Probabilistic causal logic in discrete event training simulations

by
Joost Kraaijeveld

Bachelor Thesis

Department of Computing and Information Sciences
Radboud University Nijmegen

Supervisor: Prof. dr. Peter Lucas

Summer of 2011

Nijmegen The Netherlands

Abstract

This thesis answers the research question whether it is possible to use a first-order probabilistic causal logic as the event handling logic in a discrete event training simulation. Discrete event simulations are very useful for the training of staff in organisations. Such simulations require a great resemblance with the environment they simulate. The environment is dynamic, partially observable, probabilistic and goal or utility driven. Using first-order probabilistic causal logic as the event handling logic in a discrete event training simulation can contribute to that resemblance. Based on the OMG MDA framework, AORSL is a discrete event simulation language that uses agents in its modelling. ICL is a framework that is capable of modelling agents under uncertainty using logic programming, probability, game and decision theory. It is a high-level language that makes use of other embedded formalisms. CPL, a first-order probabilistic causal logic, is integrated and unified with ICL as embedded formalism. AORSL and ICL then are integrated to create an environment in which discrete event training simulations can be specified. Through this integration the answer to the research question is affirmative: first-order probabilistic causal logic can be used as the event handling logic in a discrete event training simulation.

Contents

List of Tables

List of Figures

1. Introduction

After each incident the Dutch Safety Board appears to draw the inevitable conclusion: the handling of the incident was not up to standard. Insufficient preparation, insufficient training, bad decision-making and bad procedures by disaster relief organisation and its personnel are seen as possible causes. In general there are enough possibilities for the training and preparation of the lower staff. It varies from regular training to advanced in-the-loop simulations. For the higher staff however, there are not so many possibilities for training. And it is on the higher level that most of the shortcomings are present.

For higher staff the emphasis of the training is on procedures and cognitive skills in much more abstract situations, which is by its nature much more difficult for both the trainees and trainers. The tasks of higher staff consist mainly of making decisions using domain dependent rules, influenced by external information. The task environment is very dynamic and changes continuously. The uncertainty is very high: not all facts are available at all times and have to be guessed. Often they change or are even retracted. Often decisions must be made with incomplete or even erroneous information. New events happen all the time and whenever a new event happens all the facts have to be (re)checked and new decisions must be taken. In this process people are dependent on each other for information and have to work together to achieve their individual or common goals. To manage the dynamic and complicated environment and to be independent of incidental decision making elaborate procedures and decision rules are established and must be adhered to. Often the procedure and rules are "default" procedures and rules, and it is left to the people to interpret those according to specific circumstances.

Simulation could help with these problems. Simulations can be used to imitate the task environment an train the people in that simulated task environment. But the characteristics of the task environment are challenging for the development of training simulations. If the simulation should imitate the task environment it is logical to model the simulation in terms of the task characteristics: very dynamic, with uncertainty, with events, communicative, information driven and centred around procedures and decision rules. Not every type of simulation is suited for training simulations. Discrete event simulations appear to be very well suited for this task.

The basic idea of a discrete event simulation is that of an "event pump", a mechanism with which events are distributed to rules that handle those events, generating new events which are distributed to rules and so forth. The rules have the form of "if this event has happened and the world looks like this, then that action will be taken". They contain the logic that drives the simulation. In this thesis the emphasis will be on the logic that is used to handle the events. There are many types of logic that could be used.

In this thesis I will look into first-order probabilistic causal logics.

The research question in this thesis is:

Is it possible to use a first-order probabilistic causal logic as the event handling logic in a discrete event training simulation?

This question is studied in the following way. In chapter 2, I introduce the concept of training and the requirements that training has for a simulation. In chapter 3, a particular implementation of a discrete event simulation that uses the concept of agents to model simulations is presented. Chapter 4 introduces a framework that uses first-order probabilistic logic to model agents in a uncertain world. Two possible logics that could be embedded in the framework will be discussed and one logic will be chosen. In chapter 5, I will try to integrate the framework with its first-order probabilistic causal logic in the discrete event simulation implementation. In chapter 6, a conclusion is drawn on the suitability and possibility of the integration and I will suggest some further research to enhance the solution. The result of the research underlying this thesis is that it appears indeed possible to use a first-order probabilistic causal logic as the event handling logic in a discrete event training simulation.

2. Training

Most people are familiar with the terms education and training. As Romiszowski [24, p. 3] notes

> *"...training is akin to following a tightly fenced path, in order to reach a predetermined goal at the end of it. Education is to wander freely in the fields to left and right of this path — preferably with a map. "*

The line between training and education it not always clear or relevant. Most training will lead to some unplanned learning (education) and most education will involve almost always some goal-oriented teaching (training). In an organisational setting, training is about the job or task performance in the current job, education is about preparation for job or task performance in possible future jobs.

In this thesis I will only look at training that is aimed at improving the performance of individuals and groups in their current job in an organisational setting. A job is a collections of tasks that are performed by a single person. A task is a responsibility to achieve a goal by performing activities within a (predefined) period of time. In the remainder of the thesis I will only speak of task(s) and not of jobs.

Before looking at the nature of the tasks there are a few characteristics of the task environment that are important with regards to the training of the higher staff. The scope of the tasks is that of disaster relief. Disasters do not happen often but if they happen they have a great impact on society and the people involved. The environment in which a disaster happens is large and complex. Many people, materials and complex natural phenomena are involved. The financial and human costs involved are high. The tasks of higher staff, decision making under difficult circumstances, are critical in disaster relief. There is no time for learning or trial and error during task performance. This means that the workers should always be capable of performing their tasks. And, if the environment changes, the workers should remain capable of performing their task. Hence, training for higher staff is of utmost importance, both in preparation for a job and during the job performance itself. If the task performance is critical, then so is training.

When designing and implementing task-centred training, the intuition is that the task characteristics define the underlying knowledge and skills that a person should have to perform the task. A further intuition is that the characteristics of the knowledge and skills define the instructional design.

The tasks of the higher staff consist mainly of making decisions using domain dependent rules and influenced by external information. The task environment is very dynamic and changes continuously. The uncertainty about the facts in the environment is very high: not all facts are available at all times and they have to be guessed. They often change or are even retracted. Often decisions must be made with incomplete or even erroneous information. New events happen all the time and whenever a new event happens all the facts have be (re)checked and new decisions must be taken. In this process people are dependent on each other for information and have to work together to achieve their individual or common goals. To manage the dynamic and complicated environment and to be independent of incidental decision-making elaborate procedures and decision rules are established and must be adhered to. Often the procedures and rules are "default" procedures and rules and it is left to the people to interpret according to specific circumstances.

To summarise the characteristics of the tasks:

- The environment is complicated in terms of people, materials and natural phenomena.
- The whole environment changes often and is dynamic.
- The environment changes continuously.
- Not all facts are known, implying a partially observable world.
- Facts are observed with noise and can be retracted, making the world non-monotone.
- The unknown facts have to be guessed, making the facts probabilistic.
- Events happen all the time in a sequential fashion.
- People are dependent on each other and have to work together.
- Extensive procedures and rules are used.
- Procedures and rules must be adapted to situations, procedures, concepts, principles.
- Actions do not always have the expected effect and are thus uncertain and probabilistic.

When designing training for tasks, it is important to look at the type of task. More in particular, it is important to look at the required knowledge and skills for the performance of the task. Romiszowski [24] describes a taxonomy for the training environment. The main categories in Romiszowski's taxonomy are knowledge and skills. Each main category has subcategories of its own.

Knowledge is divided into:

- Facts: details concerning concrete events, situations, people or matters.
- Procedures: assignments that consist of a step plan.
- Concepts: definitions of abstract matters or grouping of perceptible objects or matters.

- Principles: rules or patterns.

These categories of knowledge yield a gliding scale of difficulty. Each category is more difficult to understand, apply and transfer than its predecessor.

Skills are divided into:

- Cognitive skills: decision-making, problem-solving, logical thinking.
- Psycho-motor skills: physical action, perceptual acuity.
- Reactive skills: dealing with oneself, attitudes, feelings, habits, self-control
- Interactive skills: dealing with others.

Romiszowski states that a person can perform these skills in a reproductive and in a productive manner. Reproductive means applying procedures or algorithms, productive means applying principles and strategies. Performing on a productive level is more difficult than on a reproductive level.

In every instructional system all elements of Romiszowski's taxonomy are present but their importance in that system varies.

As can be seen in the characterisation, the tasks of higher staff demand a lot of both knowledge and skills. The tasks consist mainly of making decisions using procedures and rules in a dynamic, communicative and eventful environment. Almost all elements of both the knowledge and skill taxonomies are present. All elements of the knowledge taxonomy are important and required in the task environment. They are all extensively used during the task performance. Especially because there is often a lack of objective facts during the actual performing of the task the actual application of procedures, concepts and principles is important. From the skills, the cognitive and interactive skills are especially important. In some circumstances, the reactive skills are important, especially when the staff has to perform under pressure. Psycho-motor skills are mostly irrelevant because the core activities are concerned with decision making and not performing complex motor tasks.

The effectiveness of an instructional system is determined by transfer and retention. Patrick [16] mentions that from research is known that both transfer and retention are difficult in general and for cognitive procedural skills in particular.

Transfer is the ability of persons to effectively apply the knowledge and skills they have learned in one situation to another situation, which could be e.g. another training situation or a job performance situation. The intuition is that the transfer depends on how much two tasks are the same. But as Patrick [16, p. 91] notes:

> *"unfortunately, there is considerable evidence that positive transfer does not always occur between tasks, even when, from an 'identical elements' perspective, it might be expected tot do so. "*

Especially

> *"lack of positive transfer has been found primarily between cognitive tasks involving the transfer of both specific and general problem-solving methods and cognitive tasks involving the transfer of factual knowledge between two situations"... "Learning, thinking and problem-solving skills which have been trained and are expected to generalise in a range of situations, often fail to do so, "*

even if the situations or problems are isomorph and differ only very slightly. This lack of transfer is even greater if the task is discrete, e.g. when it consists of a sequence of small steps. There appear to be two main reasons why this happens. Both reasons have repercussions for the training. First, it appears to be difficult for the trainees to see that the training situation and the transfer situation are actually isomorphic and therefore the trainees are not triggered to use the correct procedure. Second, in addition to the previous point, it appears to be that the application of knowledge and skills is highly context sensitive. This means that in a training situation one must take good care of both isomorphism and context sensitivity.

Retention is the ability of persons to remember how to effectively apply the knowledge and skills they have learned. While psycho-motor skills, e.g. riding a bicycle, are remembered even after many years of not using them, cognitive procedural skills are forgotten fairly quickly, especially if unused. In general, the level of retention is positively related to the level of learning at the end of the training. The retention also gets worse if the knowledge is not used after the training. This means that during the training the number of training session or exercises is relevant, the more the better. In the period after the training the amount of retraining or exercises and the interval between those are important, especially if the tasks are not done regularly which is typically the case in disaster relief.

The transfer requirements of isomorphism and context sensitivity dictate that the training environment must bear great resemblance with the task environment. The retention requirements dictate frequent exercises, both during and after training. Because the training of higher staff is critical for the task performance, special attention should be given to these requirements when designing the instructional systems

Isomorphism and context sensitivity, as required by transfer, are costly and difficult to realise. The task environment is large and complex. It involves many people, materials and complex natural phenomena with high costs attached. Often it is even not possible to create a a training environment that resembles the real task environment because it is too dangerous or socially unacceptable. Frequent exercises, as required by retention, are often not possible because the resources that are needed for the exercises are not available, apart from the costs. A simulation could be the solution for such training environment. In general, a simulation is an imitation of a real system and has all the key characteristic of that system. Therefore, a simulation of the task environment that takes in account all the requirements can be a solution for all these requirements. In a simulation the environment of a trainee is simulated which is less costly or dangerous than using the real environment [1]. Because a simulation does not use any of the real world resources in the environment, it is not dependent on the availability of those resources.

One of the requirements of the simulation is that the simulated persons or actors should behave in the simulation just as in the real world. They should use the same knowledge (facts, procedures, concepts and principles) and skills (cognitive and interactive) as the persons in the real world. One can use knowledge systems to implement the behaviour of the actors. Knowledge systems are systems that are able to solve problems in a particular domain using knowledge about the domain and goals given by the domain. Whenever an event happens in the simulated world and the simulated actor needs to react on that event, the knowledge system of that actor is used to determine its response. Early incarnations of knowledge systems consisted of rule bases with "if some condition then some action"-type of rules. The main inference methods were forward and backward chaining. Modern knowledge systems use more sophisticated logical methods as abduction and functional or causal models of behaviour. They also use probabilistic and decision theoretic methods to reason with uncertainty and make decisions on actions given goals.

In chapter 3, I will look into a specific implementation of a discrete event simulation that uses the concept of agents with rules bases to model and implement simulations.

[1]I do not take into account the cost for creating the simulation that can be substantial.

3. Simulation and agents

Historically, simulations are used to *study* systems. Simulations are imitations of some real system. They simulate the operation of a *model* of that system. Such a model usually consists of a mathematical or logical description (or both) of the system, the components of the system, the relation of the components and the way they interact. Typically, simulations can be be classified along three different dimensions [11]:

- Static versus dynamic. A static simulation model is a representation of the system at a certain moment in time, or a representation in which time does not play a role. A dynamic simulation model is a model in which time does play a role: it describes the system in time.
- Deterministic versus probabilistic. Simulation models that do not have any random variables and where the algorithms "behave predictably" are called deterministic. Simulation models that contain random variables or uses algorithms that "behave unpredictable" are called probabilistic.
- Continuous versus discrete. A continuous simulation model is a model in which the state variables change continuously with respect to time. A discrete model is a model in which the state variables change atomically at specific points in time. Between state transitions nothing (relevant) happens.

As with almost all classifications, no simulation exactly fits one classification. Almost all simulations contain elements of all categories.

If one looks at the description of the task environment and the simulation classification it is clear that a simulation that resembles the task environment should be dynamic and probabilistic. Although the task environment is both continuous and discrete, the tasks themselves are more discrete than continuous: people react on events by making decisions. Therefore the emphasis will be on dynamic probabilistic discrete simulation models. Such simulations are called discrete event simulations.

3.1. Discrete Event Simulation

A Discrete Event Simulation (DES) is a simulation in which the state of a model changes with discrete steps and where nothing happens between the steps. The discrete steps can be driven by the time, i.e. every certain time interval a "time-has-changed" event is generated and the model updates itself. The discrete steps can also be determined by the

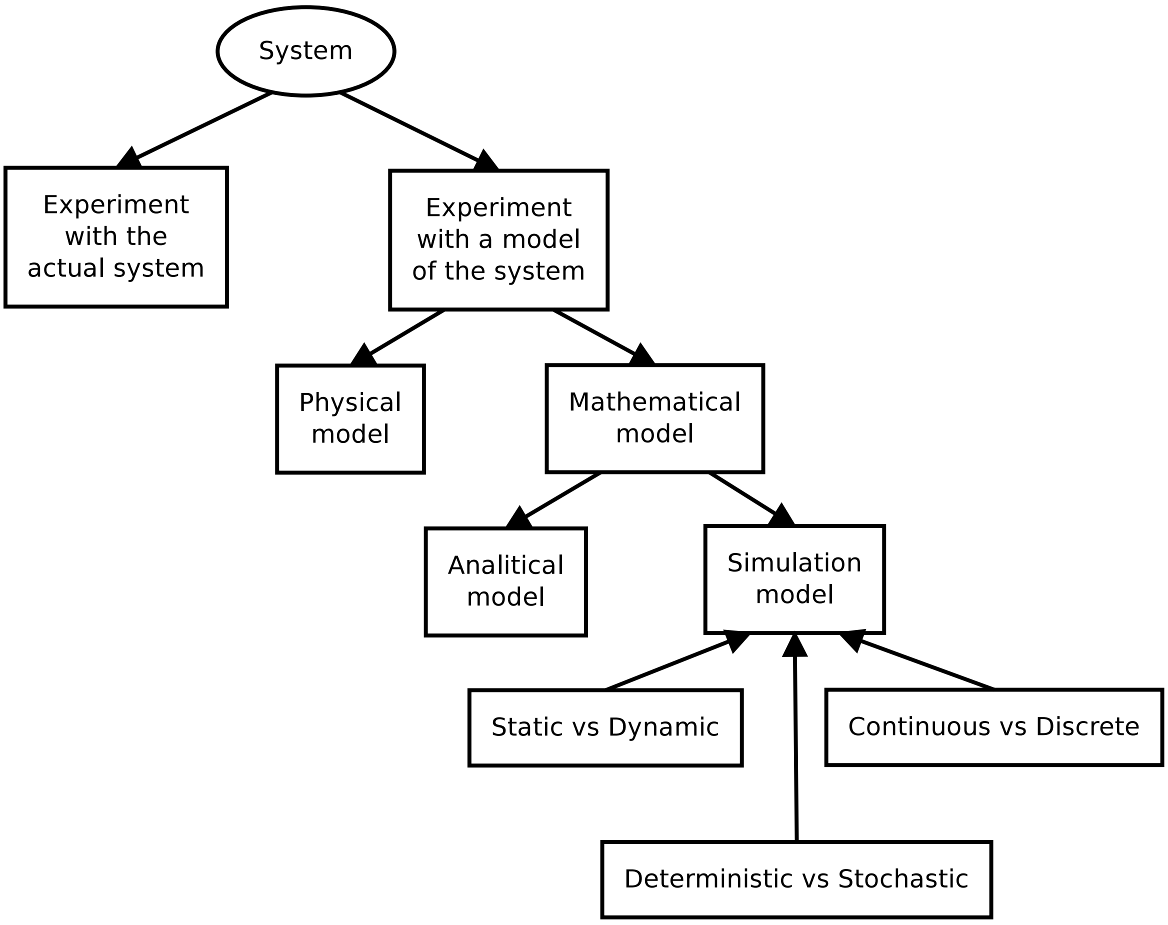

Figure 3.1.: Studying systems with simulation

activities in the model, i.e. whenever a start or end of an activity occurs a "something-happened" event is generated and the model updates itself. In this thesis only the latter type of simulation is considered.

The high-level description of a DES implementation is rather straightforward. Every state change in the simulation is translated by the DES into an event. The event is sent to the event queue. As long as the event queue is not empty or until a "stop simulation"-event is encountered the simulation will take the first event from the event queue and send it to an event handler which will handle the event appropriately, according to the logic of that handler. The handling of an event by the handler will often result into one or more events which in their turn are sent to the event queue and so on.

3.2. Agent systems

The implementation of a DES is not standardised. One of the modern forms of software engineering is object-oriented software engineering. Among many things, this means in general that variables and logic are encapsulated in classes and objects. In the task and training environment several concepts and objects appear: people, rules, facts, events, procedures an decisions. Part of the classes and objects in the software resemble the problem domain which facilitates the development of object oriented software. It seems natural to implement DES in objects oriented systems. It allows for a modelling and implementation of the domain classes and objects in the DES.

A refinement of object-oriented software engineering is agent-oriented software engineering. In agent-oriented software engineering some objects in the system, called agents, are active objects that function continuously and autonomously in an environment in which other processes take place and other agents exist that interact with each other. A specific implementation of agent oriented software engineering is the Belief-Desire-Intention (BDI) software model [22]. It models agents with concepts as "beliefs, desires and intentions", terms that suggest almost human traits in agents. They actually refer to typical technical problems that occur in (multi-)agent systems. *Beliefs* refer to the fact that it is possible that an agent has an incomplete view of the environment it operates in. To make its view complete the agent can use techniques as inference to reason about the unknown facts. *Desires* are about objectives an agents wants to accomplish or states that it would like to reach. *Intentions* are about finding and executing a plan with which the agent tries to achieve its desires.

It is easy to see that the classes and objects that refer to people in the task environment could be implemented in an agent-oriented DES as agents.

Russell and Norvig [25] distinguish different kind of environments in which agents operate:

- Fully observable vs partially observable
- Deterministic vs stochastic (probabilistic)

- Episodic vs sequential
- Static vs dynamic
- Discrete vs continuous
- Single agent vs multi-agent

The task environment maps nicely to a particular agent environment, making it a natural candidate for agent-oriented software engineering. A description of the task environment can be given in terms of agent environment:

- Partially observable
- Probabilistic
- Sequential
- Dynamic
- Discrete and continuous
- Multi-agent

Wagner [30] describes an agent-oriented approach to the conceptual modelling of organisations and organisational information systems, called Agent-Object-Relationship (AOR) modelling. In AOR modelling an entity is either an agent, an event, an action, a claim, a commitment, or an ordinary object. Special relationships between agents and events, actions, claims and commitments supplement the fundamental association, aggregation/composition and generalisation relationship types of Entity-Relationship (ER) and UML class modelling. Because AOR modelling can be done in UML the modelling can be highly formalised.

3.3. Agent-Oriented-Relationship modelling

Commonly agent-oriented software engineering and DES are not integrated and are viewed as two separate fields with different concepts and techniques. Wagner [31] describes an integration of agent-oriented software engineering and DES. The integration shows that the AOR metamodel and the metamodel of discrete event simulation can be combined into a model of agent-based discrete event simulation in a natural way. The integration is described in the Agent-Object-Relationship Simulation (AORS) metamodel.

The AORS metamodel is an OMG [1] MOF [2] model described in UML [3] that specifies the abstract syntax of the AOR simulation language. A MOF model means that a AORS metamodel implements a OMG MDA [4] M3 model. The OMG MDA defines a 4-tier layered system of models that are capable of describing models.

[1]Object Management Group, for more info see the OMG's website at http://www.omg.org/

[2]MetaObject Facility, for more info see the OMG's website at http://www.omg.org/mof/

[3]Unified Modelling Language, for more info see the OMG's website at http://www.omg.org/uml

[4]Model Drive Architecture, for more information on MDA, see [9] for a nice introduction or the official documentation which can be found at the OMG website at http://www.omg.org/mda/

- Layer M3: the meta-metamodel, the language used by MOF to build metamodels,
- Layer M2: the metamodel, e.g. the UML metamodel,
- Layer M1: The model of the system, typically written in UML,
- Layer M0: the instances, the software representation of real world items.

Schematically the relation between the AORS metamodel and the OMG MOF model is depicted in figure 3.2. For a more detailed description of the AORS metamodel at the M2/M3 level in UML see appendix B.

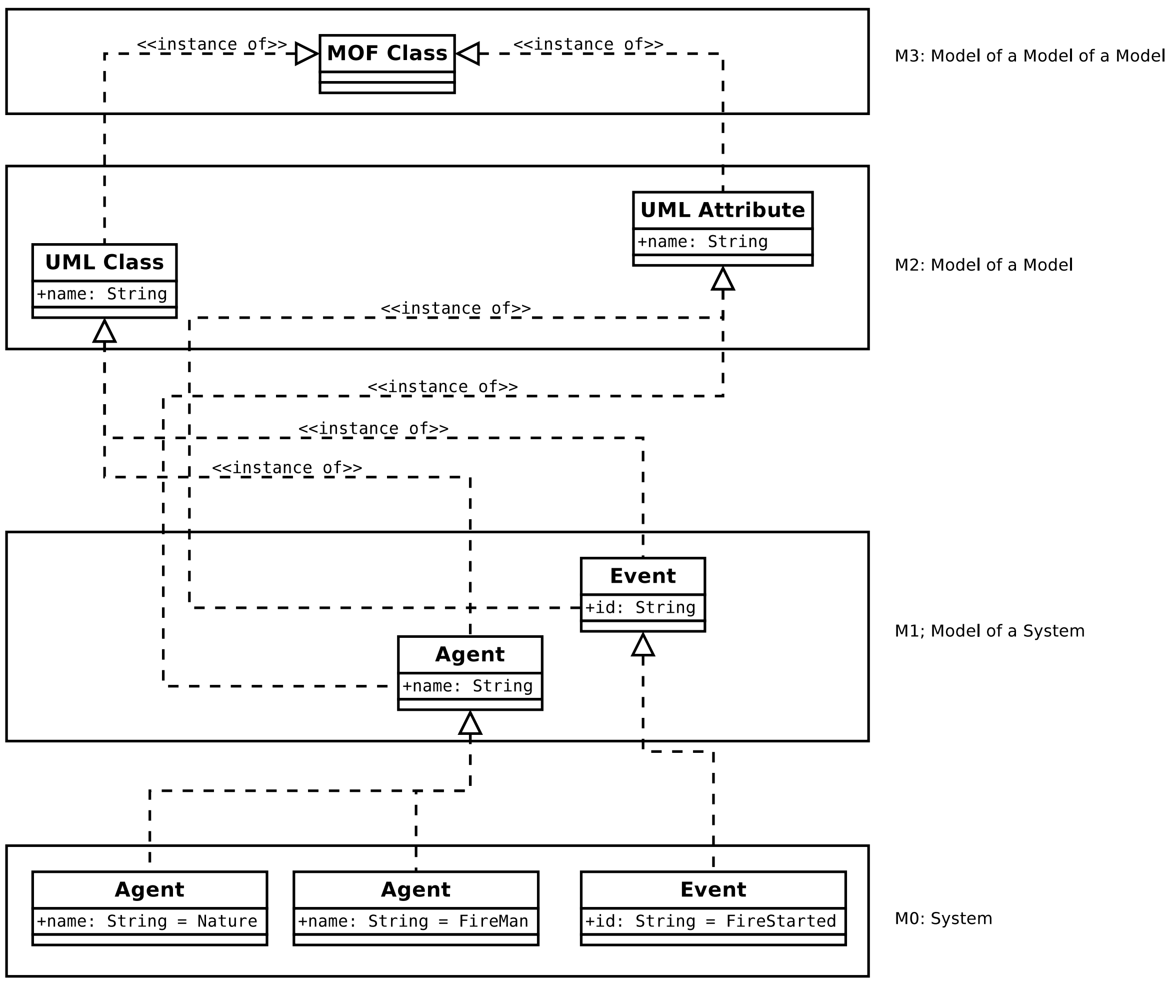

Figure 3.2.: AORS metamodel and MOF

Because the languages that are described by a MOF model are well-defined or formal languages they are suitable for automated interpretation by a computer. The fact that they are formal languages is used to define model transformations where a source model is transformed to a target model by automated tools. The most tempting and ambitious model transformation is of course the transformation from specification to running application without any manual coding. The OMG has defined the MDA framework that helps to do that.

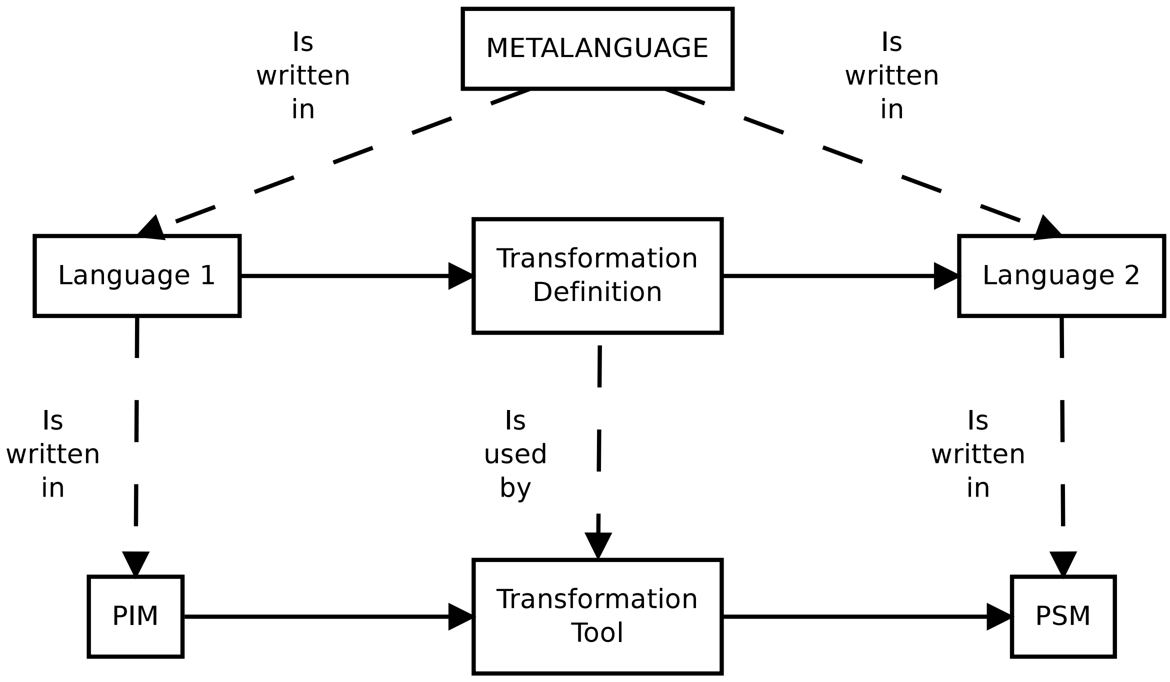

Figure 3.3.: MDA PIM and PSM framework

In the MDA framework, an OMG PIM [5] language is a platform independent language that describes a software system that supports some business. It is a MOF-based language with a high level of abstraction and is independent of any implementation technology. An OMG PSM [6] language is a platform dependent language that specifies the implementation constructs that are available in one specific implementation technology. Most of the time a PSM is an actual programming language which can be interpreted by an interpreter or compiled into an executable program by a compiler, supported by a suitable runtime environment if needed. Ideally a PIM and a PSM can be translated into each other by automated tools. Schematically the whole MDA framework is shown in figure 3.3.

The AORS metamodel is the basis for AOR Simulation Language (AORSL). AORSL is an extension and refinement of the AORS metamodel with the necessary objects and classes for a DES implementation. AORSL can be used to describe (discrete event) simulation models. A simulation model is defined by a space model, a set of entity types and a set of rules. It defines a simulation scenario, a set of simulation parameters,

[5] Platform Independent Model
[6] Platform Specific Model

and a set of entities defining the initial state of the simulation. AOR simulations may be implemented in different programming languages, but they must conform to the AOR simulator architecture and to the AOR simulator execution model as defined in [31]. A simulation model written in AORSL combined with the simulation model can be considered a model written in a OMG MDA PIM language. A confirming implementation of an AOR simulation can be considered a model in a OMG MDA PSM language. Because both model are formal models it should be possible to provide appropriate tools that translate the simulation model to a running simulation. Schematically the relation between the OMG MDA framework and AORSL is shown in figure 3.4.

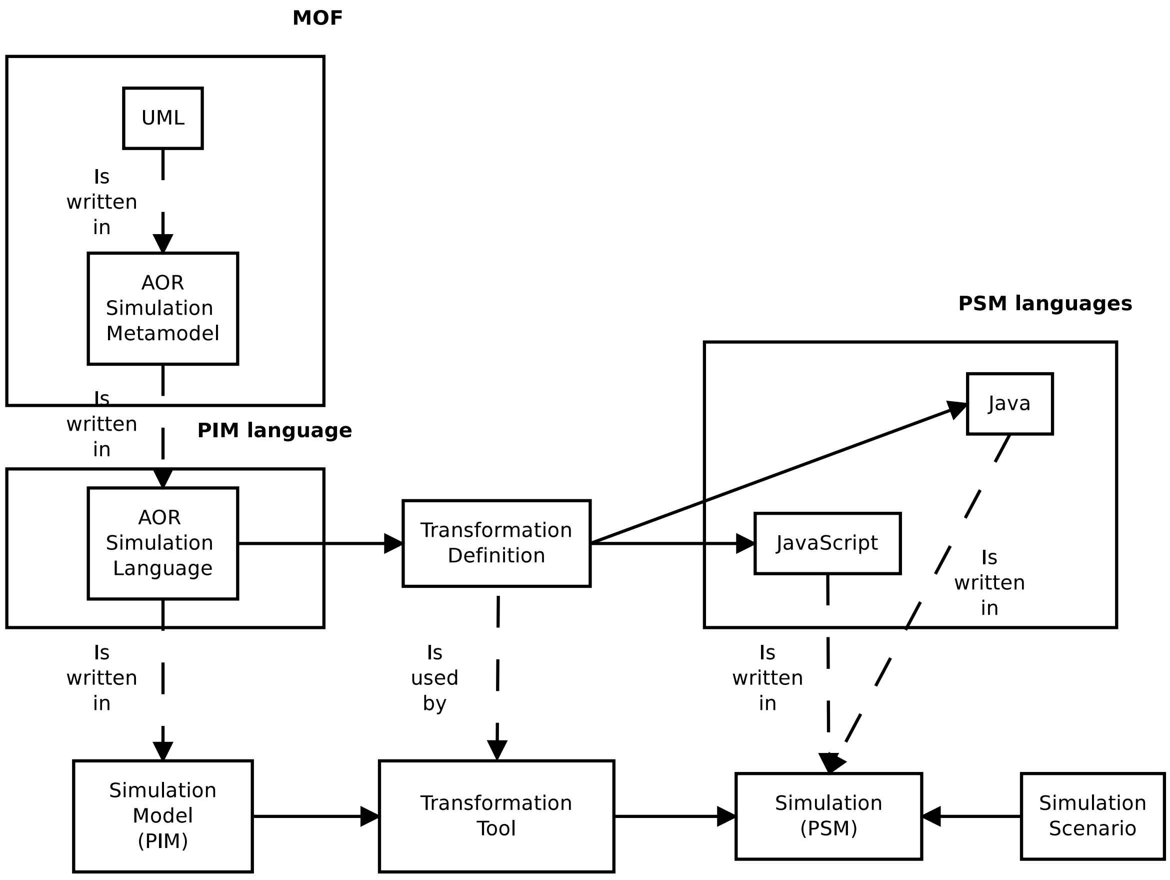

Figure 3.4.: MOF/PIM/PSM/AORSL framework implementation

There is a reference implementation available, called AOR-JavaSim, which is a standalone desktop program written in Java with a graphical user interface. The application is intended as a universal multi-purpose simulation framework in science, engineering, education and entertainment and contains several sample simulations. In this implementations simulation model are describes in XM-file. There is a tool available that uses XSLT transformations to transform the simulation model into a running simulation. The simulations are written in the target languages Java or JavaScript. The running simulation written in Java or JavaScript can be considered models in a OMG MDA PSM language. AOR-JavaSim shows that it possible to create runnable specifications: specifications that can automatically be transformed into running applications without manual coding.

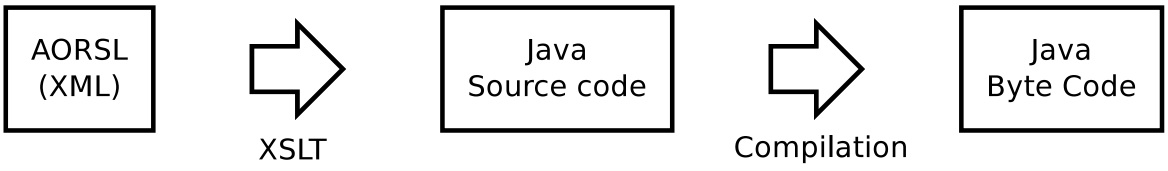

Figure 3.5.: AORSL to Java

3.4. AORSL in detail

In this section I will look in some detail at AORSL. The focus will be on those constructs and functionality that are relevant for the understanding of the language given the scope of this thesis, which is the usage of first-order probabilistic logic as the event handling logic in discrete event training simulations.

3.4.1. Ontological categories

The AORS metamodel is based on ontological principles [7]. On of the aspects of these ontological principles is the relation between a modelling language and a set of real-world phenomena in a given domain. The relation can be described by the ontological adequacy of a given modelling language. The ontological adequacy of a modelling language is a measure for how close the models produced using a modelling language are to the situations in reality they are supposed to represent. A high adequacy means that the models and the domain share many concepts. The AORS metamodel aims to closely model the domain in terms of that domain. In this case this means that the concepts, classes and objects of both the DES and agent domain are recognisable modelled in the AORS metamodel.

The upper level ontological categories of AOR simulation are messages, events and objects, which include agents, physical objects and physical agents. The task and training environment contain all these categories in some form. The people must cooperate to perform their task. This means that people have to communicate and exchange information with each other through messages. The events in the task environment map to the events in AORSL. The people and other active parties in the task environment map to agents in AORSL. The event type has an elaborate ontology of its own. It has subtypes as internal events that are those events that happen “in the mind” of the agent. For modelling distorted perceptions, both a perception event type and the corresponding actual perception event type can be defined and related with each other via actual perception mapping rules. AORSL also supports the notion of both exogenous and endogenous events, events from outside the system that have effects inside the environment and events from inside the system. For a full overview see the UML overview in appendix B.

The entities are grouped together in a simulation scenario. A simulation scenario consists of a simulation model, an initial state definition and zero or more view definitions. A simulation model consists of

- an optional space model, if needed for physical object/agents,
- a set of entity types, including different categories of event, message, object and agent types,
- a set of environment rules, which define causality laws governing the environment state changes.

An agent type is defined by:

- a set of (objective) properties,
- a set of (subjective) self-belief properties,
- a set of (subjective) belief entity types,
- a set of agent rules, which define the agent's reactive behaviour in response to events.

An agent's property is the same as an attribute in ordinary object orientation. An agent has objective and absolute knowledge of its own properties. Knowledge of itself are facts and objective. It is possible that an agent has no definitive knowledge of a property but that it has some subjective belief of itself, e.g. the position of the agent in terms of its coordinates. In that case the agent has a subjective self-belief. The knowledge that an agent has about the rest of the world is described by the set of belief entity types. How subjective the self-belief properties and belief entity types are is unknown to the agent in AORSL.

For the usage of a DES in a training simulation all elements except the space model are relevant as they have a direct mapping to the task and training environment.

3.4.2. Rule-based modelling

There are two kind of rules that are involved in handing the events in an AOR simulation. Both the behaviour of the environment and the behaviour of agents are modelled using rules. Environment rules describe the reaction of the environment to actions by agents and other environment events. Agent rules describe the reaction of agents on events received.

An environment rule is a 5-tuple $\langle \mathit{TriggerEvt}, \mathit{VarDecl}, \mathit{Condition}, \mathit{UpdateState}, \mathit{ResultEvt} \rangle$, where

- *TriggerEvt* is the type of event that triggers the rule,

- *VarDecl* is a set of variable declarations, such that each variable is bound either to a specific object or to a set of objects,
- *Condition* is a logical condition formula, allowing for variables,
- *UpdateState* specifies an update of the environment state,
- *ResultEvt* is a list of resulting events, which will be created when the rule is fired,

which can roughly be read as:

```
on  TriggerEvt
for VarDecl do
  if Condition then
    UPDATE UpdateState
    SCHEDULE-EVENT ResultEvt
  end if
end for
```

An agent rule is a 4-tuple $\langle \mathit{TriggerEvt}, \mathit{Condition}, \mathit{UpdateState}, \mathit{ResultEvt} \rangle$, where

- *TriggerEvt* is the type of event that triggers the rule,
- *Condition* is a logical condition formula, allowing for variables,
- *UpdateState* specifies an update of the environment state,
- *ResultEvt* is a list of resulting events, which will be created when the rule is fired,

which can roughly be read as:

```
on  TriggerEvt
if Condition then
  UPDATE UpdateState
  SCHEDULE-EVENT ResultEvt
end if
```

The basic difference is that an environment rule can be bound to multiple objects but an agent rule is always bound to one agent. Both rules have an arbitrary logical condition formula to determine if the rule will actually "fire", i.e. the condition evaluates to true. This allows for a sophisticated determination of the current state of the world by the environment or agent. If it fires, it will possibly update arbitrary variables in the simulation and it will possibly send the resulting events to the simulation engine. This mechanism makes both the environment and the agents reactive in nature: both the environment and the agents can be seen as purely simple reflex agents as described in [25].

3.4.3. Beliefs

As mentioned before, beliefs refer to the fact that it is possible that an agent may have an incomplete view of the environment it operates in. Most agent worlds are partially observable. AOR simulation supports the distinction between facts and beliefs, including self-beliefs (the agent's beliefs about itself). The language supports the notion of facts and beliefs and allows an implementation to store and retrieve facts or beliefs as a form of property-values tuple. An agent can always retrieve its belief during its event handling in the condition part of the rule. Updating its beliefs can be done in the update expression. Diaconescu and Wagner [4] describe in their article that the W3C Resource Description Framework (RDF) query language SPARQL [7] can be used for expressing queries that an agent can ask another agent about its beliefs. However, the support is limited. To make its view complete the agent has to use techniques as inference to reason about the unknown and uncertain facts. AORSL as a modelling language has no concepts that actually support inference of unknown or uncertain facts besides the possibility of storing beliefs and facts. As such there is a limited support for model-based reflex agents [25].

3.4.4. Desires

Desires are about the goals an agent wants to accomplish. Goal-based agents [25] make a distinction between goal states and non-goal states of their world. Their aim is to end in a goal state. More advanced agents are utility-based agents [25]. They use the concept of a utility function which is a function that calculates how desirable a particular state is and has therefore a more sophisticated idea of goal state. The term *the goal* is ambiguous in the context of agents: *the goal* of a goal-based agent is to end in one of the goal states whereas *the goal* of a utility-based agent is to end in a state with the highest utility. In the remainder of this thesis I will simply refer to *the goal* of an agent and make no distinction between the two approaches. AORSL has no concept of either goal or utility. It is currently not possible to implement goal or utility-based agents in AORSL.

3.4.5. Intentions

Intentions are about finding and executing a plan. A plan is a sequence of actions that an agent can perform and that, ideally, result in the accomplishment of the desires or goals. The current version of AORSL does not have the concept of a plan. But adding a minimal extension to AORSL, it is possible to model business processes as described by Wagner, Nicolae and Werner in [32]. The main idea is to define an activity as a complex event having a "start event" and an "end event" caused by a start action and an end action. A nice feature is that they also show how to use this extension in combination with the OMG's BPMN.[8]. According to the authors a full simulation model can be specified by combining an UML class model with a BPMN process model.

However, BPMN has some limitations. BPMN has no support for probabilistic processes except for the duration of activities. Also, the sequential, flowchart-like modelling nature

[7] for more info see the W3C website at http://www.w3.org/2001/sw/wiki/SPARQL

[8]Business Process Modeling Notation, for more info see the OMG website at http:/www.omg.org/spec/BPMN/

of BPMN is not always compatible with the event driven nature of DES where events do not necessarily follow the flowchart because it is very dynamic. BPMN is rather static as a modelling language: one specifies all possible processes beforehand and no run time building of the flowchart or planning is done. AORSL's concept of causality has some limits. In general causality means that some events cause other events. The only cause-effect relation that can be described is that of a single event and its resulting events. Multiple events that result in a common effect cannot be modelled. This actually means that there is no support for (probabilistic) planning of actions based on the state of the past, current and future states of the world. For utility-based agents planning is important. If an agent cannot plan its actions beyond the first action, it is easy to find a local utility maximum and not finding a global maximum or better local maxima.

3.5. Extending AORSL

Clearly that the task environment has some characteristics that make reasoning about the state and the consequences of actions difficult. The environment is dynamic, partial and noisy observable, probabilistic and goal or utility driven. People handle these circumstances naturally. Computer programs do not have a natural feeling for handling uncertainty, probability, causality, non-monotonicity and planning, which are the more technical terms for the task environment characteristics. Computers use special logic to handle this. Uncertainty can be handled by probabilistic logic. Probabilistic logic is an extension of classical logic where a truth value is true or false, with a certain probability. Causality can be handled by causal logic. In causal logic a relationship is established between two events where one event is the consequence of the other even. The latter is said to be the cause of the former. The non-monotonicity can be handled by non-monotonic logic. Non-monotonic logic allows for defeasible inference which allows for inferring conclusions that can be retracted by new information that is added to the available knowledge. AORSL does not support non-monotonic probabilistic causal logics.

If AORSL is to support training simulations then a solution for these problems is needed. One of the solutions is adding a causal probabilistic first-order logic. In the next chapter, I will look at another agent-based modelling environment that is based on probabilistic first-order logic, Independent Choice Logic (ICL) from Poole [20]. I will look at ICL with the purpose to embed it into AORSL, extending the possibilities of AORSL with first-order probabilistic causal logics.

4. Probabilistic Logic

In this chapter I will look into first-order languages that support probability and causality. Milch and Russell [15] give an overview of first-order probabilistic languages in a taxonomy as shown in figure 4.1. The languages of interest are languages that use relational structures (causality) and conditional probability distributions (probability). ICL (Independent Choice Logic), PHA (Probabilistic Horn Abduction) and LPAD (Logic Programs with Annotated Disjunctions) are such languages in the taxonomy. In this chapter I will look at three implementations of probabilistic logic, Poole's ICL [20], Poole's PHA [18] and Vennekens' CP-Logic (CPL)[28], the successor of LPAD [29].

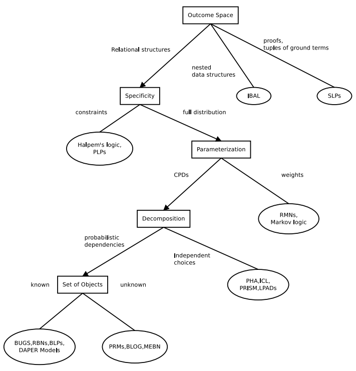

Figure 4.1.: A taxonomy of first-order probabilistic languages

Although Poole's ICL [20] is listed as a first-order probabilistic language, it is in fact more than that. It uses elements from several techniques to create a framework that can be used to represent a specification or model for multiple agents that make observations and decisions in a stateful and uncertain world. Given the fact that OARSL is an *agent oriented* simulation language, embedding the ICL framework appears appealing. ICL is a high-level abstract language specification without a concrete syntax and makes use of other embedded formalisms for more detailed modelling and concrete syntax. ICL acts in this context as specification language for the embedded formalism: ICL is used for specification and the embedded formalism is used for the implementation. Poole shows embeddings for three embeddings in his article: influence diagrams, Markow decision problems and a strategic form of games.

The most natural embedding however is that of probabilistic Horn abduction (PHA) [18], which is the precursor and a restricted form of ICL itself. It also is the basis of aforementioned influence diagram embedding. PHA is a simple first-order probabilistic logic language that uses Horn-clauses with probabilities. It can represent the same probabilistic knowledge that can be represented in a Bayesian networks PHA into Bayesian networks and vice versa.

Another possible embedding is that of CPL [28], which, like PHA, is another simple first-order probabilistic causal logic language. It uses logic to describe probabilistic causal laws which are causal relations between events. Where PHA is a restricted form of ICL, CPL can be considered a superset of ICL, allowing for a less restricted syntax for specifying probability and adding the notion of probabilistic causal processes or state transitions. therefore it may not only be possible to use CPL as embedded formalism but also to extend the possibilities of ICL as specification language. It is possible to translate CPL into Bayesian networks and vice versa.

I will not look at a *specific* monotonic logic. Non-monotonic logics as default reasoning or abductive reasoning use standard values for true and false as truth values in the logical formulae. Probabilistic logics assign the truth values real valued probability values, which can be interpreted as true or false using some threshold values. The addition of evidence leads to adjustment of the probabilities, introducing non-monotonicity. Probabilistic logics therefore support a certain form of non-monotonicity, although McCarthy [12] shows that probabilistic logics cannot cover all instances of non-monotonic logic. For this thesis the support of non-monotonicity by probabilistic logic is sufficient.

4.1. Preliminaries

Before looking into ICL, PHA and CPL, I will briefly recapitulate some elements of first-order logic, probability theory and the concept of causality. Just as with AORSL, the focus will be on on those elements that are relevant for the understanding of the languages, given the scope of this thesis. In this recapitulation I assume that the reader has knowledge about the aforementioned subjects on the level of an undergraduate computer science[1].[2]. If one is confident enough this section can safely be skipped.

4.1.1. First-order logic

An *object* is a thing. A domain is a set of individual objects. The domain of discourse is the set of domains of interest. The domain of discourse is often abbreviated to domain.

A *constant* is a specific object of some domain. Constant are customary represented by lowercase letters or lower (camel) case words. If letters are used it is customary to use letters from the beginning of the alphabet.

A *variable* is an unspecified member of some domain of objects or class of objects. Variables are customary represented by lowercase letters from the end of the alphabet.

[1]For more information on first-order logic, see [3], the primary source for this section, or [8]
[2]For more information on Bayesian networks, see [10]

A *function* assigns a combination of objects of some domains to a member of some domain. It expresses dependencies between the argument terms. *Function symbols* are customary represented by lowercase letters or lower (camel) case words. If letters are used it is customary to use letters from the middle of the alphabet.

A *predicate* maps a combination of objects from some domains to a truth value. They can be viewed as a function with the range $true, false$. Predicate symbols are customary represented by uppercase letters or upper (camel) case words.

If a function or predicate has arguments, they are given between parentheses and separated by commas. The *arity* of a function or predicate is the number of arguments of the function or predicate. A function with an arity of 0 is the same as a constant. A predicate with an arity of 0 is called an *(atomic) proposition.*

A *first-order language* consists of

- a set of constants;
- a set of variables;
- a set of functions and associated arities;
- a set of predicates and associated arities.

A *term* is defined as

1. A constant is a term.
2. A variable is a term.
3. If f is a function symbol and $t_1, \ldots, t_n$ are terms, then the function $f(t_1, \ldots, t_n)$ is a term.

An *atom* is a predicate with arity 0 or a predicate of the form $P(t_1, \ldots, t_n)$, where P is a predicate symbol and $t_1, \ldots, t_n$ are terms.

There are two *quantifiers*: the universal quantifier $\forall$ and the existential quantifier $\exists$. If x is a variable, then $\forall x$ is read as "for all x" and $\exists x$ is read as "there exists an x".

The *logical connectives* which are the same as in propositional logic: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$.

A *well-formed* first-order *formula* or *sentence* is defined as:

1. The constant truth values $true$ and $false$ are formulae.
2. A single predicate with its arguments is a formula, also called an *atomic* formula.
3. If ϕ is a formula, then $\neg\phi$ is also a formula.
4. If ϕ and ψ are formulae, then $\phi \vee \psi, \phi \wedge \psi, \phi \rightarrow \psi$ and $\phi \leftrightarrow \psi$ are also formulae.
5. If x is a logic variable and ϕ is a formula, then $\forall x\phi$ and $\exists x\phi$ are formulae.

A *clause* is a specific type of formula and is either an atom or has the form $\alpha \rightarrow \beta$ where α is a literal or a conjunction of literals and β is an atom. α is called the *body* of the clause and β is the *head* of the clause. A clause of the form $\alpha \rightarrow \beta$ is called a *rule.* A clause is a formula or sentence that consists of a finite disjunction of literals, where a literal is an atom (positive literal) or a negation of an atom (negative literal) because a rule can be re-written as $\neg\alpha \vee \beta$ or $not\ body \vee head$. A specific type of clause is the *Horn clause.* A Horn clause is a clause that contains at most one positive literal. A Horn clause with exactly one positive literal is a *definite clause.* An other specific type of clause is the *empty clause*, which is a clause with null literals. It is often denoted as $\Box$.

A *logical theory* is a conjunction of formulae. A collection of formulae separated by commas is interpreted as a conjunction of formulae. A *(logic) program* is a set of clauses. Because every formula can be rewritten as a set of clauses in Conjunctive Normal Form (CNF) (see [3]), every logical theory can be considered a logical program.

A *ground term, atom or clause* is one that does not contain any variables. A ground instance of a clause c is a clause obtained by uniformly replacing ground terms for the variables in c.A theory of which all clauses are grounded is called a *grounded theory.*

In first-order logic, a formula has no meaning of itself, it is a meaningless symbolic formula. To determine the meaning or truth value of formulae, one has to connect the symbols with objects in the domain of discourse. An *interpretation* is a mapping from the symbols of a first-order language to their meaning, possibly in the real world. An *assignment* or *valuation* is assigning every variable in the first-order language a specific value in the domain. Given an interpretation and a valuation one can determine the truth value of a formula. If an interpretation assigns the value $true$ to a formula or theory, the interpretation is called a *model* of that formula or theory.

The *Herbrand universe* of a first-order language is the set of all ground terms. It is enumerable, and infinite if a function with an arity greater than 0 exists. The Herbrand universe U_H for the set of clauses S is given by:

- All constants which occur in S are in U_H.
- If no constant appears in S, a single constant with an arbitrary name, e.g. a, is to be assumed in U_H.
- For every n-ary function symbol f in S, and for every $t_1, \cdots, t_n \in U_H$, $f(t_1, \cdots, t_n)$ is in U_H.

The *Herbrand base* of a first-order language is the set of all ground atoms that can be formed from predicate symbols in the original set of clauses and terms in its Herbrand universe. It is the set of all possible ground goals that the theory can represent. Like the Herbrand universe, it is enumerable, and infinite if the Herbrand universe is infinite and there is a predicate symbol with an arity greater than 0. The Herbrand basis for a set of clauses S and the Herbrand universe U_H is given by:

- The set of ground atoms $P(t_1, \ldots, t_n)$, where P is a n-ary predicate symbol from S and $t_1, \ldots, t_n \in U_H$.

A *Herbrand interpretation* is an interpretation in which all constants and function symbols are assigned a very simple meaning. Every constant is interpreted as itself, and every function symbol is interpreted as the function that applies it. Every grounded predicate is assigned a truth value, specifying the predicate to be $true$ or $false$. If a Herbrand interpretation is true, it is called a *Herbrand model.*

4.1.2. Probability

In real life things can be uncertain. Probability is a method to quantify uncertainty. Intuitively probability has something to do with the approximate number of times something is the case and the approximate number of times that that something is not the case, together forming the approximate total number of cases. Surprisingly enough, there is no definitive definition of probability. There are at least two approaches of probability:

- Physical of frequency probability. Frequency probabilities are about probabilities in the context of experiments that are random and well-defined. The probability of a random event denotes the relative frequency of occurrence of the outcome of an experiment, if the experiment is repeated many times or even indefinitely.
- Evidential or Bayesian probability. Bayesian probabilities are about measuring the state of knowledge or degree of belief in a statement about facts in the world, given any available evidence.

As mentioned in 3.2 and 3.4.3, agents operate in a partially observable and probabilistic environment and have believes about the world. The Bayesian interpretation of probability correspondents closely with the notion of probability as measurement for uncertainty that is used in an agent-oriented environment. Therefor, in this thesis the Bayesian approach is used.

More formally the intuition of probability is described as follows. A probability space is a triplet $\langle \Omega, \mathcal{F}, \mathcal{P} \rangle$ where

- Ω is the sample space, also called *universe of discourse* U, that it is the (arbitrary) set of all possible outcomes.
- $\mathcal{F} \subseteq 2^{\Omega}$, a set of subsets of Ω called *events*, such that $\emptyset \in \mathcal{F}$, and $\mathcal{F}$ is closed under complement, i.e. if $f \in \mathcal{F}$ then also $(\Omega \setminus f) \in \mathcal{F}$, and countable union, i.e. if $f_1 \ldots f_n \in \mathcal{F}$ then also $(\cup_n f_n) \in \mathcal{F}$;
- $\mathcal{P}$ is a function $P : \mathcal{F} \to [0,1]$, called *probability measure*, such that $\sum\limits_{f \in \mathcal{F}} P(f) = 1.0$.

Given a probability space $\langle \Omega, \mathcal{F}, \mathcal{P} \rangle$, the function $\mathcal{P}$ should obey the so-called *Kolmogorov* axioms:

- $P(\Omega) = 1$.
- For all events $E \in \mathcal{F}$: $P(E) \geq 0$.
- For all events $E, F \in \mathcal{F}$: if $E \cap F = \emptyset$ then $P(E \cup F) = P(E) + P(F) \geq 0$.

In addition to the probability space and the Kolmogorov axioms, several other concepts are introduced. *Conditional probability* is the probability that a certain event occurs, given the fact that some other event occurred. Conditional probability is defined as:

Definition 4.1. $P(X|Y) = \dfrac{P(X \cap Y)}{P(Y)}$

Independence or *marginal independence* says that two events are independent of each other: the probability that one event occurs is not connected to the occurrence of the other event. Independence is defined as:

Definition 4.2. $X \perp\!\!\!\perp Y \equiv P(X|Y) = P(X)$

Conditional independence is a generalisation of independence and is defined as:

Definition 4.3. $X \perp\!\!\!\perp Y|Z \equiv P(X|Y,Z) = P(X|Z)$

Based on the definition of conditional probability the Bayes' theorem is defined as [3]:

Theorem 4.1. $P(H|E) = \frac{P(E|H)P(H)}{P(E)}$ *where*

- *H represents a hypothesis*
- *E represents the (observed) evidence*
- *$P(H)$ is the* prior probability *of H that was inferred* before *evidence became available.*
- *$P(E|H)$ is called the* conditional probability *of seeing the evidence E if the hypothesis H happens to be true. It is also called a* likelihood function *when it is considered as a function of H for fixed E.*
- *$P(E)$ is called the* marginal probability *of E: the* a priori probability *of witnessing the evidence E given all hypotheses. It can be calculated as the sum of the product of all probabilities of any complete set of mutually exclusive hypotheses and corresponding conditional probabilities:*
  $P(E) = \sum P(E|H_i)P(H_i)$
- *$P(H|E)$ is called the* posterior probability *of H given E and is the new estimate of the probability that the hypothesis H is true* after *the evidence E became available.*

A collection of conditionally dependent variables and their probabilities can be describes by a *Bayesian* or *belief network* A Bayesian or belief network is a graphical probabilistic model that represents random variables and their conditional dependencies. The graphical model is a directed acyclic graph (DAG). The nodes or vertices of the DAG represent the random variables, the edges or arcs represent the conditional dependencies between the variables. More formally a Bayesian network is defined by:

- a directed acyclic graph $G = \langle V, E \rangle$;
- a set of random variables $X = (X_v)_{v \in V}$ indexed by V
- the joint probability distribution $P(X)$ can be written as the product of the individual density functions, conditional on their parent variables, or factorises according to:
  $P(x) = \prod_{v \in V} P(x_v | x_{parent(v)})$

[3] This follows from $P(X|Y) = \frac{P(X \cap Y)}{P(Y)}$ and $P(Y|X) = \frac{P(Y \cap X)}{P(X)}$ and $P(X \cap Y) = P(Y \cap X)$

When reasoning over the world, the Bayesian network can also be used by Bayesian inference which can be used to reason about the world, especially in the context of appearing evidence. Whenever evidence appears, e.g. by making observations in the real world, this evidence is applied to the state of the variable in the model. Then the state change is propagated throughout the network and a new probabilities are calculated. Depending on the size and structure of the network, the recalculations can be a costly operation. As mentioned in the definition, the probabilities originally encoded in the model are known as *prior probabilities*, because they are known before any evidence is known about the situation. The probabilities computed after evidence is entered are known as *posterior probabilities*, because they reflect the levels of belief computed in light of the new evidence.

4.1.3. Causality

Just as probability, causality is surprisingly difficult to define. However, the problem is largely technical of nature. For this thesis, an intuitive notion of causality will suffice. Causality is the relationship between two events where one event is seen as the cause and the second event the effect. Whenever the cause happens, the effect will follow, while the reverse is not the case. Causation implies that by varying one factor, another factor can be made to vary.

Causality can be characterised as transitive, reflexive and anti-symmetric or contraposition:
If x *causes* y, or as predicate $Causes(x, y)$, then:

- Transitivity: $\forall x\ \forall y\ \forall z\ ((Causes(x, y) \wedge Causes(y, z) \rightarrow Causes(x, z)$;
- Reflexivity: $\forall x\ Causes(x, x)$;
- Antisymmetry: $\forall x\ \forall y\ ((Causes(x, y) \rightarrow \neg Causes(y, x)$.

If not written as a predicate, but using the logical connective $\rightarrow$, $x \rightarrow y$:

- Transitivity: $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$;
- Reflexivity: $\models X \rightarrow X$;
- Contraposition (no antisymmetry): $\{X \rightarrow Y, \neg Y\} \models \neg X$.

Often, two couples of terms are used to describe causation in more detail. First, the terms necessary and sufficient causes. These say something about the cause of an effect if the effect is observerd. If C is a necessary cause of E, then the presence of E necessarily implies the presence of C. The presence of C, however, does not imply that E will occur. If C is a sufficient cause of E, then the presence of C necessarily implies the presence of E. However, another cause Z may alternatively cause E. Thus the presence of E does not imply the presence of C.

Second the terms strong and weak causality. These say something about an effect if a cause is observed. Strong causality, $C \rightarrow E$, is if C is present than E *must* also be present. Weak causality, $C \wedge \alpha \rightarrow E$, is if C is present than E *may* also be present, depending on α which may absence.

4.1.4. First-order logic, probability and causality

First-order logic, probability and causality can be combined, using a unified notation for the three concepts.

A logical or or "$\vee$" can be considered a probabilistic statement. However, it is not known what the underlying probability distribution is of such statement. A reasonable enhancement of the logical or statement would be to encode the underlying probability distribution in the statement by adding the distribution. One way of doing that is to annotate the atoms with their probability. A formula of the form "$a : p$." then should be read as "a is $true$ with a probability of p". This implies that the constituents are assumed to be independent. This is no problem, because it does not limit the representation of the world[18].

A statement of the form "$(a_1 : p_1) \vee \ldots \vee (a_n : p_n)$" should be read as "each a_i has a probability p_i and the sum off all probabilities should be 1.0 or less". If it is less than 1.0 the statement should be read as "and there is an unmentioned a_{n+1} that is responsible for the remainder up to 1.0". Note that the logical or, $\vee$, here is an exclusive or: just one a_n will be true and the remaining will be false. This allows the modelling of mutual exclusion of the values of a random variable.

To describe a probabilistic "if-then-else" the implication, $\rightarrow$, is used. A formula of the form $(a_1 : p_1) \vee \ldots \vee (a_n : p_n) \leftarrow \phi$ should be read as "if ϕ is $true$ than one of $a_1, \ldots, a_n$ is true with a probability of p_n". A formula of the form "$a \leftarrow (a_1 : p_1) \vee \ldots \vee (a_n : p_n)$" should be read as "if one of $a_1, \ldots, a_n$ with a probability of p_n is $true$ than a is $true$".

The same implication notation can be used to describe causality. Analogous to the probabilistic implication a formula of the form $a_1 \vee \ldots \vee a_n \leftarrow \phi$ should be read as "if ϕ is $true$ than it causes one of $a_1, \ldots, a_n$ to be $true$. A formula of the form "$a \leftarrow a_1 \vee \ldots \vee a_n$" should be read as "if one of $a_1, \ldots, a_n$ is $true$ than that will cause a to be $true$".

The combination of probability and causality then is trivial.

4.2. ICL

Independent Choice Logic (ICL) is a large framework that is capable of modelling agents under uncertainty. It combines logic programming, probability, game and decision theory. Agents try to accomplish their goals by making choices about the actions to be performed. The choices are considered independent and have a probability distribution over each choice outcome. A logic program gives the consequences of the choices, defining possible worlds based on the choices. Agents make the choices that result in the worlds that realise their goals as much as possible. In this section I will introduce the most parts of ICL but I will leave out the specific game related parts about Nash Equilibria and Pareto optimality. All definitions and examples are directly taken from Poole [20].

Definition 4.4. *A base logic is a triple $\langle \mathcal{L}_\mathcal{F}, \mid\!\sim, \mathcal{L}_Q \rangle$ such that $\mathcal{L}_\mathcal{F}$ and $\mathcal{L}_Q$ are languages and $\mid\!\sim$ is a consequence relation.*

$\mathcal{L}_{\mathcal{F}}$ and $\mathcal{L}_Q$ are both specified as logical languages. Poole does not specify a concrete syntax for the languages. For the actual concrete syntax of $\mathcal{L}_{\mathcal{F}}$ and $\mathcal{L}_Q$ the language of the embedded formalism is used. In his article he assumes a Prolog-like syntax and other conventions to be able to give explanations and examples . I will do the same, as Prolog [4] is firmly rooted in first-order logic which is a large part of the research in this thesis. Except when mentioned otherwise, I will forgo the specification nature of $\mathcal{L}_{\mathcal{F}}$ and $\mathcal{L}_Q$ and will use a concrete Prolog-like syntax for the languages as an implementation of that specification.

Language $\mathcal{L}_{\mathcal{F}}$ is the language of facts. It is used to specify the rules of the possible world using some specific ICL rules. In his article, Poole considers a language $\mathcal{L}_{\mathcal{F}}$ to consist of logic programs with a unique stable model and a consequence relation to be truth in the stable model [6]. This means that if there is a logic program P that $P \mid\!\sim q$ if q is true in the unique stable model of P. To ensure that a unique stable model exists the logic program P is required to be a-cyclic [1]. Recursive definitions are allowed if they are well-founded. The language supports negation-by-failure.

Language $\mathcal{L}_Q$ is the language of queries. $\mathcal{L}_Q$ is a propositional language which allows for arbitrary logical connectives. Therefore it has support for the most common connectives as negation, conjunction, disjunction and implication. The atoms of $\mathcal{L}_Q$ correspond to set of ground atoms of $\mathcal{L}_{\mathcal{F}}$.

Definition 4.5. *An independent choice logic theory on base $\langle \mathcal{L}_{\mathcal{F}}, \mid\!\sim, \mathcal{L}_Q \rangle$ is a pair $\langle \mathcal{C}, \mathcal{F} \rangle$ where*

1. *$\mathcal{C}$, called the choice space, is a set of sets of ground atomic formulae from language $\mathcal{L}_{\mathcal{F}}$, such that if $\chi_1 \in \mathcal{C}, \chi_2 \in \mathcal{C}, \chi_1 \neq \chi_2$ then $\chi_1 \cap \chi_2 = \{\}$. An element of $\mathcal{C}$ is called an* alternative. *An element in an alternative is called an* atomic choice.
2. *$\mathcal{F}$, called the facts or the rule base, is a set of formulae in $\mathcal{L}_{\mathcal{F}}$.*

An alternative is a set of ground atoms, all sharing the same logical variable. The logical variable has one of the atoms as its value, i.e. *one* of the atomic choice of the alternative is $true$ and its other atomic choices are $false$. An alternative is in a certain way comparable to a variable of an enum-type in languages as Java or C++: a variable with a discrete set of values and at any time the variable has exactly one of those values. A choice space is a set of alternatives such that none of the alternatives unify with each other: no two alternatives in a choice space share any atoms. A choice space can be thought of as the set of all distinct variables in a program.

Definition 4.6. *Given an independent choice logic $\langle \mathcal{C}, \mathcal{F} \rangle$ a selector function is a mapping $\tau : \mathcal{C} \rightarrow \cup\mathcal{C}$ such that $\tau(\chi) \in \chi$ for all $\chi \in \mathcal{C}$. The range of a selector function τ , written $\mathcal{R}(\tau)$ is the set $\{\tau(\chi) : \chi \in \mathcal{C}\}$. The range of a selector function will be called a total choice.*

[4]Prolog is an ISO-standardised logic programming language. For more information see the documentation from the International Organization for Standardization (ISO), ISO/IEC 13211-1: Information technology - Programming languages - Prolog and ISO/IEC 13211-2 Modules or http://en.wikipedia.org/wiki/Prolog

A selector function is a function that chooses a particular value for each logical variable from its alternatives, maybe using a probability distribution over the atoms of the alternative. A selector function selects or determines a *possible world*, a world that could exists according the possible values of the variables. There are as many selector functions as there are possible worlds. If the Herbrand universe of the logical program is finite, i.e. there are no function symbols, the number of worlds is the product of the number of atoms per alternatives and the number of logical variables and constants. If the Herbrand universe of the logical program is infinite, i.e. there are function symbols, the number of worlds in infinite.

It is possible to calculate the probability distribution of the worlds and the propositions that are part of those worlds. The selector function defines the probability of a world which is the product of the probabilities of the atomic choices of that possible world. The sum of the probabilities of the possible worlds in which a proposition is true is the probability of that proposition. If the number of worlds is infinite it is more difficult but it is still possible to determine the probabilities of worlds. This is however not part of this thesis. For simplicity I assume that a Herbrand universe is finite.

As an example, take the following ICL theory:

- The choice space $\mathcal{C} = \{\{a_1, a_2, a_3\}, \{b_1, b_2\}\}$
- The facts or rule base $\mathcal{F} = \{c \leftarrow a_1 \wedge b_1, c \leftarrow a_3 \wedge b_2, d \leftarrow a_1, d \leftarrow \neg a_2 \wedge b_1, e \leftarrow c, e \leftarrow \neg d\}$

The ICL theory has two alternatives: $\{a_1, a_2, a_3\}$ and $\{b_1, b_2\}$. The set $\{a_1, a_2, a_3\}$ are the atomic choices for the first alternative, $\{b_1, b_2\}$ are the atomic choices for the second alternative. Further the theory has a number of derived atoms, c, d and e, which are only used in the rule base $\mathcal{F}$.

Given the two alternatives, there are 6 selector functions for the ICL theory that select one value for each alternative: $\tau_1(a_1, b_1), \tau_2(a_1, b_1), \tau_3(a_3, b_1), \tau_4(a_1, b_2), \tau_5(a_2, b_2)$ and $\tau_6(a_3, b_2)$. The 6 selector functions define 6 different possible worlds for the given ICL theory:

$w_{\tau_1(a_1,b_1)}$	$\models$	a_1	$\neg a_2$	$\neg a_3$	b_1	$\neg b_2$	c	d	e
$w_{\tau_2(a_2,b_1)}$	$\models$	$\neg a_1$	a_2	$\neg a_3$	b_1	$\neg b_2$	$\neg c$	$\neg d$	e
$w_{\tau_3(a_3,b_1)}$	$\models$	$\neg a_1$	$\neg a_2$	a_3	b_1	$\neg b_2$	$\neg c$	d	$\neg e$
$w_{\tau_4(a_1,b_2)}$	$\models$	a_1	$\neg a_2$	$\neg a_3$	$\neg b_1$	b_2	$\neg c$	d	$\neg e$
$w_{\tau_5(a_2,b_2)}$	$\models$	$\neg a_1$	a_2	$\neg a_3$	$\neg b_1$	b_2	$\neg c$	$\neg d$	e
$w_{\tau_6(a_3,b_2)}$	$\models$	$\neg a_1$	$\neg a_2$	a_3	$\neg b_1$	b_2	c	$\neg d$	e

Table 4.1.: The possible worlds for the ICL theory

Definition 4.7. *A multi-agent independent choice logic theory is a tuple* $\langle \mathcal{C}, \mathcal{F}, \mathcal{A}, controller, P_0 \rangle$ *where*

- *$\mathcal{C}$ is the choice space as defined before,*
- *$\mathcal{F}$ are the facts as defined before, and more specific, an a-cyclic logic program such that no atomic choice unifies with the head of any rule,*
- *$\mathcal{A}$ is a finite set of agents, including a distinguished agent called "nature" or "agent 0",*
- *$\mathit{controller}$ is a function from $\mathcal{C} \rightarrow \mathcal{A}$. If $\mathit{controller}(\chi) = a$ than agent a is said to control alternative χ. If $a \in \mathcal{A}$ is an agent, the set of alternatives controlled by a is $\mathcal{C}_a = \{\chi \in \mathcal{C} : \mathit{controller}(\chi) = a\}$. Note that $\mathcal{C} = \cup_{a \in \mathcal{A}} \mathcal{C}_a$*
- *P_0 is a function $\cup\mathcal{C}_0 \rightarrow [0,1]$ such that $\forall \chi \in \mathcal{C}_0, \sum\limits_{a \in \chi} P_0(\alpha) = 1$.*

The facts of an ICL theory introduce two constraints on the Prolog-based language $\mathcal{L}_{\mathcal{F}}$. First, the program must be a-cyclic. This basically means that the program terminates and has a unique Herbrand model [1]. Second, no atomic choice may unify with the head of any rule which means that not atom from any rule body may appear in any head of any rule.

A *multi-agent* ICL theory also adds the concept of one or more agents. An agent is something that acts in the world. Poole distinguishes two types of agents. Purposive agents prefer some states of the world to other states and try to achieve their preferred worlds. All non-purposive agents together are called "nature". Agents reason and act in time. An agent should react to the world. It adapts its actions according to changes in the world. All alternatives are uniquely controlled by some agent and no two agents share control over the same alternative. There is one agent, called nature or agent 0 that is always present and the alternatives controlled by that agent have a probability distribution such that the sum of the probabilities of all elements in an alternative is 1.0. The control over the alternatives by nature is probabilistic. The probability of a possible world is the product of the probabilities of the choice by nature per rule.

Definition 4.8. *Utility is a function from agent and world to a cardinal value which represents the worth of that world for an agent.*

Utility is a way to represent the "happiness" of an agents with regard to a certain world. An agent will try to maximise its utility by acting in the world.

Definition 4.9. *ICL theory $\langle \mathcal{C}, \mathcal{F}, \mathcal{A}, \mathit{controller}, P_0 \rangle$ is*

- *utility consistent for agent $a \in A$ where $a \neq 0$ if, for each possible world w_r, $w_r \models \mathit{utility}(a, u_1) \wedge \mathit{utility}(a, u_s)$ implies $u_1 = u_2$. The theory is utility consistent if it is utility consistent for all agents (other than nature).*
- *utility complete for agent $a \in A$ where $a \neq 0$ if, for each possible world w_r, there is a unique number u such that $w_r \models \mathit{utility}(a, u)$. The theory is utility complete if it is utility complete for all agents (other than nature).*

Poole assumes that all ICL theories are utility consistent and complete, which means that no agent has two different utility values for the same world and for all agents and for all worlds the utility value is known. Nature, although it is an agent, has no concept of utility. This is not surprising as nature is not utility driven: nature does not want things to happen but just lets them happen.

Below is an example of a set of utility functions for two agents for the example ICL theory.

$utility(agent_1, 5)$	$\leftarrow$	$\neg e$
$utility(agent_1, 0)$	$\leftarrow$	$c \wedge e$
$utility(agent_1, 9)$	$\leftarrow$	$\neg c \wedge e$
$utility(agent_2, 7)$	$\leftarrow$	d
$utility(agent_2, 2)$	$\leftarrow$	$\neg d$

Table 4.2.: The utilities for two agent

Given the utility functions for $agent_1$ and $agent_2$, the example ICL is both utility consistent and complete for $agent_1$ and $agent_2$ as can be ssen in the next table. Every agent has a utility defined in every world as can be seen in the next table.

$w_{\tau_1(a_1,b_1)}$	$\models$	a_1	b_1	c	d	e	$utility(agent_1, 0)$	$utility(agent_2, 7)$
$w_{\tau_2(a_2,b_1)}$	$\models$	a_2	b_1	$\neg c$	$\neg d$	e	$utility(agent_1, 9)$	$utility(agent_2, 2)$
$w_{\tau_3(a_3,b_1)}$	$\models$	a_3	b_1	$\neg c$	d	$\neg e$	$utility(agent_1, 5)$	$utility(agent_2, 7)$
$w_{\tau_4(a_1,b_2)}$	$\models$	a_1	b_2	$\neg c$	d	$\neg e$	$utility(agent_1, 5)$	$utility(agent_2, 7)$
$w_{\tau_5(a_2,b_2)}$	$\models$	a_2	b_2	$\neg c$	$\neg d$	e	$utility(agent_1, 9)$	$utility(agent_2, 2)$
$w_{\tau_6(a_3,b_2)}$	$\models$	a_3	b_2	c	$\neg d$	e	$utility(agent_1, 0)$	$utility(agent_2, 2)$

Table 4.3.: The utilities of the agents in the possible worlds

Definition 4.10. *If $\langle \mathcal{C}, \mathcal{F}, \mathcal{A}, controller, P_0 \rangle$ is an ICL theory and $a \in \mathcal{A}, a \neq 0$, then a strategy for an agent a is a function $P_a : \cup\mathcal{C}_a \rightarrow [0,1]$ such that $\forall \chi \in \mathcal{C}_a \sum_{\alpha \in \chi} P_a(\alpha) = 1.0$*

Definition 4.11. *A pure strategy for agent a is a strategy for agent a such that the range of P_a is $\{0,1\}$*

In other words, P_a assigns a probability of 1.0 (selects) to one atom for every alternative that is controlled by agent a and assigns a probability of 0.0 to the other atoms of that alternative. A pure strategy thus corresponds to a selector function on $\mathcal{C}_a$ and chooses an atom or value for each alternative an agent controls. Just like with utility and for the same reasons, nature has no concept of strategy.

Definition 4.12. *A strategy profile is a function from agents other than nature into strategies for the agents. If σ is a strategy profile and $a \in \mathcal{A}, a \neq 0$ than $\sigma(a)$ is a strategy for agent a. We write $\sigma(a)$ as P_a^σ to emphasise that σ induces a probability over the alternatives controlled by agent a.*

Definition 4.13. *If ICL theory $\langle \mathcal{C}, \mathcal{F}, \mathcal{A}, controller, P_0 \rangle$ is utility consistent and complete, and σ is a strategy profile, then the expected utility in world w_τ for agent $a \neq 0$, under strategy profile σ is*

$$\epsilon(a,\sigma) = \sum_{\tau} p(\sigma,\tau) \times u(\tau,a)$$

(summing over all selector functions τ)

where

$$u(\tau,a) = u \text{ iff } w_\tau \models utility(a,u)$$

(this is well defined as the theory is utility consistent and complete),

and the probability of world τ given strategy profile σ is

$$p(\sigma,\tau) = \prod_{\chi \in \mathcal{C}} P^{\sigma}_{controller(\chi)}(\tau(\chi))$$

This definition means that given a strategy profile the probability of the resulting world and the utility of that world for an agent can be determined. This means that an agent in ICL is a utility-based agent.

So far the model of ICL was static. ICL also has a dynamic model. Before going into detail some concepts on the dynamics of the model must be introduced. First the concept of time is introduced. Time can be both discrete and continuous but Poole considers in his article only discrete time that is totally ordered and has some metric over time intervals. Second the concepts of trace and transduction are introduced. A trace is a function from time into some domain. There are two kind of traces, input traces and output traces. Input traces are the things that can be observed at a certain time. Output traces are the choices that are made or actions that are done at a certain time. A transduction is a function from input traces into output traces such that the output at some time t_{output} depends on the input at some time t_{input} and $t_{input} < t_{output}$. In other words, the outputs depend on the previous inputs. The third concept is that of the state of an agent. The state of an agent is the information that an agent has (or should have remembered) in order for the output to be a function of the state and the current inputs. The amount of state is not fixed for all types of agents. Simple reflexive agents do not maintain state more than their current inputs whereas more sophisticated model- and utility-based agents may maintain their entire history as state. The three concepts together result in a chain of agent states that are connected through input and output traces.

An agent is specified in an so called agent specification module (ASM).

Definition 4.14. *An agent specification module for agent $a \neq 0$, written ASM_a, is a tuple $\langle \mathcal{C}_a, \mathcal{O}_a, \pi \rangle$ where*

- *$\mathcal{C}_a$ is a set of alternatives controlled by agent a.*
- *$\mathcal{O}_a$ the observables, a set of sets of ground atomic formulae.*
- *π, the observable function, is a function $\pi : \mathcal{C}_a \to 2^{\mathcal{O}_a}$*

$\mathcal{C}_a$ are the same alternatives as the alternatives defined by the controller in a multi-agent ICL theory. The observables are facts or percepts of the world that an agent "can see". Elements of $\mathcal{O}_a$ are called observation alternatives and the elements of observation alternatives are called atomic observations. The observable function is very much like the selector function mentioned earlier. Notice that although nature is defined as an agent that is always present, it has no agent specification module.

The definition of a multi-agent independent choice logic theory is now altered to support the dynamic nature.

Definition 4.15. *A dynamic multi agent independent choice logic theory is a tuple* $\langle \mathcal{A}, \mathcal{C}_0, \mathcal{F}_0, P_0, ASM \rangle$
where

- $\mathcal{A}$ *is a finite set of agents, including a distinguished agent called "nature" or "agent 0"*
- $\mathcal{C}_0$ *is the choice space controlled by nature,*
- $\mathcal{F}_0$ *nature's facts, an a-cyclic logic program such that no atomic choice unifies with the head of any rule,*
- P_0 *is a function* $\cup C_0 \to [0,1]$ *such that* $\forall \chi \in C_0 \sum_{\alpha \in \chi} P_0(\alpha) = 1$ *,*
- ASM *is a function on* $A\{0\}$ *such that* ASM_a *is an agent specification module for agent a.*

such that $\mathcal{F}_0$ *is a-cyclic with an a-cyclic ordering where* $\forall a \in \mathcal{A}, \forall \chi \in \mathcal{C}_a, \forall \alpha \in \chi, \forall O \in \pi(\chi), \forall \alpha' \in O, \alpha' < \alpha$ *in the a-cyclic ordering. That is, there is an a-cyclic ordering where the actions are after their corresponding observables.*

The above definition says that in a dynamic ICL theory agents can always look at observables before making choices about the variables they control (actions). Nature controls the chances which determine the actual outcome of those choices (actions). Which observable can be seen by which agent is determined by a *choice function.* If multiple agents can look at the same set of choice variables the function is called *non-exclusive*, otherwise it is called *exclusive.* If every choice variable can be seen by some agent the function is called *covering.* A dynamic ICL theory is *observation consistent* if the choice function ensures that every agent only has its own choice variables as observables. The dynamic ICL theory is *observation complete* if the choice function ensures that an agent can see all possible values for its observables. Theories are required to be observation consistent but not observation complete. This means that an agent always can see the observables the agent knows of but that there may unknown choice variable it is not aware of.

Just as in a static ICL theory an agent will use a strategy to determine what the agent will do but in a dynamic ICL theory the agent can use what it has observed and remembered. A strategy is a program of the form "if observed state of the world is like this than my actions will be these so that my utility is as high as possible". More formally:

Definition 4.16. *If $\langle \mathcal{C}_a, \mathcal{O}_a, \pi_a \rangle$ is an agent specification module for agent $a \in \mathcal{A}$, then a pure strategy for agent a is a logic program $\mathcal{F}_a$ such that*

- *$\mathcal{F}_a$ is a-cyclic with an a-cyclic ordering such that, for every $\chi \in Ca$, every element of each element of $\pi_a(\chi)$ is before every element of χ.*
- *For every $\chi \in \mathcal{C}_a$, and for every selection function $\tau_{\pi(\chi)}$ on $\pi(\chi)$, there is a unique $\alpha \in \chi$ that is true in the unique stable model of $\mathcal{F}_a \cup \mathcal{R}(\tau_{\pi(\chi)})$.*
- *The heads of rules in $\mathcal{F}_a$ only unify with either local atoms (that do not unify with atoms in the agent specification module of any other agent or in $\mathcal{F}_0$) or members of choice alternatives in χ_a.*
- *For each $\chi \in \mathcal{C}_a$, the only formulae that can appear in the bodies of rules in $\mathcal{F}_a$ to prove an element of χ are:*
  - *elements of members of $\pi(\chi)$,*
  - *local atoms,*
  - *atoms whose definition do not depend on the choices of other agents (and formulae built from these).*

This means three things. First, agents can observe all their observables before making a decision. Second, the decision or logical program specifies what the agent will do. Third, the decisions can only involve variables the agent has knowledge off or can compute and will only be about the alternatives the agent actually controls.

Given the pure strategy for each agent, the pure strategy profile is defined, analogous again with the static ICL theory.

Definition 4.17. *Given a dynamic ICL theory, a (pure) strategy profile is a selection of one (pure) strategy for each agent (other than nature). Thus a strategy profile is a logic program $\mathcal{F} = \cup_{a \in \mathcal{A}} \mathcal{F}_a$ that specifies what each agent will attempt to do.*

There are two (equivalent) ways to define the semantics.

One is it to have a possible world for each selection of an element from each alternative controlled by nature, and to have $\mathcal{F}$ specify what is true in each world. In this case, the probability of a world is independent of the strategy, but a strategy profile specifies what is true in each world.

The second is to have a possible world for each selection of one element from each alternative. In this case, what is true in a world does not depend on the strategy profile chosen, but it depends on the probability of a world. The second has many possible worlds with zero probability that were not created in the first scenario.

Poole only defines the first method formally.

Definition 4.18. *If dynamic ICL theory $\langle \mathcal{A}, \mathcal{C}_0, \mathcal{F}_0, P_0, ASM \rangle$ is utility complete, and $\mathcal{F}$ is a pure strategy profile, then the expected utility in world w_τ of strategy profile $\mathcal{F}$ for agent a is*

$\epsilon(a, \mathcal{F}) = \sum_{\tau} p(\tau) \times u(\tau, a, \mathcal{F})$

(summing over all selector functions τ on $\mathcal{C}_0$)

where

$u(\tau, a, \mathcal{F}) = u$ *if* $\mathcal{R}(\tau) \cup \mathcal{F} \mid\!\sim$ *utility*(a, u)

(this is well defined as the theory is utility complete)

and the probability of world τ is

$p(\tau) = \prod_{\chi \in \mathcal{C}_0} \mathcal{P}_0(\tau(\chi))$

Like in static ICL, this definition also means that given a strategy profile the probability of the resulting world and the utility of that world for an agent can be determined.

A pure strategy profile is based on one specific pure strategy per agent. However, multiple pure strategies per agent are possible. Hence it is possible to define a probability distribution over multiple pure strategies per agent. This of course leads to probabilistic strategy profiles and expected utilities. The formal definitions are in Poole [20] and are not repeated here.

ICL is a large framework that is created to model agents and their behaviour using probability. ICL has four basic concepts: enum-like variables, the possibility to choose a value for a variable in a possible world, probability distributions over the values and the resulting worlds and utilities that result from the resulting worlds. It also introduces facts and relations between the facts about the variables by providing a Prolog-based language. With these constructs ICL can be used to express other formalisms which are needed to implement suitable semantics. Poole describes implementations of several formalisms: PHA, influence diagrams (whose formalism is largely based on PHA), Markov decision problems (MDPs) and strategic form of games. First, I will look into Poole's PHA theory, which is in fact the first and best described formalism that is used in combination with ICL. Second, I will look into Vennekens' CPL. At the end of this chapter I will compare both theories in relation to ICL and each other.

4.3. PHA

Probabilistic Horn abduction (PHA) is a framework that combines Horn clauses with probabilities and hypotheses. Poole used a simplified form of an implementation of default and abductive reasoning, Theorist [17], with the addition of probabilities to develop PHA. It is a Prolog-based logical language and is in essence a restricted form of ICL. It has only choices by nature, has no negation as failure and has restrictions on the rules. PHA provides probabilities over possible worlds. A system specified in PHA is easily translated into a Bayesian network and vice versa. All definitions are directly taken from Poole [18].

PHA uses two syntactical constructs, the definite clause, as described in 4.1, and the disjoint declaration.

Definition 4.19. *A* disjoint declaration *is of the form $disjoint([a_1 : p_1, \ldots, a_n : p_n])$. where the a_i are atoms, and the p_i are real numbers $0 \le p_i \le 1$ such that $p_1 + \ldots + p_n = 1$. Any variable appearing in one a_i must appear in all of the a_j (i.e., the a_i share the same variables). The a_i will be referred to as* hypotheses *or* assumables*.*

The atoms h_i in this definition correspond to atoms in some definitive clause. This means that a definite clause of the form $a \leftarrow a_1 \wedge \ldots \wedge a_n$. together with the complimentary disjoint declaration in the form of $disjoint([a_1 : p_1, \ldots, a_n : p_n])$ should be read as that if one of the hypothesis $a_1 \wedge \ldots \wedge a_n$ with a probability distribution of $a_1 : p_1, \ldots, a_n : p_n$ is true, then deterministic and definitely a is true. Although not mentioned in this definition, it is not allowed for a hypothesis to appear in any head of a clause.

Definition 4.20. *A* probabilistic Horn abduction (PHA) theory *is a collection of definite clauses and disjoint declarations such that if a ground atom a is an instance of a hypothesis in one disjoint declaration, then it is not an instance of another hypothesis in any of the disjoint declarations.*

There may be equal or more definite clauses than disjoint declarations in a theory but not less. Note that the collection of definite clauses together form an ordinary Prolog program. All Prolog constructs, including functions, are allowed. In this thesis I will only consider simple logical programs with a finite Herbrand universe. The most important consequence of this is that the number of possible worlds is finite. This simplifies the explanation of PHA but it is not a restriction on PHA itself. If function symbols are used in a logical program the Herbrand universe is infinite and therefore the number of possible worlds is infinite. The general idea is to define a measure over sets of possible world as is described by Poole[19]. An other possibility is to limit the nesting level of functions. This is in general no problem as the probability of a world decreases with each nesting and eventually approaches 0.

An example of a PHA theory as described above:
$disjoint([A(yes) : 0.3, A(no) : 0.7])$.
$disjoint([B(yes) : 0.7, B(no) : 0.3])$.
$C \leftarrow A \wedge B \wedge C \wedge C_{AB}(C, A, B)$.
$disjoint([C_{AB}(yes, yes, yes) : 0.2, C_{AB}(no, yes, yes) : 0.8])$.
$disjoint([C_{AB}(yes, no, yes) : 0.1, C_{AB}(no, no, yes) : 0.9])$.
$disjoint([C_{AB}(yes, yes, no) : 0.5, C_{AB}(no, yes, no) : 0.5])$.
$disjoint([C_{AB}(yes, no, no) : 0.75, C_{AB}(no, no, no) : 0.25])$.
$D \leftarrow C \wedge D_C(D, C)$.
$disjoint([D_C(yes, yes) : 0.36, D_C(no, yes) : 0.64])$.
$disjoint([D_C(yes, no) : 0.82, D_C(no, no) : 0.18])$.

Given a PHA theory T, some more terms are introduced.

- F_T the facts, is the set of definite clauses in T, together with the clauses of the form $false \leftarrow h_i \wedge h_j$ where h_i and h_j both appear in the same disjoint declaration in T , and $i \neq j$. Let F'_T be the set of ground instances of elements of F_T.

- H_T the hypotheses, the set of h_i that appears in some disjoint declaration in T. Let H'_T be the set of ground instances of elements of H_T.
- P_T is a function $H'_T \mapsto [0,1]$. $P(h'_i) = p_i$ where h'_i is a ground instance of hypothesis h_i, and $h_i : p_i$ is in a disjoint declaration in T. $P(h'_i)$ will be the prior probability of h'_i.

In the remainder of this thesis the subscript T will be omitted if T is understood from the context.

The above definition of a PHA theory corresponds closely with an ICL theory: the collection of definite clauses with the ICL facts, the disjoint declarations with the ICL choice space and its probability distributions.

The semantics of a PHA theory comes from the two primary ideas on which the theory is founded, abduction and probability. Both abduction and probability require certain assumptions on the syntax, world and probability that introduce some restrictions in PHA. Because the basic reason for the restrictions comes from the wish to be able to calculate the probabilities of explanations in a system, I will shortly introduce abduction before going into the assumptions and the restrictions on the rules that follow from those. Abduction is a kind of logical inference that tries to explain one ore more observations by finding a reason: given evidence, the observation(s), and a number of explanations, the reasons for the observation(s), one infers the reason that best explains the observation(s)[5].

Definition 4.21. *An abductive scheme is a pair $\langle F, H\rangle$ where*

- *F is a set of Horn clauses. Variables in F are implicitly universally quantifies. Let F' be the set of ground instances of elements of F.*
- *H is the set of (possible open) atoms called the assumables or "possible hypotheses". Let H' be the set of ground instances of elements of H.*

Definition 4.22. *If g is a closed formula, an explanation of g from $\langle F, H\rangle$ is a set D of elements of H' such that*

- *$F \cup D \models g$, and*
- *$F \cup D \not\models false$.*

The first condition says that D is sufficient to imply g: it is possible to prove g from $F \cup D$. It is possible that multiple sets of D can be found that imply g. The second condition says that D is possible: it can not be the case that $F \cup D$ is inconsistent.

Definition 4.23. *A minimal explanation of g is an explanation of g such that no strict subset is an explanation of g*

[5] For a more elaborate treatment of abduction see The Stanford Encyclopedia of Philosophy (Spring 2011 Edition)

If there are multiple sets of D, the smallest D is a minimal explanation. The smallest D is the intersection of all Ds. It is possible to have multiple minimal explanations if the union two sets of D is empty.

To be able to calculate the probability of explanations Poole has to introduce some assumptions about the knowledge base. The assumptions can be grouped into syntactical, world and probabilistic assumptions.

The syntactical assumptions are:

- There are no rules in F whose head unifies with a member of H,
- If F' is the set of ground instances of elements of F, it is possible to assign a natural number to every ground atom such that for every rule in F' the atoms in the body of the rule ate strictly less than the atom in the head.

The first assumption says that rules may not imply a hypothesis that is used in another rule, i.e. a hypothesis that appears in the body of some rule may not appear as the head of another rule. Hypotheses may not be chained through rules. The second assumption means that the logical program must be a-cyclic and that all recursion must be well-defined.

The world assumptions are:

- The rules in F' for every ground non-hypothesis represented atom are covering,
- The bodies in the rules in F' for an atom are mutually exclusive.

The first assumption means that there are no missing atoms in the set of hypothesis in the body of a rule: all hypothesis are known. The second assumption states that no body may be re-used in any rule. Combined with the first syntactical assumption this means that there is global and complete knowledge on the possible causes for a effect and that this is stated in one rule.

The probabilistic assumptions are:

- Ground instances of hypotheses are consistent (with F_T) are probabilistically independent.

This assumption states that the rules are probabilistic independent. This assumption is not enforced in the world model and therefore it is the responsibility of the developer of a PHA theory.

Using these assumptions Poole concludes the following probabilities for explanations.

If $D = \{h_1, \ldots, h_n\}$ is a minimal explanation then the probability of that explanation is

$$P(e) = P(h_1 \wedge \ldots \wedge h_n) = \prod_{i=1}^{n} P(h_i)$$

If $expl(g, T)$ is the set of minimal explanations of conjunctions of atoms g from PHA theory T than the probability of that set of explanations is

$$P(g) = P\left(\bigvee_{e_i \in expl(g,T)} e_i \right) = \sum_{e_i \in expl(g,T)} P(e_i)$$

These calculations can be generalised for every combination of hypothesis, whether that combination is an explanation or not. Such generalisation can be used to translate Bayesian networks into PHA theories and vice versa. The translation of a Bayesian network to a PHA theory is as follows. We have a random variable a with the variables $b_1, \ldots, b_n$ as its parents. The value of the random variable a is represented as as the proposition $a(v)$. This variable and its parents can be translated in to the following PHA rule: $a(v) \leftarrow b_1(v_1) \wedge \ldots \wedge b_n(v_n) \wedge c\_a(v, v_1, \ldots, v_n)$ where $c\_a(v, v_1, \ldots, v_n)$ is a possible hypothesis.

This states the hypothesis that variable a has value V because all the parent variables b_i have the value v_i. The probability of this hypothesis than is $P(a = v|b_1 = v_1 \wedge \ldots \wedge b_n = v_n)$. The probability of $a = V$ in general then is $P(a = V|b_1 = V_1 \wedge \ldots \wedge b_n = V_n)$. This can be translated into a disjoint declaration of the form $disjoint([c_a(v_1, v_1, \ldots, v_n) : p_1, \ldots, c_a(v_n, v_1, \ldots, v_n) : p_n])$

The reverse translation is also possible. To translate a PHA theory into a Bayesian network, one grounds the theory, substituting ground terms for the free variables. The ground atoms than are the nodes of the Bayesian network. All rules are arcs in the network where the atoms in the bodies of the rules that imply an atom a become the parents of that atom a. The CPT of a node follows from the disjoint declaration of the body atoms.

A translation of the PHA theory of the the previous example would give the following Bayesian network as in figure 4.2.

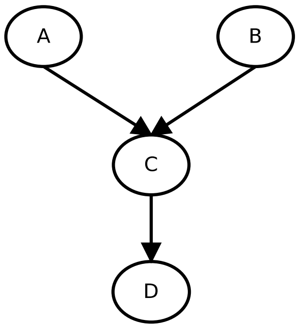

Figure 4.2.: A PHA theory as Bayesian network

In appendix A, section A.2, a more elaborate example is given of a translation of Bayes to PHA and vice versa.

4.4. CP-Logic

CP-logic (CPL) is a logical language for specifying probabilistic causal laws. Its starting point is that causality has an inherent dynamic aspect that translates into events and that one event leads to (causes) another event or events. It extends the notion of Shafer's event trees [26] with a logical language. Using the event tree representation probabilities over possible worlds are specified. LPAD, the precursor of CPL, is used by Meert [13] to show that a system specified in CPL is easily translated into a Bayesian network and vice versa. All definitions are directly taken from Vennekens [28].

ICL has just one syntactical construct, the causal probabilistic law or rule.

Definition 4.24. *A* causal probabilistic law, or CP-law, or rule *is of the form*
$\forall x(h_1 : \alpha_1) \vee \ldots \vee (h_n : \alpha_n) \leftarrow \phi$
where

- h_i *are atoms, such that the universally quantified tuple of variables* x *contains all free variables in* ϕ *and* h_i*.*
- *all* $\alpha_i \in (0, 1]$ *and* $\sum_{i=1}^{n} \alpha_i \leq 1$,
- ϕ *is a first-order formula.*

The set of $(h_i : \alpha_i)$ is called the head of a rule and ϕ is called the body of a rule. If the head contains only one atom $h : 1$ the probability may be omitted. A rule should be read as "for all x, if the body of a rule (ϕ) "fires", that is, the body is "true", then an event is caused whose effect is that at most one of the atoms of the head $((h_1 : \alpha_1) \ldots \vee (h_n : \alpha_n))$ becomes true, where the probability of atom h_i becoming true is α_i." A rule can be thought of as a probabilistic causal law and can be read as "the body causes one of the possible effects" or in traditional backward Prolog style "possible effects $\leftarrow$ cause". The use of the term "event" differs from its standard use in probability theory, where it denotes a set of possible outcomes of an experiment. Shafer uses the term *Humean event* for an event that causes a transition between states and the term a *Demoivrian* for a a set of possible outcomes [26]. In CPL "event" means the occurrence of something that causes a transition from one state to the next, a Humean event.

All first-order formulae are allowed for ϕ, including functions. Just as with PHA, and for the same reasons, I will consider only grounded formulae with a finite Herbrand universe. Riguzzi and Swift have described the semantics of LPADs with function symbols [23]. They follow the approach used by Poole as described in [19].

Definition 4.25. *A CP-Logic theory consists of a finite set of causal probabilistic laws.*

PHA introduced some (syntactical) restrictions on the rules of a theory. A CPL theory has almost no restrictions. More in particular, it explicitly allows that multiple rules attribute to the same result. A CPL theory is a multi-set of rules. Both body and head can appear multiple times in a CPL theory. CPL also supports negation. Atoms that are part of the head of a rule can also be part of the body of another rule. Under certain circumstances recursion is allowed (see later in this chapter).

An example of a CPL theory as described above:
$(A : 0.3)$.

$(B : 0.7)$.
$(C : 0.3) \leftarrow A$.
$(C : 0.7) \leftarrow B$.
$(D : 0.4) \leftarrow C$.

The semantics of a CPL theory are based on the Shafer's event of probability trees. Event trees are a trees built from all (logical) possible paths that can be formed from all events. A path is formed by the order of firing of the events. The order of firing is specified by the rules. Events are synchronised by the rules: all events in the body should happen before an event in the head of a rule. Because events in the body of one rule can also be part of the head of other rules chains of events can happen. Meert lists the principles that govern the firing of rules [13]:

- The principle of universal causation states that an endogenous property can only be true if it has been caused by some event, i.e., all changes to the endogenous state of the domain must happen as the consequence of an event.
- The principle of sufficient causation states that if an event has a cause, then it must eventually occur.
- The principle of independent causation states that every event affects the state of the world in a probabilistic independent way, i.e., knowing the outcome of one event does not give any information about the outcome of a different event. This principle ensures the modularity and robustness of the representation.
- The principle of temporal precedence states that, whenever a property ϕ might cause an event E, then the part of the process that is involved in determining the truth of ϕ happens before the event E itself can happen. This principle is motivated by the fundamental property of the physical world that a cause must always preceed its effects. This principle ensures that cyclic causal processes in a CP-theory happen in a consistent way.

For a formal treatment of how Vennekens builds a tree using these concepts see his article [28]. Assuming that the CP-events have been grounded and that the Herbrand universe is finite, Meert [13] describes the algorithm to build a probability tree as follows:

1. Start with a root $\perp$, with $Pr(\perp) = 1$ and an interpretation $\mathcal{I}(\perp) = \{\emptyset\}$ (the set of events that have happened).
2. Associate with the current node n a CP-event e from the CP-events that fulfils the following properties:
   - The ground CP-event e is not yet associated with another node.
   - The ground CP-event e has at least one atom in $head(e)$ that is not in $\mathcal{I}(n)$.
   - All the literals in $body(e)$ are true given the current interpretation $\mathcal{I}(n)$.
   - For all negative literals in $body(e)$, all CP-events with those literals in the head are associated with a node on the path to the root or their body cannot become true in the current or a future state. In other words, the precondition is in its final state, it is not only true at this point but is guaranteed to

remain true in all potential future states. This step ensures that cyclic causal processes are handled consistently. We first need to apply all possible causes for something to happen before concluding something is not caused.

3. For every $(h_i : \alpha_i)$ in $head(e)$, create a child node c_i with $\mathcal{I}(c_i) = \mathcal{I}(n) \cup h_i$ and $Pr(c_i) = Pr(n)\alpha_i$.
4. Go to step 2 for every child node. If no event is found in step 2, stop.

This builds an event tree that represents all possible final states that are described by the CPL theory and the event paths that lead to those states, including a probability distribution over the states and a probability for an event path.

In a CPL theory there are two sorts of events: *exogenous* and *endogenous events*. The former are events that are introduced in the head of an empty rule, i.e. a rule without a body. The latter are events that are introduced in the head of a rule. Every exogenous event can be seen as the start of a chain or tree of rules. Exogenous events are events that happen outside the system but that have their effect inside the system. They are the start of a sequence of endogenous events, the events in the system. If a CPL theory has only one exogenous event then it can be represented a a tree. If a CPL theory has more than one exogenous event it can be represented either as a tree or a multi-root graph. The tree is build by chaining the individual trees of the exogenous events one by one. This is the representation Vennekens uses in his article. The multi-root graph is built by taking each of the trees with an exogenous event as root and create the graph, where two or more trees are joined by the rules that have events from those trees in the body of a rule. The result is a kind of flowchart that represents the possible chain of events. I will look into this representation in a later chapter of this thesis.

Vennekens calls the tree of events a *probabilistic* Σ*-process*. Every path from the root of such tree to an event somewhere in tree is called an *execution model*. It does not exactly mean that the events will exactly happen in the order of the given path but it will adhere to the principles of firing of rules. Because of the principle of independent causation, every path with exactly the same events and that adheres to the principles of firing of rules will have the same probability, irrespective of the exact order of the events.

Vennekens also shows that some more complex processes can be modelled in CPL. It is possible to use negation. It is also possible to use temporal rules in a CPL theory, although under specific circumstances it is up to the designer of the CPL theory to design his theory in such a way that the intended causal processes satisfy the temporal precedence assumption.

Vennekens shows that it possible to translate a Bayesian network into a CPL theory. He notices however, that a CPL theory is more expressive than a Bayesian network. In particular, a CPL theory allows for multiple independent causes for the same effect. A CPL theory also supports first-order logic representation of causes. It also supports cyclic causal relation as long as the relation results in a valid execution model. All these constructs can not be represented in a Bayesian network. The fact that these constructs cannot be represented has no repercussions for the probability distribution. The extra constructs are part of the modelling possibilities, not of the probability distribution.

Vennekens has defined LPAD, a probabilistic extension of logic programming based on disjunctive logic programs. LPAD is syntactic identical to ICL but has different semantics.

Definition 4.26. *An LPAD is a set of rules of the form:*
$(h_1 : \alpha_1)$. *or* $(h_1 : \alpha_1) \ldots \vee (h_n : \alpha_n) \leftarrow \phi$
where

- h_i *are ground atoms,*
- *all* $\alpha_i \in (0,1]$ *and* $\sum_{i=1}^{n} \alpha_i \leq 1$,
- ϕ *is a sentence.*

Such LPAD rule can be read as a probability distribution over rule as the conjunction of programming rules of the form $\{(h_i \leftarrow \phi) | 1 \leq i \leq n\}$. An instance of an LPAD is a selection of one head h_i for each distinct body ϕ. An LPAD is sound if all instances of of that LPAD have an exact well-founded or a-cyclic model. The set of sound LPADs than describes all possible worlds over which a probability distribution can be calculated.

Based on the LPAD semantics Meert [13] shows that a CPL theory with well founded recursion and a finite Herbrand universe can be translated into a Bayesian network via which he calls an equivalent Bayesian network (EBN) by using the following algorithm.

First, the CPL theory needs to be grounded. From here on, CPL theory refers to the ground CPL theory. Next, the following three steps construct the EBN of the CPL theory.

1. For every atom in the CP-theory, a Boolean random variable is created in the EBN. This is a so-called atom variable and it is represented by an atom node in the network.
2. For every rule in the CP-theory, a choice variable is created in the EBN. This variable can take $n + 1$ values, where n is the number of atoms in the head. It is represented by a choice node in the network. Note that the choice nodes are unobserved, they are artificial nodes that do not correspond to atoms in the domain. These nodes are included instead of the indexed atoms as explained before. When $Ci = j$, this means the j-th atom of rule i has been selected. $Ci = 0$ will be used to denote that no listed head atom was selected, which may be either because the rule body is false, or because no atom was selected when the probabilities do not add up to one.
3. If an atom is in the head of a rule, an edge is created from its corresponding choice node towards the atom's node. If a literal is in the body of a rule, an edge is created from the literal's atom node towards the rule's choice node.

For positive body literals, "$\leftarrow$" is used for the edge to the choice node. For negative body literals, a dashed arrow, "←--", is used. This notation makes distinction possible between positive and negative body literals in the EBN, and it ensures that there is a one to one mapping between the EBN structure and the CP-theory rules. Both types of edges are regular BN edges, the difference in meaning between positive and negative body literals is encoded in the CPT of the choice node.

For the CPTs there are two cases:

1. The CPT of a choice variable. Such a variable can take $n+1$ values with n the number of atoms in the head. The variable takes the value i if the i-th atom from the head is chosen by the probabilistic process. It takes the value 0 if none of the head atoms is chosen (this can only happen if the α_i do not sum to one). If the body of the rule is true, then the probability that the variable takes the value $i = 0$ is precisely the causal probability α_i given in the head of the rule. The probability that n it takes the value 0 is equal to $1 - \sum_i \alpha_i$. If the body is not true, then the probability that the choice variable takes the value 0 is 1.0 and all the other values have probability 0.0. Formally,

   $Pr(Ci = j|all\ parents\ true) = \alpha_j$, for all $j > 0$
   $$Pr(Ci = 0|all\ parents\ true) = 1 - \sum_j \alpha_j$$
   $Pr(Ci = 0|not\ allparents\ true) = 1$
   $Pr(Ci = j|not\ all\ parents\ true) = 0$, for all $j > 0$

   The exact column of the CPT that corresponds to "the body is true" will depend on which body literals are positive and which are negative (as indicated with "←" or "←--" in the EBN).

2. The CPT of an atom variable is structured differently. It essentially represents a deterministic OR function of the different rules having the atom in the head. More specifically, if one of the choice variables representing a rule with the given atom in position i of its head takes the value i, then the atom variable will be true with probability 1.0. In all other cases it will be false, also with probability 1.0. This is because the second part of our CP-theory is essentially a standard logic program. Formally, the CPT can be represented as follows:

   $Pr(hi = true|all\ parents\ false) = 0$
   $Pr(hi = true|at\ least\ one\ parent\ true) = 1$

The procedure above translates a CPL theory to a specific form of EBN. It is however possible to translate the same CPL theory to other EBNs without losing any probabilistic information. Meert [14] introduces what he calls the *Bayesian network space.* A Bayesian network space consists of all possible Bayesian networks that can represent the same CPL theory. All definitions are directly taken from Vennekens [27] and [28].

A translation of the CPL theory of the previous example would give the following Bayesian network as in figure 4.3.

A translation from a CPL theory has at least in three situations a choice of to represent certain CPL constructs.

- *CPL theories with multiple rules with the same resulting event:*
  Such theories have the form:

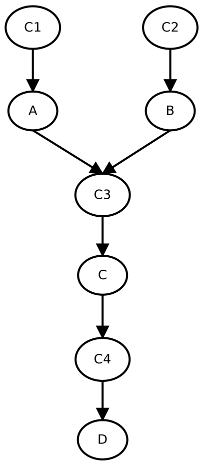

Figure 4.3.: A CPL theory as Bayesian network

$x : \alpha \leftarrow y$
$x : \beta \leftarrow z$

The two graphical representations are given in figure 4.4

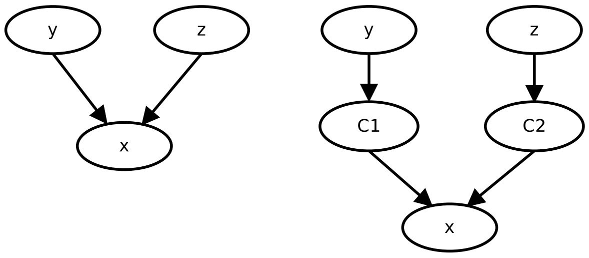

Figure 4.4.: CPL theories with multiple rules with the same resulting event

- *CPL theories with multiple literals in the rule bodies:*
  Such theories have the form:
  $x : \alpha \leftarrow w, y$
  $x : \beta \leftarrow z$

  The two graphical representations are given in figure 4.5.
- *CPL theories with multiple atoms in the rule heads:*
  Such theories have the form:
  $(x : \alpha) \vee (y : \beta) \vee (z : \gamma) \leftarrow .$

  The two graphical representations are given in figure 4.6.

The fact that a choice that is available for the representation of the Bayesian network can be used to make the translation from CPL theory to Bayesian network more semantics preserving. In both the second and third case the right representation is closer to the

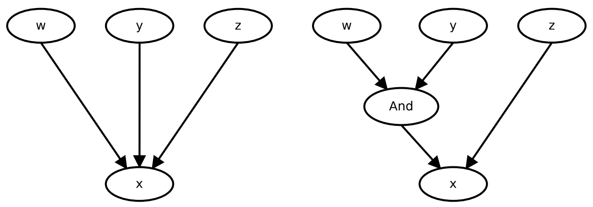

Figure 4.5.: CPL theories with multiple literals in the rule bodies

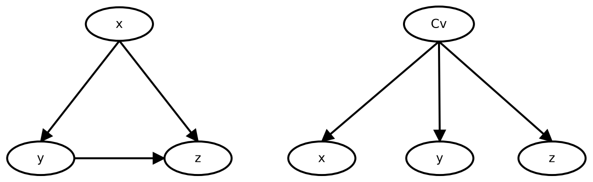

Figure 4.6.: CPL theories with multiple atoms in the rule heads

semantics of the theory than the left representation, while the probabilistic information is the same. In the last case the CPTs of the BN are also much more complicated and less intuitive than the CPs of the EBN. See table 4.4 and table 4.5 for the CPTs that belong to figure 4.6.

		x	
		T	F
y	T	0	$\frac{\beta}{1-\alpha}$
	F	1	$1-\frac{\beta}{1-\alpha}$

x	T	α
	F	$1-\alpha$

		x,y			
		T,T	T,F	F,T	F,F
z	T	0	0	0	$\frac{\gamma}{1-\alpha-\beta}$
	F	1	1	1	$1-\frac{\gamma}{1-\alpha-\beta}$

Table 4.4.: CPT tables for BN

In appendix A section A.4 a more elaborate example is given of a translation of Bayes to CPL and vice versa.

4.5. Comparison

ICL is a large framework that encompasses everything that is needed for modelling multiple agents under uncertainty. An important part of that framework is a combination of a logical language and probability for which formal semantics are defined. As a high-level modelling language ICL uses embedded formalisms for implementation related modelling. As the precursor of ICL, PHA is the most natural embedding. However, PHA has some restrictions on the syntax and uses some assumptions that limit what effectively can be said in ICL if one uses PHA as embedded formalism. As another possible embedding, CPL can be seen as a superset of ICL, allowing for a less restricted syntax for specifying probability and adding the notion of probabilistic causal processes

	a	b	c
Cv	α	β	γ

		Cv		
		a	b	c
a	T	1	0	0
	F	0	1	1

		Cv		
		a	b	c
b	T	0	1	0
	F	1	0	1

		Cv		
		a	b	c
c	T	0	0	1
	F	1	1	0

Table 4.5.: CPT tables for EBN

or state transitions. The comparison will be twofold. First, ICL and CPL are compared as high-level specification language. Second PHA and CPL are compared as embedded formalism. Afterwards a conclusion based on the comparison is drawn.

4.5.1. Comparison of ICL and CPL as specification language

The syntax of CPL [6] is arbitrary more natural than the syntax of ICL. The basic model of ICL with PHA is that the actions of an agent are deterministic and the events are probabilistic. This is however not the model that is used in a DES. In a DES an agent reacts on an event depending on the event and the state of the world. Specifying the reaction on an event is more natural under these circumstances. Most people can also easier find the possible effects of an event than all possible causes for an event, especially in the context of the definition of a training DES where the basic question is often "what should we do if this event happens under these circumstances?". The locality of the causality and probability adds to this. As can be seen in the examples in appendix A, ICL as a language is also more complex to understand. This is partly because it requires two separate specifications for an ICL theory that are dependent on each other. It requires a separate choice space and a separate fact database. The atoms in the choice space are connected to the rule base: they may only appear in one body of one rule base. In addition it is not allowed to chain rules in the rule base by using any alternative as head of a rule.

The syntax of both languages appears to be the opposite of each other: ICL's syntax has probabilistic causes or bodies of rules and deterministic effects, CPL has deterministic causes, i.e. the causing events have definitely happened, and probabilistic effects. Vennekens [27] however shows that each ICL/PHA theory is syntactically also a CPL theory. It is also possible to translate a-cyclic CPL theories without exogenous events to ICL theories. It is however not possible to translate cyclic CPL theories or CPL with exogenous events to ICL. This means that CPL as a language is more expressive than ICL: everything than can be expressed in ICL can be expressed in CPL but the reverse is not true. This is only in in terms of syntax: according to Poole ICL "is a Turing-complete language that can represent arbitrary finite probability distributions"[21]. However, there is one thing that one cannot model: ICL explicitly prohibits that two agent control the same alternative somehow together. All alternatives are assigned to different agents and

[6] As mentioned in 4.2, ICL has no concrete syntax but is a specification of what a logical language should support. However, in this section for the ease of the comparisson I will use ICL with PHA as its embedding as if is the concrete language of ICL.

no two agents can be assigned the same alternative. This can be modelled in CPL as can be seen in the shopping example from Meert [13] where two agents both can buy spaghetti.

ICL (with PHA as its embedded formalism) requires global knowledge of probability. It requires the modeller to describe all possible choices and their probability distribution together with the deterministic effect in one rule/disjoint combination. CPL does not require global knowledge of probability because a theory is a multiset of rules where atoms can appear in heads and bodies of multiple rules (not in the same rule of course). The probability distribution is embedded in the rules themselves. The resulting locality of probability leads to a greater modularity of the language. Using CPL, it is possible to develop CPL theories per agent without the need of coordination on the global level.

ICL has, as a probabilistic language, a limited view on causality. There is some form of causality in dynamic ICL in the sense that input always proceeds output and that the input trace and state are used to calculate output trace. PHA as embedded formalism has no concept of causality. CPL has causality as its very starting point, it is a logical language for representing probabilistic *causal* laws. However, it is not very difficult to extend ICL with causality. Finzi and Lukasiewics have described a bidirectional mapping for ICL and causal models [5]. Also, Vennekens mentions that the equivalence between CPL and ICL provides a causal interpretation for ICL, although a different one than Finzi and Lukasiewics.

Both languages use the notion of possible world semantics. The logic determines the possible states a world can be in, and the probability of a world is the product of all probabilities as given in either the disjoint declarations or as embedded in the rules. There is however subtle and related differences in the semantics of the defined world. First, ICL has its roots in abduction and explains why we are here in this state: it gives the choices made in the bodies that led to the choice. It has however no concept of the order of choices in the past nor in the future. It uses a rather static concept of the world. CPL has its roots in causality and has a dynamic concept of the world. It has both an explanation for the past in terms of the order of events and a notion of future: just follow all possible path(s) from the latest event in the multi-root event graph. Second, CPL has no explicit notion of the value of variables (choices) within the world. It knows that an event has occurred but it does not know what the underlying cause of the event is, typically an agent making a choice. The state of the world in CPL is not defined in terms of variables having a value but in terms of events that have led to the world. It is possible to connect both semantics by adding some interpretation to the theories. First, I will create an interpretation of an agent rule for ICL and second, I will do the same for CPL. I will conclude that it is possible to unify the semantics.

In ICL, a trace is a function from time into some domain. Only discrete and totally ordered time with some metric over time intervals was considered. ICL allows for all possible inputs and outputs in a trace at a certain time. I restrict the multiplicity and allow only one input per input trace and only one output per output trace. Further, I assume that the values of observables are discrete or discretisable of the input traces and that the choices or actions of the output traces are atomically. Then every trace can be considered an event that signals the change of either an input or output. Trace and event can be considered synonymous.

Within a dynamic multi-agent ICL theory, ICL assigns all alternatives to individual agents, creating a rule base per agents that describes which agent determines what alternative. In an ICL theory, agents choose a distinct atom or value for each alternative that they control per transduction as specified by its strategy. The agent uses observables and a logic program to determine the strategy. When using PHA as embedded formalism, the actual effect of the chosen action is determined by nature based on a probability distribution as defined in a disjoint declaration of the form $disjoint(\chi_1 : p_1, \ldots, \chi_n : p_n])$. This means that an agent rule base can be described as a set of rules that look like:

Example 4.1. $(\chi_1 : p_1) \vee \ldots \vee (\chi_n : p_n) \leftarrow \phi$ *where*

- $(\chi_1 : p_1) \vee \ldots \vee (\chi_n : p_n)$ *are the atoms and their probability of an alternative* c *in* C *and where the chosen atom is the output trace or event,*
- ϕ *is an input trace or event,*
- $\leftarrow$ *should be read as "execute the strategy logic program and choose one atom of the alternative using all available knowledge of the world".*

CPL has a global rule base per ICL theory. As CPL places almost no restriction on the rules, it is trivial to group the rules per agent based on the events on which it is supposed to react. By doing so, each agent has a rule base that defines its behaviour in terms of events. CPL uses the concept of Humean events, that which causes a transition between states to define the rules. An event in a CPL rule is a named label for the change of a variable. A CPL event can be considered the same as an ICL trace. CPL has no concept of logical program that is involved by determining the resulting event when a rule is fired: it defines the resulting event entirely in terms of probability. But a resulting event has to materialise somehow, as the result of some not described procedure. This means that an agent rule base in CPL consists out of rules that could actually be read as:

Example 4.2. $(\chi_1 : p_1) \vee \ldots \vee (\chi_n : p_n) \leftarrow \phi$ *where*

- $(\chi_1 : p_1) \vee \ldots \vee (\chi_n : p_n)$ *are the possible resulting events with their probability and where the chosen event is the output trace or event,*
- ϕ *is sentence with one or more causing events and where each event is an input trace.*
- $\leftarrow$ *should be read as "execute a logic program and choose an resulting event using all available knowledge of the world".*

As can be seen in the above rules, the semantic concepts can easily be unified. Given the rules above, an ICL rule is a subset of CPL rules. They only differ in the number of causing events. ICL allows one causing event, CPL multiple causing events. For that, the concept of event had to introduced in ICL and the concept of logical program in CPL. It has also provides a procedural implementation of transduction using strategies.

The last point in the comparisson is about exogenous events, the events that happen outside the system but that have their effect inside the system. Because ICL has no native concept of events, it has no notion of exogenous events. CPL has the notion of exogenous events. If one groups the rules per agent then even *two* groups of exogenous events are created. The first group consists of the original exogenous events. The second group is formed by endogenous events generated by an agent that appear to be exogenous to a second agent. Exogenous events are a natural way of bootstrapping any sequence of endogenous events in a system.

4.5.2. Comparison of PHA and CPL as embedded formalism

As embedded formalism PHA and CPL both translate the logic into possible world semantics with probability distributions over those worlds. In terms of probability distribution over possible worlds they are for all practical purposes identical. Both can be translated from and to Bayesian networks. There is a difference between the two languages in the translation from the theory to Bayesian networks. A round trip translation for CPL will not always result in the original representation because there are choices in the translation of the probabilistic causal rules that result in loss of information. This is not the case for PHA: a round trip translation results in an identical representation.

The strategy and utility parts of ICL depend on the possible world semantics which is provided by the embedded formalism. Because both PHA and CPL in the end describe the same probability distribution over the possible worlds choosing for either PHA or CPL has no influence on the strategy and utility part of ICL.

CPL adds a process semantic to ICL. CPL has two graphical representation of a theory, a tree or a multi-root graph. In both cases there is the concept of previous and next (possible) event. Inferences that use the “rules of firing” can be used to reason over the order of events. Such inference is not possible in CPL combined with PHA.

In practical terms PHA and CPL both use a finite Herbrand universe and grounded logical theories for the determination of possible worlds and probabilities. This is comparable to a OMG MOF layer M0 model as described in chapter 3. The theories in their first-order form with variables corresponds to a OMG MOF layer M1 model. This means that there is a connection between the OMG MOF model and the Herbrand universe based logical models.

4.5.3. Conclusion

Using CPL as a probabilistic causal logical language within the ICL framework appears to be attractive and possible. It gives the modeller of the rule bases an easier to use language with more flexibility and less restrictions without loss of formal semantics in the rest of the framework. CPL allows for a more modular rule base development than ICL with PHA. It allows for multiple agents that cause the same event. Its event-based nature is a natural fit for DES. The semantics of PHA and ICL are identical and both resolve to a possible world semantics. CPL supports the more advanced features of utility and strategy of ICL. The conclusion therefore is that the ICL framework with CPL as the logical language and embedded formalism offers more functionality and flexibility than the combination of ICL and PHA.

In the next chapter I will integrate the specification language ICL with CPL and the specification language OARSL. I will also shortly look at some of the consequences and possibilities of the runtime support that result from the integration.

5. Integration in AORSL

The primary research question of this thesis is whether is it possible to use a first-order probabilistic causal logic as the event handling logic in a discrete event training simulation. In chapter 2 and 3 the functional requirements of a discrete event training simulation have been described. In chapter 4 the ICL framework for modelling multiple agents under uncertainty was introduced. I have used CPL, a probabilistic causal language, to add flexibility, expressivity, modularity and causality to the ICL framework without loosing the formal semantics of ICL in combination with PHA. In this chapter I will describe a possible integration of OARSL and the (improved) ICL framework. I will sketch an integration of AORSL and the improved ICL framework that will keep the advantages of AORSL and add the advantages of ICL so that the resulting environment can support training simulations.

The AORS metamodel and AORSL are tied to the OMG MDA framework that tries to realise runnable specifications. The current implementation fully supports that framework. Any integration with ICL should not break that support. The integration should therefore be on three levels:

- Specification
- Translation
- Runtime

The primary focus will be on the integration of the specification possibilities of OARSL and ICL. The translation of a specification (PIM) to an actual running program (PSM) will be ignored in this thesis. I will only briefly discuss the necessary runtime support for such program.

5.1. Extending the specification

The specification mainly concerns two separate areas:

- Behaviour of the environment and agents
- Utility per agent

5.1.1. Rule-based modelling and beliefs

The main aspect that is modelled by AORSL is the behaviour of the environment and agents by using rules. A rule describes how the environment or agent reacts when it receives an event, based on knowledge of the world. An environment rule allows for binding to specific objects or set of objects. The other elements of the rule is the same as with an agent rule. This means that the specification of the actual behaviour is identical for both rules, the environment just adds routing of events.

An AORSL agent rule is a 4-tuple consisting of:

- *TriggerEvt* is the type of event that triggers the rule,
- *Condition* is a logical condition formula, allowing for variables,
- *UpdateState* specifies an update of the environment state,
- *ResultEvt* is a list of resulting events, which will be created when the rule is fired,

A CPL causal law is described by $(\chi_1 : p_1) \vee \ldots \vee (\chi_n : p_n) \leftarrow \phi$ where

- $(\chi_1 : p_1) \vee \ldots \vee (\chi_n : p_n)$ are the possible resulting events with their probability and where the chosen event is the output trace or event,
- ϕ is an input trace or event,
- $\leftarrow$ should be read as "execute a logic program and choose an event using all available knowledge of the world".

It is easy to see that these two representations largely coincide. The main difference is that a AORSL allows for the specification of only one triggering event *TriggerEvt* while CPL allows for an arbitrary logical sentence ϕ which more specifically can contain multiple triggering events. The *Condition* and *UpdateState* of AORSL is largely the same as the interpretation of CPLs $\leftarrow$. OARLS' *ResultEvt* and CPLs $(\chi_1 : p_1) \vee \ldots \vee (\chi_n : p_n)$ event list are comparable, with in case of ICL the addition of annotations with probabilities. A resulting integration could be an AORSL agent rule described by a 4-tuple consisting of

- *TriggerEvt* is a list of type of events that triggers the rule,
- *LogicalProgram* is a logical program that possible updates the environment state,
- *UpdateState* a list of (environment) variables that may be updated by the *LogicalProgram*
- *ResultEvt* is a list of resulting events annotated with probabilities, which will be created when the rule is fired.

It depends on the implementation of the specification environment in combination with the runtime environment whether and how exactly *LogicalProgram* and *UpdateState* should be specified. A smart specification environment could filter out the used variables in the *LogicalProgram* and a smart runtime environment could filter out any runtime changes in a program variable caused by the program. In both cases the modeller is relieved from the need to specify it explicitly.

Beside the rules, agents have knowledge of facts and beliefs. In AORSL the facts of an agent are defined by a set of properties, its beliefs by sets of belief-properties and belief entities. In ICL there is no difference between facts and beliefs. All fact or beliefs, which correspond to atoms in the rules and logical programs, are believed to be true with the probability that follow from the rules or observations. Adding the attribute "probability" to AORSL's properties, belief-properties and belief entities enhances the support for "beliefs" greatly, transforming the simple reflex agents from AORSL to model-based reflex agents

5.1.2. Desires

To support utility-based agents, ICL has the concept of utility. Utility is a function from agent and state to a cardinal value. Support in AORSL can easily be added by specifying a class *Utility* with function *getUtility* that takes a logical expression that represents the state of the world for which the utility is requested. Objects of this class can be added to an agent analogous to objects of the class AgentRule. Adding utility makes the agents utility-based agents.

5.1.3. Intentions

In ICL, a strategy is a (total) choice for the alternatives of an agent. The purpose of a strategy is determine the choices that result in the highest utility. The specification does not mention *how* the strategy is chosen. During a transduction any number of choices can be made by an agent, from none to a total choice. The horizon of such a decision is the immediate next world, as ICL has no concept of planning beyond the current transduction. Interestingly, the unification of ICL and CPL transforms the logical theories into a STRIPS-like planning problem [25]. Informally a STRIPS planning problem is described by an initial state, a set of actions with a description of its preconditions (what must be established before the action is performed) and postconditions (what is established after the action is performed) and a specification of the goal states. Omitting some details, a unified theory can be considered as STRIPS-like planning problem:

- Initial state: the first exogenous event takes the system into its initial state,
- Action: the $\leftarrow$ of the rules or *LogicalProgram* can be considered as an action,
  - Precondition: the body of the rule or *TriggerEvt*,
  - Postcondition: the head of the rule or *ResultEvt* including any *UpdateState*.
- Goal state: the possible world with the highest utility,

However according to Blythe [2], until recently most STRIPS-based classical planning algorithms focused on planning under rather restrictive assumptions. The most notable restrictive assumptions are

- The goal of the planner is a logical description of a world state.
- The actions taken by the planner are the only sources of change in the world
- Each action can be deterministically described by the conditions under which it can be applied and its effects on the world.

It is clear that a planner in an AORSL/ICL-based world should relax all three assumptions. Agents can have multiple goal states if the utility of different possible worlds are identical. All the agents in the world make a plan and influence each other. The actions are not deterministic but probabilistic. In his article Blythe describes four implementations of planners that all have a different approach for planning under uncertainty. They all more or less relax the assumptions mentioned before. Each of the planners however has a different approach and characteristics. Whether one of those planners is suitable for usage in an agent-based discrete event simulation for training or what the adaptions to either the planner of AORSL should be is unknown at this moment.

Integrating ICL in AORSL gives the opportunity to implement the concept of intentions. For a full fledged training simulation planning of the actions of an agent is required. Determining the required adaptions to OARSL to support planning under uncertainty is beyond the scope of this thesis.

5.2. Extending the translation

In the current implementation a specification is translated by XSLT transformations. Because the target environment is not clear (see the next section), it is uncertain what the functionality of the translation should be. But luckily, XSLT is proven to be Turing complete [1]. Apart from sensible syntax and expressivity, there is no theoretical limit on what could be achieved with XSLT.

5.3. Extending the runtime

The OMG MDA framework tries to realise runnable specifications. The purpose of the runtime is to enable a specified simulation to actually run. To a large extend this is a matter of engineering and as such beyond the scope of this thesis. However, there are a few problems that I want to address in this section.

Currently Java and JavaScript are used as the PSM language in which the actual simulation is written. The OMG MDA framework does not mandate any specific PSM language. In the context of the integration of AORSL and ICL/CPL is may be useful to rethink the PSM language. OARSL and ICL/CPL use different and largely incompatible programming paradigms. OARSL uses an object oriented imperative paradigm whereas ICL/CPL uses a logical declarative paradigm.

One possibility is to use Prolog as the PSM language. Prolog has excellent support for first-order probabilistic causal logical. There is a Prolog implementation available for ICL, AILog2, designed by Poole[2]. There is excellent support for the calculation of

[1]see http://www.unidex.com/turing/utm.htm if you really think that XSLT a solution for the proof its Turing completeness

[2]see http://www.cs.ubc.ca/ poole/aibook/code/ailog/ailog2.html for the implementation of AILog2

probabilities in Prolog in the form of ProbLog. There are however also some drawbacks. AORSL started as a prototype written in Prolog [30] but it was abandoned in favour of Java. According to Wagner (in private communication) the transition from Prolog to Java was

> *"simply because the AORS framework project is a large software engineering (SE) effort, requiring e.g. the integration of other libraries (such as physics engines and 3D graphics libraries) and I figured that SE is much better supported by a standard programming language like Java. "*

In addition, Prolog is a logical programming language with a declarative nature. A DES is procedural and event driven. Prolog does not have real good support for (imperative) procedural or event-driven programming although some efforts are made to enhance the support, e.g. Logtalk which has support for object-oriented and event-driven programming. GUI programming with Prolog is not very well developed. Inter process communication is Prolog is limited to raw socket interfaces. High-level interfaces as CORBA or DCOM are not available. As a general purpose programming language, Prolog does not seem to be an attractive candidate. An additional extra problem is that there is no (good) mapping of OMG UML to Prolog and vice versa. This would mean that the OMG MDA framework would be lost.

The other possibility would be to keep using the current choice Java as PSM language. Java is an excellent general programming language and is used on a wide scale for that. However, Java does not support directly first-order probabilistic logic as part of the language. There are also no Java libraries available that support such logic. Fully implementing the required logic environment in Java as a library is possible. There are Java Bayesian network libraries available. The calculation of the "a priory" probabilities is in general not very difficult. Another possibility would be the embedding of a Prolog engine in Java. Yap, the Prolog on which ProbLog is build, can be embedded in a Java program using its bi-directional Java interface (Java can call Prolog and vice-versa). There is one big problem with this solution. It is not possible to have multiple independent versions of the same (Prolog) database in one (operating system) process. This a technical problem that the maintainers of Yap (or SWI-Prolog as a possible replacement for Yap), the Prolog version that ProbLog uses, are unwilling to solve. There are two different solutions for this problem. The first is a multiple process model where each agent will run in a separate process and where all communication will be by TCP/IP sockets, which both Prolog versions supports. This is however not a very high-level IPC and cannot be compared to CORBA or DCOM. The second solution is embedding a (lightweight) Prolog engine written in Java. I have not looked in great detail into such a solution. More specific, I do not know what which embedded Java-based Prolog engines do or do not support multiple independent version of the same database.

Both solutions require a large software engineering effort. Not loosing the possibilities of the OMG MDA framework is very important. The better support for large scale software engineering in Java is important. Supporting tools like UML editors, IDEs and additional libraries are more abundant for Java than Prolog. Java programmers and knowledge are more available and widespread than Prolog programmers and knowledge. In my opinion, it seems more likely that a Java solution is easier to obtain than a Prolog solution. However, without further research it is impossible to estimate which solution requires the larger effort.

In this chapter I tried to integrate AORSL with ICL/CPL. As far as the specification part of an OMG MDA framework is concerned, the integration results in the transformation of AORSL from simple reflex agent environment into a full utility-based agent environment which is required by a training simulation. Integrating planning into AORSL would even provide more functionality. The problems with extending the translation and runtime part of an OMG MDA framework are substantial in terms of software engineering effort.

6. Conclusion

Very often the performance of higher staff in organisations during incidents is deemed to be lacking, caused by insufficient preparation, insufficient training, bad decision-making and bad procedures. Task training could help to improve the task performance. The most important characteristics of the task environment are that it is dynamic, partially observable, probabilistic, event driven and rule-based. For optimal transfer the training environment should have a great resemblance with the task environment. Discrete event simulations are dynamic, event driven and rule based. By adding a causal probabilistic first-order logic, the event handling logic becomes non-monotonic and probabilistic and support for partial observability is added.

The research question of this thesis is:

> *"Is it possible to use a first-order probabilistic causal logic as the event handling logic in a discrete event training simulation? "*

In this thesis I have looked at a specific DES implementation, AORSL from Wagner. OARSL has two important features. Firstly, it is based on the OMG MDA framework where the goals is to make "runnable specifications" in a formal language. Secondly, it uses agents in its modelling. OARSL actually achieves the runnable specification goal for simple reflex agents. However, to be truly useful for training simulations, model and utility-based agents are required. This means that AORSL as a language must be extended. This can be done by embedding a first-order probabilistic causal logic.

For this reason I introduced ICL. ICL is a framework that is capable of modelling agents under uncertainty using logic programming, probability, game and decision theory. It is able to specify the model- and utility-based agents which are required for training simulations. By unifying ICL's logical language with CPL and embedding it in the framework as the first-order probabilistic causal logic language, ICL is provided with a more expressive, modular and causal language while keeping the formal semantics of ICL intact.

In chapter 5 I have integrated ICL/CPL framework in the specification environment of AORSL. Because CPL can be considered as implementing a superset of the logical language specification, the main problem is the unification of the semantics of ICL's agent rule and CPL's causal law. By adding the concept of Shafer's Humean event to ICL and the concept of logical program to CPL the semantics of both languages can be unified. The unification also provides for a procedural embedding of transduction using strategies. Replacing PHA by CPL in ICL does not change the possible world semantics with probabilities over the worlds. It adds the event tree and multi-root graph semantics of CPL to ICL. By integrating the ICL/CPL unified logic in AORSL rules and properties the agents in AORSL are transformed from simple reflex agents to model-based reflex agents. The subsequent integration of ICL's utility into AORSL makes the agents utility-based agents. Although not worked out in AORSL, I showed that the unified ICL/CPL can be seen as STRIPS-like planning problem, which would make the behaviour of agents even more sophisticated.

Extending the translation and runtime environment by integrating of AORSL and the ICL/CPL framework proved to be more difficult. The most difficult parts were the technical difficulties that arise from the use of different programming paradigms for AORSL and ICL/CPL. AORSL is currently implemented in Java and JavaScript while ICL/CPL is Prolog-based. Integration of these technologies is not trivial. Without further research it is impossible to determine which solution should be chosen.

Based on the research in this thesis the answer to the research question is that is indeed possible to use a first-order probabilistic causal logic as event handling logic in a discrete event training simulation. In fact, using such language creates an environment that almost fulfils all requirements for such training simulation. As far as a specification language is concerned, further research is needed on the subject of planning under uncertainty. To fully support an OMG MDA framework implementation for a full utility-based agent environment, even more research will be needed.

Bibliography

[1] K. R. Apt and M. Bezem. Acyclic programs. In *ICLP*, pages 617–633, 1990.

[2] J. Blythe. An overview of planning under uncertainty. In M. J. Wooldridge and M. Veloso, editors, *Artificial Intelligence Today*, chapter An overview of planning under uncertainty, pages 85–110. Springer-Verlag, Berlin, Heidelberg, 1999.

[3] S. Burris. *Logic for Mathematics and Computer Science.* Prentice Hall, 1998.

[4] I. M. Diaconescu and G. Wagner. Agent-based simulations with beliefs and sparql-based ask-reply communication. In *Proceedings of the 10th international conference on Multi-agent-based simulation*, MABS'09, pages 86–97, Berlin, Heidelberg, 2010. Springer-Verlag.

[5] A. Finzi and T. Lukasiewicz. Structure-based causes and explanations in the independent choice logic, 2003.

[6] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski, Bowen, and Kenneth, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press, 1988.

[7] G. Guizzardi. *Ontological foundations for structural conceptual models.* PhD thesis, Centre for Telematics and Information Technology, University of Twente, 2005.

[8] J. K. J. L. W. M.-V. J.F.A.K. Benthem, H.P. Ditmarsch. *Logica voor informatici.* Pearson, 2003.

[9] A. G. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[10] K. Korb and A. Nicholson. *Bayesian Artificial Intelligence.* Chapman and Hall, 2nd edition, 2010.

[11] A. M. Law. *Simulation Modeling and Analysis 4th edition.* McGraw-Hill Higher Education, 4rd edition, 2007.

[12] J. McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[13] W. Meert. *Inference and learning for directed probabilistic logic models.* PhD thesis, Katholieke Universiteit Leuven, 2011.

[14] W. Meert, J. Struyf, and H. Blockeel. Learning ground cp-logic theories by leveraging bayesian network learning techniques. *Fundam. Inf.*, 89:131–160, January 2009.

[15] B. Milch and S. Russell. Inductive logic programming. In S. Muggleton, R. Otero, and A. Tamaddoni-Nezhad, editors, *Inductive Logic Programming*, chapter First-Order Probabilistic Languages: Into the Unknown, pages 10–24. Springer-Verlag, Berlin, Heidelberg, 2007.

[16] J. Patrick. *Training: Research and Practice.* Academic Press, 1992.

[17] D. Poole. Representing knowledge for logic-based diagnosis., 1988.

[18] D. Poole. Probabilistic horn abduction and bayesian networks. *Artif. Intell.*, 64:81–129, November 1993.

[19] D. Poole. Abducing through negation as failure: Stable models within the independent choice logic. *Journal of Logic Programming*, 44:2000, 1995.

[20] D. Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artif. Intell.*, 94:7–56, July 1997.

[21] D. Poole. The independent choice logic and beyond. In L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton, editors, *Probabilistic inductive logic programming*, chapter The independent choice logic and beyond, pages 222–243. Springer-Verlag, Berlin, Heidelberg, 2008.

[22] A. S. Rao and M. P. Georgeff. Modeling rational agents within a bdi architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991.

[23] F. Riguzzi and T. Swift. An extended semantics for logic programs with annotated disjunctions and its efficient implementation. In *Proceedings of the 25th Italian Conference on Computational Logic (CILC2010), Rende, Italy, July 7-9, 2010.*, number 598 in CEUR Workshop Proceedings, Aachen, Germany, 2010. Sun SITE Central Europe.

[24] A. J. Romiszowski. *Designing instructional systems.* Kogan Page, New York,, 1984.

[25] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[26] G. Shafer. *The art of causal conjecture.* MIT ress, 1996.

[27] J. Vennekens. *Algebraic and logical study of constructive processes in knowledge representation.* PhD thesis, Informatics Section, Department of Computer Science, Faculty of Engineering, May 2007. Denecker, Marc and De Schreye, Danny (supervisors).

[28] J. Vennekens, M. Denecker, and M. Bruynooghe. Cp-logic: A language of causal probabilistic events and its relation to logic programming. *Theory Pract. Log. Program.*, 9:245–308, May 2009.

[29] J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic programs with annotated disjunctions. In B. Demoen and V. Lifschitz, editors, *Logic Programming, 20th International Conference, ICLP 2004, Saint-Malo, France, September 6-10, 2004,*

Proceedings, volume 3132 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 2004.

[30] G. Wagner. The agent-object-relationship metamodel: towards a unified view of state and behavior. *Inf. Syst.*, 28:475–504, July 2003.

[31] G. Wagner. Aor modeling and simulation âĂŞ towards a general architecture for agent-based discrete event simulation. In *In P. Giorgini et al. (Eds.): Agent-Oriented Information Systems, Springer LNAI 3030*, pages 174–188, 2004.

[32] G. Wagner, O. Nicolae, and J. Werner. Extending discrete event simulation by adding an activity concept for business process modeling and simulation. In *Winter Simulation Conference*, WSC '09, pages 2951–2962. Winter Simulation Conference, 2009.

A. Example translations

A.1. Original Bayesian network

$P(fire) = 0.01$
$P(smoke|fire) = 0.9$
$P(smoke|\neg fire) = 0.9$
$P(tampering) = 0.02$
$P(alarm|fire \wedge tampering) = 0.5$
$P(alarm|fire \wedge \neg tampering) = 0.99$
$P(alarm|\neg fire \wedge tampering) = 0.85$
$P(alarm|\neg fire \wedge \neg tampering) = 0.0001$
$P(leaving|alarm) = 0.88$
$P(leaving|\neg alarm) = 0.001$
$P(report|leaving) = 0.75$
$P(report|\neg leaving) = 0.01$

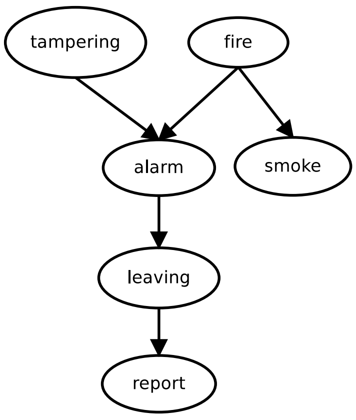

Figure A.1.: Bayesian network for smoking alarm

A.2. Translation Bayesian network to PHA

The PHA theory for the Bayesian network:

$disjoint([fire(yes) : 0.01, fire(no) : 0.99]).$
$smoke(Sm) \leftarrow fire(Fi) \wedge c\_smoke(Sm, Fi).$
$disjoint([c\_smoke(yes, yes) : 0.9, c\_smoke(no, yes) : 0.1]).$
$disjoint([c\_smoke(yes, no) : 0.01, c\_smoke(no, no) : 0.99]).$
$disjoint([tampering(yes) : 0.02, tampering(no) : 0.98]).$
$alarm(Al) \leftarrow fire(Sm) \wedge tampering(Ta) \wedge c\_alarm(Al, Fi, Ta).$
$disjoint([c\_alarm(yes, yes, yes) : 0.50, c\_alarm(yes, yes, yes) : 0.50]).$
$disjoint([c\_alarm(yes, yes, no) : 0.99, c\_alarm(no, yes, no) : 0.01]).$
$disjoint([c\_alarm(yes, no, yes) : 0.85, c\_alarm(no, no, yes) : 0.15]).$
$disjoint([c\_alarm(yes, no, no) : 0.0001, c\_alarm(no, no, no) : 0.9999]).$
$leaving(Le) \leftarrow alarm(Al) \wedge c\_leaving(Le, Al).$
$disjoint([c\_leaving(yes, yes) : 0.88, c\_leaving(no, yes) : 0.12]).$
$disjoint([c\_leaving(yes, no) : 0.001, c\_leaving(no, no) : 0.999]).$
$report(Le) \leftarrow leavin(Al) \wedge c\_report(Le, Al).$
$disjoint([c\_report(yes, yes) : 0.75, c\_report(no, yes) : 0.25]).$
$disjoint([c\_report(yes, no) : 0.01, c\_report(no, no) : 0.99]).$

A.3. Re-translation ICL to Bayesian network

The translation of the ICL theory above back into Bayesian network creates the original network. This means that the two representation are translatable into each other without any loss of information.

A.4. Translation Bayesian network to CPL

The CPL theory for the Bayesian network as given in the introduction:

$fire : 0.01$
$tampering : 0.02$
$alarm : 0.5 \leftarrow tampering \wedge fire$
$alarm : 0.85 \leftarrow tampering \wedge \neg fire$
$alarm : 0.99 \leftarrow \neg tampering \wedge fire$
$alarm : 0.0001 \leftarrow \neg tampering \wedge \neg fire$
$smoke : 0.9 \leftarrow fire$
$smoke : 0.01 \leftarrow \neg fire$
$leaving : 0.88 \leftarrow alarm$
$leaving : 0.001 \leftarrow \neg alarm$
$report : 0.75 \leftarrow leaving$
$report : 0.01 \leftarrow \neg leaving$

The translation from a Bayesian network to a CPL-theory is straight forward.

A.5. Re-translation CPL to Bayesian network

Retranslating the CPL theory into a Bayesian network creates a different network than the original network because the translation from a CPL theory into a Bayesian network introduces choice nodes. Figure A.2 shows the Bayesian network that results from the above CPL theory. It is clear that a naive translation of this network will lead to different CPL theory than the original CPL theory. Whenever a CPL theory-based Bayesian network is translated back into the original CPL theory the choice nodes should be eliminated in the translation by putting the choices in the rules of the atoms. The information necessary for this however is not guaranteed encoded in the Bayesian network

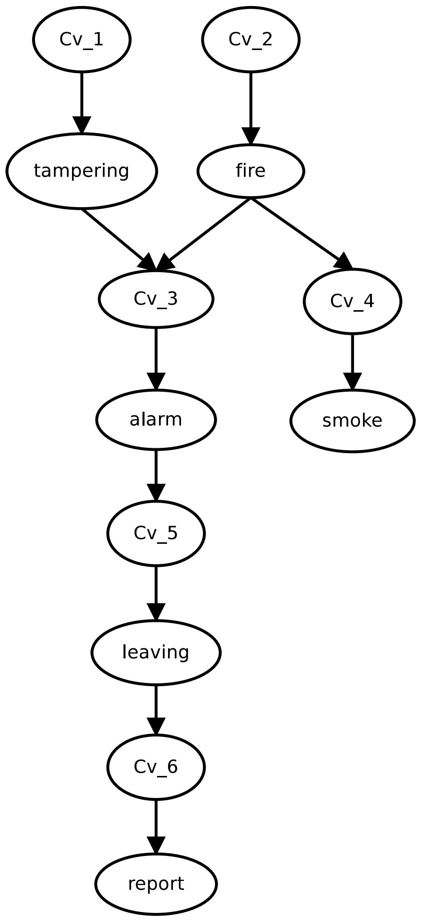

Figure A.2.: re-translation from CPL to Bayesian network

B. AORSL UML

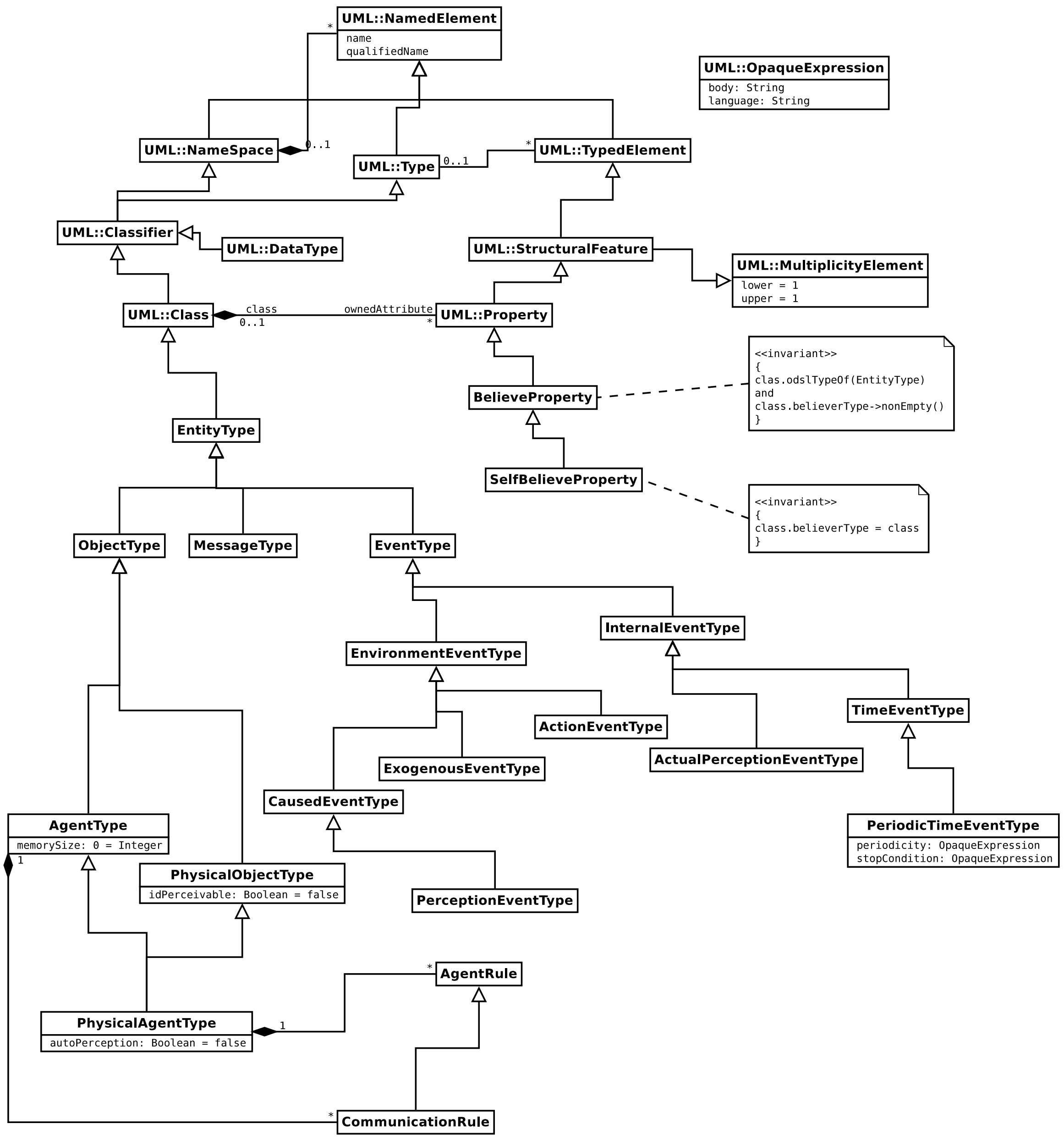

Figure B.1.: Entity types

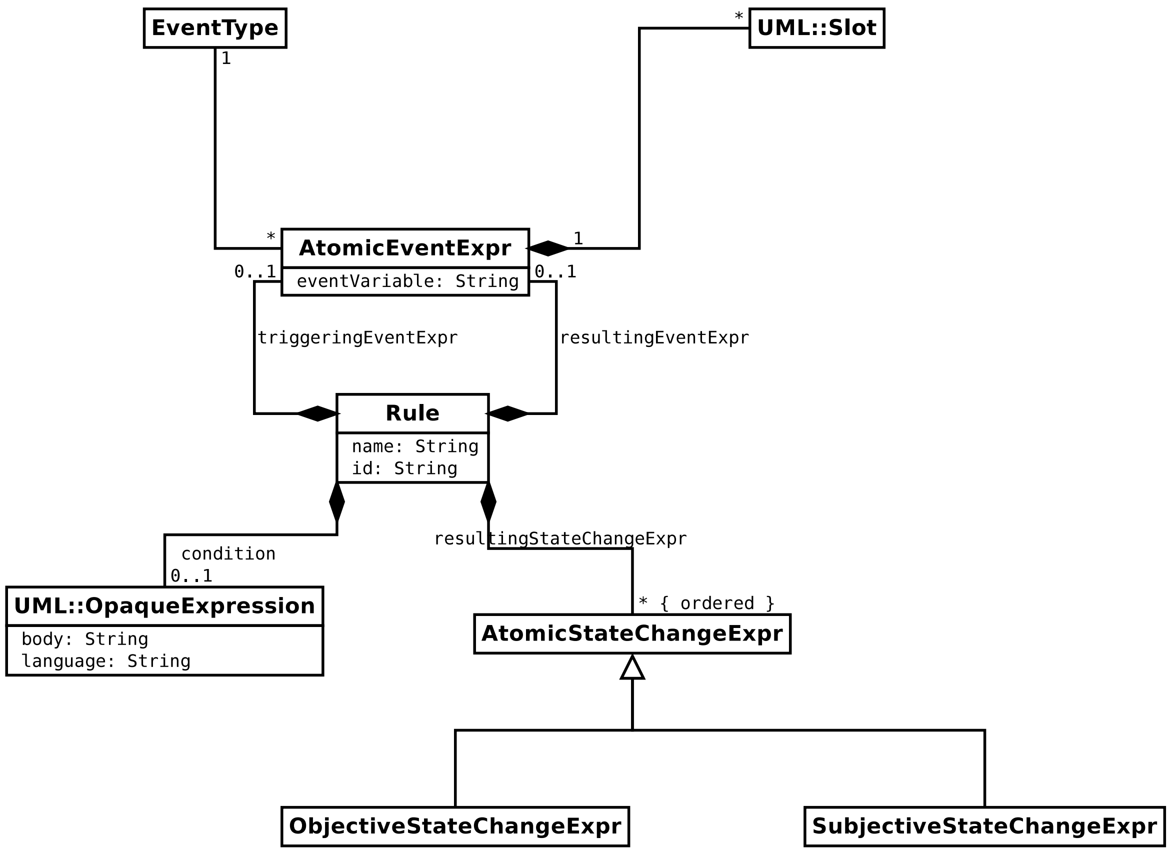

Figure B.2.: Rules

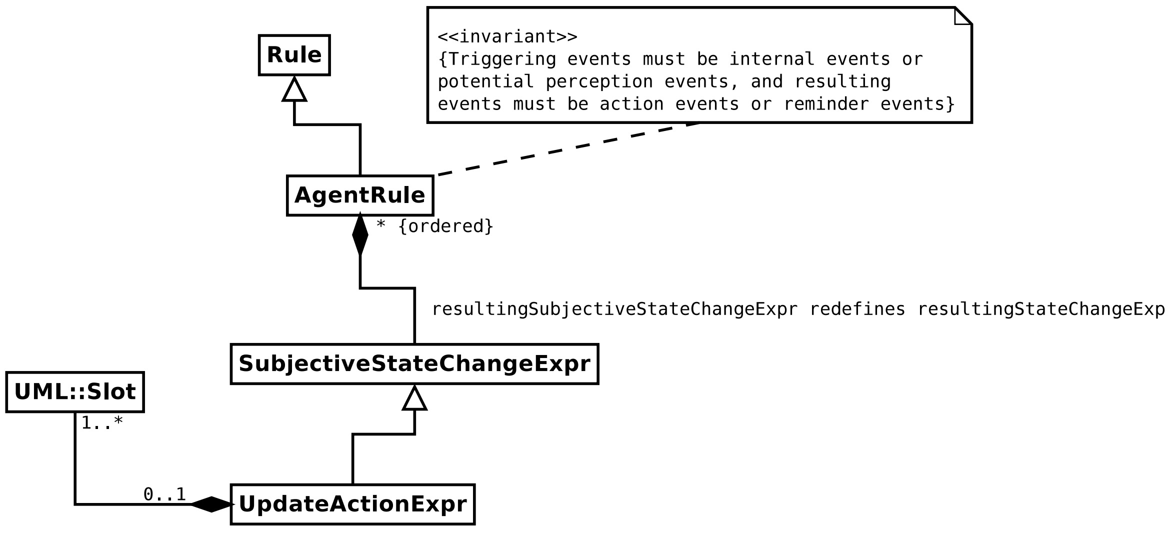
Rule
<<invariant>>
{Triggering events must be internal events or
potential perception events, and resulting
events must be action events or reminder events}
AgentRule
* {ordered}
resultingSubjectiveStateChangeExpr redefines resultingStateChangeExp
SubjectiveStateChangeExpr
UML::Slot
1..*
0..1
UpdateActionExpr

Figure B.3.: Agents behaviour

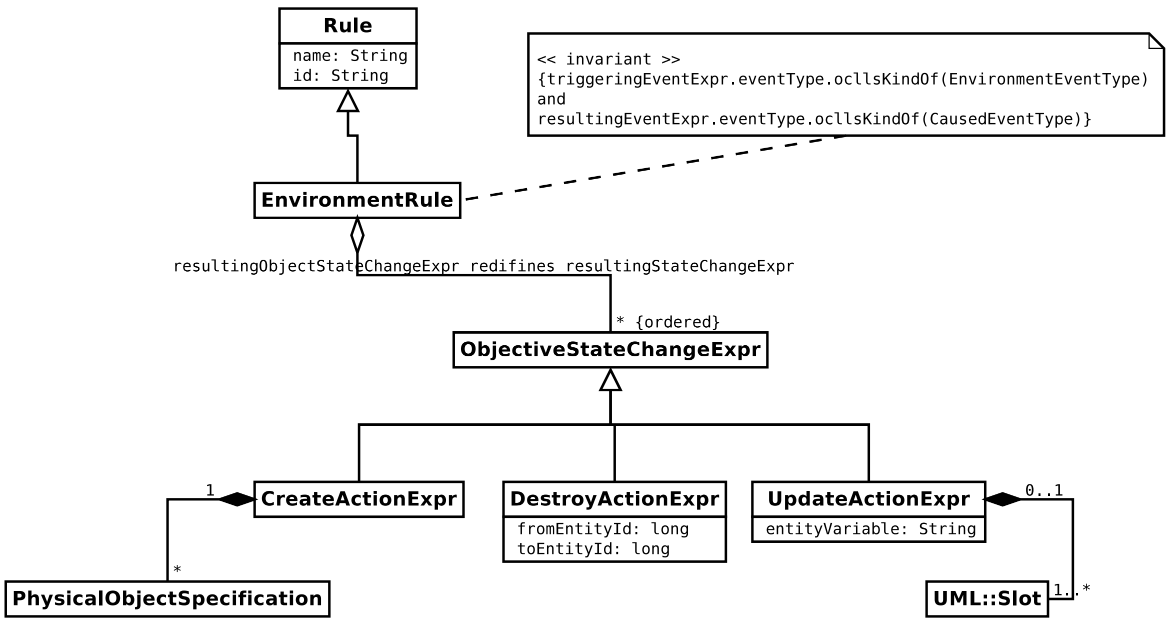

Figure B.4.: Environment (causality) rules

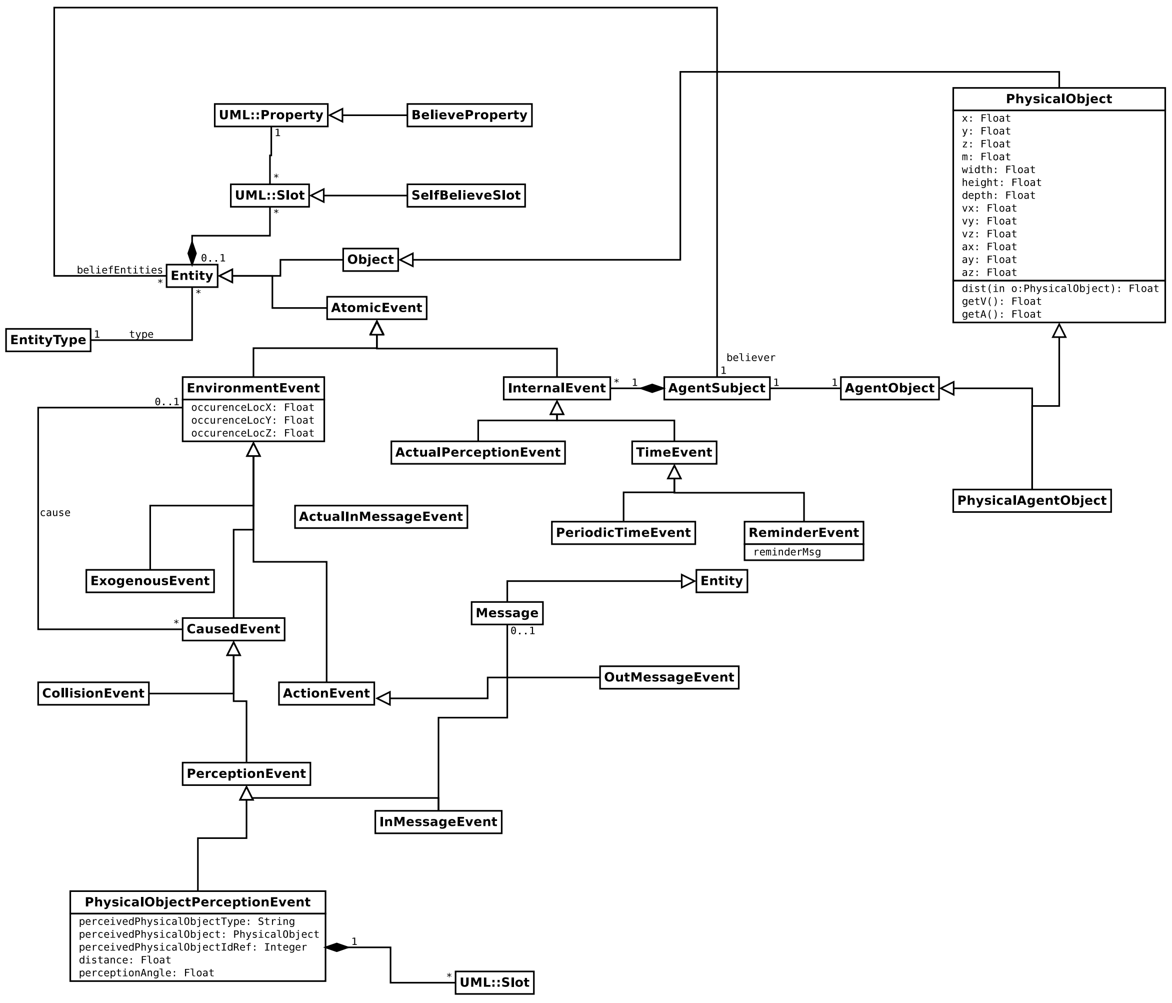

Figure B.5.: Entities

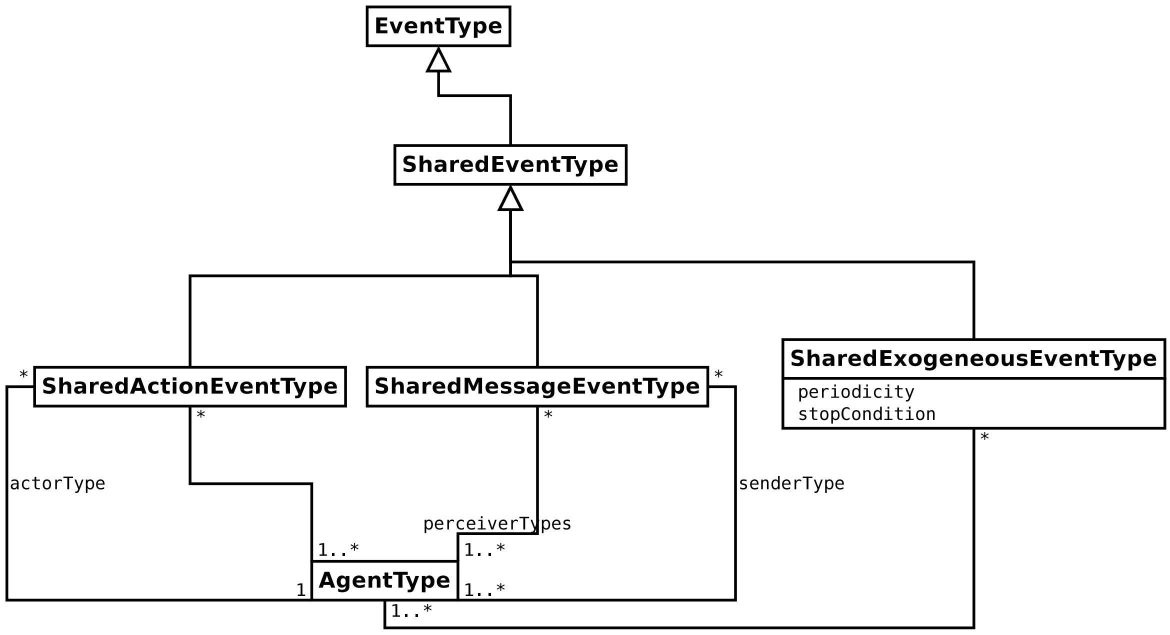

Figure B.6.: Shared event types (convenience constructs)