonly used in categorial (i.e., symbolic) data.

3.4.1 Decision Trees

3.4.1.1 Description and Methods

Decision trees are a supervised learning method which achieves good performance for medium-sized data sets. A strictly symbolic decision tree works on categorial data (n-tuples of values); continuous numerical data can be admitted here as well. Certainly, however, decision trees for continuous fields such as image or video processing would rather be regarded as belonging to the sub-symbolic class of learning methods. An extended version with even better performance is called "Random Forests". Decision trees and their extensions are widely used, included in many well-known software libraries, and well understandable for human users for limited numbers of attributes. Consequently, decision trees are scalable and efficient from a technical point of view, but not quite as much for explainability.

A decision tree is constructed by evaluating the relative importance of attributes in the example (training) set, based on entropy or information gain. The most important attributes become the root node of the tree and so forth. In other words, the most important or decisive questions are asked first. Once the tree is configured, it can be used for evaluating queries which are new entities similar to, but not included in, the example set. Those entities have values for the same attributes as the examples. However, it is possible that not all attributes have values, which makes it more difficult to answer them.

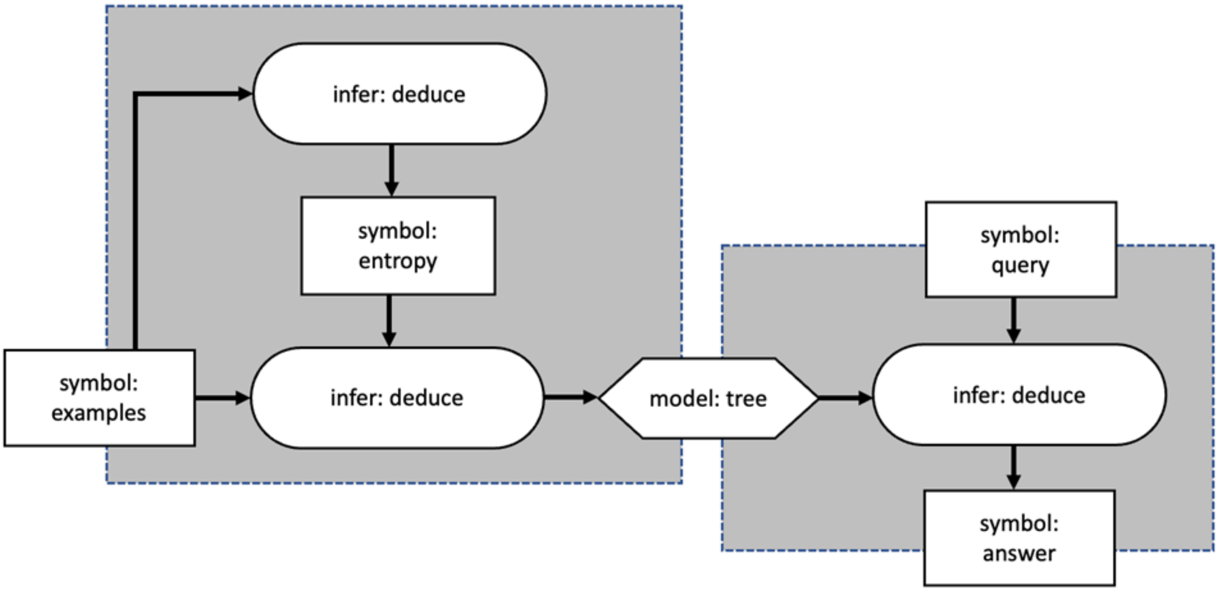


Figure 3.13: Generic design pattern for decision tree learning methods

Decision trees can be used for classification, such as the selection of options regarding several relevant attributes. Alternatively, predictions can be made (regression), but this approach is used less frequently. Although decision trees are easily understandable and quite scalable, their main drawback is that they depend very much on the example set. The addition of one example in the training set can result in a very different tree.

Gradient Boosting is one of several machine learning methods that use multiple models instead of just one. This approach is called ensemble learning and relies on combining multiple outcomes of simpler models (also called base models or hypotheses) in order to reduce bias and variance. The simpler models can be decision trees, in which case the ensemble is called a Gradient Boosted Tree.

With boosting, a model is constructed incrementally and optimised according to a loss function. An example is AdaBoost that can be used with many different base models. An initial vector of weights for the examples of the training set is repetitively adjusted to improve the final outcome. In general, the wrongly classified examples receive a higher weight, because they appear to be more difficult to classify with the current model, until all examples are reliably classified. For comparing the different hypotheses, a gradient (differentiable loss function) between the predicted and correct answers is calculated to influence the adapted weights.

3.4.1.2 Applications and Security

Brewitt et al. (2021) propose a goal recognition approach for autonomous driving based on decision trees. With the term goal recognition, they refer to the task of predicting the trajectories of traffic vehicles. The authors use decision trees that are trained from (symbolic) vehicle trajectory examples. They furthermore propose a verification by mapping the learned trees into propositional logic using an SMT (Satisfiability Modulo Theory) solver. However, we only look at the actual decision tree method, not at the extras around it. The trajectories are represented as time series based on pose (location and orientation). A goal of a vehicle is described as reaching a certain state i.e., a target location (a junction or the visible end of a lane), which is relative to the static scene information (local road layout). As input during inference, the decision tree takes the past observed trajectories of local vehicles and the static scene information. As output, it gives a probability distribution over possible goals for a vehicle. Decision trees are used to infer a likelihood for each goal, before inferring a posterior probability distribution over goals.

Kashifi et al. (2022) propose a system for predicting the severity of car accidents based on histogram-based gradient boosting (HistGBDT). The goal of the model is to facilitate efficient emergency response management. The experiments were conducted on French accident data from 2005 to 2018. The HistGBDT model showed an overall accuracy of 82.5%, recall of 76.7%, and precision of 81.9%. It outperformed other models. An analysis of feature importance indicated that safety equipment was the most important feature and vehicle category, department, localization, and region were other significant features. Without a doubt, such an application belongs to the class of those that absolutely must be protected against a cyberattack. In Kashifi et al. (2022) it becomes particularly clear that if the attacker has knowledge that the safety equipment plays an important role, he can exploit this information for an attack. Depending on how data is transmitted to the control centre, the attacker can try to inject a falsified assessment of the safety equipment. The consequences could be fatal if, for example, rescue vehicles were not dispatched optimally for a longer period of time until the attack was detected.

Regarding the security aspects, it must first be noted that decision trees are a form of aggregated information derived very directly from the original dataset (Zhu, et al., 2010). Therefore, a decision tree still contains a certain amount of private information and thus can lead to a potential privacy violation. Data poisoning in decision trees has also been studied extensively (see, e.g., (Drews, et al., 2020) for a list of related work). This work has shown that attacks can sometimes dramatically degrade the accuracy of decision trees. Instability in decision trees is a well-known problem (ibid). For example, decision tree learning algorithms are generally shown to be vulnerable to data poisoning attacks. Decision trees represent sets of "rules" so one can analyse the conditions under which these rules remain stable. However, Drews et al. (2020) point out that mitigation should focus on formalizing the underlying decision tree learning. Under these conditions, one can also allow the "rules" to change as long as the final classification is not altered. Zhang et al. (2020) study the problem of efficient adversarial attacks on gradient boosting decision trees (GBDTs) and random forests (RFs). They formulate the attack problem as a discrete search problem specifically designed for these types of decision trees. The search objective is to find the shortest path (i.e., least perturbation) to a valid "leaf tuple" that leads to misclassification. In their research, the authors show that this can be done quickly and efficiently using experimental results on several large GBDT and RF models with up to hundreds of trees.

3.4.2 Bayesian Networks

3.4.2.1 Description and Methods

Bayesian networks (BN) are probabilistic graphical models used to represent probabilistic knowledge and make inferences from it (Derks, et al., 2020). Therefore, BNs could also be called probabilistic reasoning methods and classified as such in Section 3.2. However, due to the possibility of learning both conditional probabilities and network topology from data, we decided to classify them here.

To illustrate BNs, we consider the classic "Asian network" (Lauritzen, et al., 1988): A patient complains of shortness of breath (dyspnea, D). The physician knows that possible causes of dyspnea are lung cancer (C), bronchitis (B), and tuberculosis (T). Smoking increases the likelihood of lung cancer and bronchitis, while both tuberculosis and lung cancer lead to abnormal radiographic findings (X). Previous residence in Asia increases the likelihood of tuberculosis. Based on these statements, the nodes and values are defined and then the graphical structure of the network is constructed (or learned). This is followed by quantifying the tables of so-called conditional probabilities (CPTs) for each node. These are either defined by a domain expert (in this case, a physician) or learned from data.

The resulting Bayesian network is shown in Figure 3.14. The arrows represent the direction of the actual causal relationship: B causes D, A causes T, etc. Some nodes are directly observable (A, S, X, and D), others are not (T, C, B, P). The posterior probabilities of all nodes in the network are updated after each observation. Thus, a concrete probability for C, for example, can eventually be determined.

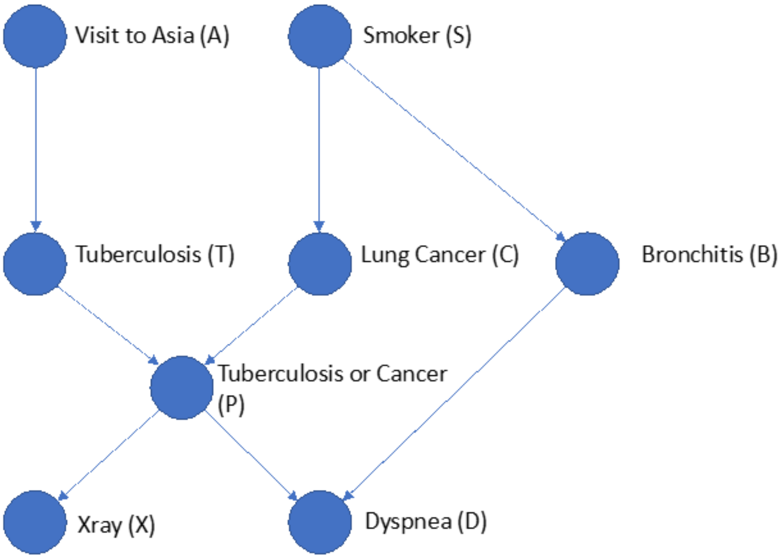


Figure 3.14: Bayesian Net "Asia" according to Lauritzen et al. (1988)

3.4.2.2 Applications and Security

Bypassing Credit Card Fraud Detection. Anomaly detection, for example fraud transactions with credit cards, are a typical scenario for dynamic Bayesian networks. Two transactions executed in close succession in distant cities are extremely unlikely. A card payment made again shortly afterwards in the first city would be particularly unusual. In such a case, the credit card company will contact the credit card owner at short notice. The case becomes somewhat more complicated if there is a certain semantic proximity between the actions. For example, an attacker who has knowledge of the card number and possibly also the card verification value and intends to use it to rent a car in Barcelona could first create the semantic proximity to an intention to travel by making bookings and cancellations with an airline. This will significantly increase the probability of using a car rental and the actual rental will no longer stand out as a suspicious case. In this case, we have a classic evasion attack. The security mechanism is bypassed by entering unsuspicious, yet misleading, transactions. This application suggests the following design pattern description in Figure 3.15.

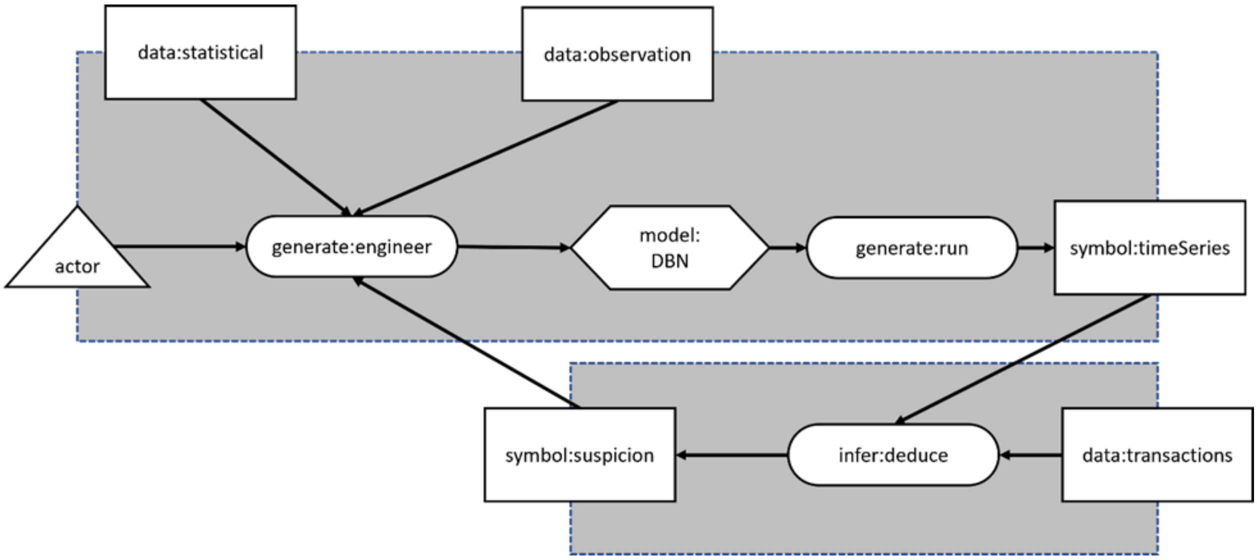


Figure 3.15: Design pattern of DBN-based application Bypassing Credit Card Fraud Detection

Regarding the security properties of Bayesian Networks, it is important to distinguish whether or not structure learning or CPT learning occurs. Alsuwat et al. (2019) study poisoning attacks in the context of structure learning. The attacker's goal is assumed to either invalidate the resulting BN or (and this is the more interesting case) to suppress or add certain edges. Alsuwat et al. focus on constraint-based algorithms for structure learning, in particular the Principal Component (PC) algorithm. For their implementation, they use the Hugin PC algorithm. Hugin is a widely used tool for Bayesian networks (Olesen, et al., 1992). The authors use a number of examples to show how data poisoning can be used to suppress or add edges selectively. When evaluating this case, however, it must be taken into account that this was an attack on the PC algorithm, not on the Bayesian network itself. In principle, the well-known wisdom applies here that a chain is only as strong as its weakest link. So, if a symbolic method is based on a numerical, possibly gradient-based component, then an adversarial attack can "propagate", especially if this component affects the structure of the model, as in this case. In Chapter 4, we will encounter more such cases, and they will occur whenever a sub-symbolic component informs a symbolic one.

3.4.3 Inductive Logic Programming

3.4.3.1 Description and Methods

According to Cropper et al. (2022), inductive logic programming (ILP) is a combination of machine learning and knowledge representation. Based on a given symbolically represented background knowledge (domain knowledge) and given positive and negative examples of target predicates, an ILP procedure can generalize the example inputs or infer hypotheses from them. In this way, ILP procedures are similar to general induction in logical reasoning. Unlike other symbolic learning procedures, whose representation format is restricted to propositional logic, ILP procedures use restricted forms of predicate logic as the representation format for examples, background knowledge, and hypotheses. Since the learned models are represented as logical programs (set of logical rules), relations are learned in terms of Horn clauses rather than functions (Cropper et al., 2021). Among other things, this leads to humans being able to better interpret resulting models or hypotheses by their symbolic nature, or to understand reasons for the output of an ILP model. The generic design pattern of this symbolic AI method for ILP-based learning is shown in Figure 3.16.

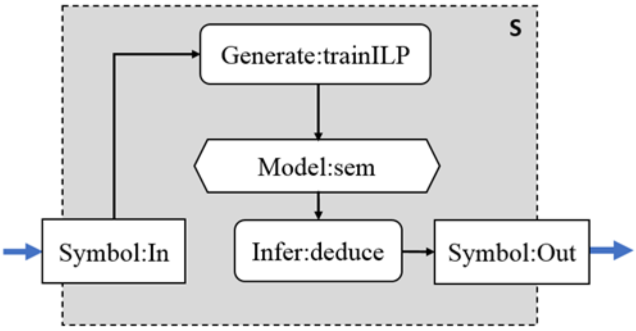


Figure 3.16: Generic design pattern of ILP-based learning methods

In addition to these properties that are relevant with respect to explainable AI, ILP methods typically require significantly less data (number of examples) than other machine learning techniques to inductively generate a hypothesis. In some cases, a single example is sufficient to conclude a model (Muggleton, et al., 2018). ILP itself has its origins in the 1990s (see (Muggleton, 1991), (Quinlan, 1990).

Textbook examples for ILP often use toy language problems such as

Input	Output
BSI	I
study	y

that are catchy and easy to explain. In ILP, these examples are represented as logical atoms, such as f([s,t,u,d,y], y), where f is the target predicate that we want to learn (the relation to generalise). Auxiliary information are provided as background knowledge (BK), also represented as a logic program -- logical definitions for string operations, such as empty(A), which holds when the list A is empty; head(A,B), which holds when B is the head of the list A; and tail(A,B), which holds when B is the tail of the list A. In the above example, the goal would be to find statements in the background knowledge that represent the generalisation, which is in this case tail(A,B).

Assuming that an appropriate *background knowledge* exists for the given domain, the problem is to find the right statements. For this analysis, we follow the approach proposed by Cropper et al. (2021), which is called "*learning from failures*" (LFF). In this approach, an ILP system (the learner) decomposes the learning problem into three separate stages: generate, test, and constrain. In the generate stage, the learner generates a hypothesis (a logic program) that satisfies a set of hypothesis constraints (constraints on the syntactic form of hypotheses). In the test stage, the learner tests the hypothesis against training examples. A hypothesis fails when it does not entail all the positive examples or entails a negative example. In this sense, one could argue that (depending on the input data that can be sub-symbolic or symbolic) the approach already belongs to the class of Hybrid AI, which will be subject of Chapter 4. The examples would then belong to the machine learning part, while the constraints represent a form of symbolic knowledge base that forces the learning to remain within certain boundaries (constrains the learning). We will inter alia see this in the general design pattern of "Learning with semantic priors". A key idea in LFF is to translate failed hypotheses into sets of constraints.

In the given case of ILP according to Cropper et al. (2022), if a hypothesis fails the learner learns constraints from the failed hypothesis to prune the hypothesis space. For instance, if a hypothesis is too general (entails a negative example), the constraints prune generalisations of the hypothesis. If a hypothesis is too specific (does not entail all the positive examples), the constraints prune specialisations of the hypothesis. This loop repeats until either (i) the learner finds a hypothesis that entails all the positive and none of the negative examples, or (ii) there are no more hypotheses to test.

3.4.3.2 Applications and Security

ILP methods should be considered as a basis for real-life applications if 1) explainability is required and 2) a learning result is to be generated on the basis of a few examples. However, the prerequisite is that a common knowledge base in the form of a hypothesis space is available. Thus, the domain must not be too open - pixel-based object recognition in images is therefore not expected to be one of the application domains of ILP. Gulwani et al. (2015) mention the automation of repetitive tasks as a concrete possible application, e.g., in the domain of interactive software tools such as spreadsheet programs. Meli et al. (2021) describe a repetitive-task application of ILP in the field of medicine. According to the authors ILP is a good choice for task planning in robot-assisted surgery because “it supports reliable reasoning with domain knowledge and increases transparency in the decision making”, which are precisely the benefits already pointed out. However, they also highlight the problems that need to be addressed as “prior knowledge of the task and the domain is typically incomplete, and it often needs to be refined from executions of the surgical task(s) under consideration to avoid sub-optimal performance”.

Muggleton et al. (2018) argue that human observation (in that case Galileo’s observation of the moon using a telescope) does not require many examples because it is relying on knowledge such as geometry, the straight-line movement of the light of the sun, and the sun as on out-of-view light source. They propose an ILP to derive such logical hypotheses from a small set of real-world images. The background knowledge is comprised of statements that model some physical aspects of light propagation thus being able to reason about the shape of the object (e.g., convex, concave). In their studies, this helped to recognize a ball (in a robo-soccer setting) even when it was partially occluded, because the light reflection properties clearly revealed its spherical shape. The relevance of this approach in safety-critical applications such as autonomous driving is obvious. As pointed out above, besides the training examples, the background knowledge would then be an attractive target for the intruder: an injected set of logical statements that describe for example light reflection in seldom conditions may remain undetected in tests and cause misclassifications. Like in the case of instance-based systems, this attack could be combined with the manipulation of the environment, i.e., altering the visual criteria of for example a road sign in order to meet the physical rules injected before. Note that the pair of rules and object can be artificial, which makes the attack hard to uncover.

3.4.4 Instance-Based Learning

3.4.4.1 Description and Methods

Instance-based learning is a class of learning algorithms that do not perform explicit generalization. Instead, they compare new problem instances with instances seen in training. Those instances are directly stored in memory. The methods are called “instance-based” because they build hypotheses directly from the training instances themselves. This means, however, that the hypothesis complexity grows with the size of the data set. One advantage that instance-based learning has over other methods of machine learning is its ability to adapt its model to previously unseen data. Instance-based learners may simply store a new instance or throw an old instance away. If instances are human understandable even though they get transformed in non-symbolic form but remain identifiable and interpretable as such during the whole learning process with symbolic output, the process is considered as kind of symbolic machine learning.

Example methods. One of the best-known example methods of instance-based learning is kNN (k Nearest Neighbours). In kNN, clever indexing is used to store all examples. Then typical examples are selected. For a new observation, the most similar examples are found, whereby different similarity measures can be used. Based on a decision function, such as the maximum, average or majority, the result is then determined. The similarity can be determined as for individual attributes or thus combination of the attribute similarities, whereby the relative importance of the attributes can be considered by weighting. In instance-based methods, the criteria must be considered carefully for each data set (model selection).

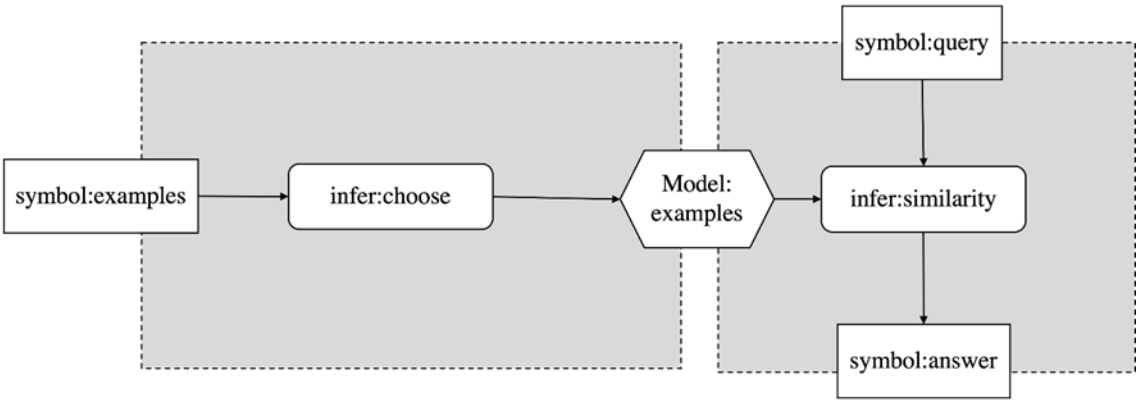


Figure 3.17: Design pattern of instance-based learning methods on the example of kNN method

Support Vector machines (SVMs) can also be classified as instance-based methods. Here, training examples stored as "support vectors" build the basis for determining a decision hyperplane similar to the examples in kNN. However, in SVM, a transfer to a hyperdimensional space takes place by applying certain functions that are combined in a "kernel".

3.4.4.2 Applications and Security

Instance-based learning methods are relevant in practice especially due to their explainability. Any explanation for a decision can be derived from the similarity to previously seen cases. Although instances in IBL models are task dependent, the decision process is generic and thus the same theoretical concepts apply to many different tasks.

Putra et al. (2020) propose a computer vision-based approach using kNN classification for vehicle (car vs. no car) detection in highly automated driving. They use histograms of oriented gradients (HOG) as features. The HOG feature extraction produces a vector of about 4000 individual features for each frame. According to the authors, due to an optimized sliding window method, the required computation time could be reduced to the point where the method can be used in practice. Regarding the choice of the instance-based method kNN for the classification, the authors state that it is a very simple algorithm that can be implemented well as supervised learning. They use Euclidean distance as the similarity measure, and the value for k was determined experimentally. Here, the trade-off between overfitting on the one hand (due to a too small k) and a suboptimal accuracy (due to a too large k), which is typical for the class, had to be solved. The empirical results are in the range of just below 90% accuracy on own data (i.e., not belonging to a publicly available challenge). The authors compare the kNN approach with a support vector machine (SVM), yielding no significant difference.

As far as the *practical relevance* of the example application is concerned, it must be clearly pointed out that the classification performance is not sufficient to compete against the current methods based on deep neural networks. It cannot be evaluated whether the HOG features do not allow a more accurate classification or whether the decision hyperplane of the kNN or SVM was too simple. It is reasonable to assume that it was due to both. However, this does not mean that the use case is irrelevant. First, improvements are always conceivable, and second, there are a number of tasks in the architecture of autonomous vehicles for which non-neural methods may be more appropriate than the vision-based perception components that are a classic domain of DNNs.

According to Li et al. (2019), adversarial attack methods known from DNNs are not immediately applicable to kNNs because they would require large perturbations and are not successful with a large k. They do, however, propose a new attack method for (numerical) kNNs called “AdvKNN”. The main idea of the approach is to learn a neural network based “deep kNN block”, which is a module that “emulates” the behaviour kNN classifiers, leading to adversarial examples with minimal perturbations. For SVMs, adversarial attacks are likewise possible. According to Biggio et al. (2012), it is possible that an “intelligent adversary” can predict changes in the SVMs decision function and construct input that increases the test error. They propose a gradient ascent attack strategy that can even be applied to non-linear kernels. Biggio et al. (2014) furthermore demonstrate the feasibility evasion, poisoning and privacy attacks against SVMs. Regarding evasion attacks, they showed that SVMs can still be evaded if the adversary can only surrogate a model of the classifier. As a means to improve the classifiers security, they suggest introducing a “tight enclosure” of the legitimate class, i.e., penalize “blind spot” areas with a low p(x). For poisoning Biggio et al., refer mostly to the earlier contribution cited above, pointing out that in future work the possibility of attacking with surrogate data should be explored, i.e., without training data being known to the adversary. Regarding privacy, the authors suggest adding just as much noise to the SVM’s weight vector that would not decrease its stability.

The *security assessment* of the previous application example, which is purely on the symbolic learning part of kNN, yields the following: In principle, adversarial attacks in the sense of the known cases are possible. However, we assume that a manipulation of the environment, which would result in a substantial change of the HOG feature vector, would require more far-reaching changes than a comparatively small perturbation. It is therefore reasonable to assume that these changes would be very noticeable. As for the process itself, the following would be worthwhile targets: Similarity function, value for k, and of course the examples. Tampering with any of these places would severely compromise the recognition According to the literature, performance. If the intruder were to succeed in injecting its own examples, the attack could also be very targeted to produce a particular result in a particular traffic situation (analogous to using a speed 50 sign instead of a stop sign). Probably the most perfidious attack would be a combination of manipulation of the environment and infiltration of own examples, which sensitize the model for this manipulation.

3.5 Summary

Neural Networks (NNs) are excellent and fast in extracting patterns from raw unstructured data and learning powerful representations. However, they are fragile methods with regard to small perturbations in input space both in computer vision and natural language understanding. For example, it’s been shown that slight modifications to the natural images can place them close to the decision boundaries learned by classifiers changing their final predicted labels. Therefore, an attacker can often find adversarial examples by searching in the vicinity of the input image space, comprising the security of the NN model.

According to the literature, white-box attacks are currently the dominant types of attack in NNs in which the attacker usually has detailed information about the target model. In contrast, in black box variants (Guo, et al., 2019), the attacker only has access to the public APIs and solely relies on the predicted label (decision-based attacks) (Brendel, et al., 2018) and, in some cases, a confidence score (score-based attacks) as the response. In white box attacks on NNs, the searching problem is often formulated as gradient-based attacks in which the search is guided by the gradient of the loss with regard to the input.

Symbolic methods, in contrast, are expected to be robust to such small changes due to their discrete, transparent, and verifiable nature. Gradient-based attacks lose their power against most symbolic approaches, often due to the non-differentiability of symbolic modules. For example, decision trees and their ensembled variants, mentioned in the symbolic learner section, fall into this category. Therefore, the attacker requires detailed knowledge of the symbolic methods and their special structures, e.g., trees, to conduct a successful attack (Zhang, et al., 2020). Since accessing such information regarding deployed models is usually hard to obtain in practice, the likelihood of an attack is lower.

In real-world scenarios, for example, in Google Cloud Vision APIs, access to the internal model architecture/parameters or training data is strictly restricted, and decision-based black box attacks are more probable. Therefore, the attacker must search the input space by querying the model using APIs. The number of queries should also be minimized (Cheng, 2020) because they are usually expensive in terms of time and money.

We can combine the symbolic methods presented in this chapter with NNs, called neuro-symbolic, to address their deficiencies in robustness, transparency and interpretability, sample inefficiency, and adversarial attacks. In neuro-symbolic models, mainly when NNs are used for high-level concept detection and representation (neural perception), the intermediate concepts and symbols are manipulated with deterministic, verifiable, and transparent symbolic modules preventing the attacker from performing white box attacks. We argue that combining NNs with symbolic methods has the potential to make the final model and its prediction more interpretable both for designers and end-user, respectively, while exposing the neural modules less directly to the public. In other words, this makes the model black box from the attacker's point of view by providing only high-level symbols as final predictions or explanations.

Note that in score-based attacks, even the predicted scores (e.g., class probabilities or logits) of the model can be used by the attackers to estimate the gradients numerically and, therefore, narrow down the search space. In such cases, a symbolic explanation removes the need for exposing the raw scores and provides a better rational high-level description of how the decision is made.

In order to aggregate our findings on the security traits of symbolic AI, let us review a recent study on statistical machine learning or deep learning (DL). In a survey on attacks and defences for DL-based Autonomous Driving system, Deng et al. (2021) relate attack types, attack objectives, methods, settings, and experiment settings. Attack types are evasion and poisoning attacks; attack objectives refer to components in the autonomous vehicle architecture (for example end-to-end driving model or 3D object detection); methods refer to the manipulation (e.g., drawing black strips on the road by Gradient-based Optimization method, pasting adversarial stickers that generated by optimization-based approach on traffic signs); the attack setting is white box (intruder has access to the interiors of ML system) or black box (intruder can only apply the system); experiment setting is real word, digital dataset or simulation.

Without going into details, it becomes apparent that any of those attacks require experiments with a good deal of knowledge about the ML/DL method implementation at hand. Adversarial textures or image triggers in poisoned data are themselves results of a learning procedure. They operate on the lower levels of the process close to the sensor. In this sense, the attacks take advantage of the fact that in DL the system is "left alone" with mapping features on the lowest possible level onto output categories. In symbolic AI, this is – by definition – not the case. The lowest level of features we described in our study are FOL primitives, a level on which the possibilities of completely obscure adversarial phenomena are already abstracted away.

This does not mean that symbolic AI is not attackable. As we have seen, attacks of various kinds are possible. However, the stimuli are not obscure – they lie in the open. This has advantages as well as disadvantages. From the perspective of the intruder, the advantage is that s/he can get away without complex experiments. Once s/he has access to the model (we must assume the same conditions described above for DL), s/he can analyse everything (since everything is abstract and human-readable) and design his/her poisonous examples, fake background rules, manipulate application-specific planning problems as input, or malicious similarity measures. To his/her disadvantage, however, the attack is much easier to detect as well, because nothing can be obscured. Formal verification and consistency checks can be applied as well (while in DL all information is distributed in the neural network). Which side prevails depends on the circumstances. The less a manipulation exposes itself in "normal" application, and the more indirect its relationship is with the planned attack, the more likely it will be undetected (until the attack is triggered). In this sense our example of adding physical properties of light reflection that correspond to adversarial objects is such an attempt, because the rules themselves are in the background model and do not necessarily appear suspicious.

Clearly, for many real-world applications, there must be components translating raw input data to symbols. However, the opportunities for adversarial (texture) based attacks are greatly diminished, since there is always the level of the symbols mediating between input and output. In other words, finding an adversarial texture that would first translate into a weird symbol, which again leads to a corrupt output is not a likely scenario.

Another point that should be considered in this context is the way in which the models come about. A DL model for a specific application is often not developed from scratch but is at least based on an already known neuronal network architecture, which is then adapted or extended at best. So, apart from the hyperparameters, it is not unlikely that the attacker knows how the model "works". Due to the fact that the number of popular frameworks is limited, it can even be assumed with some probability that the attacker has a similar implementation. The work of Data Scientists developing new AI applications often consists of downloading a suitable model from public project websites or open-source repositories, providing appropriate datasets, and experimentally optimizing hyperparameters. Attack experiments, as described above, thus do not necessarily have as a prerequisite that direct access to the system is available.

The effort (especially for pure Data Scientists) in manipulative attacks of symbolic AI methods compared to purely data-driven DL methods) should not be underestimated. The effort alone of a syntactically correct and consistent representation of manipulative input with appropriately suited, specialized software tools is high and the handling of these tools requires its own learning curve in practice. This is independent of the import of available standard models such as various ontologies, because their modification and extension, even for the smallest fact changes or additions, as well as the necessary logical checking is often very time-consuming even for experts. Manipulation of the input (data poisoning) in the case of a planner requires manipulative intervention in a readable, interpretable but complex description of problem AND domain in a PDDL (Plan Domain Definition Language) language variant AND the subsequent verification of this manipulation with a suitable tool.

Symbolic methods can also be based on standard models if the knowledge base is appropriately large. However, in most of the applications we analysed, it is more likely that the model has mainly application-specific facets. Furthermore, it can be assumed that symbolic procedures are more often available in a custom implementation than is the case with DL. Evasion attacks on the symbolic level without direct access to the system are nevertheless conceivable, as our example "Bypassing Credit Card Fraud Detection" in Section 3.4.3.2 shows.

All approaches in the area of knowledge representation and logical reasoning have in common that the problem under consideration is represented in a model that consists of a collection of formal sentences and that there exists a deduction mechanism that allows the reasoner to infer further knowledge to be added to the model. What the approaches do *not* have in common is the level of abstraction in both the knowledge representation and the inference machinery. For instance, reasoning with SAT solvers requires that the problem under consideration can be encoded as a SAT problem. Once this has been done successfully, the structure of the original problem is completely lost. As a result, any successful manipulation of the model can hardly be detected. Even the change of a single propositional variable would have devastating effects on the meaningfulness of the model. The same applies to the inference mechanisms used. If, for example, the problem to be processed is described by formulas of predicate logic, there are a multitude of deduction methods that could be used. In the case of "natural deduction", both the problem description and the individual deduction steps are easily comprehensible for an experienced person. However, if the decision is made to use "resolution" as an inference mechanism (e.g., due to efficiency considerations), the predicate logic representation of the problem is first converted into a normal form suitable for resolution. This has the consequence that along with original problem structure the readability of the inference steps is lost, and a manipulation of the model can hardly be detected. Such attacks can therefore only be countered if the possibility of manipulation is excluded from the very beginning by means of suitable access restrictions.

Dynamic symbolic AI methods are more vulnerable to attacks than their offline variants. In online planning, the process of feedback from the external plan execution controller to the planning system (Section 3.3.2) could be intercepted (and understood). This does not require any access to the planner or its model. This is not possible with the offline planner since it does not have this "open" feedback. This can be dangerous in applications, e.g., in the case of online collision-free manoeuvre planning or online planning of manufacturing or medical support services (Section 3.3.3).

While symbolic AI methods are already used for security applications (verification), the issue of securing these methods against attacks and integrated attack detection is only moderately, if at all, a current research topic in AI. In all of the symbolic AI methods studied, securing authorized access to system interface functions, external data sources (such as the domain and problem models), and feedback communication is ultimately critical in dynamic methods. Additional, integrated attack detection by the method itself is missing. In addition, a use of existing methods proposed in symbolic AI for more flexibility to tolerate logical inconsistencies and uncertainty in the symbolic knowledge representation would, in our estimation, make attack detection and recovery more difficult.

The attacks on concrete application examples we listed in this document, are mostly hypothetical, because there is little published work about security of symbolic AI. What we recommend doing is concrete empirical research on some of those. This research could for example have the goal to describe the trade-off of human interpretability between easier to attack and easier to defend (detect the attack, consistency checks). A very interesting study could be, for example, to implement a perception system based on light propagation rules as described in Section 5. In a series of experiments, one could find out, among other things, whether it is possible to describe adversarial properties in this way, what kind of access to the system one would need as an attacker, how well the changes can be detected, and ultimately what effect one can achieve. It makes sense to do this in simulation first before experimenting in real environments. The project could be extended by using a hybrid (neuro-symbolic) method. In Section 4.5 we further elaborate on the case.

4 Hybrid AI Method Classes

In this chapter, we discuss the security aspects of the following top-most prominent classes of hybrid neuro-symbolic or neuro-explicit AI methods in the literature (van Harmelen, et al., 2019), (van Bekkum, et al., 2021):

1. Learning intermediate abstractions for reasoning or planning
2. Informed learning with prior knowledge
3. Learning logical representations for statistical inferencing
4. Explainable learning through rational reconstruction

Like in the previous chapter, for each of these method classes, we first outline its generic functionality and design pattern together with selected examples of such methods in the literature. For reasons of readability, in the generic design pattern of each class of neuro-symbolic methods, its combined neuronal and symbolic AI modules are labelled 'N', respectively, 'S', and called N- and S-module, respectively. This is followed by a general discussion of security aspects of the method class in terms of applicable types of attack patterns, and then an exemplification of attacks for a selected representative method of this class in the context of a practical use case.

4.1 Preliminaries

4.1.1 Hybrid System View

The *input and output of a hybrid neuro-symbolic system* refer to the external input and output of its N- and S-modules, which can be accessed and manipulated through the hybrid system interface. The *hybrid system model* consists of the semantic model of the S-module (cf. Section 3.1) and the statistical model of the N-module. We assume a hybrid system to be equipped with a *hybrid system interface* that allows to retrieve and update (modify, delete, insert) the specific input, output, and models of its interacting N- and S-modules. This holds differently for the training and operational phase of the hybrid system. Please also note, that the external hybrid system inputs to the N- and S-modules can be provided through the hybrid system interface in parallel.

The *hybrid system training* phase refers to the training of the N-module model and, if applicable, the S-module model. In the latter case, the training of both modules may be conducted either sequentially or entangled at the same time, depending on the type of their I/O dependencies. In the first case, for example, the training of some neural network (N-module model) may require the output of a previously trained dynamic Belief network (S-module model) as an additional input (Muscholl, et al., 2020). In case the S-module model does not have to be trained in prior, such as in automated planning, its produced output still may be required online in the training phase of the hybrid system to train the N-module model using, for example, some cross-entropy loss function (Pusse, et al., 2019).

The *hybrid system operational* phase refers to the usage of its N- and S-modules running in applications. We restrict our study to operational cases where the adaptivity of the hybrid system comes as a feature of its given models, such as the generalisation to unforeseen situations by its trained neural network. We assume that the trained N-module model itself is not modifiable during the operational phase of the system. This excludes, for example, the replacement of a trained neural network with another one during this phase.

A *hybrid system architecture type* is called (a) *sequential* (SEQ), if the I/O dependency between the N- and S-modules is unidirectional (sequential hybrid system types: N→S, S→N), and (b) *entangled* (ENT), otherwise, that is the modules are in some way mutually dependent on each other (entangled hybrid system types: N→S→N / N←→S and S→N→S / S←→N), as shown in Figure 4.1.

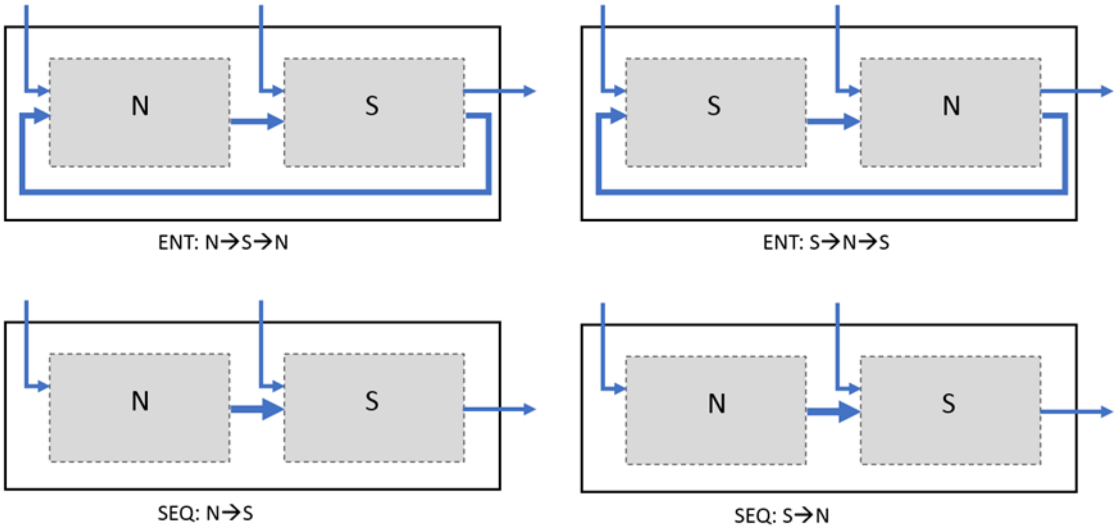


Figure 4.1: Generic hybrid system architectures of type sequential and entangled with I/O data flow

The indicated feedback loop between entangled N- and S-modules in the I/O data flow of the hybrid system is not restricted to one execution pass, and the displayed order of the modules in Figure 4.1 depends on which of them produces the main hybrid system output. Further, we distinguish between the training and operational phases of a hybrid system during which the architecture can be of the same or different type but coherent with respect to the hybrid system input and output. For example, a hybrid system architecture can be of type entangled (N→S→N) during the training phase, and of type sequential (N→S) during the operational phase, or sequential (N→S) during both phases.

The N-module model is the statistical model (Model:stat), which refers to an either initiated or already trained neural network (architecture, weights) together with hyperparameters, and auxiliary data for internal processing by the N-module. In the operational phase, the model is assumed to be intrinsic to the N-module but not part of its input.

The *N-module input* is data input (Data:In) for parametric function approximation, which refers to either training data or operational data of the neural network architecture. Training data includes all data including reward signals that is used to train the neural network (training or development phase). Operational data refers to input data for inferences by the trained network in applications (operational phase). Please note, that postprocessed S-module (symbolic) output can serve as additional data input for the N-module.

The *N-module output* is the data output (Data:Out) produced by the neural network either during its training or application (operating). In this context, data includes numbers (numerical measurements), numeric encoded text, tensors (multi-dimensional spaces, including bitmaps), streams (real-time sequences of data, including video and other sensor inputs) (van Bekkum, et al., 2021). That implies that, if required, the appropriate transformation from Symbol:Out of the S-module or any symbolic knowledge representation from the external to Data:In of the N-module happens at the N-module interface post-processing side; we do not explicitly indicate this transformation process in the design pattern of the N-module.

For the description of the S-module model, input, and output, we refer to Section 3.1.

4.1.2 Generic Attack Patterns for Hybrid Systems

In general, *attacks on a hybrid neuro-symbolic system* may *target* its input, output, and model either directly or indirectly and during the training or operational phase of the system. We focus on six types of hybrid system attack patterns with two variants each as indicated in Table 4.1 and briefly explained below.

Table 4.1: Generic attack pattern types for hybrid neuro-symbolic systems

	Data:In	Model:stat	Data:Out	Symbol:In	Model:sem	Symbol:Out
Manipulation of hybrid system model [MHM]		X			X	
Manipulation of hybrid system input [MHI]	X			X		
Manipulation of hybrid system output [MHO]			X			X
Theft of hybrid system model [THM]		X			X	
Theft of hybrid system input [THI]	X			X		
Theft of hybrid system output [THO]			X			X

Manipulation of hybrid system model [MHM]: We differentiate between several variants of this type of attacks for manipulating the hybrid system model, that is the set of models (Model:stat, Model:sem) of its N-module and S-module based on the considered system type being either sequential or entangled (cf. Section 4.1.1). These variants are indicated in Appendix B.1.

For each of those variants, we indicate *direct* or *indirect* model manipulation attack methods depending on their execution during the training or the operational phase of the hybrid system. This holds for all subsequently listed generic attack pattern types for hybrid systems. Please note, that during the operational phase of the hybrid system, the N-module model itself is assumed to be not modifiable through the system interface but the S-module model only.

Direct manipulation of the model of one module M1 of a hybrid system refers to (a) manipulation of the input inducing a change of the model and output of M1; this holds analogously for a self-loop from the output back to the input of M1. *Indirect manipulation* of a model of one module M1 refers to changes of the model that are implied by manipulations of the input, model, or output of the other module M2 of the hybrid system it is interacting with.

In other words, *direct model manipulation* attacks are targeting the model of only one module of the hybrid system by operating through the system interface directly on this model. For example, the parameters of the statistical model of a N-module can be changed during the training phase directly through the respective system interface operation. The same holds for the manipulation of the semantic model of some S-module for learning, reasoning, or planning in a hybrid system. Any direct attack on one model does not aim at and not necessarily lead to a change of the other model as well, though that might unintendedly happen.

Indirect model manipulation attacks may focus on the model of only one module, but with the ultimate purpose to implicitly change the main target of the attack, that is the model of the other module as well depending on its use of this manipulated and transformed output of the preceding module in the hybrid system. The attack of manipulating a neural network model (architecture, weights, parameters) during training may result in a network producing unintended output (Data:Out). Now, in case the N-module output is part of the hybrid system output, any N-module model manipulation attack can serve as a variant of manipulating the hybrid system output. In case the N-module precedes the S-module in the hybrid system data flow, the N-module output serves as transformed input (Symbol:In) of the subsequent S-module, hence can imply an unintended output (Symbol:Out) of the latter. Please note, that the goal of indirectly manipulating the S-module output through directly manipulating the N-module model and its output does not necessarily require or imply a change of the semantic model (Model:sem).

Model manipulation attacks for entangled system types have to take the internal I/O interaction between the modules (feedback loops) into account. That is, depending on the order of model-based processing in the hybrid system, a manipulation attack on the model of one module that produces the respective module output as feedback to the other module has to be specifically designed to be intendedly (on purpose) implicative for the feedback receiving (and not directly attacked) model to produce unintended output. The design of such indirect model manipulation attacks in entangled hybrid systems typically requires the highest degree of detailed knowledge about the module interactions and their processing.

Analogously, we also consider other generic types and variants of attack patterns for hybrid system input, output and model as listed below.

Manipulation of hybrid system output [MHO]: This type of attack refers to the manipulation of the output of the hybrid system produced by one of its S- and N-modules, through the interface of the hybrid system, or later on in some external output data store. The variants of [MHO] attacks are denoted [MHO-SEQ] and [MHO-ENT] for manipulation of the hybrid system output for system type SEQ, respectively, system type ENT.

Manipulation of hybrid system input [MHI]: This type of attack refers to the manipulation of the input of the N- and S-modules of the hybrid system via the hybrid system interface and depending on which of both modules appears first in the hybrid system workflow producing the hybrid system output. The variants of [MHI] attacks are denoted [MHI-SEQ] and [MHI-ENT] for the manipulation of the hybrid system input for system type SEQ, respectively, system type ENT. The [MHI] attack type can be a prerequisite for the attack type [MHM]; this is not the case for direct attacks on only the model without input manipulation such as in (Salem et al., 2020).

Theft of hybrid system model [THM]: This attack type refers to the theft of the hybrid system model, either all or only one of the models of the S- and N-modules through the interface of the hybrid system or the interface of some external model store or by intercepting the respective data communication between both or developer and system interface. This holds analogously for the other attack types related to the theft of the hybrid system input and output. The variants of [THM] attacks are denoted [THM-SEQ] and [THM-ENT] for the theft of the hybrid system model for system type SEQ, respectively, system type ENT. The remaining attack pattern types concerning the **theft of hybrid system input [THI]** and **output [THO]** have analogously denoted variants.

In the following sections, these generic attack patterns for hybrid neuro-symbolic methods will be related to prominent types of attacks of individual neuronal or symbolic AI method, and then applied to and described in more detail for each of the selected method classes.

4.1.3 Relationships to Prominent Attack Typs

Prominent types of attacks on individual neuronal AI methods are (1) evasion or adversarial attack, (2) data poisoning attack, (3) model stealing and privacy attack. We briefly indicate their general relationship to the generic attack patterns for hybrid systems introduced in the previous section.

Model stealing and privacy attacks on hybrid system. These attack types relate to the attack pattern [THM] and [THI] for the theft of the hybrid system model and input of the respective system modules. Model or input theft attacks are carried out to enable the subsequent extraction of either the model functionality or information on training data from the stolen model, as well as the extraction of information about the model functionality from a privacy violating surrogate model that is created from stolen or given input data for the original model and subsequently obtained output data such as reported in (Shen et al., 2022).

Adversarial attacks on hybrid system. This attack type refers to a *manipulation of the hybrid system output through some adversarial attack on N- and/or S-module inputs during the operational phase, without changing the respective model(s) of involved modules*. In this sense, the attack relates to variants of the generic attack pattern type [MHO] for hybrid system outputs depending on their sequential or entangled architecture *during the operational phase*.

In case of the attack variant [MHO-SEQ] SEQ = N→S, an adversarial attack [MHO]|N on the N-module through manipulation [MHI]|N of its input obviously affects the working of the dependent S-module such that it may produce unintended system output. Analogously, this also holds for adversarial attacks on the other types of hybrid systems with [MHO] variants [MHO-ENT] ENT = N→S→N, [MHO-ENT] ENT = S→N→S, and [MHO-SEQ] SEQ = S→N. But now the attacker must also take the additional, unchanged input from the S-module into account for any manipulation of the N-module input.

However, adversarial attacks on S-modules are limited. For example, an adversarial attack on an automated action planner through changing its input to enforce unintended plans (actions) as its output, *while leaving its internal planning model unchanged*, is not possible. The reason is that the planner input (domain and problem) is initially part of the internal planning model used by the planner to produce the output. Moreover, the internal planning structures created during planning such as incrementally created belief and action policy trees are part of the internal planning model and may change at any time online or whenever the domain and problem descriptions change. Besides, adaptive S-modules such as those with static models during operation like dynamic belief networks are trained but remain prone to adversarial attacks when the attack provides rare input feature data which the developer did not check when testing the S-module.

In general, compared to adversarial attacks on individual deep learning systems, to design and run the same kind of attack on a hybrid system requires typically more knowledge and skills due to the inherently hybrid nature of the system. In fact, to design adversarial attacks for such systems with [MHO] attack pattern variants as indicated above, requires knowledge about the N- and S-module model, input, and output, as well as the architecture type of and respective data dependencies within the overall system. Though, for evasion or adversarial attacks the attacker does not necessarily have to have stolen all models and I/O data, i.e., there does not have to be an absolute white-box scenario for an evasion attack to be possible. For example, as mentioned above, the creation of a surrogate model for approximating the targeted model of the N-module only requires obtaining its input data.

In an absolute white-box scenario for evasion attacks on a sequential hybrid system, the attacker has not only to steal (a) the N-module model together with the benchmark I/O data used for the N-module training via attacks [THM]|N, [THI]|N, [THO]|N, but also (b) the S-module model with used I/O data offline or online via the hybrid system interface with attacks [THM]|S, [THI]|S, [THO]|S. For the design of adversarial examples for the hybrid system, the attacker then may attempt to locally simulate manipulation of hybrid input attacks [MHI]|N on the N-module with assumed internal data flow between modules without changing anything but the N-module input. In case of entangled hybrid systems, adversarial attacks become even more difficult for an attacker to design without access to the system code for local simulations in preparation of real and targeted attacks. However, in general, there might be cases where the necessary knowledge can be guessed from the concrete application context, e.g., if there are only few standard models for a certain problem, which are frequently used by the community in a transfer learning setup.

Data poisoning attacks on hybrid system. This attack type refers to the *manipulation of the training data of the N- and/or S-module model during the training phase through which in the following operational phase the trained hybrid system reacts to (specific) inputs with an output not intended by the developer*. In this sense, the attack relates to variants of the generic attack pattern type [MHO] for hybrid system outputs depending on their sequential or entangled architecture *during the training phase*.

While for deep learning systems these attacks are generally hard to detect post-hoc due to the large amount of data used to train and the lack of transparency, this in principle also holds for the training of N-modules in hybrid systems with an additional dependent S-module. But, again, due to the fact that in hybrid systems two different kind of models (sub-symbolic and symbolic) are used in both the training and operational process, it requires more effort from the attacker to understand the internal I/O dependencies between N- and S-module (independent from the hybrid system architecture being sequential or entangled) to design data poisoning attacks during the training phase, or adversarial attacks during the operational phase. Of course, like with any other AI system, oracle attacks on any type of hybrid AI system where only its external I/O data streams are monitored (during training or operational phase) but no model is stolen or manipulated are also possible.

4.2 Learning Intermediate Abstractions for Reasoning or Planning

4.2.1 Description and Methods

In this category of the hybrid AI system, the N-module is trained to infer from raw data an intermediate abstraction in symbolic representation, which then serves as an input for symbolic reasoning or planning method of the S-module that logically deduces the final symbolic result of the overall system. Such intermediate abstraction is the N-module data output (Data:Out) taken as an input (Symbol:In) semantically interpreted and represented by the S-module for its symbolic inference. In this sense, a neural learner supports a symbolic reasoner or planner which produces the hybrid system output.

During the *training phase* of the hybrid system, the statistical model (Model:stat) of the N-module (i.e., the neural network weights) is learned with raw input data (Data:In), while the semantic model (Model:sem) of the symbolic reasoning or planning method of the S-module is generated with given symbolic input knowledge such as (observed and pre-processed) facts, concepts, or rules. In case of planning, the semantic model covers the internal planning model including the domain and problem models given by human actors as symbolic input. The S-module output can be used during the training of the N-module model as a feedback or reward signal embedded into the training input Data:In. In this sense, during training of the Model:stat both modules would operate entangled, but do not have to. The generic design pattern of this hybrid system category during training is shown in Figure 4.2.

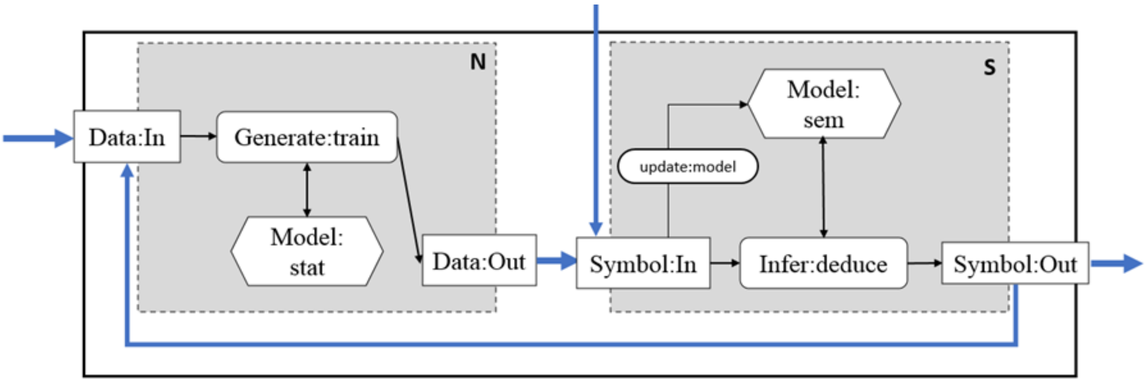


Figure 4.2: Generic design pattern of hybrid AI methods for learning an intermediate abstraction for reasoning or planning in the training phase, and in special cases the operational phase

During the *operational phase* of the hybrid system, its N-module uses a previously learned statistical model for its inference (prediction) for given input data with an output that is then interpreted by the S-module as an intermediate abstraction of knowledge that is subsequently leveraged for reasoning or planning based on its given semantic model.

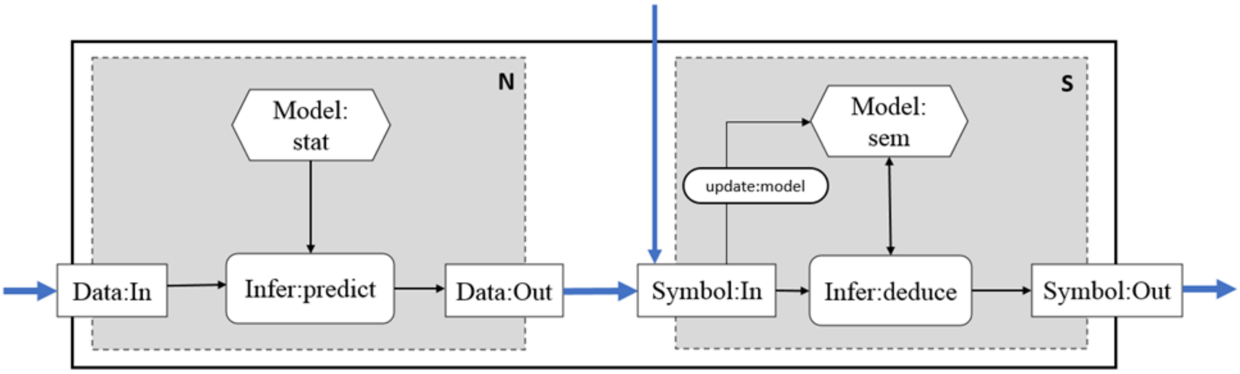


Figure 4.3: Generic design pattern of hybrid AI methods for learning intermediate abstractions for reasoning or planning in the operational phase

In this sense, the hybrid system of this category usually operates in a sequential way, where the N-module supports the S-module in its production of the hybrid system output, though both may receive external input through the hybrid system interface (cf. Figure 4.3). Please note, however, that we allow for special cases of methods of this category for which the training and operational architectures do not differ. In such cases, the S-module always generates (part of the) input of the N-module such that the system operates entangled in both training and operational phases.

Please also note, that we assume that the trained model of the N-module cannot be (ex-) changed during the operational phase (cf. Section 4.1.1). Similarly, for planning, the domain and problem model parts of the internal planning model (semantic model) are externally provided (as part of Symbol:In, cf. Section 4.1.1) for different domains and tasks to be pursued by the system. Depending on the type of online or offline planning or reasoning, the S-module may update its model during operation or not.

In the following section, we describe one representative instance of this category of hybrid methods, that is HyLEAP (Pusse, et al., 2019), and indicate other methods of this category as well. This is followed by a discussion of security aspects of the selected representative method HyLEAP in general (in Section 4.2.3) and what do they mean in a practical use case of HyLEAP (in Section 4.2.4). A discussion of our findings of security aspects for this method category is included in Section 4.2.3.

4.2.2 Example Methods

We first describe and classify HyLEAP in relative detail that is necessary to understand the possible attacks on HyLEAP, and then summarize other hybrid methods as further examples of this hybrid method category. Later on, more practical, and detailed information about HyLEAP is provided.

HyLEAP is an experience-based online planner for POMDPs. It combines an actor-critic deep reinforcement learner in the N-module with an online approximated POMDP planner IS-DESPOT in the S-module to solve problems modelled as a POMDP (partially observable Markov decision process). In short, the online planner is assisted by the learner for experience-based look-ahead action planning at each time step. For determining action alternatives, the planer uses belief trees. The learner is trained off-policy and entangled with the online planner. The job of the learner is to evaluate the utility of simulated action path alternatives in the belief tree of the planner more realistically based on experience instead of some pre-defined heuristic (lower and upper bound) values.

In particular, for each description of a terminal belief (leaf) state in the belief tree reachable with a n-step look-ahead planned action path the N-module receives from the S-module to evaluate, it uses its model (Model:stat) to produce and then return an estimated state value (Data:Out) to the S-module. The S-module, in turn, interprets these values as *intermediate abstractions* (Symbol:In) of an heuristic upper bound of optimal utilities (rewards) for the explored belief states. In fact, the planner leverages the neural state estimates to recursively determine the lower and upper bound values at each belief node in a post-order traversal on the belief tree to the root belief. Once the gap between the lower and upper bound values at the root belief in the tree reaches a target level (after a series of network guided explorations in the incremental belief tree construction) or time runs out, the root action with highest value lower bound is finally selected for execution in the environment. This holds for both operational and training phase.

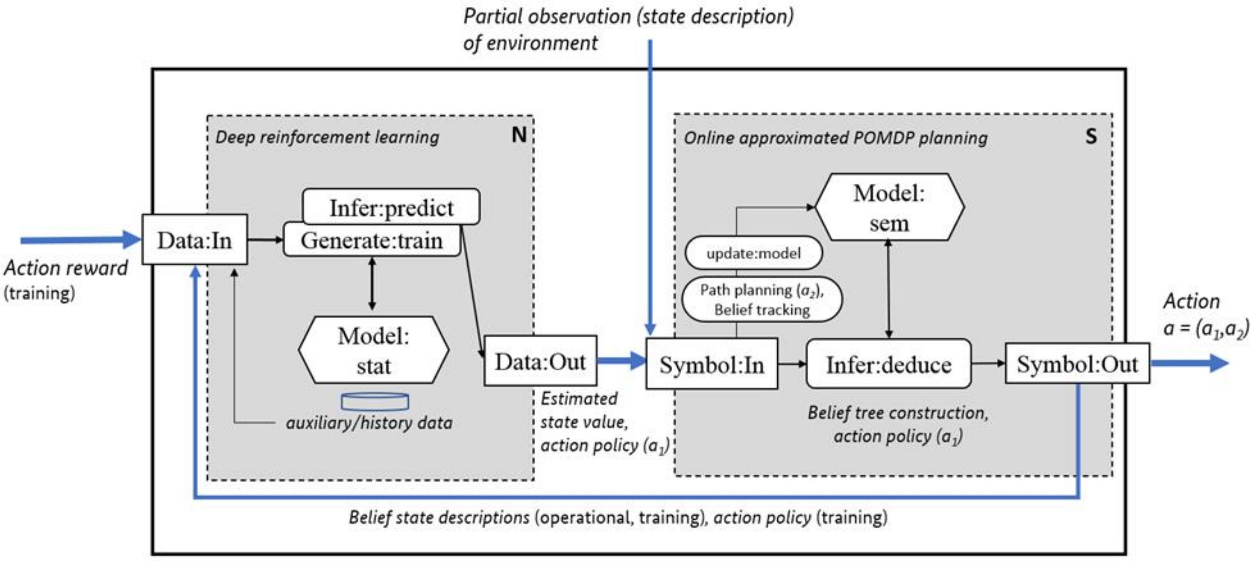

Partial observation (state description) of environment
Deep reinforcement learning
N
Online approximated POMDP planning
S
Action reward (training)
Data:In
Infer:predict
Generate:train
Model: stat
auxiliary/history data
Data:Out
Estimated state value, action policy (a_1)
Model: sem
update:model
Path planning (a_2), Belief tracking
Symbol:In
Infer:deduce
Symbol:Out
Belief tree construction, action policy (a_1)
Action $a = (a_1,a_2)$
Belief state descriptions (operational, training), action policy (training)

Figure 4.4: Generic design pattern of example method HyLEAP

shows the specific instance of the generic design pattern of the considered hybrid method category (in Figure 4.2) for HyLEAP. Please note that in HyLEAP, the N-module model (Model:stat) comprises the deep reinforcement learning network model together with auxiliary data such as history of rewards, state descriptions (observations) and action policies. The S-module model (Model:sem) includes the currently constructed belief tree and action policy of the online planner, and auxiliary data used for internal planning such as lower and upper bounds of estimated belief values (utilities, rewards) at each node of the tree. The N-module input (Data:In) includes rewards from the environment, descriptions (observations) of current terminal (belief) states of look-ahead action plans and (only during training) action policies from the S-module, and other auxiliary input from the N-module model such as last action and history. The S-module output (Symbol:Out) includes the action to be executed in the environment, and the above mentioned data from the S-module model (belief state descriptions, action policy) to serve as input for the N-module. The S-module input (Symbol:In) comprises descriptions of partially observed environmental states and the estimated (reward) values produced by the N-module. Note, that the N-module itself does not receive any state description from the environment but from the S-module, namely those, the learner is supposed to (learn to) evaluate on request of the online planner.

In HyLEAP, an *action a = (a_1, a_2)* consists of two sub-actions a_1 and a_2. The sub-action a_2 for currently tracked (root) belief state for a point in time of interaction with environment is computed with a path planner as part of the S-module. This sub-action is used together with input from the environment and respectively updated model (Model:sem) by the POMDP planner IS-DESPOT to determine (Infer:deduce) sub-action a_1 guided by its evaluation input from the N-module, which eventually yields the full action *a* as final output of the hybrid system.

During the *training phase*, the N-module model (Model:stat) is trained on input data (Data:In) with a cross-entropy loss function that checks the action policies produced by both learner and planner together with estimated state values and real rewards from the environment. The network model gets updated only at the end of each training scene based on gradient descent over the accumulation of loss function gradients for each scene time point. During training scene simulation, the network model does not change and is used to (feed-forward) generate the estimated state values for the online planner.

During the *operational phase*, at each point in time of operating in the environment, the trained learner uses its model to evaluate the simulated action alternatives during the belief tree constructions of the planner.

Note, that unlike in the training phase, the N-module does not receive the action policy from the S-module and reward from the environment but only the state descriptions during its belief tree constructions for evaluation.

To summarize: For each observation at given point in time in both training and operational phases, the online planner (a) iteratively constructs a belief tree for n-step look-ahead planned (simulated) action alternatives that are evaluated by the learner, (b) selects the action at the root with highest lower bound as an output of the hybrid system for execution, (c) finally destroys the belief tree, and (d) restarts this process for the next point in time of interaction with the environment. As consequence, *HyLEAP classifies as a hybrid system with architecture of type entangled in both training and operational phases*.

Other example methods of this category of hybrid neuro-symbolic methods are summarized in the following. In the entangled hybrid system AlphaGO (Silver, et al., 2017) a deep reinforcement learner (N-module), similar to HyLEAP, gets trained to best evaluate and assist a Monte-Carlo-Tree-Search based action planning process (S-module) restricted to MDPs (not POMDPs) and optimal Go game playing. Micheli et al. (2021) extend the temporal logic planner TAMER* (Valentini, et al., 2020) with a deep reinforcement learner for solving MDPs in a hybrid system TAMER* during both training and operational phase. The trained neural learner (N-module) assists the planner (S-module) through the construction of a value function which can be interpreted as an intermediate abstraction of and transformed into a planning heuristic (function) for search exploration by the planner TAMER*. The N-module takes planning instances, i.e., domain and problem including deadlines, as input (Data:In) and learns to synthesize an estimated optimal value function as output (Data:Out). This output corresponds to an intermediate abstraction of the approximated optimal distance heuristic value used by the S-module (Symbol:In) during its action planning for given planning instance. TAMER* is sequential (SEQ = N→S) during operation, since TAMER* uses the trained neural network for given planning instance, and in an entangled mode (ENT = N→S→N), since TAMER* used as planning environment simulator provides feedback to the N-module during its episodic training. The planner in TAMER* has been experimentally shown to benefit from learned insights on given planning instances to solve temporal planning problems better than with reinforcement learning and planning alone.

Other work on neural learning for classical planning is relatively rare. Ferber et al. (2021) suggests a feed-forward neural network (N-module) that takes as input (Data:In) only states (no goal, no object universe) of given planning instances to learn the parameter values (Data:Out) as intermediate abstractions that define the neural network approximated search heuristic functions. As in TAMER*, these heuristic functions could then be used by a classical planner (S-module) in a hybrid system. O'Toole et al. (2022) proposes regression-based supervised learning of the neural network to select relevant sets of planning states at a range of different distances from the given planning goal. The N-module is trained to produce parameter values that define the search heuristic of the planner based on selected states labelled with exact or estimated distances to goal states and tested over a set of different initial world states of planning. Speck et al. (2021) propose reinforcement learning of dynamic configuration of search heuristic parameters as output to be taken as input by a planner (S-module). This is done per instance and based on the planning states of the S-module taken as input by the learner. This would allow a portfolio planner to select a particular heuristic depending on the current instance before solving it without depending on the current time step or the internal state of the planner. The reward as additional input to the learner comes as feedback from the S-module, which makes the hybrid system entangled, though not neuro-symbolic for which a deep reinforcement learner as N-module might be considered.

In addition to the above type of methods, there are a few methods of neural learning of intermediate abstractions for learning, where the neural learner (N-module) supports another but symbolic learner (S-module) such as in Manhaeve et al. (2018), Toro Icarte et al. (2018), and Garnelo et al. (2016). For example, Garnelo et al. (2016) present a sequential hybrid system (SEQ = N→S) for the purpose of so-called deep symbolic reinforcement learning. In other words, the system performs neural learning of intermediate abstractions for symbolic learning through the combination of a convolutional autoencoder as N-module with a symbolic reinforcement learner as S-module. Roughly said, the N-module learns to map highdimensional input (Data:In) such as raw pixel data of an observation image to encoded lower-dimensional and more abstract (state) representations (Data:Out), which are intermediate abstractions of states describing type and positions of detected objects. The S-module transforms and processes this input (Symbol:In) from the N-module into symbolic spatial temporal representations of states and then applies reinforcement learning to map them to an optimal action (Symbol:Out) based on rewards it receives as additional input.

4.2.3 Security Aspects

This section discusses attack patterns for this type of hybrid systems exemplified for the selected, representative method HyLEAP (cf. Section 4.2.2). This is followed by a very short summary of security aspects of other example methods and discussion.

Attack patterns for hybrid system HyLEAP. In both training and operational phases, the HyLEAP system input from the environment (or simulator) is provided through the system interface to both the (trained) learner and online planner with the latter producing the hybrid system output. As mentioned above, during the *training* phase, HyLEAP is of type *entangled* (cf. Figure 4.1: hybrid system type ENT:N→S→N) with the online planner producing the hybrid system output, that is the optimal action to be executed at each time of the partially observed environment (scene simulation). The component of HyLEAP to be trained is the N-module, that is the off-policy actor-critic deep reinforcement learner, which is coupled with the online POMDP action planner IS-DESPOT. While the N-module model of HyLEAP is trainable, the S-module model of HyLEAP is not. In this context, the *relevant attack variants* for the training phase are concerned with the manipulation and theft of the HyLEAP model [MHM-ENT, THM-ENT], HyLEAP input [MHI-ENT, THI-ENT], and HyLEAP output [MHO-ENT, THO-ENT].

During the *operational* phase, HyLEAP is of type *entangled* as well. The S-module requests the N-module to evaluate its online planned action alternatives upon which the N-module returns action policy values (rewards) to the planner that allow for final action selection and execution. The HyLEAP output is not produced by the N-module but the S-module, while the latter always produces part of the input of the N-module (during the belief tree construction with simulated observations and actions). Thus, the *relevant attack variants* for the operational phase are concerned with the manipulation of the HyLEAP model [MHM-ENT, THM-ENT], HyLEAP input [MHI-ENT, THI-ENT], and HyLEAP output [MHO-ENT, THO-ENT]. They differ from those in the training phase only in that, by assumption, the N-module does not change, that is, any hybrid model manipulation attack [MHM] is restricted to the S-module ([MHM]|S).

In HyLEAP, both neural and planning models can be *directly* and *indirectly* manipulated via the hybrid system interface. As mentioned above, *direct manipulation* of a model of one module refers to (a) manipulation of the input inducing a change of the model and output of the module; this holds analogously for a self-loop from the output back to the input of the module. *Indirect manipulation* of a model of one module refers to changes of the model that are implied by manipulations of the input, model, or output of the other module it is interacting with. For example, indirect manipulations in HyLEAP are possible because of (a) the dependency induced by the feedback loop (S→N) for training the neural model (Model:stat) with a loss function that also includes the action policy value of the planner (S-module output), and (b), vice versa (N→S), the action policy value of the learner being used by the planner to (heuristically) drive its final selection of best (root) action in the constructed belief tree (and the corresponding policy tree derived from the latter).

Attack Pattern Name: **[MHM-ENT]** *Manipulation/hybrid system model* for system type ENT,

Case: ENT = N→S→N; **Category instance: HyLEAP** (cf. Sect. 4.2.2)

Description: The attacker manipulates the hybrid model (of HyLEAP), that are the N-module model and/or the S-module model either directly or indirectly through the hybrid system interface.

Methods of Attack:

1. The N-module model (Model:stat) can be *directly* manipulated *during training* with CRUD (create, update, read, delete) operations on (a) any part of the *model itself* such as network weights, replay buffer, and hyperparameters, and (b) the N-module *input from the environment*, that are the rewards used for learning. For this purpose, there is no need to manipulate the input part (Data:In) the N-module receives from the S-module.
2. The N-module model (Model:stat) can be *indirectly* manipulated *during training* through the change of the N-module *input from the S-module*. The N-module input (Data:In) that is received as part of the S-module output (Symbol:Out) are: (1) the belief state descriptions (observed from environment as part of Symbol:In, or simulated during belief tree construction and stored in the S-module model Model:sem), and (2) the action policy of the planner used in the cross-entropy loss function for off-policy learning. Any such belief state description can be changed through direct manipulation of the respective part of either (a) the S-module input (Symbol:In) from the environment, or (b) the updated S-module model (Model:sem), or (c) the S-module output (Symbol:Out). The action policy of the planner stored in the S-module model and communicated to the learner during training can be changed through direct manipulation of the respective part of either (a) the S-module output (Symbol:Out), or (b) the S-module model.
3. The S-module model (Model:sem) can be *directly* manipulated *during training and operational* phases through the hybrid system interface with CRUD operations on (a) any part of the *model itself* such as the current belief tree and sub-action a_2, observed and simulated belief state descriptions in the tree, and initial lower and upper bounds for heuristic evaluation, and (b) the belief state descriptions received as *input from the environment*.
4. The S-module model (Model:sem) can be *indirectly* manipulated *during training and operational* phases through the hybrid system interface with changes of the part of the *input from the N-module*. This part concerns the estimated state values of the N-module output (Data:Out) that are passed to the S-module as input (Symbol:In), interpreted as heuristic upper bound utility of a reached (real or simulated) state during belief tree constructions (planning) and stored in the semantic model (Model:sem). Any change in such heuristic evaluation can misguide the belief tree construction, therefore lead to final selection of less optimal actions to execute in the environment.

Attack Motivation-Consequences: The indirect manipulation of the neural model with manipulated state descriptions or wrong action policies (Methods 2.1, 2.2) during the training phase could yield a learner which support of the online action planner in the operational phase then would become a disaster. The same holds for the direct manipulation (Methods 1.a, 1.b) such as through some "brute force" replacement of the whole network at the end of the training phase, or with a change of the rewards from the environment in favour of more preferred action policies for trained original or appropriately changed state descriptions. The motivation of model attacks could be either purely destructive – the overall hybrid system shall perform badly or not at all, or manipulative in the sense that the system performs action planning that is biased towards own or third-party interests.

Attacker Skill or Knowledge Required: All types of model attacks require quite an in-depth knowledge on the concrete interaction between the modules and the respective part of the hybrid system interface to use for model manipulations. Besides, the attacker must understand not only the online action planning process carried out by the approximate POMDP planner DESPOT (cf. Section 3.3.2.1), the details of (syntax and semantics of) the internal planning model and the belief tracker, but also the way of how the deep reinforcement learner is integrated into this process. As with attacking planner or learner individually, skills for an undetected interception of potentially encrypted communication between the environment (e.g., driving simulator) or developer's data store(s) and the hybrid system interface are also required.

Resources Required: The attacker requires appropriate tools for an interception of communication between developer or operator data store or execution controller (environment) with the system and remote illegal usage of the hybrid system interface.

Solutions and Mitigations: The usage of the hybrid system interface shall be secured by appropriate authentication and cryptographic means, and trustworthy input data sources used. Besides, the fact that comparatively immense in-depth knowledge and skills are required from the attacker to design specific targeted attacks of HyLEAP might be considered as their inherent mitigation (cf. Section 4.2.4).

Attack Pattern Name: [MHO-ENT] *Manipulation/hybrid system output* for system type ENT,

Case: ENT = N→S→N; **Category instance: HyLEAP**

Methods of Attack: The hybrid system output (Symbol:Out) is the action policy produced by the planner (S-module), which can be both *directly* and *indirectly* manipulated. In the first case during both training and operational phases, the attacker can use the hybrid system interface to monitor and change the action policy before it is being executed in the environment (execution controller). Due to the feedback-loop between the modules, the hybrid system output can also be *indirectly* manipulated through manipulations of the N-module output, model, or input. For example, during the training phase, the attacker can use the hybrid system interface to directly manipulate the N-module input ([MHI-ENT]|N, [MHI-SEQ]|N) from the environment (rewards) such that both the N-module model and its produced N-module outputs (i.e., the action policy and state value) are directly affected and may significantly change (like in data poisoning). This holds analogously during the operational phase, where the N-module model is not changed but its input such that the neural network output is changed (like in adversarial attacks). Any change of N-module outputs, however, may cause the action policy of the planner to change as well (since this manipulated output is passed as evaluation of plan alternatives during the belief tree construction of planning). As consequence, the internal planning model and thereby potentially the action policy as S-module output of the system is indirectly manipulated.

Attack Pattern Name: [MHI-ENT] *Manipulation/hybrid system input* for system type ENT,

Case: ENT = N→S→N; **Category instance: HyLEAP**

Methods of Attack: The hybrid system input (Data:In, Symbol:In) can be directly manipulated through the respective functional part of the hybrid system interface during training and operational phases. During the *training phase*, the *direct* manipulation of the N-module input ([MHI]|N) - in terms of changing rewards from the environment or the state descriptions and action policy from the S-module - may also *directly* change the neural model and its generated output like in data poisoning attacks. Due to the hybrid nature of the system, any changed neural output may also *indirectly* change the S-module input during belief tree construction (internal planning model), hence *indirectly* the hybrid system output (cf. [MHO-ENT]). During the *operational phase*, like in adversarial attacks, the *direct* manipulation of the neural input may change the output, without changing the model; as consequence, the model and output of the hybrid system may be *indirectly* manipulated as well (cf. [MHM-ENT], [MHO-ENT]). One major challenge of such attacks on the neural module in the entangled hybrid system is that the input part coming from the S-module must be considered in relation as well. That is, methods for data poisoning and adversarial attacks of the neural module must be adapted accordingly. That requires the attacker to have in-depth knowledge and skills to understand the production of the S-module output which is passed as input to the N-module. In this regard, please note that the input from the S-module is different in both phases (cf. Figure 4.4).

Attack patterns [THM, THI, THO] for HyLEAP. The attack methods for *stealing* the hybrid model, input, or output of HyLEAP during training or operational phases are concerned with the interception of communication with and the illegal usage of the respective functional parts of the hybrid system interface (cf. Section 4.1.2). The goal of these attacks is to steal intellectual property or to prepare manipulation attacks described above.

Attack patterns for other example methods. In general, the above-mentioned generic attack patterns for the category instance HyLEAP also apply to the other example methods of this category summarized in Section 2.2 but need to be properly adjusted to the specific input, output, and models of N- and S-modules in detail. For example, the hybrid method TAMER* (Micheli, et al., 2021) uses a supervised feed-forward network not a deep reinforcement learner as N-module, an offline temporal logic planner for MDPs not an online approximated planner for POMDPs and is not of type entangled but sequential during the operational phase. However, the generic attack methods for model, input, output manipulation and theft remain the same in principle (cf. Section 4.1.2) but with slightly less effort required, since the feedback between both modules does not need to be considered during operation but training phase only, and the offline planning restricts the relevant part of the hybrid system interface to be used for an attack. The other methods for neural learning in support of classical planning are prone to the type of generic attack methods on N-modules alone such as adversarial and model stealing attacks during the operational phase, and data poisoning during training.

Discussion. Hybrid neuro-symbolic methods for *learning intermediate abstractions for reasoning or planning* can be directly and indirectly manipulated with more kinds of attacks than their individual neuronal and symbolic AI components. In this regard, any direct or indirect change of the intermediate abstractions produced by the N-module for further use by the S-module in sequential or entangled order is of importance. Due to the functional dependency of the S-module on these abstractions as input, any manipulative attack (adversarial, data poisoning) on the individual N-module can lead to an indirect manipulation of the output of the overall system without any direct attack on the input or the model of the S-module through the hybrid system interface.

On the other hand, in entangled systems of this type, the intermediate abstractions can also be indirectly manipulated through direct manipulations of the S-module output (used in the cross-entropy loss function) during training of the N-module. In this regard, a hybrid method of this type appears not more robust against attacks than its individual N-module.

However, the hybrid nature of such methods can inherently mitigate these manipulation attacks that aim at system outcomes unintended by the developer. In general, the skills and knowledge required to design and conduct these attacks is comparatively higher with a possibly steeper learning curve. In particular, the meaning and use of intermediate abstractions by the S-module has to be understood in detail. Of course, that does not hold for any “brute force” trial-error manipulation attacks through the hybrid system interface with mere intention to hamper the system during training or operation no matter what.

Moreover, the use of symbolic AI methods for reasoning or planning in the S-module allows to explain the decision or action policy making of the overall hybrid system easier than the intermediate abstraction generated by the neuronal method in the N-module. Roughly speaking, such types of hybrid neuro-symbolic methods become not more explainable than their individual N- or S-modules.

4.2.4 Use-Cases

What do the general security aspects of HyLEAP mean in practical use cases? Let us first provide an example of how HyLEAP can be applied in practice, that is the collision-free navigation of self-driving cars in critical, hence virtually simulated traffic scenes based on the major German in-depth accident study GIDAS.

HyLEAP in practice. The practical application of HyLEAP is to navigate on a drivable way to a given goal with minimal number of collisions and in minimal amount of time. This optimization problem can be modelled as a partially observable Markov decision problem (POMDP) and solved in general with either (deep) reinforcement learning or approximate online POMDP planning or a combination of thereof such as with HyLEAP. The technical implementation of HyLEAP is currently interacting with the virtual environment, that is the open-source driving simulator OpenDS (Pusse, et al., 2019). From this environment and based on the underlying car and pedestrian model for traffic scene simulations, HyLEAP receives (a) partial state descriptions (observations) including the full car state (position, speed, orientation) and the partial pedestrian states (only actual positions but not orientation, predicted positions, goal, and speed), (b) the current 2D cost map of the scene, and (c) rewards for executing car driving actions each with sub-actions for steering and velocity of the car.. HyLEAP has not been used on-board of a real automated vehicle yet.

Security aspects of HyLEAP in practice. There are numerous ways to malevolently attack this hybrid system for collision-free navigation as described in generic terms and independent from the use case in the previous section. in HyLEAP, the manipulation of the hybrid system input (cf. [MHI-ENT]) can be used to manipulate the hybrid system model directly or indirectly (cf. [MHM-ENT]), which, in turn, eventually leads to an indirect manipulation of the hybrid system output (cf. [MHO-ENT]). Of course, the model of the deep reinforcement learner might also be directly manipulated during training without any input manipulation. In general, HyLEAP itself can be directly and indirectly manipulated with more kinds of attacks than its individual N- and S-module.

For example, one kind of indirect manipulation attack on HyLEAP as a system is to "just" directly manipulate the observations (state descriptions) the planner (S-module) receives from the environment (cf. [MHI-ENT]) as an input (Symbol:In) via the hybrid system interface. The rest goes through the system without any need of additional attack activities: The manipulated S-module input could be wrong indications of minimal or no costs of driving in no-go areas like sidewalks, and wrong or deleted positions of pedestrians on the street. That, in turn, can lead to a direct manipulation of the planner output being passed to the N-module during the belief tree construction. As consequence, the neural learner (N-module) model gets indirectly manipulated through the S-module during training in the sense that it now gets trained to evaluate the look-ahead simulated action alternatives of the planner for these manipulated scenes rather than for the original critical traffic scenes the planner was intended to safely act in. Besides, a rather brute force attack on the neural model is to directly replace it with a pretrained one for malevolent purpose at the end of the training phase.

In addition, the attacker could attempt to only change intercepted reward signals of the driving simulation environment for the N-module (without changing the planning heuristics) during training to influence the output of the hybrid system, that are its driving actions. For example, a malevolent manipulation of reward signals such that crashes instead of their avoidance are rewarded directly manipulates the neural model during training. As consequence, the use of such manipulated neural model for the experience-based (upper bound) evaluation of the action alternatives planned by the S-module can lead to a comparatively restricted performance of the overall driving system based on the (unchanged) lower bound value of the planner only. The safety of driving may decrease compared to that of HyLEAP without such manipulation but not worse than that of the individual online planner with equal lower and upper bound evaluation heuristics. In this sense, the S-module processing itself helps to restrict the effect of indirect attacks on the hybrid system through manipulation attacks on the neural input, hence neural model, and output, to only conservative, not optimal collision-free navigation - unless the attacker does not appropriately also change the planning model, that is the heuristic lower bound and default policy at the same time.

Moreover, the hybrid nature of this method would require an attacker to invest comparably more efforts and skills to understand the details of the entangled interaction between the N-module and the S-module, and the used (evaluation) heuristics for online planning in compliance with the reward scheme for the learner. On top of that, the attacker needs to get acquainted with the respectively complex hybrid system interface for how to best conduct attacks compared to, for example, a deep learning system alone.

For example, for entering manipulated observations into HyLEAP, the attacker needs to know about the details of either how to provide input to the S-module and not (as usual in deep learning approaches only) the N-module, or the replacement of correct observations the N-module receives from the S-module with the manipulated ones in the internal interaction pipeline between both modules via the hybrid system interface. However, in both cases, the attacker needs to know about the dependencies between used parameters of learning and planning in the hybrid system; we consider this possible but with a very steep learning curve.

On the other hand, the action selection of HyLEAP by the online planner in the S-module is explainable to the extent its belief- and action policy tree construction with symbolic knowledge representation is retraceable and human-interpretable by experts. This appears to be easier than to post-hoc explain the experience-based heuristic values (intermediate abstractions for planning in the S-module) generated by the deep reinforcement learner in the N-module with XAI (explainable AI) methods such as layer-wise relevance propagation.

4.3 Informed Learning with Prior Knowledge

4.3.1 Description

In these hybrid AI systems, a given semantic model is used by a S-module to logically infer some symbolic output, which then serves the N-module as a kind of semantic prior (appropriately encoded) for its respectively more informed learning process. In particular, the intermediate symbolic input together with raw input data are used by the neural network to generate a statistical model with which it eventually can infer a post-hoc symbolically encoded output from given raw input data. In other words, a semantical model (Model:sem) allows the hybrid system to infer information that supports the neuronal learning process in generating more sophisticated statistical models (Model:stat). In this regard, the intermediate symbolic knowledge representation of the S-module guides or informs the learning of the N-module (cf. Figure 4.5).

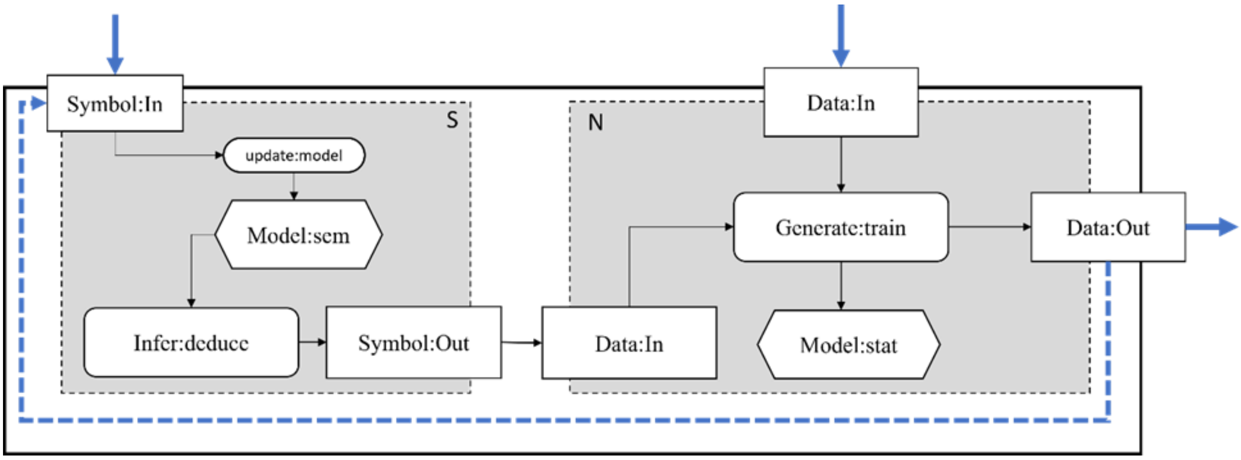


Figure 4.5: Generic design pattern for hybrid AI methods of informed learning with prior knowledge

In a simplified version of this pattern, the semantic prior might also be just given by the symbolic AI module in form of an explicit knowledge base to enable the neural learner to perform better on the abstracted symbolic data than on the raw data. Such a knowledge base usually consists of semantic networks or ontologies that restrict potential data output of the N-module. More complex, entangled versions of this category of hybrid methods involve a feedback from the output (Data:Out) of the N-module into the S-module such that the latter is able to derive consequences in a more goal-directed manner (as illustrated in Figure 4.5 by the dashed line) or checking logical consistency with symbolically represented (task, domain, common-sense) knowledge. This is particularly useful in case earlier behaviour of the N-module can be utilized to further strengthen the S-module's semantical model. During the operational phase prior knowledge may also help the neural network to arrive at more sophisticated or educated answers (cf. Figure 4.6).

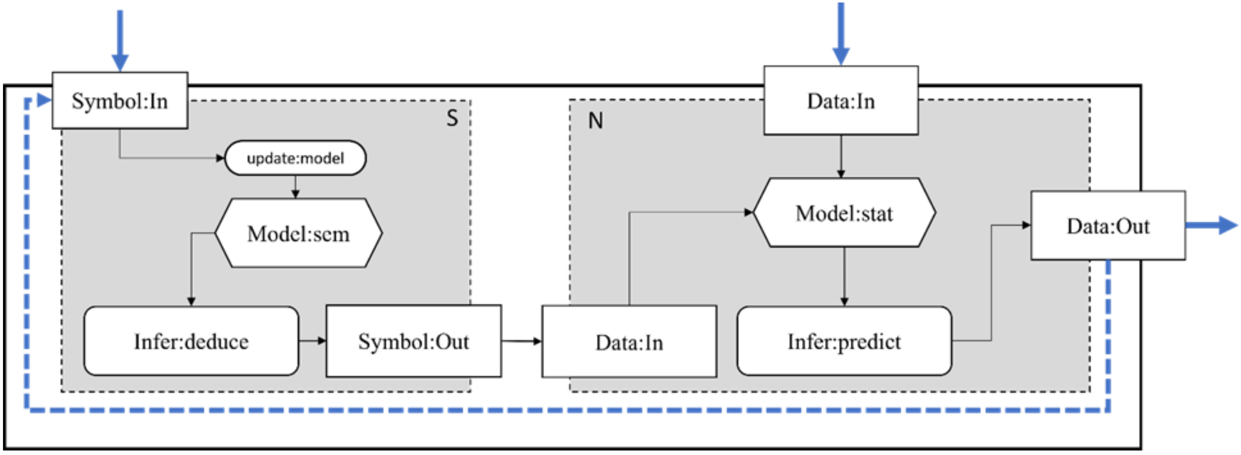


Figure 4.6: Generic design pattern of hybrid AI methods for informed learning with prior knowledge during operational phase

In analogy to the pure *training phase* there exists an entangled version of this design pattern where the N-module's output serves as an additional input for the S-module that enables the latter to infer a more goal-directed guidance for the statistical model.

4.3.2 Example Methods

Semantical models and their related inference mechanisms appear in various forms and sizes. Typical examples are logical formulas (SAT, classical, modal, temporal, etc.) together with appropriate deduction techniques, rule-based systems, or ontologies. Also, probabilistic and/or non-monotonic reasoning may come into play here.

In this section, we describe a selected, representative hybrid method, called CrisGen of this category. This will emphasize on the synthesis of critical driving simulator scenarios for an autonomous car, where the scenario generation is entirely built upon classical reasoning mechanisms. The training of autonomous cars to avoid collisions is usually supported by suitable driving simulators, mainly because real-world experiments are expensive and potentially dangerous, at least as long as the training process is in its early or mid-term stages. One of the key aspects of utilising driving simulators is to define suitable scenarios (street geometry, behaviour of pedestrians and other traffic participants) that cover possibly all aspects of driving situations an autonomous driver might get involved in.

CriSGen (Nonnengart, et al., 2019) is a generator for such scenarios in the sense that it produces *critical* small variants of already existing scenarios - in which the autonomous car behaved well - that nevertheless enforce the car to react differently to avoid an accident. Thus, we assume to have symbolic knowledge that consists of facts together with logical inference mechanisms that allow us to infer additional knowledge such that this intermediate symbolic input together with raw input data is utilized by the neural network to generate a statistical model that comprises both inputs. The symbolic knowledge therefore allows us to guide the learning mechanism towards a direction that is determined by knowledge beyond the raw input data. During the training phase the neural module's symbolic output is then compared to the assessment mechanism of the symbolic module. Depending on this assessment, the neural module either must adjust its internal weights or the symbolic module can make use of this output to generate additional knowledge in a goal-directed manner. The design pattern of CrisGen is shown in Figure 4.7.

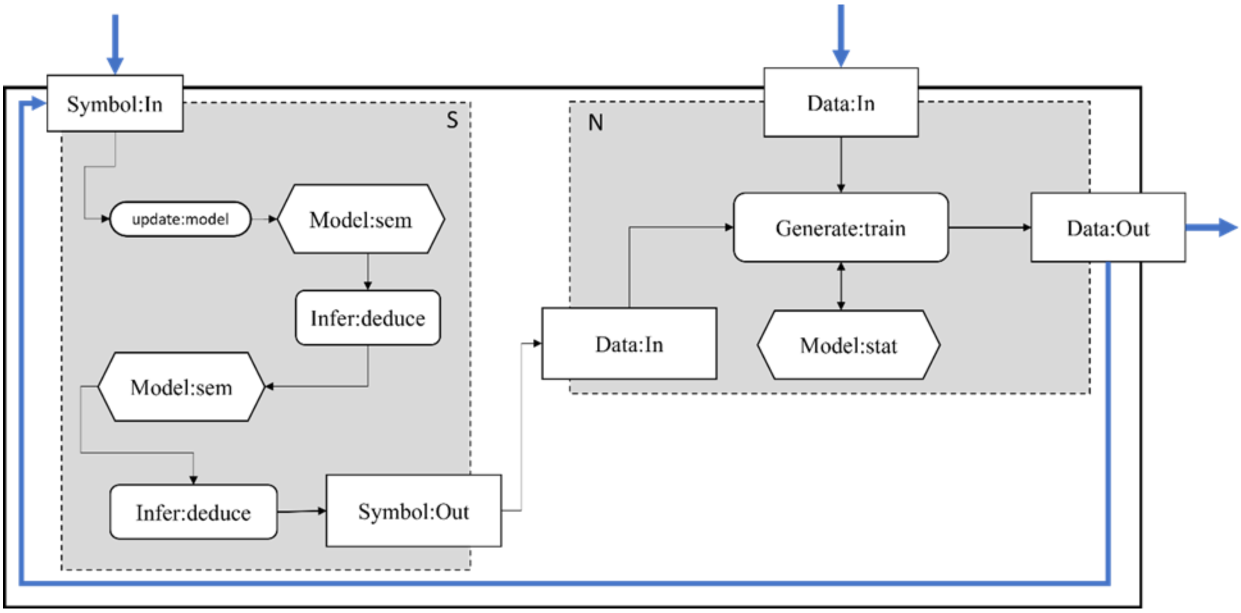


Figure 4.7: Design pattern for CriSGen as producer of critical scenarios for an autonomous driver. The driving simulator is considered implicit within the hybrid AI system as it is situated between the S-module's output and the N-module's input (see section 4.3.3)

The information flow between the two modules of this type of hybrid method is bi-directional (entangled). On the one hand the logical inference mechanism generates symbolic data to guide the learning process of the neural module. On the other hand, the symbolic output of the neural module serves as a trigger for the symbolic reasoner to infer concrete new symbolic data that "challenges" the neural module. As can be seen in Figure 4.7, the major extension to the generic design pattern for informed learning with prior knowledge lies in the S-module rather than the N-module. In this pattern a symbolic model is generated from former training data and the N-module's symbolic outputs to derive new training data. It thus considers the knowledge of the neural network behaviour for a given input data. The newly derived training data is supposed to be as similar as possible to the given training data, however with the proviso that the N-module's former output would lead to a negative reward. It thus forces the neural network to behave differently in situations that are similar yet challenging.

The two semantical models within the S-module are particularly interesting here. The first model is generated (operation update:model) from a given scenario, the autonomous driver's behaviour within this scenario and a suitable abstraction mechanism that generalizes the given scenario in the sense that it represents a whole bunch of similar scenarios together with a formal description of the driver's manoeuvres. From this model another formal description is inferred that represents all scenarios within the abstractions mentioned above that would lead to an undesired outcome. Finally, from this formal description some concrete instances are extracted that are as similar as possible to the original scenario. Any one of these instances serves as a new, challenging input scenario for the autonomous driver.

Other example methods of this category of hybrid neuro-symbolic methods are summarized in the following.

The hybrid system in Wang et al. (2016) leverages given semantic model (ontology) as semantic prior to guide the design of the neural network in form of deep restricted Boltzmann machines for informed learning of recreational sports classification of items. The resulting neural network structure, called ontology-based deep restricted Boltzmann machine, represents structured knowledge from the ontology such that the respectively informed learning and execution complies with ontological domain and task knowledge (implicit semantic validation). This is similar to the approach presented in Phan et al. (2015).

The hybrid AI system proposed in Ma et al. (2018) combines reasoning on prior semantic knowledge in form of knowledge graphs to enable informed learning with a neural autoencoder for multi-omic data analysis in the medical domain. The knowledge comprises essentially biological interaction of human cells and the autoencoder is supposed to learn typical cancer features. The proposed architecture fits well into the design pattern of Figure 4.6 Where the semantical model contains the biological knowledge graph and the statistical model of the N-module (in this case the autoencoder) predicts pathological cases of cancer cells.

Marra et al. (2020) propose a hybrid approach that integrates a recurrent neural network and a first-order probabilistic logic reasoner into what is called a relational neural machine. The symbolic part consists of first-order predicate logic formulas – often in form of clauses that act like production rules – with which some *noise* can be associated that represents different grades of uncertainty. The deduction mechanism then serves as a guide that strengthens the N-module's output. The authors compared various inference mechanisms and training methods such as expectation maximization with the standard neural network approach and reported improvement of performance; the corresponding design pattern perfectly matches Figure 4.6.

The following methods are variants of this type of hybrid system in the sense that the learning module is not a neural network but a reinforcement learner, though one might consider a deep reinforcement learner instead.

In De Giacomo et al. (2019), and Geibel et al. (2006), a constrained reinforcement learning approach is presented. The exploration behaviour of a reinforcement learning (RL) agent is constrained through symbolic constraints that enforce safety conditions. In particular, two distinct sets of features are extracted from the world, the first set by the RL agent and the second set by the "authority", that is the symbolic reasoner imposing restraining specifications, also called bolts, in linear time logic on finite traces over this set. The RL agent obtains additional reward and features thereby learning to act in compliance with the restraining bolts. The corresponding design pattern fits well to Figure 4.6 where the feedback is explicitly enabled (entangled version).

Illanes et al. (2020) use high-level symbolic plans as semantic prior from the symbolic AI module to guide a reinforcement learner towards efficiently learning a policy. These plans are represented in terms of linear temporal logic formulas. There is no reasoning within the linear temporal logic involved, they rather represent sequences of actions in terms of this logic. While executing a plan, the restricting temporal logic formulas evolve to future sub-sequences. The potential planning policies are thus always compared to future action sequences that are represented symbolically.

4.3.3 Security Aspects

In this section we examine relevant security aspects of the selected method CriSGen (cf. Section 4.3.2) with respect to relevant variants of generic attack pattern types for hybrid systems (cf. Section 4.1.2). This is followed by a very short summary of security aspects of other example methods and discussion of our findings for this category of hybrid methods in general.

Attack patterns for the hybrid system CriSGen. Recall that CriSGen is active solely during the training of the autonomous car. The system input from the environment (from the driving simulator) is provided through the system interface to the learner alone, the S-module does not take this information into account. During this *training* phase, CriSGen is of type *entangled* (cf. Figure 4.1: hybrid system type ENT:S→N→S). The scenario generator's output (Symbol:Out) – which is also the N-modules input - consists of a symbolic driving scenario that can be interpreted by the driving simulator (not part of the corresponding design pattern) to produce sensory data (Data:In) for the N-module. Such sensory data consists of camera images, radar signals, ultra-sonic information, etc. depending on the assumed capabilities of the ego car under consideration. The link between these two therefore covers much more than a mere data connection; it also represents all the computational effort the driving simulator has to spend in order to interpret the scenario details. Similarly, the N-modules output (Data:Out) actually consists of (a sequence of) driving manoeuvres that can be interpreted/generated by the simulator. But also, this output serves as input for the S-module after transforming the manoeuvre data into a symbolic form that is readable for the S-module. Therefore, the *relevant attack variants* for the *training* phase are concerned with the manipulation and theft of the three CriSGen models [MHM-ENT, THM-ENT], the CriSGen inputs [MHI-ENT, THI-ENT], and the CriSGen outputs [MHO-ENT, THO-ENT].

In CriSGen, all statistical and semantical models can be *directly* and *indirectly* manipulated via the hybrid system interface. There are two different semantical models of the S-module involved. Only the one that is directly connected with the hybrid AI system's symbolic input (Symbol:In) can be directly manipulated through the system's input interface. The second semantical model has no immediate connection to the environment through the system interface and therefore can only be manipulated indirectly via changes of the first semantic model. The statistical model in the N-module, however, can be directly manipulated via the system interface by changing the data input (Data:In) and indirectly by changing the internal transformation from the S-module's output (Symbol:Out) to the N-module's input (Data:In). Recall that this transformation involves a complex mechanism realized in the driving simulator implementation. Similarly, the connection between the N-module's output and the S-module's input involves the driving simulator and is as vulnerable as the simulator itself.

The attack patterns [MHM-ENT, MHO-ENT, MHI-ENT] as well as [THM, THI, THO] for CrisGen are described in Appendix B.2.

Attack patterns for other example methods. In a figurative sense the attack patterns that have been identified for CriSGen also apply for the other example methods mentioned above. Whereas Wang et al. (2016), Ma et al. (2018), and Marra et al. (2020) define hybrid systems that are not entangled (see Figure 4.6 with the dashed connection deleted), the approaches in De Giacomo et al. (2019), Geibel et al. (2006), and Illanes et al. (2020) rather follow the entangled variant as defined for CriSGen. The various approaches differ essentially in the way symbolic knowledge is represented as well as in the sub-symbolic system that is used. For instance, Wang et al. (2016) represent their symbolic knowledge in terms of ontologies (see Section 3.2.1) and use a deep restricted Boltzmann machine for sub-symbolic inference. Possible attacks are therefore based on the attack patterns for theft or manipulation of the input models (ontologies), or manipulation of the input/output. Marra et al. (2020) on the other hand utilise a first-order probabilistic reasoner (see Section 3.4.2) within their S-module and a recurrent neural network for their N-module. Here the attack patterns concerning manipulation of input/output as well as theft or manipulation of the semantic or statistical models apply here as well.

De Giacomo et al. (2019), Geibel et al. (2006) as well as Illanes et al. (2020) use reinforcement learners within their N-module, whereas the former approaches apply symbolic constraints to suitably guide the learner, the latter make use of linear temporal logic formulas (see Section 3.2.7) that evolve with the action sequences inferred by the neural network in order to monitor its behaviour.

Discussion. Many of the aspects mentioned in the discussion of security aspects of the previous category of hybrid methods in Section 4.2.3 also apply here. Nevertheless, there are some slight differences regarding explainability that should be emphasized. Recall that in the case S → N as well as in its entangled version S → N → S the S-module is supposed to guide or challenge the learning mechanism within the N-module. Here *challenging* means that a former input to the learner which resulted in a positive reward is taken by the S-module and changed as little as possible but nevertheless in a way that forces the learning system to behave differently in order not to gain a negative reward for the modified example input. The idea behind this assumes that, in general, very little changes in an input example (which the learner handled well) will result in no or only minor changes of the learner's output. If, however, there exists a modified version of the example for which the same output would end up with a faulty result then this modification might safely be called a challenge for the learning system.

For instance, suppose a driving situation in which a driver, whether human or an AI system, slows down a bit although the pedestrian crossing the street in front of the car is still quite far away. If asked why s/he slowed down, it would be very surprising to hear the answer that the driver was faced with exactly the same situation and had learned to slightly decelerate. It is more likely, though, that the driver explains that s/he cannot know the pedestrian's behaviour and that s/he might change his/her mind and make a turn right before s/he reaches the other side of the street. Such a modification, i.e., letting the pedestrian make a turn instead of straight crossing the street would be a typical challenging example for an S-module to infer, regardless of whether that driver was trained with such an example or not. Such a challenging situation can thus be seen as some kind of *a posteriori* explanation of a behaviour that could hardly be foreseen.

In general, in such methods the output of the N-module is also the output of the entire hybrid system. In the non-entangled version, the knowledge represented in the S-module is typically knowledge about the domain of discourse in which the information processing of the neural network operates, possibly extended by the external inputs to the N-module. Any direct attack on the statistical model or its inferences is thus similar to that of a pure neural network. The task of the S-module is to guide the derivations of the N-module into specific paths. If an attacker has the opportunity to manipulate this symbolic output, the learning success will be jeopardised, and the learner will probably even behave worse than if it receives no additional input at all. Such an attack can be prevented by designing the hybrid system as a monolithic block so that the internal communication between the symbolic and sub-symbolic components cannot possibly be disrupted. The same applies to changes in the semantic model within the S-module. In the entangled version, the S-module additionally receives information about the output of the N-module. This enables further targets of attack, namely the manipulation of the N-module's output during or after the conversion into semantic information that can be interpreted by the S-module. In the case of CriSGen, this would typically be a manipulation of the output of the driving simulator. A skilled attacker could perform such an attack without affecting the externally visible actions of the driving simulator. Since the simulator is typically a stand-alone component within this scenario, integrating it into the monolithic block of the hybrid system is in general not possible. For this reason, it is necessary to secure the communication between the modules and the simulator using suitable authentication mechanisms and cryptographic means.

4.3.4 Use-Cases

As use case for this category of hybrid AI methods, we consider the practical application of the hybrid learning and scenario generation method CriSGen (cf. Section 4.3.2) for collision-free navigation of self-driving cars in a multitude of traffic scenarios.

CriSGen in practice. The practical application of CriSGen is to train an autonomous driver through a driving simulator (in this case the free simulator OpenDS) in a goal-directed manner by synthesizing critical traffic scenarios from previous scenarios that have been well mastered by the driver. It is thus based on the suspicion that, although the autonomous driver gained a positive reward for a given scenario, it is by no means sure that it behaved well for the right reasons. The synthesis is based on a purely symbolic calculation of new scenarios (i.e., road and building geometries, as well as behavioural descriptions of other road users), based on the given scenario, the associated driving behaviour judged to be good, and a formal description of possible misbehaviour. For this purpose, first an abstraction of the scenario description is determined which, in contrast to the original scenario, contains free parameters. This abstraction is then translated together with a symbolic representation of the driving behaviour (and the possible misbehaviour) into a so-called hybrid automaton (not to be confused with hybrid AI systems). From this hybrid automaton, a set of arithmetic constraints can then be derived with the help of a forward reachability analysis, which describes all variants of the abstraction that lead to a misbehaviour. These constraints ultimately describe a subspace in an n-dimensional hyperspace (where n represents the number of parameters) in such a way that each point within this subspace describes a critical scenario. The geometric proximity to the original scenario (also a point in this n-dimensional space) in turn represents a measure of the similarity of scenarios. Finally, points that are as similar as possible are determined from the subspace and translated back into real simulator scenarios. The resulting new, critical scenarios then serve as a challenge for the autonomous driver (see Figure 4.8).

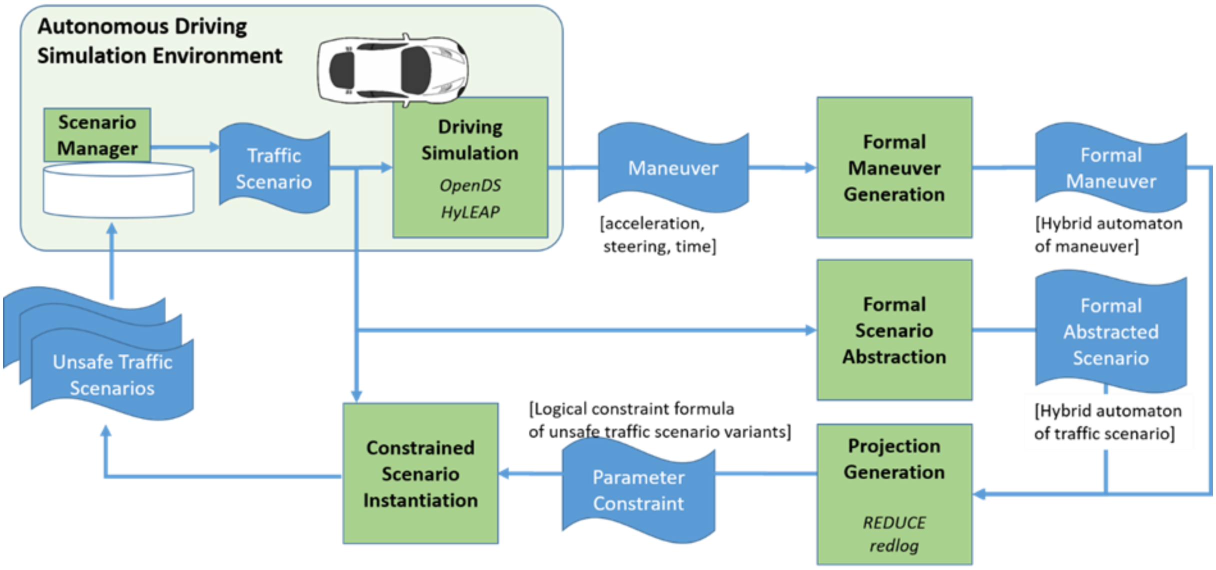

Autonomous Driving Simulation Environment
Scenario Manager
Traffic Scenario
Driving Simulation
OpenDS
HyLEAP
Maneuver
[acceleration, steering, time]
Formal Maneuver Generation
Formal Maneuver
[Hybrid automaton of maneuver]
Formal Scenario Abstraction
Formal Abstracted Scenario
[Hybrid automaton of traffic scenario]
Unsafe Traffic Scenarios
Constrained Scenario Instantiation
[Logical constraint formula of unsafe traffic scenario variants]
Parameter Constraint
Projection Generation
REDUCE
redlog

Figure 4.8: Technical workflow of CriSGen to train an autonomous driver in a goal-directed manner

Security aspects for CriSGen in practice. Figure 4.7 focuses on the symbolic aspects of the S-module of CriSGen. The N-module together with the driving simulator OpenDS with HyLEAP is hidden behind the *Driving Simulation* box. The processing of the various symbolic inference steps does not take place in a monolithic system but in several subsystems that communicate with each other. In particular, an external system (REDUCE with quantifier elimination system redlog) is used to calculate the arithmetic constraints that describe the critical scenarios. The individual systems communicate their intermediate results with each other by means of file transfer. This happens partly via the file system, but also via remote procedure calls.

Each of these communication paths can be the target of an attack on the entire system. For example, an attacker who has read and write access to the shared file system could easily read the data to be exchanged (loss of confidentiality) or even change it (violation of integrity). The same applies to a man-in-the-middle attack against remote procedure calls.

The points of attack are manifold: If an attacker succeeds in manipulating the transmitted manoeuvre, a hybrid automaton will emerge that may no longer have anything to do with the original behaviour of the driver. The scenarios that ultimately arise would not serve the actual purpose of the approach. The same applies to the manipulation of formal models, but this requires much more detailed knowledge (about syntax and semantics of the transmitted data) on the part of the attacker. The artifact P*arameter Constraint* could, however, be manipulated without any problems in such a way that a syntactically meaningful, but semantically useless, subspace is described.

Finally, there is another point of attack, namely the *Scenario Manager* who is responsible for the management of a large number of training scenarios that are to be sent to the autonomous driver during the training phase. In addition to many initial scenarios, the *Scenario Manager* is also supplied with new, critical scenarios by CriSGen. An attacker could intercept the sent data with the appropriate means and forward it with or without manipulation. In both cases, detecting such an attack would only be possible with a great deal of effort. The only remedy would be to convert the S-module into a single monolithic system. In this case, it would be sufficient to cryptographically secure the communication paths between the S-module and the N-module.

4.4 Learning Logical Representations for Statistical Inferencing

4.4.1 Description

In many use cases and domains there is extensive knowledge available that has been formalised in centuries of research, such as the well-known laws of physics and chemistry. It makes a lot of sense to use this reliable knowledge in the machine learning component, rather than restarting from scratch by performing experiments from data exclusively. It allows, among others, to use logic calculus in the neural network. In this type of hybrid AI systems, a neural AI module (N) is trained using appropriately encoded data (Data:In) from a symbolic representation, such as a knowledge graph (Symbol:Out) of a semantic AI module (S). The generic design pattern is shown in Figure 4.9.

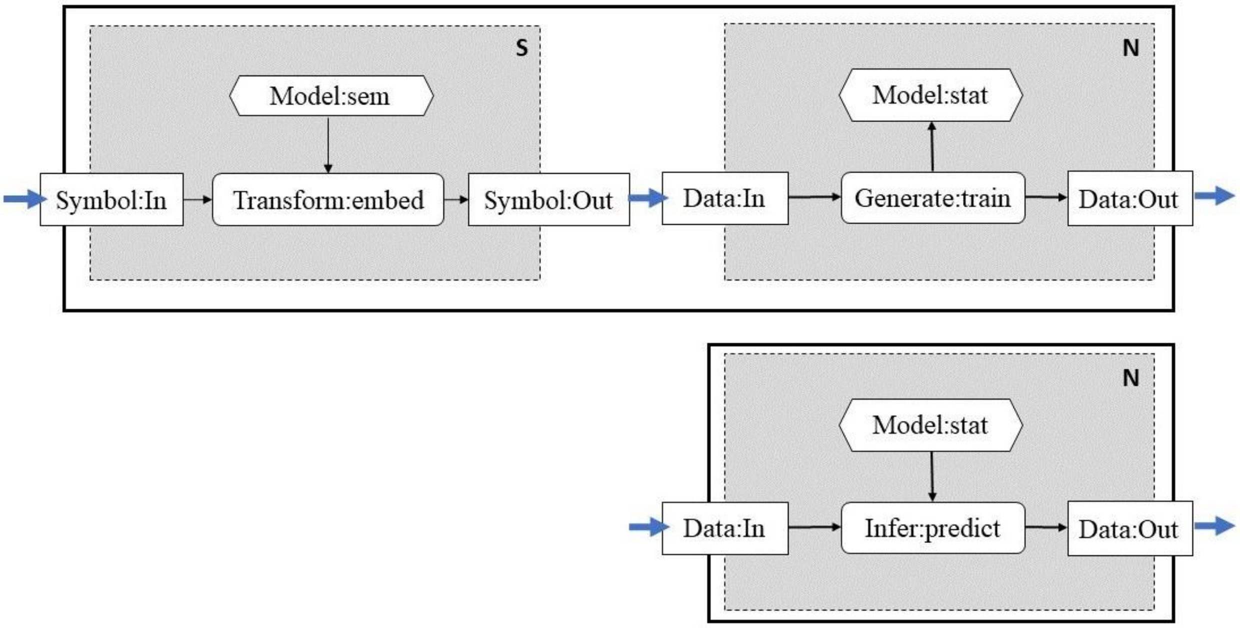


Figure 4.9: Generic design pattern of hybrid AI methods for learning logical representations for statistical inferencing (top: training phase, bottom: operational phase)

According to the above-mentioned classification scheme, this is a sequential (SEQ) hybrid system of type S→N. That is, the S-module leverages a semantic model for transforming a symbolic input (Symbol:In) into an intermediate semantic data representation (Symbol:Out) which is then (after appropriate encoding) used as data input (Data:In) to train the N-module to statistically infer ("deduce") symbolic output (semantically interpreted output data). In other words, the hybrid AI system first transforms a semantic model into vector/tensor representations and uses them to train a neural network to learn making statistical inferences (Infer:predict) based on its embedded logic. Consequently, the knowledge incorporated in the semantic model (Model:sem) of the S-module is embedded in the structure of the neural network and model (Model:stat) of the N-module such that the AI system can profit from existing logical knowledge. During the operational phase the N-module uses its trained model in the same way as in traditional machine learning approaches to make statistical inferences (Infer:predict) that are based on the embedded knowledge of the semantic model.

4.4.2 Example Methods

Below, two representative methods of this type of hybrid AI method class are explained: Graph neural network (GNN) and logic tensor network (LTN).

Graph neural networks Zhou et al. (2018), Battaglia et al. (2018), Hamilton (2020), Chen et al. (2020) and Jegelka (2022) incorporate knowledge graphs in neural networks by embedding their structure in a form that is suitable as input to an N module. The advantage is that the N module does not need to learn from scratch, but can reuse existing knowledge, therefore requiring less data, and increasing robustness through relying on proven knowledge models.

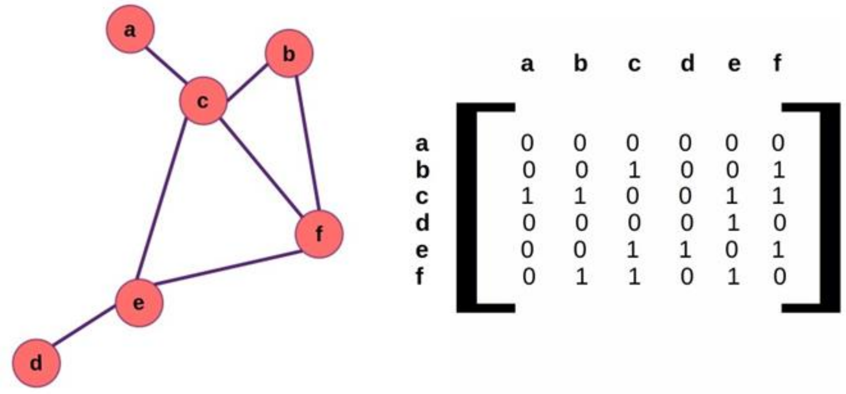
GNNs are a specific type of graph representation learning (GRL) and include graph convolutional networks (GCN) and spatio-temporal graph neural networks (STGNN). In general, a GNN makes use of knowledge, that is the semantic model Model:sem of the S-module, represented as semantic graphs (e.g., knowledge graphs or ontologies) as appropriately transformed input to a neural network (N-module) as training data. Semantic graphs consist of concepts (nodes) and their relationships (links) with their corresponding attributes and instances. Consequently, the statistical model (Model:stat) of the N-module is based on the knowledge in the graph. Alternatively, or in addition, the structure of the neural network can represent the structure of the graph.

The method of using GNNs can be described as follows. As a first step, the appropriate graph structure in the problem domain is identified. Second, a graph type is selected (such as directed/undirected, homogeneous/heterogeneous, static/dynamic) where these properties are orthogonal. In the third step, the loss function for the learning task of the neural component is defined as either node-, edge- or graph-level tasks. Finally, a model is constructed using computational modules for propagation, sampling, and pooling. The resulting model is instantiated in a GNN pipeline that matches the graph structure to the task to be performed.

GNNs can be used for various purposes:

- *Node Classification*: nodes and their attributes are evaluated and assigned to certain types or classes (for example, nodes can be identified as representing actors, directors, movie titles, genres, or soundtracks in a movie database).
- *Graph Classification*: classify the whole graph as a certain type or class.
- *Clustering*: determine groups of nodes that form subgraphs according to some common attributes or links (for example, by grouping actors and movies according to genres).
- *Graph Completion*
  - *Node Prediction*: nodes can be introduced as new concepts or instances using data analysis (for example, to detect new movie genres or to find movies that are missing).
  - *Link Prediction*: missing links can be found by analysing the similarity between nodes and their relationships (for example, to detect missing links between actors who played in movies of the same director).

Before a graph can be used for training the statistical model (Model:stat), it needs to be converted to a suitable format for the neural network (N-module). The process of transforming the semantic model (Model:sem) into a tensor representation is known as *embedding*. Embedding can be performed using various methods, such as random walks, deep walks and adjacency matrices (see Figure 4.10), where the connections of each node of a graph are converted to a numerical representation in a vector space with different depths, walking strategies and incorporating various attributes of each node and its (neighbours') links. Another class of embedding methods is *node2vec* (and *graph2vec*, *owl2vec*, etc.), which uses a language model and is based on *word2vec*. The result of embedding is a representation of (parts of) the information of the graph that the neural network can process without modification.



$$\begin{array}{c|cccccc}
 & a & b & c & d & e & f \\
\hline
a & 0 & 0 & 0 & 0 & 0 & 0 \\
b & 0 & 0 & 1 & 0 & 0 & 1 \\
c & 1 & 1 & 0 & 0 & 1 & 1 \\
d & 0 & 0 & 0 & 0 & 1 & 0 \\
e & 0 & 0 & 1 & 1 & 0 & 1 \\
f & 0 & 1 & 1 & 0 & 1 & 0
\end{array}$$

Figure 4.10: Adjacency Matrix [Source: https://rivesunder.github.io/public_domain/2021/07/25/public_domain.htm]

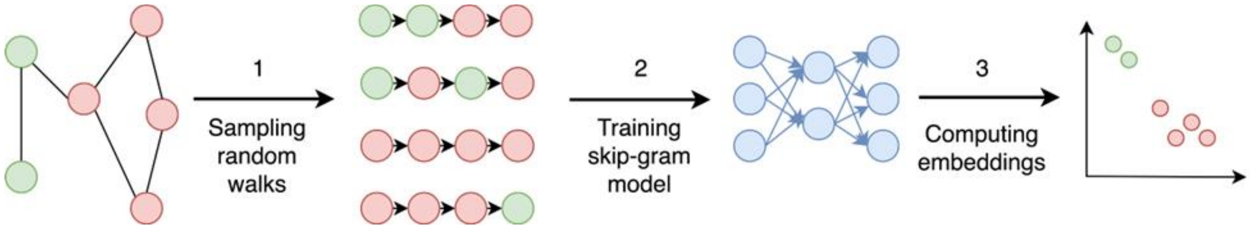


Figure 4.11: Random Walk [Source: https://dsgiitr.com/blogs/deepwalk/]

Alternatively, the GNN can be considered as *incorporating* the graph *structure*, where each node is a neural network processor and the communication among nodes occurs via message passing (see Figure 4.11). Messages are sent from each node to its connected nodes containing information for updates in a vectorised form. Receiving nodes update their own information and proceed to distribute further messages.

Logic tensor networks enable their N-modules to make inferences based on their embedded logic. In other words, in LTNs relational data is embedded in a (convolutional) neural network (Serafini, et al., 2016). The symbolic (first-order logic) input of the S-module is transformed into data which is subsequently used by a series of convolutional neural networks to learn how to produce symbolic output for a given task such as knowledge graph completion. This approach is extended in Donadello et al. (2017) to a hybrid LTN approach: It utilizes fuzzy logic reasoning on given prior factual knowledge and a tensor neural network for indoor object classification, in particular the classification of an image's bounding boxes and the detection of the relevant part-of relations between objects. In particular, it is an example of a statistical relational learning approach for reasoning under uncertainty and learning in the presence of data and rich knowledge applied to the task of semantic image interpretation. The proposed logic tensor network integrates neural networks with first-order fuzzy logic to allow (i) efficient learning from noisy data in the presence of given (prior) logical constraints, and (ii) reasoning with logical formulas describing general properties of the data.

Other example methods of this category of hybrid neuro-symbolic methods are summarized in the following.

In Zhou et al. (2018), an introduction and classification of graphs and approaches to GNNs is given. A large number of papers is listed and explained with very different applications and domains. The paper gives an overview of approaches and use cases and a generic design pipeline for a GNN model that enables systematic comparison according to the presented classification.

In Almasan (2019), Deep Reinforcement Learning (DRL) is combined with Graph Neural Networks for network traffic routing optimisation. A GNN-based DRL agent evaluates diverse routing scenarios and network topologies to find an optimal solution. The GNN models the network environment and the DRL agent can then operate in networks that were not available and used for training. When new traffic arrives the DRL agent simulates the effect in the current network and the GNN allows for adapting the network, accordingly, based on the embedded network graph structure as input to the N-module. The network graph is required for this simulation and the N-module can then perform predictions for efficient use of the network.

In Cohen et al. (2020), a reified knowledge base with reasoner provides transformed data input to train the neural module to perform multi-hop inferencing tasks (such as for knowledge graph-based question answering) with symbolic output. The relations in the knowledge base are represented as sparse matrices which are suitable for processing by a neural network. They conclude by claiming that this approach is much faster (partly due to distribution to multiple GPUs) and efficient without losing the semantics of the knowledge base.

4.4.3 Security Aspects

In this subsection, we provide the main attack patterns for this category of hybrid AI systems.

Attack Pattern Name: [**MHM-SEQ**] *Manipulation/hybrid system model* for system type SEQ,

Description: An attacker manipulates

1. the S-module model directly through the hybrid system interface, or
2. intercepts the embedding of the S-module model to modify the training data.

Methods of Attack:

1. The direct manipulation of the S-module model (Model:sem) is enabled in the same way as described above.
2. The communication line for sending requests and receiving results to and from the S-module model is manipulated such that they reach a different model elsewhere. For example, the IP address can be manipulated, or a socket can be redirected.
3. When the output of the embedding process is attacked, wrong data is fed from the S-module into the N-module.

Attack Motivation-Consequences: An attacker might want to change the concepts and relationships in the semantic model of the S-module such that a certain bias or "alternative" truth is reflected in the model, which could reflect an attacker's interests.

Attacker Skill or Knowledge Required: The availability of an open API facilitates sharing access to the model, but also allows attackers to easily manipulate the model. An attack to the communications line, or even the internal embedding module are hard to achieve and require detailed knowledge of the internal system architecture.

Resources Required: Documentation of the external and internal interfaces is required, either via open access or via reverse engineering and hacking.

Solutions and Mitigations: State-of-the-art authentication and authorisation methods can prevent attackers from accessing the model and the process that transforms it (embedding).

Attack Pattern Name: [**MHI-SEQ**] *Manipulation/hybrid system input* for system type SEQ,

Description: An attacker manipulates

1. the input data of the S-module (Symbol:In), or
2. the input data of the N-module (Data:In), or

Methods of Attack:

1. Either the data set itself can be attacked and (partially) replaced or the connection is disrupted, and the data is modified.
2. Same as above.

3. The communication line for receiving input data for the S-module model is manipulated such that it reaches different input data elsewhere. For example, the IP address can be manipulated, or a socket can be redirected.

Attack Motivation-Consequences: Attackers could change the symbols to be embedded by the S-module, and hence learned by the N-module in the training phase, for their own profit. They could also change the data input that is used by the N-module in the operational phase for their own purposes.

Attacker Skill or Knowledge Required: Access to an open API facilitates access to the input data set and allows attackers to manipulate the data. An attack to the communications line is hard to achieve and requires detailed knowledge of the communication protocol.

Resources Required: Documentation of the external and internal interfaces is required, either via open access or via reverse engineering and hacking.

Solutions and Mitigations: State-of-the-art authentication and authorization methods can prevent attackers from accessing the input data.

Attack Pattern Name: [**MHO-SEQ**] *Manipulation/hybrid system output* for system type SEQ,

Description: An attacker manipulates the output data (Symbol:Out) by redirecting the output data connection, such that the receiver receives a wrong data set as a result. This attack is equal to other well-known attacks on neural networks, known as adversarial attacks, because it is irrelevant how the output was produced. Consequently, any black box attack pattern is applicable if only the output (e.g., labels of classifications) is manipulated.

Attack Pattern Name: [**THM-SEQ**] *Theft/hybrid system model* for system type SEQ,

Description: An attacker copies the model for use in different applications, thereby saving effort to produce the model by itself (lack of resources or access to training data). A model can be accessed and stolen through an API, but also using repetitive legitimate requests and model extraction.

Attack Pattern Name: [**x^**] *Theft/hybrid system input* for system type SEQ,

Description: An attacker redirects (a copy of) the input stream to use the (possibly private) input data for other purposes.

Attack Pattern Name: [**THO-SEQ**] *Theft/hybrid system output* for system type SEQ,

Description: An attacker manipulates the output stream in order to send alternative results to the receiver.

4.4.4 Use-Cases

Logical representations for statistical models are used, e.g., in medical applications, where known chemical relationships and properties are embedded and used as input for a neural system that explores previously unknown molecules for discovering new drugs (protein interface prediction). Another application in the medical domain is disease prediction (Cosmo, et al., 2020), also known as Computer Aided Diagnosis (CADx), where similarities among patients are represented as a graph, which allows to classify (potential) patients to predict their probability to get a disease. Because the graph already contains the relevant causes and symptoms of known diseases, these need not be learned. Therefore, less training is required, and the predictions are more robust (based on medical experience and knowledge) and explainable.

Graph neural networks in practice. GNNs are used in particle physics (Shlomi, et al., 2021), where relations among elementary particles (quarks and leptons) include strong, weak, and electromagnetic forces modelled as nodes and links in graphs. The GNN can then be used, for example, for particle flow reconstruction, by link and node predictions and event classification in comparison to experimental results. GNNs can also be used for designing a recommendation system, where the graph consists of users, items, and reviews (nodes) and interactions (edges) and is structured as a large-scale directed heterogeneous graph on which missing links can be predicted. In similar ways, physical geographical models (earthquakes, fires, floods, climate change) or social (migration, pandemic, social network analysis) and economic models (fraud detection, creditworthiness) can be analysed. All these graphs contain structural knowledge that cannot practically be modelled using neural networks, while, at the same time, embedding mechanisms perform exactly this conversion. As a result, the best of both worlds is combined: explicit representation of domain knowledge and adaptive learning mechanisms to find hitherto undiscovered facts and relations.

4.5 Explainable Learning Through Rational Reconstruction

4.5.1 Description

Statistical machine learning, especially deep learning, outperforms other machine learning methods in many tasks. Problems remain, however, inter alia in the form of insufficient robustness in the presence of perturbations and the general lack of explainability. Therefore, especially in safety-critical applications, an additional layer or perspective is required. The class of hybrid methods of explainable learning through rational reconstruction are meant to solve this problem; their generic design pattern is shown in Figure 4.12.

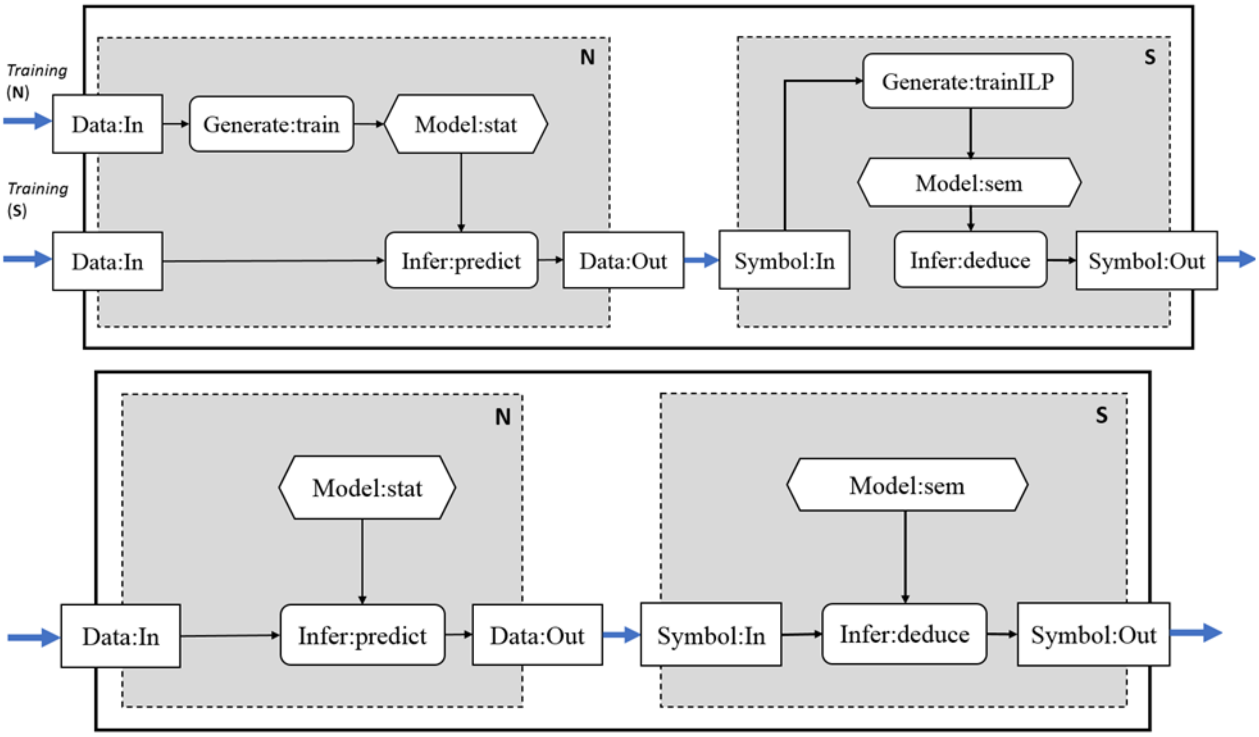


Figure 4.12: Generic design pattern of hybrid AI methods for explainable learning through rational reconstruction (top: training phase with N-module trained first, S-module trained second with trained N-module, bottom: operational phase with trained N- and S-modules)

The N-module essentially stays the same, i.e., it can be a black box. An S-module is linked to it in such a way that the outputs correspond to each other. The output of the S-module can then be used for explaining and plausibility checking of the output of the N-module. The strength of the approach depends on the strength of the relationship between the two. In its simplest form, the S-module will be a mere "second opinion" for each given instance. The training of both modules is then conducted independent from each other, and the hybrid system architecture type is purely sequential (SEQ, N→S).

However, the idea of the design pattern aims at stronger links, that can for example be achieved through entangling (ENT). In practise, the relation between the N-module and the S-module will always be less than 100% in expressiveness or coverage (otherwise the S-module would be redundant after all). Hence, the S-module will in some cases only be able to give indications of whether the N-module is right or not. In other cases, it will perhaps not have an answer for all instances in the N-module's domain.

Pacaci et al. (2019) provide a four-step procedure, named RICE, for this design pattern of hybrid AI: Step one they call "Training of the model". This can either be meant for the case of a pre-trained N-module or jointly trained module (we provide examples for both cases below). The second step is called "Extract examples from the model (probing)", in other words connecting the output level to the S-module. The third step is

“Synthesize a human-readable program from examples (synthesis)” and the fourth step “Inspect the synthesized program as an indirect representation of source artefact (validation)”.

4.5.2 Example Methods

Muggleton et al. (2018) introduce a new approach on making perception systems based on visible light more robust. The core idea of the approach is to create background knowledge that describes some of the physical properties of light propagation. The idea is best explained on the example of light reflection on convexly vs. concavely shaped objects. A ball, for example, would reflect sunlight in a distinct kind of way, having the lighter areas in the top hemisphere and the shadowy areas in the bottom one. Additional knowledge about the available light sources would make the reflection hypotheses about the objects even stronger. Muggleton et al. store these as rules in a background knowledge base using inductive logic programming. In principle, this approach can be categorized as “physics-based learning” since assumptions from physics are applied to steer machine learning. The architecture in the original paper thereby follows the hybrid design pattern of Informed Learning with Prior Knowledge (see Section 4.3). Here, we investigate the application of the approach in perception for autonomous driving. Robustness of camera-based object recognition could be greatly enhanced if the underlying three-dimensional properties could be utilized. Much like in the original example in Muggleton et al. (2018) with a Robo-Soccer ball, partially occluded object could then be better recognized. The hybrid design principle of rational reconstruction is a plausible variant. We would then expect a state-of-the-art visual perception N-module (object recognition module) based on a DNN plus ILP S-module containing a background knowledge base. Additionally, a visual-computing component is necessary translating the camera image into features for the S-module. When we follow the approach in Muggleton et al., the symbolic background module can be regarded as rather application independent, since it describes light propagation constraints from physics, while the visual computing component is dependent on the exact application. The latter can also be based on machine learning including deep learning.

Other example methods of this category of hybrid neuro-symbolic methods are summarized in the following.

One representative method of this category of hybrid AI methods is described in Sarker et al. (2017). The method uses structured data from the semantic web to explain the input-output behaviour of a trained N-module, which acts as a classifier. The input layer is mapped onto a background knowledge base of the S-module. The output layer provides negative and positive examples. The S-module is implemented with inductive logic programming (ILP) applied on the example sets. It provides explanations for the classifications based on the background knowledge. The explanations have the form of logical class expressions such that all positive examples are described by the expression and all negative ones are not.

Let us assume that we have a classifier for cats. All images containing cats can then be considered positive examples and all images not containing cats as negative ones. Let us use the example Sarker et al. There are trains with cars varying in number, size, shape, content, and openness. They are arranged in two colums, the left column representing the positive examples, the right the negative ones. The S-module comes up with a logical expression ∃hasCar(closed ∧ short), which includes all positive examples and excluded all negative ones.

Pacaci et al. (2019) propose examples from the autonomous driving domain in a very similar setup. A scenario of a car approaching a stop light is described. The feature vector contains the state of the traffic light (0, 1, 2 for red, orange, green) and the distance to the traffic light in meters. The (postprocessed) output of the S-module produces statements like “If the red light is 1.0, when the Dist is less than 0.6 assign Go to 1.0, otherwise assign Go to 0.0.”

Ciravegna et al. (2020) use multi-label (or multi-class) classification for the N-module. Instead of being mapped on exactly one output label, the examples can be associated with multiple labels. In this case, the labels correspond to logical terms that can be combined to a first-order logic (FOL) expression. The result is very similar to the examples given above for Sarker et al. The main difference is that Ciravegna et al. propose the use of a second neural network as the “S-module” that operates on the output space (concept space) of the first one. Both the explanation related second network and the classification related first network are jointly learned, thus implicitly introducing a latent dependency between the development of the explanation mechanism and the development of the classifier. However, the variant of using a multi-label neural network as “S-module” can also be applied in a setting similar to the one in Sarker et al., assuming a pretrained N-module.

The variant proposed by Yang et al. (2022) is particularly interesting in the context of this study, because they use the S-module to defend and explain adversarial attacks on the N-module. The N-module is represented by an object classifier operating on RGB images like the one common in autonomous driving and many other applications. The S-module is implemented as inductive logic programming in the domain of the scene graph of the images. The idea is to reason about plausible spatial relations: A cat (that is detected due to adversarial manipulation) is not likely to have doors on the side and hover above train tracks. In other cases, a logical rule “Airplanes have engines” is violated by adversarial patches covering the respective image area. The others point out that the approach can be extended with common-sense databases in order to reason beyond the scene graph. This shows the potential of the given design pattern to reduce security risk inherent in the N-module.

4.5.3 Security Aspects

With Yang et al. (2022), we exemplified a case, in which the given design pattern can be used to reduce the inherent security risk in the N-module. The authors use information in the scene graph to create an S-module capable of discovering and explaining pixel-based adversarial attack on the N-module, an image-based object classifier. Despite the fact that the idea is very recent, we believe that there is a large potential. Any structured logical knowledge including multi-label classification can be used to help mitigating adversarial threats in a sense that the latter would be assumed to lead to inconsistencies in the former.

We investigate further a security threat in the variant of **Muggleton et al.** (2018) as described above. We pointed out that the knowledge base with rules about light propagation is domain independent. This could be an attractive target for the intruder. Since the rules are symbolic and human readable, it can be regarded a plausible case that the intruder manipulates these rules or adds new rules. The attack would be adversarial and have a similar effect as the well-known poisoning attacks on DNN N-modules.

The attack falls under the attack pattern type of “manipulation of hybrid system model” [MHM] attack pattern type, more specifically Model:sem its S-module. The attack is executed by directly manipulating Model:sem via hybrid system interface. Note that not the entire Model:sem is affected but only the part of the ILP which represents the background knowledge. In this specific case, the background knowledge can be considered application independent. Hence, the intruder does not necessarily have to have specific knowledge about the hybrid system. However, such knowledge would be required in order to create a specific effect rather than merely corrupting the system.

4.5.4 Use-Cases

Method of Sarker et al. (2017) **in practice.** The authors apply their scheme (cf. Section 2.5.3) on scene classification based on images (e.g., Section 4.5.2) on scene classification based on images (e.g., "outdoor warehouses" as positive cases vs. "indoor warehouses" as negative ones). As background knowledge, they use an ontology with common terms such that the positive class is described in terms of ∃contains.Roadw etc. and the negative ones are described in terms of ∀contains.Ceiling etc. In another experiment, the authors used pictures of mountains as the positive class. Here, the expression ∃contains.BodyOfWater was created as explanation by the S-module. Indeed, the authors found out that all examples of mountain pictures in the dataset contained either a river or a lake. This result can be used in a plausibilisation process in order to identify limitations of the connectionist approach (N-module) due to insufficient or inappropriate data. Sarker et al. (2020) refine the approach using Wikipedia, creating a pre-processed Wikipedia knowledge graph as background knowledge. Instead of roundabout five thousand concepts previously, the knowledge graph contains almost two million.

Regarding the variant of **Muggleton et al.** (2018), the most perfidious method of attack would be the following (we describe it from the perspective of autonomous driving, but the example is not limited to that). If the intruder gains access to the background knowledge, he could create a new rule, which does not correspond to any object in the usual environment of the car. The rule could made in such a way as to correspond to an artefact designed by the intruder (equivalent to the "sticker on the stop sign"). In the context of the hybrid design pattern of rational reconstruction, this would lead to an inconsistency between the result of the N-module (assumed to be unaffected by the attack) and the result in the S-module (manipulated by the attack). The consequences that arose from this inconsistency depend on the exact relationship between the N-module and the S-module. If S is used for explanations only (N→S), the explanation would be wrong, but the performance of the systems would be unchanged. If we are still dealing with a situation where consistency between N and S is monitored, the system could be tricked into a minimal-risk manoeuvre (e.g., pulling to the side). If the system is designed as entangled (N→S→N), we could essentially have all kinds of effects including driving forward with 50 km/h at a stop sign.

4.6 Summary

In general, hybrid neuro-symbolic (or neuro-explicit) AI methods or systems of the categories studied in this chapter can be potentially more or at least not less resilient against attacks than mere neural networks from sub-symbolic or connectionist AI. For example, the symbolic AI component (S-module) of hybrid methods can help to detect attacks through means of verifying input it receives from the neuronal AI component (N-module) against symbolically encoded task, domain or common-sense knowledge and issuing an alert with an appropriate explanation in case of intolerable inconsistencies or implausibility with this knowledge.

For example, due to the brittleness of deep learning methods a minimal difference in the input can lead to dramatic differences in the output. Suppose there is a manipulation of traffic signs with stickers such that suddenly a speed limit sign on a highway is classified as a stop sign. However, the semantic model of the symbolic AI method might indicate that this neuronal classification is impossible, thereby detecting the manipulation or erroneous classification. Statistical models and methods are in comparison harder to grasp for human understanding. However, they can be attacked with brute force methods, such as probing and sensitivity analysis where the most relevant input-output relationships are detected. The monitoring of suspicious activities could contribute to preventing such attacks (anomaly detection).

However, today's hybrid AI methods do not yet include such smart semantic verification feature, but some methods, such as for constrained deep reinforcement learning (cf. Section 4.3.2) and explainable learning through rational reconstruction (cf. Section 4.5), already point in this direction. Besides, hybrid neuro-symbolic AI methods inherit the extent to which they are explainable and human-understandable compared to mere deep learning systems from their symbolic AI method.

On the other hand, the inherent complexity of hybrid neuro-symbolic AI methods could be considered as a potential weakness with the most vulnerable component being attacked. However, compared to targeted adversarial attacks on individual deep learning, or symbolic AI methods, in many cases it requires significantly higher technical skills and more knowledge to design and run the same kind of attacks on a hybrid AI method. Such required knowledge concerns not only the N- and S-module model, input, and output, but the architecture type of and respective data dependencies and workflow within the overall system of the hybrid method. Basically, this argument holds for any of the hybrid methods in this study.

Analogously for data poisoning attacks on hybrid AI methods: These attacks are hard to detect for deep learning systems post-hoc, and consequently also for the training of N-modules in hybrid systems with an additional dependent S-module. But, again, the effort to be invested by attackers typically is significantly higher. In any case, oracle attacks are always possible as with all other AI systems (cf. Section 4.1.3).

Now let us briefly summarize our findings on the studied types of hybrid neuro-symbolic methods. For more details of the discussion of security aspects of these methods, we refer the reader to the relevant sections.

In hybrid methods of learning intermediate abstractions for reasoning or planning (cf. Section 4.2) the neural network produces intermediate abstractions for further use by the symbolic component for reasoning or planning. The S-module does neither detect nor mitigate possible attacks on the preceding N-module it is entangled with in the hybrid system. In this regard, these hybrid methods appear not more robust against (adversarial, data poisoning) attacks than deep learning methods. As mentioned above, the comparatively higher effort to be invested by attackers due to the entanglement of N- and S-modules makes these hybrid methods more robust against targeted attacks. They inherit the extent of explainability from their symbolic component such as the online planner in HyLEAP but, roughly speaking, do not offer more in this regard yet.

Basically, the same holds for hybrid methods of informed learning with prior knowledge (cf. Section 4.3). Prior knowledge represented in the S-module typically concerns the domain of discourse in which the neural information processing of the N-module operates, possibly extended by external inputs to the network. Some direct manipulation of the semantic model of the S-module used to guide the neural learner would indirectly manipulate the latter, and vice versa. This calls for the interaction between both components not to be disrupted and secured.

In hybrid methods of learning logical representations for statistical inferencing (cf. Section 4.4) the output of the S-module forms the input for the N-module which produces the overall output of the hybrid system. Again, the main interface for attacks lies between both modules, but also at the input and output of data and, possibly, the statistical and semantic models. As with the above-mentioned types of methods, both modules can be directly and indirectly manipulated during training and operational phase. When the semantic model (or at least its output) is deliberately modified, the N-module would learn from faulty knowledge without being “aware” of it.

Hybrid methods of explainable learning through rational reconstruction (cf. Section 4.5) appear more robust against adversarial and poisoning attacks on its neuronal component than the other studied types of hybrid methods. This statement assumes that the N-module is regarded as a black box (which is not necessarily the case), i.e., it offers exactly the same attack possibilities on its own as outside a hybrid overall system. The S-module itself is used for plausibility check and explanation of the N-module. Thus, if the attacker has successfully manipulated the N-module, it can be assumed that the S-module recognizes this attack and enables countermeasures accordingly. Of course, this defence can fail. But then we are at the same level as with the original stand-alone N-module. The same is true if the S-module is also attacked. The only conceivable aggravation of the situation with respect to the single N-module is if the attacker manipulates the S-module in such a way that it disguises the planned attack on the N-module. However, we can assume that this attack is more difficult to carry out.

In the context of formally *certifying security aspects* with the common criteria evaluation methodology[13], integrated security functions in hybrid methods of all categories selected for this study are needed but do not yet exist and require further applied research and development. In particular, test procedures for both module functionalities within the hybrid system are needed to decide whether for example (a) not the planner is corrupt, but the learner supporting the planner, or (b) vice versa, the planner itself is corrupt but not the learner, or (c) both are not corrupt, but their internal interaction (via respective part of hybrid system interface) is attacked.

Further, the usage of the hybrid system interface (cf. Section 4.1.1) needs to be secured by appropriate authentication and cryptographic means. However, there are currently no hybrid system implementations available that offer such kind of *secure interface* to their benevolent (developers, customers) or malevolent (attackers) users.

Overall, it can be summarized that hybrid AI systems are often more complex than traditional AI systems and due to more components and connections have a broader attack surface. However, we also have seen that hybrid architectures can be used to increase the robustness of neural networks. Moreover, targeted AI specific attacks on neural networks, like poisoning and adversarial attacks, on the N-module will be, in general, more difficult to apply since the complex interaction within the whole system has to be considered. Additional and concrete empirical work is needed to make more precise statements in this regard. We propose to proceed on the basis of the design patterns we have presented and to set up an experimental system that can be used to systematically investigate concrete attack scenarios on both the N-module and the S-module. Besides, more *practical real-world applications of hybrid methods* apart from those use-cases in rather toy domains have to be investigated.

This is a call for significant investment into research and development of hybrid neuro-symbolic methods as it is in general, and with respect to their security aspects in particular, still in its infancy. Of course, there is always the possibility of further literature studies to observe the state-of-the-art. For this purpose, the present study represents a good starting point, since the keywords for the literature search are listed here and a helpful system is provided by the structure of the design patterns.

[13] Common criteria evaluation methodology CEMV3.1R3 at www.commoncriteriaportal.org

5 Summary and Outlook

This section is meant as “executive summary”, rearranging key paragraphs from the entire study without paraphrasing. We made the point that with AI-based systems becoming ever more wide-spread and complex, both the paradigms of AI security and AI safety need to be extended and, in a way, they are growing closer together. AI security tries to cope with perturbations that occur “naturally”, for example because the environment changes or because the domain gradually evolves. AI security, aside from assuming an adversary, deals with similar problems. Data poisoning is the attempt to inject examples to the training set that decrease the accuracy of the system (or increase test error), thereby trying to be as efficient and subtle as possible. Evasion attacks alter the inference situation, either by manipulating the environment or otherwise making sure that the system receives input that leads to misclassifications. If we look at the body of literature in AI security, the game of finding new attacks on the one side and inventing new ways of defending them on the other could also be framed under the umbrella of research on robustness.

In the literature, one finds by far more contributions on security for sub-symbolic AI, especially (deep) neural networks, than for symbolic or even neuro-explicit AI. However, the proximity of AI-security to AI-safety in the sense of intended vs. natural perturbations, as described above, offers the possibility of a transfer. Indeed, the safety community has long assumed that hybrid AI can make a significant contribution to overcoming the drawbacks of purely statistical AI. The issue of robustness to distribution shifts is directly transferable here. We can assume that a method that is more robust to naturally occurring perturbations will also be harder to attack with deliberately induced ones – and vice versa. In addition, the safety feature of explainability also has potential for increasing AI security. Hence, in this study, we approached the topic from multiple perspectives. For both symbolic and hybrid AI, we lined-up a selection of methods, describing them using examples, and point out cybersecurity and AI security aspects, both direct and indirect (i.e., transferred from AI safety). Altogether this paints a picture of what we can call the hypotheses on the security of symbolic and hybrid AI, which in most cases require concrete and detailed empirical underpinning, formulated as research questions for the community to tackle.

Symbolic methods, in contrast to sub-symbolic ones, are expected to be robust to small perturbations due to their discrete, transparent, and verifiable nature. Gradient-based attacks lose their power against most symbolic approaches, often due to the non-differentiability of symbolic modules. Therefore, the attacker requires detailed knowledge of the symbolic methods and their special structures, e.g., trees, to conduct a successful attack. Since accessing such information regarding deployed models is usually hard to obtain in practice, the likelihood of an attack is lower. In real-world scenarios, for example, in Google Cloud Vision APIs, access to the internal model architecture/parameters or training data is strictly restricted, and decision-based black box attacks are more probable. Therefore, the attacker must search the input space by querying the model using APIs. The number of queries should also be minimized because they are usually expensive in terms of time and money.

Hybrid neuro-symbolic (or neuro-explicit) AI methods or systems can be potentially more or at least not less resilient against attacks than mere neural networks from sub-symbolic or connectionist AI. For example, the symbolic AI component (called S-module here) of hybrid methods can help to detect attacks through means of verifying input it receives from the neuronal AI component (N-module) against symbolically encoded task, domain or common-sense knowledge and issuing an alert with an appropriate explanation in case of intolerable inconsistencies or implausibility with this knowledge.

For example, due to the brittleness of deep learning methods a minimal difference in the input can lead to dramatic differences in the output. Suppose there is a manipulation of traffic signs with stickers such that suddenly a speed limit sign on a highway is classified as a stop sign. However, the semantic model of the symbolic AI method might indicate that this neuronal classification is impossible, thereby detecting the manipulation or erroneous classification. Statistical models and methods are in comparison harder to grasp for human understanding. However, they can be attacked with brute force methods, such as probing and sensitivity analysis where the most relevant input-output relationships are detected. The monitoring of suspicious activities could contribute to preventing such attacks (anomaly detection).

However, today's hybrid AI methods do not yet include such smart semantic verification feature, but some methods, such as for constrained deep reinforcement learning (cf. Section 4.3.2) and explainable learning through rational reconstruction (cf. Section 4.5), already point in this direction. Besides, hybrid neuro-symbolic AI methods inherit the extent to which they are explainable and human-understandable compared to mere deep learning systems from their symbolic AI method.

On the other hand, the inherent complexity of hybrid neuro-symbolic AI methods could be considered as a potential weakness with the most vulnerable component being attacked. However, compared to targeted adversarial attacks on individual deep learning, or symbolic AI methods, in many cases it requires significantly higher technical skills and more knowledge to design and run the same kind of attacks on a hybrid AI method. Such required knowledge concerns not only the N- and S-module model, input, and output, but the architecture type of and respective data dependencies and workflow within the overall system of the hybrid method. Basically, this argument holds for any of the hybrid methods in this study.

Appendix A: Attack Patterns – Symbolic AI Methods

A.1 Other attack patterns for automated offline planning methods

Attack Pattern Name: *Model Theft*

Attack Prerequisites: Data stores with API through which the planning domain and problem model, as well as the plan files can be retrieved and read, or digital communication line through which these items are being passed from the stores to the planner, respectively, from the planner to the controller.

Description: The attacker gains access to the planning domain and/or problem models and/or the plan produced by the planner after its planning ends.

Method of Attack: The attacker performs an illegal access to the data stores via their APIs, and/or intercepts the communication from the actors' data stores to the planner interface. The attacker may also perform an illegal access to the plans sent by the planner as output to the controller for execution through interception of the respective communication.

Inspection of retrieved domain and problem model for private information such as available operators, methods, objects, and facts, as well as user goals and plans produced in the past. Leveraging of (adversarial) plan or goal recognition techniques to predict the victim's top-level plans or goals (interests) based on its observed actions, e.g., set of previous plans or issued goals (Sukthanar, et al., 2014). This is, in principle, possible for all of the above-mentioned offline planning methods.

Attack Motivation-Consequences: Motivations of model theft and (subsequent) inference include (a) improvement of own domain model for competitive advantage; (b) increase of profit by either selling hijacked domain and problem models, or privacy-sensitive information derived from them (e.g., credit card data; other personal profile data; preferred or most recent user goals, plans and interests) to interested third parties or using them for own second-level attacks such as extortion.

Another motivation of this attack is to prepare for subsequent attacks such as (a) flooding attack of the planner with domain model-compliant planning problems, and particularly (b) model manipulation attack for which knowledge about both concrete syntax and semantics of domain and problem models is required.

Attacker Skill or Knowledge Required: The attacker has to understand the syntax and semantics of domain and problem model descriptions and the plan as output of the considered type of offline planner. Usually, these models are specified in (or translated to) standard PDDL. The skill of understanding the semantics of large, complex models in PDDL in general, and in particular for POMDP planning (cf. Section 3.3.2.1) might be considered more difficult to obtain by attackers than for classical state-based or pattern-based planners (cf. Section 3.3.1.1). Skills for an undetected interception of potentially encrypted communication between actors' data store(s) and planner API and/or hacking the data store(s) and knowledge of their type are also required.

Resources Required: The attacker requires appropriate tools for an interception of communication or remote illegal access to data store(s) of considered type.

Solutions and Mitigations: The data store(s) and communication have to be secured by appropriate authentication and cryptographic means. This applies to the complete lifecycle of the models including their development and utilization by the actors.

Attack Pattern Name: *Manipulation Output*

Attack Prerequisites: Data store with API through which stored plan files can be retrieved and updated, or digital communication line of the planner through which plan files are passed to the controller. Alternatively, the planner provides API for read-only access of its internal model after planning ends.

Description: The attacker manipulates plan files in data store or intercepted plans communicated to the controller. As a consequence, manipulated plans can get used by the actors or executed by the controller in the application environment or distributed to third-parties.

This attack breaches integrity and confidentiality (privacy) before and after planning.

Method of Attack: The attacker either (a) gains access to and then manipulates the plan files in the actors' data store or (b) reads them off via the planner API for subsequent manipulation and update of hacked data store, or (c) wiretaps the communication between planner and controller to intercept the produced plan, then to immediately manipulate it and pass the manipulated plan to the controller for execution.

Attack Motivation-Consequences: Competitors of some planning-based service provider might be interested in knowing first-hand about the most recent plans (cf. attack *Model theft*) and at the same time being able to counter-react on critical plans by manipulating them properly for own benefit. This can comprise the unnoticed execution of manipulated plans to influence plan-based processes, or the distribution of "fake" plans for ostensible solutions to customers of the provider to damage its reputation.

Attacker Skill or Knowledge Required: The attacker must have the skill to perform sound modifications of plans specified in PDDL or other formats mentioned above depending on the offline planning method used by the planner.

Resources Required: See attack pattern *Manipulation Model.*

Solutions and Mitigations: Strong authentication mechanisms help to prevent such attacks from the outside. Validation of plans received by the controller with appropriate tools mentioned above, and a visual inspection in order to detect unwanted actions in a plan before its execution.

Attack Pattern Name: *Flooding*

Attack Prerequisites: Planner provides API for its input of problem and domain models.

Description: The attacker prevents the planner from performing its planning tasks.

Method of Attack: The attacker utilises the API of the planner server to flood it with masses of planning requests in the appropriate format. A standard Denial-of-Service attack.

Attack Motivation-Consequences: This attack is purely destructive. It prevents any planning-based application from functioning properly.

Attacker Skill or Knowledge Required: Finding and initially executing this attack requires knowledge of used (actor data store - planner) client-server communication protocol and planner API.

Resources Required: None, in principle.

Solutions and Mitigations: Leveraging of two-way authentication means for all communication between the actors' data store and the planner as well as between the planner and the application (plan execution) controller, and whitelists on the planner (server) side to filter and validate any client input.

A.2 Other attack patterns for automated online planning methods

Attack Pattern Name: *Model Theft*

Attack Prerequisites: See attack pattern *Model Theft/Inference* for offline planning (cf. Section4.1.2). The online planner API additionally provides **read-only access to the full internal planning model even during planning.**

Description: The attacker gains access to all models (problem, domain, internal) and plan files even during each planning process. Inference of private information from domain and problem models and plans.

Method of Attack: See attack pattern *Model Theft/Inference* for offline planning. In addition, the attacker gains access to the internal planning model including not only the current domain and problem model versions (only initial version in offline planning) but even more information such as current plan fragments and about the specific capabilities of the planner to handle environment changes through the extended API of the online planner.

Attack Motivation-Consequences: See attack pattern *Model Theft/Inference* for offline planning. In addition, the internal model theft attack enables the attacker to prepare for the internal model and output (plan) manipulation attacks also during planning. That increases the opportunity and success chance of targeted model and output manipulations as mentioned above.

Attacker Skill or Knowledge Required: See attack pattern *Model Theft/Inference* for offline planning. Additional knowledge of extended online planner API for sending change events and retrieving internal models is required.

Resources Required: See attack pattern *Model Theft/Inference* for offline planning.

Solutions and Mitigations: The data store(s) and communication (via API) with the planner have to be secured by appropriate authentication and cryptographic means. This also applies to the complete lifecycle of the models including their development and utilization by the actors.

Attack Pattern Name: *Flooding*

Attack Prerequisites: Planner provides API for input of problem and domain models, as well as observation records with changes of the environment.

Description: The attacker prevents the system from performing online planning tasks.

Method of Attack: The attacker utilizes the API of the planner server to flood it with masses of bot only initial planning problem requests as in offline planning but additionally may flood it with faked change events during planning. A standard Denial-of-Service attack.

Attack Motivation-Consequences: This attack is purely destructive. It prevents any planning-based application from functioning properly.

Attacker Skill or Knowledge Required: See attack pattern *Manipulation Model* for online planning described above.

Resources Required and Attack Prerequisites: None.

Solutions and Mitigations: See attack pattern *Flooding* for offline planning.

Appendix B: Attack Patterns – Hybrid AI Methods

B.1 Attack patterns for Category "Informed Learning with Prior Knowledge"

Attack Pattern Name: [**MHM-ENT**] *Manipulation/hybrid system model* for system type ENT,

Case: ENT = S→N→S; **Category instance: CriSGen** (cf. Section 4.3.2)

Description: An attacker manipulates

1. the N-module model either directly or indirectly through the hybrid system interface, or
2. the S-module model that is directly connected to the hybrid system input Symbol:In either directly or indirectly through the hybrid system interface, or
3. the internal S-module model from which the S-module's output Symbol:Out is deduced indirectly via a manipulation of the model in case 2.

Methods of Attack:

1. The *direct* manipulation of the N-module model (Model:stat) is identical to the corresponding manipulation in case of **HyLEAP**. This model can also be affected indirectly through the S-module's output Symbol:Out that gets transformed into a N-modules input (Data:In). For instance, this input might repeatedly come up with identical driving scenarios.
2. The S-module model (Model:sem) that is *directly* connected to the hybrid system input Symbol:In can be *directly* manipulated through the hybrid system interface with CRUD operations on any part of the *model itself* such as the behaviour description of the dynamics, or the description of the initial state, or the definition of the undesired states. An *indirect* manipulation can be performed through a manipulation of the input scenario. Recall that the S-module's input is supposed to be a description of a scenario in which the autonomous driver behaved *well* and the purpose of the S-module is to provide a possibly similar scenario in which the former behaviour would be a fallacy. A manipulation of the semantical model might represent a scenario in which the driver ended up with a negative reward. In such a situation a lot of computational power is wasted to infer critical scenarios for an autonomous system that had learned from the former experience and has changed anyway.
3. The S-module model (Model:sem) that produces the S-module's output (Symbol:Out) can be *indirectly* manipulated through the hybrid system interface with changes of the part of the *input from the N-module*. This part concerns the N-module output (Data:Out) that are passed to the S-module as input (Symbol:In). It thus requires an appropriate manipulation of the model in 2.

Attack Motivation-Consequences: The motivation of attacking the N-module's model (Model:stat) is certainly to let the autonomous driver behave in a faulty manner. On the one hand, this could have a purely destructive purpose, but it could also be used by competitors to discredit the system of the market opponent. The consequences are, of course, that the learning system cannot fulfil its task. On the other hand, such manipulation is generally easy to detect since the underlying driving simulator will reveal the erroneous behaviour every time. Of course, this only applies if the driving simulator itself is not also manipulated accordingly. The manipulation of the S-module's semantical models (Model:Sem) have a similar purpose. Their manipulation, if not entirely destructive but goal-oriented, leads to the generation of scenarios that do not challenge the learning system and thus slows down or even prevents the actual learning process.

Attacker Skill or Knowledge Required: Goal-oriented attacks on any semantical model of the hybrid system require a deep knowledge and understanding of the structure and purpose of the models. For instance, the model connected to the S-module's input contains information about the dynamics of the (virtual) car as well as any other traffic participant. Arbitrary changes to this model would in general lead to a situation that prevents the generation of the second semantical model, and it becomes impossible to infer a suitable output. Similarly, the second semantical model reflects the original behaviour but also critical behaviours of the traffic participants. Arbitrary changes can hardly be expected as there is no direct connection between the environment and this model. It therefore can only be changed by manipulating the first semantical model. And performing this in a goal-directed manner is even harder than the manipulation of the first model alone.

Resources Required: The attacker requires appropriate tools for an interception of communication between developer or operator data store or execution controller (environment) with and remote illegal usage of the hybrid system interface.

Solutions and Mitigations: The usage of the hybrid system interface shall be secured by appropriate authentication and cryptographic means. Manipulations of one of the models might also be detected via appropriate monitoring systems. These cannot prevent the manipulations but detect their occurrence.

Attack Pattern Name: **[MHI-ENT]** *Manipulation/hybrid system input* for system type ENT,

Case: ENT = S→N→S; **Category instance: CriSGen**

Methods of Attack: For CriSGen the S-module's main input is actually the feedback loop from the N-module's output (see Figure 3.10). After all, the main purpose of the S-module is to modify a scenario that has just been processed by the N-module. The environment essentially delivers control parameters and abstraction rules that may vary from case to case. A modification of these rules could lead to erroneous abstractions such that the S-module cannot perform its major task; the output scenarios would be useless. To perform such an attack, it is necessary to gain write access (with appropriate tools) to the communication channel but also to have detailed knowledge about how parameters and abstraction rules work. Such attacks can only be countered by ensuring that the communication channels are suitably cryptographically secured. In addition, a sufficient logging mechanism can help to determine the source of the manipulation.

The input Data:In of the hybrid system, which provides basic scenarios for the N-module, could also be used to compromise the learning success of the N-module. For example, an attacker who succeeds in gaining write access to this communication point could produce trivial basic scenarios as input and thus force a very low learning success. Depending on which abstraction rules the S-module uses, it might not be able to derive new, critical scenarios from these trivial scenarios. One way to counteract this is to define and use abstraction rules that introduce additional complexity into the new scenarios to be defined.

Attack patterns [THM, THI, THO] for CriSGen. The attack methods for *stealing* any of the hybrid models, input, or output of CriSGen during training require at least read access to the hybrid system interface (cf. Section 4.1). The motivation for these attacks is to violate intellectual property rights or to prepare one of the manipulation attacks described above.

Bibliography

Abadi, M. and Gordon, A. D. 1997. Lecture Notes in Computer Science. *Reasoning about Cryptographic Protocols in the Spi Calculus.* Berlin, Heidelberg : Springer, 1997. DOI:10.1007/3-540-63141-0_5.

Alford, R. W., et al. 2009. Proceedings of the 21st International Joint Conference on Artificial Intelligence. *Translating HTNs to PDDL: A small amount of domain knowledge can go a long way.* 2009.

Almasan, R. W., et al. 2019. *Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case.* 2019. arXiv:1910.07421.

Amos-Binks, A., et al. 2017. Proceedings of the 24th IEEE Symposium on Computers and Communications. *Efficient attack plan recognition using automated planning.* 2017.

Apt, K. 2003. *Principles of constraint programming.* Cambridge : Cambridge University Press, 2003. DOI:10.1017/CBO9780511615320.

Baader, F., et al. 2017. *An Introduction to Description Logic.* Cambridge : Cambridge University Press, 2017. DOI:10.1017/9781139025355.

—. 2007. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge : Cambridge University Press, 2007. DOI:10.1017/CBO9780511711787.

Bai, H., et al. 2015. Proceedings of the 32nd IEEE International Conference on Robotics and Automation. *Intention-aware online POMDP planning for autonomous driving in a crowd.* 2015. DOI:10.1109/ICRA.2015.7139219.

Barnum, A. S. S. 2013. [Online] 2013. [Cited: December 9, 2022.] https://www.cisa.gov/uscert/bsi/articles/knowledge/attack-patterns/introduction-to-attack-patterns.

Barták, R., Salido, M. A. and Rossi, F. 2010. Constraint satisfaction techniques in planning and scheduling. *Journal of Intelligent Manufacturing.* 2010.

Battaglia, P. W., et al. 2018. *Relational inductive biases, deep learning, and graph networks.* 2018. arXiv:1806.01261.

Berners-Lee, T., Hendler, J. A. and Lassila, O. 2001. Scientific American. *The Semantic Web.* 2001.

Bertoli, P., Cimatti, A. and Traverso, P. 2004. Proceedings of the Eurpean Conference on Artificial Intelligence. . *Interleaving Execution and Planning for Nondeterministic, Partially Observable Domains.* 2004.

Biere, A., Heule, M. and van Maaren, H. 2009. *Handbook of Satisfiability.* Amsterdam : IOS Press, 2009. ISBN: 9781607503767.

Biggio, B., et al. 2014. Support Vector Machines Applications. *Security evaluation of support vector machines in adversarial environments.* 2014. DOI:10.1007/978-3-319-02300-7_4.

Biggio, B., Nelson, B. and Laskov, P. 2012. Proceedings of the 29th International Conference on Machine Learning. *Poisoning attacks against support vector machines.* 2012. arXiv:1206.6389.

Blum, A. and Furst, M. L. 1997. Artificial Intelligence Journal. *Fast planning through planning graph analysis.* s.l. : Elsevier, 1997. DOI:10.1016/S0004-3702(96)00047-1.

Bozic, J. and Wotawa, F. 2020. Software Quality Journal. *Planning-based security testing of web applications with attack grammars.* 2020. DOI:10.1007/s11219-019-09469-y.

Brachman, R. and Leveque, H. 2004. The Morgan Kaufman Series in Artificial Intelligence. *Knowledge Representation and Reasoning.* 2004. ISBN: 9781558609327.

Brendel, W., Rauber, J. and Bethge, M. 2018. Conference Paper at the 6th International Conference on Learning Representations. *Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models.* 2018. arXiv:1712.04248.

Brewitt, C., et al. 2021. IEEE/RSJ International Conference on Intelligent Robots and Systems. *GRIT: Fast, Interpretable, and Verifiable Goal Recognition with Learned Decision Trees for Autonomous Driving.* 2021. arXiv:2103.06113.

Chen, F., et al. 2020. APSIPA Transactions on Signal and Information Processing. *Graph representation learning: a survey.* 2020. DOI:10.1017/ATSIP.2020.13.

Cheng, M., et al. 2020. 2020 International Conference on Learning Representation. *Sign-OPT: A Query-Efficient Hard-label Adversarial Attack.* 2020. arXiv:1909.10773.

Ciravegna, G., et al. 2020. Proceedings of the 29th International Joint Conference on Artificial Intelligence. *Human-Driven FOL Explanations of Deep Learning.* 2020. DOI:10.24963/ijcai.2020/309.

Cohen, W. W., et al. 2020. Proceedings of the 8th International Conference of Legal Regulators. *Scalable neural methods for reasoning with a symbolic knowledge base.* 2020. arXiv:2002.06115.

Cook, S. A. 1971. Proceedings of the 3rd annual ACM Symposium on Theory of Computing. *The complexity of theorem-proving procedures.* 1971. DOI:10.1007/978-3-030-59713-9_62.

Cosmo, L. and al., et. 2020. Proceedings of the 23rd Medical Image Computing and Computer Assisted Intervention. *Latent-Graph Learning for Disease Prediction.* 2020. DOI:10.1007/978-3-030-59713-9_62.

Cropper, A. and Dumančić, S. 2022. Journal of Artificial Intelligence Research. *Inductive logic programming at 30: a new introduction.* 2022. DOI:10.1613/jair.1.13507.

Currie, K. and Tate, A. 1991. Artificial Intelligence Journal. *O-Plan: the open planning architecture.* 1991. DOI:10.1016/0004-3702(91)90024-E.

Da Cruz, C. G. R., Ferreira, M. G. V. and Silva, R. R. 2020. Proceedings of the 22nd International Conference on Enterprise Information Systems. *State Validation in Automated Planning.* 2020. DOI:10.5220/0009411903960406.

Davis, M. and Putnam, H. 1960. Journal of the ACM. *A computing procedure for quantification theory.* 1960. DOI:10.1145/321033.321034.

Davis, M., Logemann, G. and Loveland, D. 1962. Communications of the ACM. *A machine program for theorem-proving* . 1962. DOI:10.1145/368273.368557.

DBPedia. 2021. [Online] 2021. [Cited: December 9, 2022.] https://www.dbpedia.org.

De Giacomo, G., et al. 2019. Proceedings of International Conference on Automated Planning and Scheduling. *Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications.* 2019. arXiv:1807.06333.

De, S. and Chakraborty, B. 2018. Proceedings of the 3rd International Conference for Convergence in Technology. *Case Based Reasoning (CBR) Methodology for Car Fault Diagnosis System (CFDS) Using Decision Tree and Jaccard Similarity Method.* 2018. DOI:10.1109/I2CT.2018.8529695.

Dechter, R. and Cohen, D. 2003. The Morgan Kaufmann Series in Artificial Intelligence. *Constraint processing.* 2003. ISBN: 9781558608900.

Deng, Y., et al. 2021. IEEE Transactions on Industrial Informatics. *Deep Learning-Based Autonomous Driving Systems: A Survey of Attacks and Defenses.* 2021. DOI:10.1109/TII.2021.3071405.

Derks, I. P. and de Waal, A. 2020. SACAIR 2021: Artificial Intelligence Research. *A Taxonomy of Explainable Bayesian Networks.* 2020. DOI:10.1007/978-3-030-66151-9_14.

Domshlak, C. and Hoffmann, J. 2007. Journal of Artificial Intelligence Research. *Probabilistic planning via heuristic forward search and weighted model counting.* 2007. DOI:10.1613/jair.2289.

Donadello, I., Serafini, L. and d'Avila Garcez, A. 2017. Proceedings of the 26th International Joint Conference on Artificial Intelligence. *Logic tensor networks for semantic image interpretation.* 2017. arXiv:1705.08968.

Drews, S., Albarghouthi, A. and d'Antoni, L. 2020. Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. *Proving data-poisoning robustness in decision trees.* 2020. DOI:10.1145/3385412.3385975 .

Eykholt, K. and al., et. 2018. Proceedings of the 2018 IEEE/CVF conference on computer vision and pattern recognition. *Robust physical-world attacks on deep learning visual classification.* 2018. DOI:10.1109/CVPR.2018.00175.

Fandinno, J., et al. Theory and Practice of Logic Programming. *Planning with incomplete information in quantified answer set programming.* DOI:10.1017/S1471068421000259.

Ferber, P., et al. 2021. Proceedings of the 2nd Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning. *Neural Network Heuristic Functions for Classical Planning: Reinforcement Learning and Comparison to ther Methods.* 2021.

Fourati, F., Bhiri, M. T. and Robbana, R. 2016. Proceedings of the 5th International Conference on Multimedia Computing and Systems. *Verification and validation of PDDL descriptions using Event-B formal method.* 2016. DOI:10.1109/ICMCS.2016.7905631.

Fredrikson, M., Jha, S. and Ristenpart, T. 2015. Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. *Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures.* 2015. DOI:10.1145/2810103.2813677.

Gamma, E. and al., et. 1995. *Design Patterns - Elements of Reusable Object-Oriented Software.* Boston : Addison-Wesley Professional, 1995. ISBN: 0201633612.

Garnelo, M., Arulkumaran, K. and Shanahan, M. 2016. *Towards deep symbolic reinforcement learning.* 2016. arXiv:1609.05518.

Geibel, P. 2006. Machine Learning: ECML 2006. *Reinforcement Learning for MDPs with Constraints.* 2006. DOI:10.1007/11871842_63.

Georgievski, I. and Aiello, M. 2015. Artificial Intelligence Journal. *HTN planning: Overview, comparison, and beyond.* 2015. DOI:10.1016/j.artint.2015.02.002.

Georgievski, I. 2015. Coordinating services embedded everywhere via hierarchical planning. 2015. ISBN: 978-90-367-8148-0.

Ghallab, M., Nau, D. and Traverso, P. 2016. *Automated planning and acting.* Cambridge : Cambridge University Press, 2016. DOI:10.1017/CBO9781139583923.

—. 2004. *Automated Planning: theory and practice.* s.l. : Elsevier, 2004. ISBN: 9781558608566.

Goldman, R. P. and Kuter, U. 2019. Proceedings of the 12th European Lisp Symposium. *Hierarchical Task Network Planning in Common Lisp: the case of SHOP3.* 2019. DOI:10.5281/zenodo.2633324.

Gomes, C. P., et al. 2008. Handbook of Knowledge Representation. *Satisfiability Solvers.* 2008. ISBN: 9780444522115.

Greenwald, L. and Shanley, R. 2009. Proceedings of the 28th Military Communications Conference. *Automated planning for remote penetration testing.* 2009. DOI:10.1109/MILCOM.2009.5379852.

Gulwani, S., et al. 2015. Communications of the ACM. *Inductive programming meets the real world.* 2015. DOI:10.1145/2736282.

Guo, C. and al., et. 2019. Proceedings of the 36th International Conference on Machine Learning. *Simple Black-box Adversarial Attacks.* 2019. arXiv:1905.07121.

Hamilton, W. 2020. Synthesis Lectures on Artificial Intelligence and Machine Learning. *Graph Representation Learning.* 2020. DOI:10.1007/10.2200/S01045ED1V01Y202009AIM046.

Havey, M. 2005. *Essential Business Process Modeling.* Sebastopol : O'Reilly Media, 2005. ISBN: 9780596008437.

Helmert, M. 2006. Journal of Artificial Intelligence Research. *The fast downward planning system.* 2006. See also: https://www.fast-downward.org. DOI:10.1613/jair.1705.

Helmert, M. 2009. Artificial Intelligence Journal. *Concise finite-domain representation for PDDL planning tasks.* 2009. DOI:10.1016/j.artint.2008.10.013.

Hoare, C. A. R. 2004. Prentice Hall International Series in Computer Science. *Communicating Sequential Processes.* 2004. ISBN: 9780131532892.

Hoffmann, J. and Brafman, R. 2006. Artificial Intelligence Journal. *Conformant planning via heuristic forward search: A new approach.* 2006. DOI:10.1016/j.artint.2006.01.003.

Hoffmann, J. and Nebel, B. 2001. Journal of Artificial Intelligence Research. *The FF planning system: Fast plan generation through heuristic search.* 2001. DOI:10.1613/jair.855.

Höller, D. and al., et. 2020. Proceedings of the 34th AAAI Conference on Artificial Intelligence. *HDDL: An extension to PDDL for expressing hierarchical planning problems.* 2020. DOI:10.1609/aaai.v34i06.6542.

Howey, R., Long, D. and Fox, M. 2004. Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence. *VAL: Automatic plan validatino, continous effects and mixed initiative planning using PDDL.* 2004. See also: https://nms.kcl.ac.uk/planning/software/val.html. DOI:10.1109/ICTAI.2004.120.

Illanes, L., et al. 2020. Proceedings of the 30th International Conference on Automated Planning and Scheduling. *Symbolic plans as high-level instructions for reinforcement learning.* 2020. DOI:10.1609/icaps.v30i1.6750.

Jegelka, S. 2022. Theory of Graph Neural Networks: Representation and Learning. 2022. arXiv:2204.07697.

Ji, S. and al., et. 2021. IEEE Transactions on Neural Networks and Learning Systems. *A survey on knowledge graphs: Rpresentation, acquistion, and applications.* 2021. DOI:10.1109/TNNLS.2021.3070843.

Kashifi, M. T. and Ahmad, I. 2022. Transportation Research Record: Journal of the Transportation Research Board. *Efficient Histogram-Based Gradient Boosting Approach for Accident Severity Prediction with Multisource Data.* 2022. DOI:10.1177/03611981221074370.

Klusch, M. and Gerber, A. 2006. Proceedings of the 2006 European Conference on Web Services. *Fast composition planning of owl-s services and application .* 2006. DOI:10.1109/ecows.2006.20.

Klusch, M. and Renner, K. U. 2006. Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology Workshops. *Fast dynamic re-planning of composite OWL-s services.* 2006. DOI:10.1109/WI-IATW.2006.72.

Klusch, M. 2008. CASCOM: Intelligent Service Coordination in the Semantic Web. *Semantic Web Service Coordination.* 2008. DOI:10.1007/978-3-7643-8575-0_4.

—. 1993. Proceedings of 1993 International Conference on Neural Networks. *A hybrid neural system and its use for the classification of stars.* 1993. DOI:10.1109/ijcnn.1993.714007.

Knoblock, C. A. 1995. Proceedings of the 14th International Joint Conference on Artificial Intelligence. *Planning, Executing, Sensing, and Replanning for Information Gathering.* 1995.

Kurniawati, H. and al., et. 2011. The International Journal of Robotics Research. *Motion planning under uncertainty for robotic tasks with long time horizons.* 2011. DOI:10.1177/0278364910386986.

Kurniawati, H., Hsu, D. and Lee, W. S. 2008. Proceedings of the 4th International Conference on Robotics: Science and Systems. *SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces.* 2008. DOI:10.15607/RSS.2008.IV.009.

Lauritzen, S. L. and Spiegelhalter, D. J. 1988. Journal of the royal statistical society series b-methodological. *Local computations with probabilities on graphical structures and their application to expert systems.* 1988. DOI:10.1111/j.2517-6161.1988.tb01721.x.

Li, X., et al. 2019. Proceedings of the 2020 IEEE International Conference on Acoustics, Speech and Signal Processing. *Advknn: Adversarial attacks on k-nearest neighbor classifiers with approximate gradients.* 2019. arXiv:1911.06591.

Lopez, A. and Bacchus, F. 2003. Proceedings of the 18th International Joint Conference on Artificial Intelligence. *Generalizing GraphPlan by formulating planning as a CSP.* 2003.

Luo, Y. and al., et. 2019. The International Journal of Robotics Research. *Importance sampling for online planning under uncertainty.* 2019. DOI:10.1177/0278364918780322.

Ma, T. and Zhang, A. 2018. Proceedings of the 2018 IEEE International Conference on Bioinformatics and Biomedicine. *Multi-view Factorization AutoEncoder with Network Constraints for Multi-omic Integrative Analysis.* 2018. DOI:10.1109/BIBM.2018.8621379.

Manhaeve, R., et al. 2018. Proceedings of the 32nd Conference on Neural Information Processing Systems. *DeepProbLog: Neural Probabilistic Logic Programming.* 2018. arXiv:1805.10872.

Marra, G., et al. 2020. Proceedings of the 1st International Workshop on New Foundations for Human-Centered AI co-located with 24th European Conference on Artificial Intelligence. *Inference in Relational Neural Machines.* 2020.

Mazzola, L. and al., et. 2016. Proceedings of the 8th International Conference on Knowledge Engineering and Ontology Development. *CDM-Core: A Manufacturing Domain Ontology in OWL2 for Production and Maintenance.* 2016. DOI:10.5220/0006056301360143.

Mazzola, L., Kapahnke, P. and Klusch, M. 2018. International Journal of Web Information Systems. *Semantic composition with ODERU.* 2018. DOI:10.1108/IJWIS-05-2018-0038.

Meli, D., Sridharan, M. and Fiorini, P. 2021. Machine Learning. *Inductive learning of answer set programs for autonomous surgical task planning.* 2021. DOI:10.1007/s10994-021-06013-7.

Micheli, A. and Valentini, A. 2021. Proceedings of the 35th AAAI Conference on Artificial Intelligence. *Synthesis of Search Heuristics for Temporal Planning via Reinforcement Learning.* 2021. DOI:10.1609/aaai.v35i13.17413.

Milner, R. 1999. Science of Computer Programming. *Communicating and Mobile Systems: The π-calculus.* 1999. DOI:0.1016/S0167-6423(00)00008-3.

Möller, T., et al. 2006. Health Informatics Journal. *Next-generation applications in healthcare digital libraries using semantic service compostion and coordination.* 2006. DOI:10.1177/1460458206063802..

Muggleton, S. and al., et. 2018. Machine Learning. *Ultra-strong machine learning: comprehensibilty of programs learned with ILP.* 2018. DOI:10.1007/s10994-018-5707-3.

Muggleton, S. 1991. New generation computing. *Inductive logic programming.* 1991. DOI:10.1007/BF03037089.

Müller, C. 2005. PhD Thesis. *Zweistufige kontextsensitive Sprecherklassifikation am Beispiel von Alter und Geschlecht [two-layered context-sensitive speaker classification on the example of age and gender.* 2005.

Muscholl, N. and al., et. 2020. 2020 IEEE Symposium Series on Computational Intelligence. *SIMP3: Social Interaction-Based Multi-Pedestrian Path Prediction By Self-Driving Cars.* 2020. DOI:10.1109/SSCI47803.2020.9308130.

Nau, D. and al., et. 2005. Intelligent Systems. *Applications of SHOP and SHOP2.* 2005. DOI:10.1109/MIS.2005.20.

Nethercote, N. and al., et. 2007. Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming. *MiniZinc: Toward A Standard CP Modelling.* 2007. DOI:10.1007/978-3-540-74970-7_38.

Nonnengart, A., Klusch, M. and Müller, C. 2019. Proceedings of the 23rd International Symposium on Formal Methods for Autonomous Systems. *CriSGen: Constraint-Based Generation of Critical Scenarios for Autonomous Vehicles.* 2019. DOI:10.1007/978-3-030-54994-7_17.

Olesen, K. G., Lauritzen, S. L. and Jensen, F. V. 1992. Uncertainty in Artificial Intelligence. *aHUGIN: A System Creating Adaptive Causal Probabilistic Networks.* 1992. DOI:10.1016/B978-1-4832-8287-9.50035-9.

O'Toole, S., et al. 2022. Proceedings of the 15th International Symposium on Combinatorial Search. *Sampling from Pre-Images to Learn Heuristic Functions for Classical Planning.* 2022. arXiv:2207.03336.

Pacaci, A. und Tamer Özsu, M. 2019. International Conference on Management of Data, ACM. *Experimental Analysis of Streaming Algorithms for Graph Partitioning.* 2019. DOI:10.1145/3299869.3300076 .

Palacios, H. and Geffner, H. 2009. Journal of Artificial Intelligence Research, AI Access Foundation. *Compiling uncertainty away in conformant planning problems with bounded width.* 2009. DOI:10.1613/jair.2708.

Parimi, V. 2021. PhD Thesis, Carnegie Mellon University. *T-HTN: Timeline based HTN Planning for Multi-Agent Systems.* 2021.

Pasaoglu, C., et al. 2016. Proceedings of the Institution of Mechanical Engineers, SAGE Publishing. *Hybrid systems modeling and automated air traffic control for three-dimensional separation assurance.* 2016. DOI:10.1177/0954410015619443.

Paulheim, H. 2016. Semantic Web. *Knowledge graph refinement: A survey of approaches and evaluation methods.* 2016. DOI:10.3233/SW-160218.

Peot, M. A. 1998. Thesis, Standford University. *Decision-theoretic planning.* 1998.

Phan, N., et al. 2015. Proceedings of 6th ACM Conference on bioinformatics, computational biology and health informatics, ACM. *Ontology-based deep learning for human behavior prediction in health social networks.* 2015. DOI:10.1016/j.ins.2016.08.038.

Pozanco, A., et al.. 2021. Proceedings of the 31st International Conference on Automates Planning and Scheduling. *Proving Security of Cryptographic Protocols using Automated Planning.* 2021.

Prakken, H. 2013. Law and Philosophy Library. *Logical Tools for Modelling Legal Argument: A Study of Defeasible Reasoning in Law.* 2013. DOI:10.1007/978-94-015-8975-8.

Protegé Ontology Editor. 2021. [Online] 2021. [Cited: December 2022, 9.] https://protege.stanford.edu.

Pryor, L. and Collins, G. 1996. Artificial Intelligence Research, AI Access Foundation. *Planning for contingencies: A decision-based approach.* 1996. DOI:10.1613/jair.277.

Pusse, F. and Klusch, M. 2019. Proceedings of IEEE Intelligent Vehicles Symposium (IV), IEEE. *Hybrid online planning and deep reinforcement learning for safer self-driving cars.* 2019. DOI:10.1109/IVS.2019.8814125.

Putra, F., et al. 2020. Indonesian Journal of Computing and Cybernetics Systems, IndoCEISS. *HOG Feature Extraction and KNN Classification for Detecting Vehicle in the Highway.* 2020. DOI:10.22146/ijccs.54050.

Quinlan, J. R. 1990. Machine learning. *Learning logical definitions from relations.* 1990. DOI:10.1023/A:1022699322624.

Rahayu, N. W., Ferdiana, R. and Kusumawardani, S. S. 2022. Computers and Education: Artificial Intelligence. *A systematic review of ontology use in E-Learning recommender system.* 2022. DOI:10.1016/j.caeai.2022.100047.

Rintanen, J. 2012. Journal of Artificial Intelligence. *Planning as satisfiability: Heuristics.* 2012. DOI:10.1016/j.artint.2012.08.001.

Roberts, M., et al. 2011. Proceedings of the 3rd International JCAI Workshop on Intelligent Security. *Personalized vulnerability analysis through automated planning.* 2011.

Rossi, F., van Beek, P. and Walsh, T. 2006. Foundations of Artificial Intelligence. *Handbook of Constraint Programming.* 2006. ISBN: 9780444527264.

Sarker, M. K., et al. 2020. KGSWC 2020: Knowledge Graphs and Semantic Web. *Wikipedia Knowledge Graph for Explainable AI.* 2020. DOI:10.1007/978-3-030-65384-2_6.

—. 2017. Proceedings of the 12th International Workshop on Neural-Symbolic Learning and Reasoning. *Explaining trained neural networks with semantic web technologies: First steps.* 2017. arXiv:1710.04324.

Selman, B., Leveque, H. J. and Mitchell, D. G. 1992. Proceedings of the 10th National Conference on Artificial Intelligence. *A new method for solving hard satisfiability problems.* 1992.

Serafini, L. and Garcez, A. D. 2016. *Logic tensor networks: Deep learning and logical reasoning from data and knowledge.* 2016. arXiv:1606.04422.

Shao, T. and et al. 2021. Knowledge-Based Systems. *The hierarchical task network planning method based on Monte Carlo Tree Search.* 2021. DOI:10.1016/j.knosys.2021.107067.

Shlomi, J., Battaglia, P. and Vilmant, J. R. 2021. Machine Learning: Science and Technology. *Graph neural networks in particle physics.* 2021. DOI:10.1088/2632-2153/abbf9a.

Shrinah, A. and et al. 2021. *D-VAL: An automatic functional equivalence validation tool for planning domain models.* 2021. arXiv:2104.14602.

Silva, M. C. and et al. 2022. Cancers. *Ontologies and Knowledge Graphs in Oncology Research.* 2022. DOI:10.3390/cancers14081906.

Silver, D. and et al. 2017. Nature. *Mastering the game of Go without human knowledge.* 2017. DOI:10.1038/nature24270.

Silver, D. and Veness, J. 2010. *Monte-Carlo planning in large POMDPs.* 2010.

Singhal, A. 2012. *Introducing the Knowledge Graph: things, not strings.* [Online] 2012. [Cited: December 2022, 9.] https://blog.google/products/search/introducing-knowledge-graph-things-not.

Smith, D. E. and Weld, D. S. 1998. Proceedings of the 10th International Conference on Artificial Intelligence,. *Conformant GraphPlan.* 1998.

SNOMED CT. 2022. [Online] 2022. [Cited: December 9, 2022.] https://www.snomed.org.

Speck, D., et al. 2021. Proceedings of International Conference on Automated Planning and Scheduling. *Learning heuristic selection with dynamic algorithm configuration.* 2021. arXiv:2006.08246.

Staab, S. and Studer , R. 2010. International Handbook on Information Systems. *Handbook on ontologies.* 2010. DOI:10.1007/978-3-540-92673-3.

Sukthanar, G. and al., et. 2014. *Plan, activity, and intent recognition: Theory and practice.* 2014. DOI:10.1016/C2012-0-01169-6.

Tang, X. and al., et. 2011. Proceedings of 7th IEEE International Conference on Semantics, Knowledge and Grids. *A planning engine based on SHOP2 for AFLOW.* 2011. DOI:10.1109/SKG.2011.22.

Toro Icarte, R., et al. 2018. Proceedings of 17th International Conference on Autonomous Agents and Multi-Agent Systems. *Teaching multiple tasks to an RL agent using LTL.* 2018.

Truex, S., et al. 2019. IEEE Transactions on Services Computing. *Demystifying Membership Inference Attacks in Machine Learning as a Service.* 2019. arXiv:1807.09173.

Tsang, E. 2014. Foundations of constraint satisfaction. 2014. DOI:10.1016/C2013-0-07627-X.

Valentini, A., Micheli, A. and Cimatti, A. 2020. Proceedings of the 34th AAAI Conference on Artificial Intelligence,. *Temporal planning with intermediate conditions and effects.* 2020. DOI:10.1609/AAAI.V34I06.6553.

van Bekkum, M. and al., et. 2021. Applied Intelligence. *Modular design patterns for hybrid learning and reasoning systems.* 2021. arXiv:2102.11965.

van Harmelen, F. und Teije, A. 2019. Journal of Web Engineering. *A boxology of design patterns for hybrid learning and reasoning systems.* 2019. DOI:10.13052/jwe1540-9589.18133 .

Vlachos, A., Perifanou, M. and Economides, A. A. 2022. Journal of Cultural Heritage Management and Sustainable Development. *A review of ontologies for augmented reality cultural heritage applications.* 2022. DOI:10.1108/JCHMSD-06-2021-0110.

W3C Recommendation. 2012. OWL 2 Web Ontology Language: W3C Recommendation. [Online] 2012. [Cited: December 9, 2022.] https://www.w3.org/TR/owl2-overview.

—. 2004. OWL Web Ontology Language: W3C Recommendation. [Online] 2004. [Cited: 10 22, 18.] https://www.w3.org/TR/owl-ref.

Wallace, M. 2020. *Building Decision Support Systems Using MiniZinc.* 2020. https://doi.org/10.1007/978-3-030-41732-1 .

Wang, H., Dou, D. and Lowd, D. 2016. Proceedings of the 27th International Conference on Database and Expert Systems Applications. *Ontology-based deep restricted Boltzmann machine.* 2016. DOI:10.1007/978-3-319-44403-1_27.

Weichert, D. 2021. *Monte Carlo Tree Search for high precision manufacturing.* 2021. arXiv:2108.01789.

Wilkins, D. E. 1999. *Using the SIPE-2 planning system.* 1999.

Yang, Y., et al. . 2022. Proceedings of the AAAI Conference on Artificial Intelligence. *LOGICDEF: An Interpretable Defense Framework Against Adversarial Examples via Inductive Scene Graph.* 2022. DOI:10.1609/aaai.v36i8.20865.

Younes, H. L. S. und Littman, M. L. 2004. Technical Report. *PPDDL 1.0: an extension to PDDL for expressing planning domains with probabilistic effects.* 2004.

Zhang, C., Zhang, H. und Hsieh, C. J. 2020. Proceedings of the 34th Conference on Neural Information Processing Systems. *An efficient adversarial attack for tree ensembles.* 2020. arXiv:2010.11598 .

Zhang, W. und al., et. 2022. *Knowledge Graph Reasoning with Logics and Embeddings: Survey and Perspective.* 2022. arXiv:2202.07412.

Zhou, J., et al. 2018. *Graph neural networks: A review of methods and applications.* 2018. arXiv:1812.08434v1.

Zhu, Z. and Du, W. 2010. Proceedings of the 24th IFIP Annual Conference on Data and Applications Security and Privacy. *Understanding privacy risk of publishing decision trees.* 2010. DOI:10.1007/978-3-642-13739-6_3 .

List of Figures

List of Tables