Simulation
XX(X):2–39

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Quantifying means-end reasoning skills in simulation-based training: A logic-based approach

Audun Stolpe[1] and Jo Erskine Hannay[12]

Abstract

We develop a logic-based approach for designing simulation-based training scenarios. Our methodology embodies a concise definition of the scenario concept and integrates the notions of training goals, acceptable versus unacceptable actions and performance scoring. The approach applies classical AI planning to extract coherent plays from a causal description of the training domain. The domain- and task-specific parts are defined in a high-level action description language $\mathcal{AL}$. Generic causal and temporal logic is added when the causal theory is compiled into the underlying Answer Set Programming (ASP) language. The ASP representation is used to derive a scoring function that reflects the quality of a play or training session, based on a distinction of states and actions into *green* (acceptable) and *red* (unacceptable) ones. To that end, we add to the casual theory a set of *norms* that specify an initial assignment of colours. The ASP engine uses these norms as axioms and *propagates* colours by consulting the causal theory. We prove that any set of such norms constitutes a *conservative extension* of the underlying causal theory. With this work we hope to lay the foundation for the development of design and analysis tools for exercise managers. We envision a software system that lets an exercise manager view all plays of a tentative scenario design, with expediency information and scores for each possible play. Our approach is applicable to any domain in which means-ends reasoning is pertinent. We illustrate the approach in the domain of crisis response and management.

Keywords

Simulation-based training, Means-end reasoning, Automated scoring, Answer Set Programming, Deontic logic

Prepared using sagej.cls *[Version: 2017/01/17 v1.20]*

1 Introduction

Imagine a pan, without a lid, overheating on a stove in an industrial kitchen. The pan contains cooking oil that will ignite and burn if left to itself. There is a kitchen policy and operating procedure designed to protect against workplace hazards such as this, but working in a big kitchen is a high-stress job, and continuous restaffing makes it difficult to keep all employees drilled in safety procedures at all times.

An astute employee who recognizes the danger may attempt to avert it, but even in a simple scenario such as this, his success depends on his competence and training. In an optimal performance he would notice that the pan is overheating and simply turn off the stove. But if he is new to the job, he may not realize the danger until the pan starts to give off smoke. When this happens, there is still time to prevent a fire if, say, the employee acts quickly and both turns off the stove and puts a lid on the pan. We are supposing that the cooking oil has been brought so close to the point of ignition that the residual heat in the stove will set it off unless the pan is covered.

Some actions that are open to the employee are prudent if performed at the right time but dangerous otherwise. As an example, take the action of turning on the fan to dissipate the smoke. If the stove is already turned off and the pan has a lid on it, then this is a useful thing to do. However, if the fan is on when the oil catches fire, the fire spreads to the vent shaft. Other actions are downright bad ideas, for instance pouring water on the fire. This causes a fire splash that spreads the fire to the kitchen cabinets as well. If the employee next acts with acumen, he may still redeem himself by putting out all incipient fires quickly. There are three different fire extinguishers hanging on a panel on the back wall of the kitchen. It is important that the employee selects the correct extinguisher, for as one extinguisher is for grease fires, the second for metal fires and the third for fires in combustible substances, it is easy to get things wrong.

Emergency exercises are specifically designed to prevent harmful incidents like this from evolving and to drill the staff in efficient actions to mitigate the consequences. *Simulation-based training* (SBT)[1]– understood as the systematic acquisition of concepts, knowledge, and rules in a designed practice environment with both real and synthetic elements–offers a way for personnel to practice coping with hazardous contingencies safely in their work environment. In other words, somewhere to learn from mistakes without doing any damage and somewhere for management to validate and improve emergency protocols with little overhead. A well-designed simulation enables *deliberate practice*,[2,3] in which trainees improve their procedural knowledge through tailored feedback on the effectiveness of their *means-ends reasoning*.

Deliberate practice relies on monitoring trainee performance to tailor feedback and reruns. Individual scores must be aggregated into time series if long-term learning effects are to be documented and the training regime itself fine-tuned accordingly. Current practices are widely considered to fall short in this

[1]Dept. of Applied Research in Information Technology, The Norwegian Computing Center, Pb. 114 Blindern, 0314 Oslo, Norway [2]Center for Effective Digitalization in the Public Sector, Simula Metropolitan Center for Digital Engineering, Pb. 4 St. Olavs plass, 0130 Oslo, Norway

Corresponding author:
Audun Stolpe, The Norwegian Computing Center, Pb. 114 Blindern, NO-0314, Norway.
Email: audun.stolpe@nr.no

Prepared using sagej.cls

respect. Costly exercises are regularly held with little or no means of documenting learning effects in any principled manner.[4,5] To be useful, SBT thus requires a set of well-defined performance metrics for means-ends reasoning. Moreover, these metrics, and how and when in a simulation to collect data on them, must be designed into the scenario.

The present paper elaborates on a foundational aspect of assessing means-end reasoning on the part of those training in simulations. It is concerned with the problem of finding an easy-to-interpret and easy-to-apply metric for scoring the *instrumental rationality* displayed by an agent in different *plays* of the same simulation *scenario*.[6] The central idea is that of interpreting a simulation scenario formally as a *dynamic system* in the classical symbolic AI sense.[7–10] In a nutshell, a dynamic system is a state transition diagram where the transitions are actions whose effects are recorded as properties of the adjacent vertices. Such a diagram is encoded in a *causal language* as a *causal theory* axiomatizing the effects of actions on objects and the interaction *between* objects in the domain of interest. A causal theory additionally axiomatizes a few general rules of causal reasoning, notably the default persistence of facts over time (inertia).[11]

In this paper, we adopt a two-step methodology: the domain-specific part of a causal theory is represented in an abstract *action description language*, specifically the language $\mathcal{AL}$.[8,12] This language abstracts away the logical details of causal and temporal reasoning, allowing domain experts and scenario designers to focus on defining events and possible actions. Next, the resulting domain description is translated into the declarative programming language Answer Set Programming (ASP), which automates the causal reasoning and takes care of propagating facts through time.

The mode of execution of the compiled ASP program will depend on the problem to be solved. A causal theory encoded in ASP can be used to predict and to plan. In a prediction problem, the task is to determine how the current state of affairs will change after a given sequence of actions is executed. In a planning problem, the task is to find a sequence of actions that causes an initial state to evolve into a desirable end state, aka. a goal state.[7] We have argued elsewhere[13] that the planning mode is useful in the scenario design phase to assist the designer in validating and exploring combinations of events. For instance, a designer would typically want to check that the actions and events he has defined yield coherent plays, that the different lines of evolution make sense from a skill-building perspective, and that there are no anomalies in the scenario that can cause a trainee to mistrain. Prediction, on the other hand, is useful in the actual simulation or training phase for managing the simulated events, say, in a virtual 3D environment.[14] Using the causal theory to predict the consequences of actions in a complex environment over time, the simulation engine would essentially just have to render the ensuing states. This workflow, based on a central knowledge base and the two aforementioned modes of automated reasoning, is the main principle behind the ExManSim architecture (exercise management and simulation architecture) that was proposed in an earlier article written by the present authors.[5,13]

The principal theme of the present paper, however, is that of performance scoring. We reduce the problem of scoring training sessions to the problem of defining a function from plans to integers that represents a reasoned assessment of a plan interpreted as a play. This reduction is based on identifying a training scenario with the causal theory that describes it and a play with a plan of how to move from an initial state to a goal state according to that theory. The core idea behind the performance metric is to derive a scoring function from a distinction of states and actions into *green* (acceptable) and *red* (unacceptable) ones. To that end, we add to the casual theory a set of colouring rules that we refer to as *norms* which specify the conditions under which an action or state is to be deemed acceptable or unacceptable. The set of norms constitutes a *deontic specification* that enables the ASP engine to *compute*

and *propagate* colours over all states in all plans. We prove that any set of norms as so defined constitutes a *conservative extension* of the causal theory in the sense that computing performance metrics does not affect the causal reasoning. A summary of the contributions of this paper goes as follows:

- We formalize the notion of simulation scenario in terms of dynamic systems declared in the action language $\mathcal{AL}$. This formalization is also a contribution to the modelling and simulation field itself, in that it provides a plausible clarification of an overloaded and under-specified notion.
- We axiomatize the concepts of acceptable and unacceptable actions and states in a *deontic overlay* to the action language $\mathcal{AL}$.
- We formally prove that this axiomatization has the property that any particular *deontic specification*, i.e. set of *norms*, constitutes a *conservative extension* of the underlying causal theory. This means that introducing the notion of expediency does not interfere with, or change, the scenario itself.
- We formulate a set of requirements that a scoring function should satisfy.
- We define a function that meets these requirements by derivation from the number of green and red entities contained in a plan.

We discuss a few sample runs from our prototype implementation offered as a proof of concept.

The paper is organized as follows: Section 2 puts our development in the relevant simulation-based training context. Section 3 assembles enough background material to allow the paper to be self-contained; it explains the basics of the ASP programming language, it introduces the action language $\mathcal{AL}$, and it explains how the ASP encoding of the $\mathcal{AL}$ action description defines a labelled transition diagram. Transition diagrams are very useful for visually displaying causal theories. By way of illustration, we formalize the kitchen scenario mentioned above and show parts of its associated transition diagram. It will be used as a running example throughout. Section 4 discusses a crucial idea of this paper, which is that of treating a simulation scenario as a causal theory and a play or training session as a sequence of actions that leads from an initial state to a goal, i.e. as a plan. These sequences of actions form the domain of the scoring function, which will calculate the overall quality of the play based on the combinations of acceptable and unacceptable actions and states. To that end, Section 5 defines a deontic extension to the $\mathcal{AL}$ language used to express the acceptable/unacceptable distinction. We prove that the general syntax of this deontic language is such that any particular specification of acceptable and unacceptable states and transitions in a given causal theory constitutes a conservative extension of that theory. The resulting language, which we call $\mathcal{ALD}$ for easy reference, is next used to encode a normative specification for the kitchen scenario, exemplifying how $\mathcal{ALD}$ defines a coloured transition diagrams from which the acceptable and unacceptable choices at each point in time can easily be read off. In Section 6, a scoring function is derived from the colouring of states and actions by certain algebraic operations on the number of green and red actions and states of the play. It is argued that a score function should satisfy certain desiderata, for instance that shorter plays should *ceteris paribus* be preferred to longer ones. Section 7 compares a selection of plays and explains discrepancies in their score by reference to causal laws and deontic norms. These examples serve to illustrate how the causal theory, the deontic specification, and the scoring function combine to form a single theory.

2 Background

The topic of scoring performance in simulations has been discussed in various forms in the literature; perhaps most extensively for medical simulations and military simulations. Although discussions tend toward platform-specific details of measurement, the salient issues in simulation-based performance studies are, in fact, similar to those of empirical performance studies. In particular, both simulations and empirical performance studies (especially controlled experiments) rely on administering *small representative tasks*,[15] in place of real-world tasks. This is out of necessity for conducting simulations and studies in reasonable time, but also by design to accentuate learning and systematic assessment. The question of scoring performance in simulations therefore relies on designing these small representative tasks with sufficient *construct validity*[16] and *criterion validity*[17] to allow generalizability of training and assessment results to the real world. Designing small representative tasks, in turn, relies on comprehensive domain-specific task analyses[18] and insight into how to build expertise and skills using these tasks.[19–25]

However, or focus is, in one sense, not on such detailed task analyses. The task that is the subject of our discussion is *means-end reasoning*.[26] This is a judgement and decision-making task[27–30] that consists of choosing and sequencing procedural sub-tasks (means) in order to obtain desired outcomes (ends), in other words to decide on the best, or alternatively, a *satisfying*,[31] course of action. Although the design, training and assessment of procedural tasks (handling fire extinguishers, using structured communication formats when communicating an emergency, etc.) are essential for completeness in training, the means-end reasoning that *deploys* such tasks has a generic task-independent aspect to it. It is precisely this generic aspect that lends itself to the approach we shall elaborate on in the following.

The present research develops a logic-based approach for assessing performance on means-end reasoning. The resulting framework is generic in two respects. First, the framework is domain and task agnostic. Implementing a particular training scenario boils down to describing specific events and tasks that stimulate the particular expertise and skills under training. Such instances are defined by domain specific predicates that express the characteristics of the objects and interactions that make up these events and tasks. In contrast, the logic of causation and inertia (i.e. the default propagation of statefulness through time), as well as the deontic logic of reasoning about acceptable vs. unacceptable courses of action, is handled by a background theory encoded in ASP. This separation of concerns allows the scenario designer to focus on defining the desired expertise and skills rather than on the physical simulation itself.

Secondly, since the framework is fully expressed in logic, it is also fully declarative and therefore independent of any particular simulation platform. Platform and implementation independence is a major concern in the modelling and simulation community. The high-level architecture (HLA) is a standardized simulation protocol[32] that can be implemented in various simulation frameworks and engines. HLA comes with an abstract specification format called the Base Object Model (BOM) format,[33] and more expressive abstractions have been proposed in the form of ontology-based[34] semantic extensions to the BOM format.[35] Such ontologies are currently under development for both military and civilian domains,[36–41] where they are also studied as prerequisites for the modelling and simulation as a service (MSaaS) vision.[42] To enable exercise managers, who may be non-technical experts, to compose simulations from simulation services, the services must have semantic and abstract non-technical, but machine-readable, service descriptions.[43,44] The relevance of the work discussed in this article to all of the above is that the logical framework can be seen to represent such abstract specifications (BOMs,

ontologies, service descriptions) in such a way as to enable machine reasoning over those specifications. This is essential for a structured design and analysis approach to designing skill-centred training scenarios, and so in another sense than above, or focus is indeed on detailed task analysis; namely on analyzing the task of means-end reasoning in a given domain.

From a methodological point of view, the present work is most aptly viewed as an application of classical AI planning to the problem of scoring plays of a simulation-based training scenario. The literature on classical planning spans 50 years of research, and the interested reader is referred to Russell and Norvig's overview of the subject.[10] The scoring function that is proposed is derived from a division of nodes and edges in a transition diagram (corresponding to a *causal theory* in the sense used in symbolic AI into acceptable (green) and unacceptable (red) actions and states. More specifically, the function measures the performance of an agent by calculating the ratio of acceptable and unacceptable actions to the complexity of a task and the actual number of actions performed by the agent. We regard this as the principal idea of the present paper and as its main contribution. Given the *qualitative* basis of this function (that is, the distinction between green and red) its values are most naturally interpreted as a measure of the *instrumental rationality*, i.e. the effectiveness of an agent's means-ends reasoning.

Coloured transition diagrams are defined in a language $\mathcal{ALD}$ which is a straightforward extension of the action language $\mathcal{AL}$[8] with *norms*. The idea of expressing the discrepancy between acceptable and unacceptable actions by a red-green colouring of transition diagrams is not itself one for which we claim originality. Our approach closely follows the deontic action language nC^+ of Sergot and Craven,[45] which is itself a deontic extension of the language C of Giunchiglia et al.[46] Indeed, language $\mathcal{ALD}$ is essentially a re-implementation of Sergot and Craven's deontic extension on top of the action language $\mathcal{AL}$ instead of on top of C. There are two main reasons for doing this, the first has to do with the expressiveness of the respective action languages. More specifically:

- $\mathcal{AL}$ unlike nC^+ allows recursive rules.[47] Recursion is important for any domain in which an action may have to be repeated an indefinite number of times until a condition is met.
- $\mathcal{AL}$ unlike nC^+ has explicit negation aka. strong negation which is useful for distinguishing between what is known to be false and what is not known to be true.[48]

The second reason for choosing $\mathcal{AL}$ over C is practical and has to do with the availability of software. Language $\mathcal{AL}$ comes with an encoding into ASP (we extend this encoding to $\mathcal{ALD}$). ASP is a general purpose programming language for NP-hard problems, it is actively developed, and it exists in a rich ecosystem of programming language APIs and third-party libraries. Examples of programming APIs include Clyngor* and Clorm† for Python, Clj-ansprog for Clojure‡ and EmbASP§ for Java, Python and $C^{\#}$. Third-party libraries include the Asprin[49] library for complex preference reasoning and aspStream¶ for stream reasoning. In comparison, nC^+ is implemented on top of the action language C in a special purpose tool called the Causal Calculator or CCalc. The Causal Caculator was part of Norman McCain's dissertation "Causality in commonsense reasoning about actions" from 1997. It was later incorporated

*https://pypi.org/project/clyngor/
†https://github.com/potassco/clorm
‡https://github.com/zootalures/clj-ansprog
§https://www.mat.unical.it/calimeri/projects/embasp/
¶https://github.com/potassco/aspStream

in the MAD-language (The Modular Action Description language), but does not seem to be actively maintained at present. There is a software tool called Cplus2ASP that translates C theories into ASP,[50] but due to discrepancies in the respective semantic underpinnings, the encoding is not very easy to read (for instance preconditions of causal laws are prepended by double negation as failure, and fluent constants are heads of rules with a double negation of itself as condition). In contrast the ASP encoding of $\mathcal{AL}$ is simple and fairly direct, and therefore easier to inspect, amend and debug.

Non-logical simulation-specific formalisms exists, and it is in place to comment on why we do not use any of these. Foremost among them is the Discrete Event System Specification (DEVS) of Zeigler et al.[51] DEVS is based on automata theory with mixed-in features from process modelling and object orientation. In the interest of brevity, we give a brief comparison of DEVS with ASP, taking them as representative of system-theoretical and logic-based approaches respectively:

- Research traditions:
  - DEVS derives from systems theory, particularly automata theory.[52]
  - ASP comes from non-monotonic logic and more generally symbolic AI.
- Simulation model:
  - DEVS is a discrete *event* formalism. That is, time is continuous and any pair of events can be separated by any length of time. As such it is appropriate for modelling the internal state of cyber-physical systems such as robots.[53]
  - ASP is more compatible with discrete *time* models, that is, with an abstract representation of time as a sequence of discrete steps of the same duration. ASP is appropriate for solving problems that that can be construed as logic problems, e.g. planning, diagnosis, explanation.
- Supported Operations:
  - ASP supports reasoning. An ASP theory can be used to predict events, to explain observations and to answer queries.
  - DEVS defines simulations that can be run from start to finish. To answer a query, one typically has to run the entire system until the simulation of an action sequence is complete.[54]

Foo and Peppas[54] summarize the relative merits of system-based and logic-based formalisms with the following two maxims:

1. If simulating a system that is *inertial*, meaning that facts are propagated forward in time by default, then use DEVS and not logic.
2. In query answering or planning use a logic-based approach.

The first of these maxims, Foo and Peppas argue, follows from the fact that inertia is implicit in the DEVS framework but requires verbose and intricate axiomatizations in logic-based formalisms. However, they only consider early logic-based formalisms such as the situation calculus of McCarthy[55] and the event calculus of Kowalski and Sergot.[56] ASP is a more recent language based on the stable model (answer set) semantics of logic programming proposed by Gelfond and Lifschitz.[57] The stable model semantics *does* have inertia built in, so this particular reason for preferring DEVS falls away. Moreover, since planning is the central inference operation in this paper—as we shall argue later, scoring plays of a simulation scenario is essentially scoring *plans*—ASP becomes the natural choice.

3 Causal theories and action description languages

This section presents an overview of the three formalisms we use to specify dynamic systems in our discussion. As will be apparent, each formalisms serves a particular purpose. Thus, we summarize the action description language $\mathcal{AL}$,[8,58] Answer Set Programming (ASP),[7–9,59,60] an encoding from $\mathcal{AL}$ into ASP,[8] and transtition diagrams[8,58] at the levels of detail needed for our discussion.

An action description language is a formal language for defining state transition systems modelling the effects of actions on the world. Such state transition systems will henceforth also be called *causal theories*. Action languages are commonly used in artificial intelligence and robotics to implement an agent loop comprising the steps observe-plan-execute. Well known examples from the literature are the STRIPS language,[61] the situation calculus,[62] the event calculus,[56] the language C^+ [46] and the language $\mathcal{AL}$.[58] The latter is the language of choice in the present article, for the reasons mentioned in Section 2. We start by describing this language in a bit more detail.

3.1 The action language $\mathcal{AL}$

The action language $\mathcal{AL}$ is parameterized by a *signature* σ that declares symbols of two *sorts*: fluent and action constants. Fluent constants represent properties that can be changed by actions, that is, facts that are true at one point in time and false at another. They can be thought of as atomic propositions. Action constants represent actions, which in this context are atomic (i.e. not complex) performances of an unspecified agent. We shall use $a, a_1, a_2, \ldots$ for action constants and $f, f_1, f_2, \ldots$ for fluent constants. A *fluent literal* denoted $l, l_1, l_2, \ldots$ is a fluent constant f or its negation $\neg f$. Similarly an *action literal* denoted $e, e_1, e_2, \ldots$ is an action constant or its negation. A set of fluent literals S is *complete* if for any f either f or $\neg f$ is in S. It is *consistent* if there is no f such that both f and $\neg f$ are in S.

$\mathcal{AL}$ allows three types of statement:

1. Causal Laws:

$$e_1 \ldots e_m \textbf{ causes } l \textbf{ if } l_1, \ldots, l_n$$

2. State constraints:

$$l \textbf{ if } l_1, \ldots, l_n$$

3. Executability conditions:

$$\textbf{impossible } a \textbf{ if } l_1, \ldots, l_n$$

Causal laws describe the effects on an environment of simultaneously executing a set atomic actions whilst refraining from others (the latter represented by negated action literals). This feature, i.e. concurrent actions in the heads of causal laws is a slight generalization of the $\mathcal{AL}$ language that is useful in practice. In the kitchen scenario it allows an agent to turn off the stove and put a lid on the pan at the same time. Another generalization is that we allow negations of actions to occur as conditions in causal laws. For succinctness of expression the following abbreviation[8] is adopted henceforth.

$$e_1, \ldots, e_m \textbf{ causes } l =_{df} e_1, \ldots, e_m \textbf{ causes } l \textbf{ if } \top$$

where $\top$ is an arbitrary tautology.

State constraints define causal relations between fluents and thereby also *indirect effects* of actions. An executablity condition states that an action a can *not* be performed under the circumstance described by $l_1, \ldots, l_n$. In the literature, executability conditions sometimes take the more general form

$$\textbf{impossible } a_1, \ldots, a_m \textbf{ if } l_1, \ldots, l_n$$

This generalized version says that it is impossible to execute $a_1, \ldots, a_m$ *simultaneously* in a state satisfying $l_1, \ldots, l_n$.[8] This additional expressiveness is not needed for present purposes, though.

Example 1. *The kitchen scenario*. We give an $\mathcal{AL}$ representation of a simplified version of the kitchen scenario from the introduction: There is a pan with cooking oil on a stove that is turned on too hot. There is a lid for the pan, a kitchen fan, and a vent. The actions available to the agent are

- put the lid on the pan: *put(lid, pan)*,
- turn off the stove: *turnOff(stove)*,
- turn on the fan: *turnOn(fan)*, and
- a *wait* action that has no effect on fluents.

The *wait* action will function as a device for saying that the agent does not do anything else. It is convenient for triggering negative action literals in causal rules, as explained below.

The fluents that describe the relevant properties of the domain are

- *on(stove)* representing that the stove is on,
- *on(fan)* representing that the fan is on,
- *on(lid, pan)* representing that the lid is on the pan,
- *heats(pan)* representing that the pan is overheated,
- *smoke(pan)* representing that smoke is developing from the pan,
- *fireIn(oil)* and *fireIn(vent)* representing fire in oil and fire in vent.

Note that although these actions and fluents are complex terms (e.g. built from *put*(), *lid*, *pan*), they are considered atomic here in the context of $\mathcal{AL}$; they are simply actions or fluent. In this running example, these terms anticipate the finer-grained signatures of ASP in the next section. The actions and fluents are related by the following causal laws:

turnOff(stove) **causes** $\neg$*on(stove)* (1)

turnOn(fan) **causes** *on(fan)* (2)

put(lid, pan) **causes** *on(lid, pan)* (3)

put(lid, pan), turnOff(stove) **causes** $\neg$*fireIn(oil)* **if** $\neg$*fireIn(oil)* (4)

$\neg$*turnOff(stove)* **causes** *heats(pan)* **if** *on(stove)* (5)

$\neg$*turnOff(stove)* **causes** *smoke(pan)* **if** *on(stove), heats(pan)* (6)

and state constraints:

fireIn(oil) **if** *heats(pan), smoke(pan)* (7)

fireIn(vent) **if** *fireIn(oil), on(fan)* (8)

$\neg$*fireIn(oil)* **if** $\neg$*on(stove),*$\neg$*smoke(pan)* (9)

Some noteworthy elements on this list are: The causal law (4) states that putting the lid on the pan and turning off the stove *simultaneously* always suffices to prevent a fire given that the fire has not already erupted. Causal laws (5) and (6) represent sins of omission, as it were. If the agent does anything other than turning off the stove, say he waits or turns on the fan, then the situation deteriorates. Note also state constraints (9); it says that there is one more way to avert a fire, namely to turn off the stove before smoke develops from the pan.

Finally, one may add executability conditions to prevent the agent from futile pursuits such as trying to turn off a stove that is not on, or repeatedly putting the lid on the pan:

impossible *turnOff(stove)* **if** $\neg$*on(stove)* (10)

impossible *turnOn(fan)* **if** *on(fan)* (11)

impossible *put(lid, pan)* **if** *on(lid, pan)* (12)

As well as constraints making the *wait* action incompatible with all other actions

impossible *turnOff(stove)* **if** *wait* (13)

impossible *turnOn(fan)* **if** *wait* (14)

impossible *put(lid, pan)* **if** *wait* (15)

This gives a simple action description for the kitchen scenario that will be used as a running example throughout.

An important feature that makes $\mathcal{AL}$ particularly suitable for the purposes of the present paper is that its semantics is defined in terms of an encoding into ASP. The ASP encoding allows $\mathcal{AL}$ descriptions to be compiled into computable causal theories in a logic programming language that supports a wide range of reasoning tasks such as *planning* and *prediction*.

The reader may wonder why $\mathcal{AL}$ is needed at all, and why we choose to work with more than one knowledge representation language. There are two reasons for this: First, $\mathcal{AL}$ abstracts away a lot of the nitty-gritty details of causal theories including, incrementing the time counter, axiomatizing general causality and declaring sorts, actions, and fluents. In other words, $\mathcal{AL}$ acts as a high-level API that allows non-programmers to concentrate on describing domain-specific actions and events in a very simple and abstract rule language. Secondly, although $\mathcal{AL}$ can be compiled into ASP, it does not have to be. It could equally well be compiled into another logic programming language such as Prolog, or even into imperative languages such as $C^{\#}$ for the purposes, say, of simulating a scenario in a 3D engine such as Unity. In other words, language $\mathcal{AL}$ fully decouples a causal theory from any particular implementation of it, which in turn enhances the versatility and reuse value of the causal theory.

3.2 Answer Set Programming

The signature σ of an answer set program consists of *variable symbols*, *object names* (also known as *constants* or *object constants*), *function symbols*, *predicate symbols* and *logical connectives*. The convention is that variable symbols are arbitrary strings of letters and numbers that start with an uppercase letter, while constants, predicate symbols and function symbols are strings that start with a lowercase letter. Object and function constants are used to construct *terms*, which are defined inductively:

Definition 1. *Term.*

1. *A variable is a term.*
2. *A constant is a term.*
3. *$f(t_1, \ldots, t_n)$ is a term whenever f is an n-ary function symbol and $t_1, \ldots, t_n$ are terms.*

A term is *ground* if it contains no variable symbols. An *atom* is a formula $p(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are terms. If each term t_i is ground then the atom is ground. A *simple literal* is either a simple atom or an atom preceded by *strong negation* $\neg$. We continue to denote simple literals by l, l_i, with the intent to state the correspondence between $\mathcal{AL}$ and ASP at this level of syntax, relying on context to disambiguate the two formalisms when necessary. If Γ is an ASP program, $lit(\Gamma)$ denotes the set of simple literals contained in a program Γ. A *complex literal* is defined inductively as either a simple literal or a complex literal prepended by '*not*'. The connective '*not*' denotes *negation as failure*, a notion that will be explained shortly. A rule is ground if each literal that occurs in it is ground. A program $gr(\Gamma)$ consisting of all ground instances of all rules in Γ is called the *ground instantiation of* Γ.

For the purposes of the present paper we shall work with rules restricted to the following forms:

1. Facts:
$$l. \tag{16}$$
2. Rules:
$$l \leftarrow l_1, \ldots, l_m, not\, l_{m+1}, \ldots, not\, l_n. \tag{17}$$
3. Integrity constraints:
$$\leftarrow l_1, \ldots, l_k, not\, l_{k+1}, \ldots, not\, l_m. \tag{18}$$

The expression to the left of $\leftarrow$ is the *head* of a rule and the expression to the right of $\leftarrow$ its *body*. The comma-separated lists of the body denote conjunctions of complex literals. Facts and integrity constraint are degenerate cases of a rule, in the former case a rule with an empty body and in the latter case a rule with an empty head. This redundancy in the presentation is common and convenient.

Rules of form (17) may be thought of as production rules: If the body of the rule can be deduced from a program then the head of the rule is added to the set of facts (16). Integrity constraints act as filters that exclude models satisfying the body of the constraint. Equivalently, an integrity constraint lists conditions that cannot all be true, forcing models not to have that particular combination of properties. The difference between explicit negation and negation as failure, also known as *epistemic negation,* is very important: An explicitly negated atom $\neg p(t_1, \ldots, t_n)$ is deducible from a program if that literal occurs in the head of some rule of form (17). In other words $p(t_1, \ldots, t_n)$ is asserted to be false under the conditions stipulated in the body of the rule. In contrast $not\, p(t_1, \ldots, t_n)$ is true in *absence of evidence to the contrary*; that is, it is true if $p(t_1, \ldots, t_n)$ can *not* be deduced from the rules of the program. To understand the difference between negation as failure and explicit negation, consider the typical example[48] of the distinction between the rule $cross \leftarrow not\ train$ that captures the criterion "you can cross if you have no evidence that a train coming," and the rule $cross \leftarrow \neg train$ which means "you can cross if you have evidence that no train is coming". Negation as failure induces non-monotonic behaviour in an answer set program and is crucial for formalizing default reasoning in general and causality in particular. Some examples of this as it pertains to the encoding of the causal theories in general, and the kitchen theory in particular, will be discussed in due course.

The semantics of answer set programs is defined for ground programs, and its full statement requires a fair bit of technical nomenclature. Suffice it here to outline the essentials: we start with the classical notion of an *interpretation* or *model*: a *model* I of a program Γ is a subset of $lit(\Gamma)$ that is closed under the rules of Γ, meaning that I satisfies the head of a rule of form (17) whenever it satisfies its body. The satisfaction relation $I \models \phi$ between a model I and a formula ϕ (rule or conjunction of complex literals) is defined as:

$$\begin{aligned} I \models p &\text{ iff } p \in I, \text{ for an atom } p \\ I \models not\, p &\text{ iff } p \notin I \\ I \models \neg p &\text{ iff } \neg p \in I \\ I \models \phi, \psi &\text{ iff } I \models \phi \text{ and } I \models \psi \\ I \models \phi \leftarrow \psi &\text{ iff } I \not\models \psi \text{ or } I \models \phi \end{aligned}$$

The reader should note the difference between the second and third item on this list.

The *answer sets*, also known as the *stable models*, of a program Γ is a particular subset of the set of models defined above, namely the models that can plausibly be treated as idealized representations of the belief sets of a rational agent. These are the minimal models with the property that they are are both closed wrt. to the rules of Γ and such that every fact holding in them is *supported* by applications of those rules. The reader should note that there are in general minimal models that are not answer sets in this sense.

Answer sets are easily identified for certain classes of simple programs. For instance, if a program has a finite set of rules and no epistemic negation, then it has a unique minimal model which is also its answer set—it is the single model that satisfies the set of rules with no gratuitous information. A program that contains complex literals, on the other hand, may have more than one minimal model. For example,

$$\Gamma = \left\{ \begin{array}{l} b \leftarrow not\, a \\ a \leftarrow not\, b \end{array} \right.$$

has two minimal models $\{a\}$ and $\{b\}$. These models can be identified as fixpoints of the operation of *reducing* Γ to a negation-free program relative to a model. More precisely, the *reduct*, Γ^I of Γ relative to a model I of Γ is the set of rules derived from rules of form (17) by including

$$l \leftarrow l_1, \ldots, l_m \tag{19}$$

in Γ^I whenever $\{l_{m+1}, \ldots, l_n\} \cap I = \emptyset$ for the complex literals $not\, l_{m+1}, \ldots, not\, l_n$. Thus, the general definition of an answer set; a model I of an ASP program is an answer set iff I is the minimal model of the reduct Γ^I. We will denote the answer sets of Γ as $M(\Gamma)$.

3.3 Encoding action descriptions in ASP.

$\mathcal{AL}$ is a high-level language for defining causal theories. The semantics of an $\mathcal{AL}$ description is defined in terms of its encoding into ASP. Considered in the abstract, this encoding can be viewed as an operation

$ASP(\cdot)$ that takes an $\mathcal{AL}$ description $\mathbb{A}$ and a non-negative integer n called a *horizon* and outputs an ASP program $\Gamma_n^{\mathbb{A}}$. The program $\Gamma_n^{\mathbb{A}}$ gives a computable description of all plans of alternating actions and states of length up to n determined by action description $\mathbb{A}$. One way to view $\Gamma_n^{\mathbb{A}}$ is as a labelled transition diagram where nodes are states and transitions are actions. Plans then become paths through the diagram from the initial state to a goal state at most n steps distant. We shall return to transition diagrams shortly, as they have practical utility for our discussion. For now, the following points should be noted: An action description $\mathbb{A}$ describes the effect of actions in general terms. The causal theory $ASP(\mathbb{A}, n) = \Gamma_n^{\mathbb{A}}$ models the evolution of an initial state *over time* as a consequence of actions $a \in \mathbb{A}$ being performed, in general, more than once, at various moments. Therefore, the essential step in defining the encoding $ASP(\cdot)$ is to assign time indices to fluents and actions as a means of representing *when* they are, respectively, executed and true. This involves a manoeuvre called *reification*, which in knowledge representation is the process of turning a predicate into an object. The $ASP(\cdot)$ encoding uses the meta-predicates *occurs* and *holds* for this purpose.[7] A statement $\mathit{occurs}(a, 2)$ asserts that action a is performed at time-step 2 (assuming $2 \leq n$), whilst $\mathit{holds}(f, 2)$ asserts that fluent f is true at step 2. For positive action and fluent literals, $\mathcal{AL}$ rules translate as follows:

- Every causal law $a_1, \ldots, a_n$ **causes** l **if** $l_1, \ldots, l_n$ becomes an ASP rule:

$$\begin{aligned} \mathit{holds}(f, T+1) \leftarrow \quad & \mathit{holds}(f_1, T), \ldots, \mathit{holds}(f_n, T), \\ & \mathit{occurs}(a_1, T) \\ & \ldots \\ & \mathit{occurs}(a_n, T) \\ & T < n. \end{aligned} \tag{20}$$

- Every state constraint f **if** $f_1, \ldots, f_n$ becomes an ASP rule:

$$\mathit{holds}(f, T) \leftarrow \quad \mathit{holds}(f_1, T), \ldots, \mathit{holds}(f_n, T). \tag{21}$$

- Every executability condition **impossible** a **if** $f_1, \ldots, f_n$ becomes an ASP rule:

$$\neg\mathit{occurs}(a, T) \leftarrow \quad \mathit{holds}(f_1, T), \ldots, \mathit{holds}(f_n, T). \tag{22}$$

Note that this gives the translation for the case of positive literals only. In the general case, negated action literals, say $e = \neg a$, would be translated as $\neg\mathit{occurs}(a, T)$ (as anticipated in rule 22), and negated fluent literals, say $l = \neg f$, as $\neg\, \mathit{holds}(f, T)$. It is straightforward to generalize the rules (20-22) accordingly, but rather tedious and unenlightening, so we leave it at that.

Causal laws, state constraints and executability conditions form the *domain-specific* part of a causal theory corresponding one-to-one to rules in the $\mathcal{AL}$ description in question. In addition, the ASP encoding also maintains general axioms for *holds* and *occurs* that characterize the concept of causal reasoning as such. Causal reasoning, unlike classical logical deduction, is *non-monotonic*. Informally, this means that a reasoning agent is allowed to jump to a certain conclusion in the absence of evidence to the contrary if that conclusion is warranted in *typical* cases. This is a pervasive inference pattern in commonsense reasoning. For instance, if you are told that Juliette lives in Rouen you may assume that she appreciates a good Livarot, since this is a cheese of the Normandy region. Yet, upon learning that Juliette is lactose intolerant you would probably retract this conclusion. This example illustrates that in non-monotonic

reasoning the set of conclusions does not necessarily increase as more information becomes available. Some inferences are tentative, meaning that their justification appeals to the *absence* of evidence to the contrary. Tentative conclusions can therefore be withdrawn when one learns something new.

Inference rules that are sensitive to the absence of information in this manner are known as *defaults* or *exception-allowing rules*. A default that occupies pride of place in causal theories is the so-called *inertia axiom*. It formulates the commonsense law that facts tend to persist from one moment in time to the next unless acted upon by an opposite force. The inertia axiom prevents objects from behaving erratically. For instance, moving a cup will not close the door since the door's being closed is a fact that is propagated forward in time by inertia and there is no causal rule that connects the cup to the door. The problem of finding a concise and accurate representation of inertia, known as the *frame problem*, has been a central topic in symbolic AI for more than forty years. With the development of new programming paradigms, such as Answer Set Programming and new action languages such as the ASP-based $\mathcal{AL}$, the frame problem can now be said to have been satisfactorily solved. In the ASP encoding of $\mathcal{AL}$, inertia is expressed as follows:

$$\begin{aligned} \mathit{holds}(f, T+1) \leftarrow \quad & \mathit{holds}(f, T), \\ & \mathit{not}\ \neg\mathit{holds}(f, T+1), \\ & T < n. \end{aligned} \tag{23}$$

Spelled out, this rule says that if f is true at time T and *it may, for all we know, continue to be true* at time $T+1$, then it is assumed to be true also at time $T+1$. Notice how strong negation and epistemic negation are combined here to propagate the truth of fluents through time: Given that the falsity of f at $T+1$ is represented as $\neg\mathit{holds}(f, T+1)$, the absence of evidence of the falsity of f at $T+1$ can be represented with epistemic negation as $\mathit{not}\ \neg\mathit{holds}(f, T+1)$ which warrants jumping to the conclusion that $\mathit{holds}(f, T+1)$. Of course, if one were to learn that $\neg\mathit{holds}(f, T+1)$ then the rule of inertia would yield to that exception.

Finally, the encoding adds the so-called *closed world assumption* for actions. For variable A,

$$\neg\mathit{occurs}(A, T) \quad \leftarrow \quad \mathit{not}\ \mathit{occurs}(A, T), \mathit{action}(A). \tag{24}$$

It sanctions the conclusion that an action is not performed if it has not been explicitly recorded or deduced. This rule has a crucial role to play in connection with causal rules with a negative premise, for instance rules (5) and (6) above. Recall that these rules express that if the agent does anything other than turning off the stove in any situation prior to the breakout of a fire then the situation worsens. The closed world assumptions for actions allows ASP to reason from the absence of an explicit record of the *turnOff(stove)* action at some time-step T, to the non-performance, i. e. the explicit falsity, of that action at T, thus triggering the conditions of causal rules (5) and (6).

3.4 Transition diagrams.

$\mathcal{AL}$ descriptions and their corresponding causal theories can be visualized in a convenient format as transition diagrams. Such diagrams are useful for discussing the red/green labelling of plays/histories from which the proposed scoring function will be derived. Transition diagrams for action descriptions in $\mathcal{AL}$ are discussed in the literature.[8,58] However, the definition that follows is novel and rather different,

for reasons that will be explained.

Definition 2. *A transition diagram for an $\mathcal{AL}$ action description $\mathbb{A}$ is a structure $\mathbb{D} = \langle S, A, R, s^* \rangle$, where*

1. *S is a complete and consistent set of fluent literals from the signature of $\mathbb{A}$ satisfying the state constraints in $\mathbb{A}$,*
2. *$R \subseteq S \times 2^A \times S$ is a transition relation between states labelled by a set of action constants A from the signature of $\mathbb{A}$, and*
3. *s^* is a designated initial state.*

Some additional nomenclature is needed to define the set of states S and the relation R: Let f be a fluent constant in ASP. For convenience[8] we define $h(l, i)$ as *holds*(f, i) if $l = f$ and as $\neg$*holds*(f, i) if $l = \neg f$. Whether negated or not, such statements will be referred to collectively as *holds* statements.

Let n be our horizon. For any set of fluents s and time $m \leq n$, define

$$\rho(s, m) =_{df} \{h(l, m) : l \in s\}$$

Intuitively $\rho(s, m)$ takes the $\mathcal{AL}$ fluent literals in s and converts them into reified *holds* statements of ASP, time-stamped with m. For example, consider Fig. 1, which displays a sub-graph of the transition diagram for the kitchen scenario, where the initial state s^* is outlined in bold. Then, $\rho(s^*, 1)$ gives the set of appropriate *holds* statements in ASP for each of the fluent literals in s^*. These *holds* statements are then used, together with the *occurs* statement of the relevant action literal (e.g. *occurs*(*turnOff*$(stove), 1)$) to compute the consequential *holds* statements according to the causal theory. This computation is formally given by a *nxt* operator as follows:[58] Let $M(\Gamma)$ be the operator that returns all stable models of the ASP program Γ. Then, for a set of fluents s, action a and time m,

$$\textit{nxt}(s, a, m) =_{df} \begin{cases} M(\Gamma_n^{\mathbb{A}} \cup \rho(s, m) \cup \textit{occurs}(a, m)) & \text{if } \Gamma_n^{\mathbb{A}} \cup \{a\} \text{ is consistemt,} \\ \emptyset & \text{otherwise} \end{cases}$$

This definition presupposes that the causal theory is deterministic in the sense that an action always produces a single next state. Under this assumption an action generates a next state if it is not excluded by an executability constraint in $\Gamma_n^{\mathbb{A}}$.

Conversely, let Δ be a set of *holds* statements of ASP time-stamped with m and define:

$$\begin{aligned} \rho^{-1}(\Delta) =_{df} & \{f : \textit{holds}(f, m) \in \Delta\} \cup \\ & \{\neg f : \neg\textit{holds}(f, m) \in \Delta\} \cup \\ & \{\neg f : \textit{holds}(f, m) \notin \Delta, \neg\textit{holds}(f, m) \notin \Delta, f \in Sig(\mathbb{A})\} \end{aligned}$$

The operation ρ^{-1} takes a set of *holds* statements, converts it to a set of literals of $\mathcal{AL}$, and then *completes* it to make an $\mathcal{AL}$ state out of it. Completion here means to add the negation of all fluent constants f that do not occur in the set Δ. For example, given the set of *holds* statements computed by *nxt*$(s^*,$ *turnOff*$(stove), 1)$, ρ^{-1} will convert those *holds* statements to the set of fluent literals in the

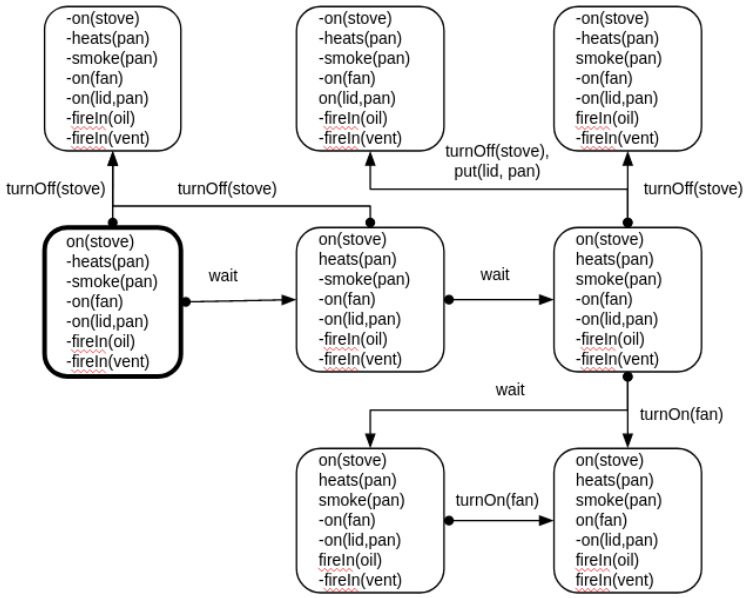


Figure 1. An excerpt of the kitchen scenario. Strongly negated fluent literals are written as $-f$.

top left state in Fig. 1. The entire step from transition diagram state via ASP computation to the next transition diagram state is given formally by

$$scc(s, a, m) =_{df} \rho^{-1}(nxt(s, a, m))$$

that is, the successor state of a state s relative to a and m in $\mathbb{D}$. It is easy to verify for any complete and consistent set s of $\mathcal{AL}$ fluent literals $\rho^{-1}(\rho(s)) = s$, that ρ^{-1} is the inverse of ρ.

The idea behind the diagram construction is to induce the set of states S and the transition relation R from the consequences according to the theory $\Gamma_n^{\mathbb{A}}$ of performing an action a in a state s. The bijection ρ, ρ^{-1} allows us to move from states in $\mathbb{D}$ to the corresponding sets of holds-statements in ASP, and back again. We are now in position to give the full definition of the transition diagram induced by an $\mathcal{AL}$ action description $\mathbb{A}$:

Definition 3. *Let $\mathbb{A}$ be an $\mathcal{AL}$ action description with action constants A and assume an initial state s^*. Let S and R be defined by a double induction as follows:*

1. $S_0 = \{s^*\}$, $R^0 = \emptyset$
2. $S_{n+1} = S_n \cup \{scc(s, a, n) : s \in S_n, a \in A\}$,
3. $R_{n+1} = R_n \cup \{\langle s, a, s' \rangle : s' = scc(s, a, n), s \in S_n\}$.

Put $R =_{df} \bigcup_{i=1}^{n} R_n$ and $S =_{df} \bigcup_{i=1}^{n} S_n$. Then $\langle S, A, R, s^* \rangle$ is a transition diagram for $\mathbb{A}$ with initial state s^*.

This definition has a rather different flavour from that of previous approaches,[8,58] and some comments are perhaps called for: First, unlike those approaches, Definition 3 ignores all complete and consistent sets of fluents that are not reachable from the initial state, since these are irrelevant from the point of view of simulation-based training. Conversely, just as unreachable states are irrelevant, the same goes for transitions that link them to each other. Honing in on only relevant states and transitions makes the two concepts interlock in a manner that requires the double induction of Definition 3.

Nevertheless, a transition diagram as so defined will still be large for all but trivial theories as it will contain all allowable combinations of actions and states. Many if not most of these possibilities are not practically interesting, for instance applying a fire extinguisher to a pan that isn't burning. Although a discerning use of executability constraints can eliminate choices like this that no one would make, transition diagrams are still best regarded as a tool for conceptual clarification or as a tool for defining reasoning problems that to be embedded in an ASP theory.

4 Plays as plans

We have proposed to give the concept of a training scenario the formal meaning of a dynamic system or causal theory (in the present context these two terms can be used interchangeably). The term "scenario" has many definitions.[63] The definition used in this paper is the one used by the Simulation Interoperability Standards Organization in their scenario development guidelines document:[6]

> "A scenario is a description of the hypothetical or real area, environment, means, objectives, and events during a specified time frame related to events of interest."

This is very close to the idea of a dynamic system as a finite number of states causally interlinked by actions. Close enough, at any rate, to treat the latter concept as a reasonable formal approximation of the former for computational purposes. Henceforth, therefore, we shall talk interchangeably about causal theories, action descriptions, and transition diagrams, on the one hand, and training scenarios on the other. That is, we are treating an action description in $\mathcal{AL}$, its ASP encoding and its associated transition diagram as different representations of the same training scenario.

As regards the concept of *a play*, which is ultimately the unit of interest here since it is plays that will be scored, it is intuitively clear that a play is just a directed path through the transition diagram. In order to assess trainee performance, one needs to distinguish between those plays in which hazards are handled with some degree of competence or success and those in which they are not, allowing for a sufficiently general criterion of success. For instance, preventing a fire from arising will count as a successful handling of the industrial kitchen scenario, but so may putting the fire out, or pressing the alarm button when things really get out of hand. These plays will be scored differently, of course, according to the training objectives as expressed in part by a deontic specification. Purely unsuccessful developments consists of the plays in which the trainee's pursuits are futile. Say he searches for an extinguisher but never finds one, or applies a standard foam extinguisher on a metal fire, etc. All courses of action in this latter group will be scored at zero. The problem of scoring plays is thus primarily about the former group.

The computational model behind ASP as a programming language is a good fit for extracting and evaluating plays from a training scenario insofar as problem solving in ASP reduces to enumerating satisfying models. Inference engines that compute the answer sets of ASP programs are usually referred to as *answer set solvers*.[64] An answer set solver attacks the computation of a program in a two-stage

process: first it starts with grounding the ASP program, which means instantiating its variables by ground terms. The resulting program has the same answer sets as the original but is essentially propositional. In the second stage, the answer sets of the grounded program are computed using substantially modified and expanded satisfiability checking algorithms, usually a variant of the Davis-Putnam resolution algorithm.[8,9] Thus, given an ASP program, an answer set solver grounds the program and generates models that satisfy all rules of the program and that violate none of its integrity constraints.

Applied to a causal theory, this model of computation can be used to extract sequences of actions that lead from an initial state to a goal state in a dynamic domain. Here a goal state is understood as any state that conforms to some criterion of success as explained above. The concept of a play we thus arrive at—a sequence of actions conforming to a criterion of success—coincides with the concept of a plan in a *classical planning problem*.[8,10] A classical planning problem has the following constituents:

- A **goal description** which in the context of a causal theory is a set of fluent literals describing features of one or more desirable state of affairs, i.e. goals.
- An **initial state** from which a goal state is to be reached.
- A **horizon** which is a limit on the length of allowed plans.

Planning is the problem of computing a sequence of actions that evolves the initial state into a goal state within the limit of steps set by the horizon. Given an ASP causal theory a planning module can be added in four lines of code, that exemplify a well-documented technique in automated planning.[7–9,60,65]

$$1\{\mathit{occurs}(A,T) : \mathit{action}(A)\}2 \leftarrow T < n. \tag{25}$$

$$\mathit{goal}(T) \leftarrow \ldots \tag{26}$$

$$\mathit{goal}(\mathit{achieved}) \leftarrow \mathit{goal}(T), T <= n. \tag{27}$$

$$\leftarrow \mathit{not}\ \mathit{goal}(\mathit{achieved}). \tag{28}$$

The rule (25) is a *choice rule*, which describes alternative ways to form an answer set by choosing elements described by the set builder notation. The numbers flanking the expression in braces are upper and lower bounds on the cardinality of the selection that is triggered when the condition in the body is satisfied. Informally, the rule says: for each time-step within the horizon n, choose at least one and at most two (concurrent) actions to be performed. Note that this rule does not conform to any of the schemata in the list (17-18) However, choice rules can be translated into a set of rules of form (17).[60] This is a matter of some importance for the proof of conservativeness of the deontic extension in Sect. 5. The second rule (26) is elliptical for a goal description where the ellipsis would be replaced by a list of fluents. Rule 27 can be read as an existentially quantified expression which states that the goal has been achieved if satisfied at *some* time point T. Dually, the rule (28) is a constraint the excludes all models in which the goal is *never* achieved.

A goal may be described in multifarious ways some of which do not say anything in particular about the domain. For instance the limiting case

$$\mathit{goal}(T) \leftarrow T = n. \tag{29}$$

deems any plan that exhausts the horizon n a successful play. This goal specification makes the answer set solver generate all plays of length n. It is useful for obtaining a complete picture of all possible plays, acceptable or unacceptable.

For an example of a domain-specific goal, consider:

$$\mathit{goal}(T) \leftarrow \neg\, \mathit{holds}(\mathit{fireIn}(\mathit{oil}), T), T < n. \tag{30}$$

This goal requires that there be no fire at any time-point within the horizon n.

Fig. 2 shows two answer sets (projected onto actions) generated by adding rules (25-28) and (30) to the kitchen scenario from the previous section. In the first play, the agent allows the pan to overheat, then successively turns off the stove and puts a lid on the pan. These actions successfully prevent a fire in the cooking oil. No further action is required and the trainee waits. In the second play, the trainee turns off the stove and puts the lid on the pan at the same time and at the last minute. This suffices to avert a pan fire. The trainee waits for two time-steps and then turns the fan on.

The action of turning on the fan is interesting. It is sometimes harmful, and sometimes OK. If there is already a fire in the cooking oil, then it is harmful according to the causal theory since it may cause the fire to spread to the vent shaft. If the fire has been averted, as is the case in the second play, turning on the fan merely dissipates the smoke which is ok. The example demonstrates that the desirability or undesirability of an action is a context-sensitive affair. An action may not be good or bad *per se*. Rather the determination of its deontic status, its value or utility or what have you, involves a *computation* that derives all the significant facts about the situation pertaining to that action.

$\mathit{occurs}(\mathit{wait}, 1)$
$\mathit{occurs}(\mathit{turnOff}(p), 2)$
$\mathit{occurs}(\mathit{put}(\mathit{lid}, \mathit{pan}), 3)$
$\mathit{occurs}(\mathit{wait}, 4)$
$\mathit{occurs}(\mathit{wait}, 5)$
$\mathit{occurs}(\mathit{wait}, 6)$

$\mathit{occurs}(\mathit{wait}, 1)$
$\mathit{occurs}(\mathit{wait}, 2)$
$\mathit{occurs}(\mathit{put}(\mathit{lid}, \mathit{pan}), 3)$
$\mathit{occurs}(\mathit{turnOff}(p), 3)$
$\mathit{occurs}(\mathit{wait}, 4)$
$\mathit{occurs}(\mathit{wait}, 5)$
$\mathit{occurs}(\mathit{turnOn}(\mathit{fan}), 6)$

Figure 2. Two example plans.

We have our work cut out for us, then: identifying plays with plans we are after a principled way to assign scores to sequences of actions such as those in Fig. 2. Clearly, such a scoring function needs to be more sophisticated than, say, just measuring the length of successful plays, since different orderings of the same set of actions implies different contexts of agency that may imbue the same action with different utility or value. This is the problem to which we turn next.

5 The deontic overlay

In this section, we turn to the task of defining a deontic overlay to the action language $\mathcal{AL}$. The basic idea is to provide a means of specifying what is to count as acceptable (i.e. useful or prudent) and unacceptable (worthless or imprudent) actions, and to have ASP work out the normative consequences of this classification. Following Sergot,[45] we represent these classes of states and actions at the ASP level by the respective colours red and green. Colours are propagated exhaustively throughout a transition diagram by computing the consequences of a set of *norms*, which are $\mathcal{AL}$/ASP rules that give the conditions under which an action or state is to be deemed acceptable or unacceptable for a particular purpose. This set of norms will be referred to as a deontic specification or overlay. The syntactical form of these norms is designed to balance *expressiveness* against *conservativeness*; we want to be able to state useful constraints on acceptable plays of a simulation scenario (expressiveness), yet we do not want norms to interfere with the causal theory itself (conservativeness). We give a proof that this is indeed so.

5.1 The language $\mathcal{ALD}$

The deontic overlay consists of rules specifying the conditions under which a state or transition is to be classified as acceptable or unacceptable. Such rules will be referred to as *norms*. Norms can be represented by extending the action language $\mathcal{AL}$ by two new kinds of rules:

1. State norms:
$$\textbf{unacceptable } f \textbf{ if } l_1, \ldots, l_m. \tag{31}$$
2. Action norms:
$$\textbf{unacceptable } a \textbf{ if } l_1, \ldots, l_m. \tag{32}$$

We shall call the new language $\mathcal{ALD}$ for easy reference. Note that both kinds of rule allows mixed formulae in the body, that is, it allows both action literals and fluent literals in conditions.*

*As pointed out by one of the reviewers, the language $\mathcal{ALD}$ has interesting parallels to the language $\mathcal{APL}$ proposed by Gelfond and Lobo.[66] They consider authorization policies that partition actions into *permitted* and *not permitted* ones. A state/action pair is defined as compliant with a policy if the state entails the permission of the action. The central reasoning problem is that of computing the compliance of a given sequence of state/action pairs with a policy. A detailed comparison of $\mathcal{ALD}$ and $\mathcal{APL}$ is out of scope for the present paper. Suffice it for now to point out a few of the more conspicuous similarities and differences: while the red/green distinction in $\mathcal{ALD}$ corresponds to permitted/not permitted in $\mathcal{APL}$, a strictly permissive reading of red and green is not mandatory. Rather the red/green distinction is intended primarily to express norms governing *instrumental rationality* understood as the prudence or utility of an action in light of a goal. In $\mathcal{ALD}$ it is conceptually the goal that renders an action green or red, whereas permission in $\mathcal{APL}$ is defined by an authorization policy.

On a formal level we note the following: unlike $\mathcal{ALD}$, $\mathcal{APL}$ allows one to express *priorities between defeasible norms*. This is an interesting candidate for further development of $\mathcal{ALD}$. Next, permissions in $\mathcal{APL}$ apply to actions only, whereas the red/green distinction in $\mathcal{ALD}$ applies to both actions and states. Therefore, $\mathcal{ALD}$ unlike $\mathcal{APL}$ can express recovery from suboptimal states as chains of red states linked by green actions, and it can express propagation rules such as the *ggg*-constraint. Finally, in $\mathcal{APL}$ the central reasoning problem is that of computing the compliance of a *given* sequence of state/action pairs with an authorization policy. In contrast, the central reasoning problem of $\mathcal{ALD}$ is to compute all action sequences that lead from an initial state to a goal with their corresponding scores. There is no assumption that these sequences are classifiable into compliant or non-compliant ones wrt. a policy. And of course, there is no assumption that all choices are even good ones, although a domain description must generate at least one sequence with only green choices if inherent task complexity, and thus scores, are to be computable.

Example 2. *To the simple kitchen theory in Example 1, one might consider adding norms that express mandatory actions for a trainee to complete the exercise successfully. For instance, if preventing fire is the training objective, then the trainee should be forbidden to allow the pan to catch fire. The following list gives some examples of the norms that may be entertained:*

- *Action norms:*

unacceptable *turnOn(fan)* **if** *smoke(pan), on(stove)* (33)
unacceptable *wait* **if** *heats(pan)* (34)
unacceptable *wait* **if** *smoke(pan)* (35)

- *State norms:*

unacceptable *heats(pan)* **if** $\top$ (36)
unacceptable *smoke(pan)* **if** $\top$ (37)
unacceptable *fireIn(oil)* **if** $\top$ (38)
unacceptable *fireIn(vent)* **if** $\top$ (39)

Here, $\top$ *denotes an arbitrary tautology, meaning that the head of the norm is unacceptable under all circumstances. Note that the first action norm is* context sensitive*. Turning on the fan is not deemed unacceptable as such, but only if the pan is overheating and smoke develops from it. These conditions may in turn depend on other causal rules in the underlying* $\mathcal{AL}$ *action description. In other words, unacceptable states and actions need not be explicitly marked as such. They are in general* computed *from the norms and the underlying causal theory.*

5.2 Translation to ASP

Norms in $\mathcal{ALD}$ are translated into ASP as follows

- Every state norm **unacceptable** f **if** $l_1, \ldots, l_m$ becomes an ASP rule:

$$\begin{aligned} \mathit{status}(\mathit{red}, T) \leftarrow \quad & \mathit{holds}(f, T), \\ & \mathit{holds}(l_1, T), \ldots, \mathit{holds}(l_m, T), \\ & T < n. \end{aligned} \tag{40}$$

- Every action norm **unacceptable** a **if** $l_1, \ldots, l_m$ becomes an ASP rule:

$$\begin{aligned} \mathit{trans}(\mathit{red}, a, T) \leftarrow \quad & \mathit{occurs}(a, T), \\ & \mathit{holds}(l_1 T), \ldots, \mathit{holds}(l_m, T), \\ & T < n. \end{aligned} \tag{41}$$

State and action norms form a group that may be called *domain norms*. These are domain- and task specific norms that label concrete actions and fluents occurring in the causal theory as either acceptable or unacceptable according to the particular scenario under development. In addition, the deontic overlay provides generic norms that are more appropriately classified as *logical* norms. Their role is not

to annotate the causal theory directly, but rather to govern the logic of the acceptable/unacceptable distinction by propagating colours throughout the diagram. This group of norms can be subdivided into the set of *closure conditions* and the singleton consisting of the so-called *green-green-green* constraint (ggg-constraint):

- Closure conditions: every state and transition is green by default.

$$\mathit{status}(\mathit{green}, T) \leftarrow \mathit{not}\ \mathit{status}(\mathit{red}, T),\ T < n. \tag{42}$$

$$\mathit{trans}(\mathit{green}, A, T) \leftarrow \mathit{not}\ \mathit{trans}(\mathit{red}, A, T), \mathit{action}(A),\ T < n. \tag{43}$$

- The ggg-constraint: An action is red if it leads from a green to a red state (note that this makes concurrent actions red if one of their elements is).

$$\begin{aligned} \mathit{trans}(\mathit{red}, A, T) \leftarrow \quad & \mathit{status}(\mathit{green}, T), \\ & \mathit{occurs}(A, T), \\ & \mathit{status}(\mathit{red}, T+1), \\ & T < n. \end{aligned} \tag{44}$$

The closure conditions guarantee that the colouring is exhaustive; there are no uncoloured nodes or transitions. Note, that the default colour of a state or transition is green. This is not strictly necessary, but it is usually easier to say what is unacceptable than what is acceptable. Conceptually, the colour green corresponds to the concept of *negative permission* from deontic logic; what is not explicitly forbidden is permitted[67,68] (in jurisprudence this principle is known as *nullum crimen sine lege*).

As regards the rationale behind the ggg-constraint, studies of database constraints suggest that few deontic constraints can be assumed to hold between information states in general. Illegal transitions can result in an acceptable state, and an acceptable or green action need not restore a red state to a green one. Examples of this are quite frequent. The only relationship that has been found to be generally plausible is the ggg-constraint.[69,70] Other more global constraints are clearly conceivable, though. For simulation scenarios in particular it makes sense to require the existence of a green path from a start state to a goal state. We shall see later how this can be expressed in ASP.

5.3 $\mathcal{ALD}$ transition diagrams.

The state and action norms (40) and (41) encode *domain-specific* qualitative criteria of the unacceptability of a state or transition symbolized by the colour red. The logical norms essentially propagate colours. The colour red is propagated by the ggg-constraints whereas the closure conditions propagate the colour green. The role of the logical norms is easier to explain by considering an $\mathcal{ALD}$ description as an extended transition diagram, that is, as a tuple $\langle S, S_r, A, R, R_r, s^* \rangle$. Here, the reduct $\langle S, A, R, s^* \rangle$ is an $\mathcal{AL}$ transition diagram as specified in Definition 3. The new components $S_r \subseteq S$ and $R_r \subseteq R$ are respectively the red states and transitions. The rules (42) and (43) are closure conditions making sure that the remainders $S_g = S - S_r$ and $R_g = R - R_r$ after colouring states and transitions red are coloured green. Considered as an abstract condition on transition diagrams the ggg-constraint (44) is the condition:

$$(s_i, a, s_j) \in R \text{ and } s_i \in S_g \text{ and } s_2 \in S_r \text{ implies } (s_i, a, s_j) \in R_r \tag{45}$$

This is equivalent to:

$$\text{if } (s_i, a, s_j) \in R_g \text{ and } s_i \in S_g \text{ then } s_j \in S_g \tag{46}$$

which gives it a more direct expression.[70] The overall effect of the rules (42 - 44) is (i) to colour a state green unless coloured red by some state norm, and (ii) to colour a transition red if it is coloured red by some action norm or by the ggg-constraint; otherwise it is green (ibid.).

Fig. 3 gives a version of the diagram in Fig. 1 coloured in accordance with the state and action norms from Example 2. Note that unacceptability is context sensitive in the aforementioned sense. For instance, once the pan overheats and smoke starts to develop, *merely* turning off the stove is an unacceptable action. If the lid is placed on the pan at the same time, however, then turning off the stove is also acceptable.

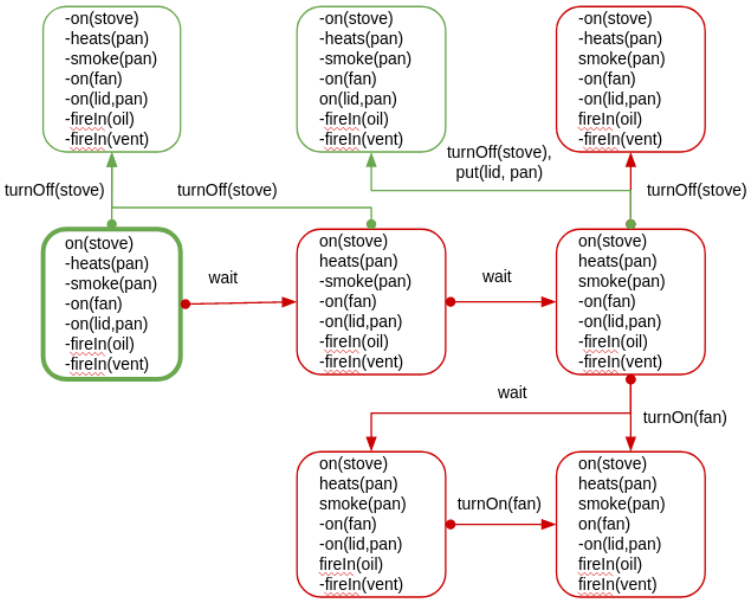


Figure 3. The kitchen scenario coloured.

5.4 The conservativeness of the deontic specification

It is reasonable to require the deontic overlay to leave the causal theory intact. The purpose of the deontic specification is merely to indicate what state of affairs are unacceptable, and, by derivation from the causal theory, when certain actions should not be performed. It should *not* interfere with the causal laws themselves.

A different way to say the same thing is to say that the norms should be of such a form that the addition of the deontic specification constitutes a *conservative extension* of the underlying causal theory. That is, every consequence that is derivable in the language of the causal theory with state and action norms added should be derivable from the causal theory without these norms. In this section we verify that this

is so for any set of rules of form (42–44), and thereby verify that the deontic specification really is an 'overlay'.

There is a rich body of research on conservative extensions in logic programming to draw on. The following definition from Baral[9] will suffice for present purposes:

Definition 4. *Let* Γ *and* Γ' *be ground programs such that* $\Gamma \subseteq \Gamma'$. *Then* Γ' *is said to be a conservative extension of* Γ *if the following condition holds:* A *is a consistent answer set for* Γ *iff there is a consistent answer set* A' *for* Γ' *such that* $A = A' \cap lit(\Gamma)$.

The following simple property must surely be recorded somewhere in the literature, although we have not been able to find a statement of it. A simple proof is therefore provided:

Lemma 1. *Let* Γ_2 *be a conservative extension of* Γ_1 *and* Γ_3 *a conservative extension of* Γ_2. *Then* Γ_3 *is a conservative extension of* Γ_1.

Proof. For the left-to-right direction of the bi-conditional: by Definition 4 we can select three answer sets A_1 of Γ_1, A_2 of Γ_2 and A_3 of Γ_3 such that $A_2 = A_3 \cap lit(\Gamma_2)$ and $A_1 = A_2 \cap lit(\Gamma_1)$. Substituting $A_3 \cap lit(\Gamma_2)$ for A_2 it follows that $A_1 = A_3 \cap lit(\Gamma_2) \cap lit(\Gamma_1)$, which since $\Gamma_1 \subseteq \Gamma_2$ is true iff $A_1 = A_3 \cap lit(\Gamma_1)$.

The hard work of proving the conservativeness of the deontic overlay can be delegated to a special case of a theorem due to Lifschitz.[71] Here $naf(r)$ denotes the set of literals in r that are prepended by negation as failure:

Theorem 1. *Let* Γ *be a ground program consisting of rules of form (17), and let* D *be a consistent set of literals such that* $D \cap lit(\Gamma) = \emptyset$. *Let* Δ *be a ground program of the same form as* Γ *such that for every rule* $r \in \Delta$*:*

1. $head(r) \subseteq D$, *and*
2. $naf(r) \subseteq lit(\Gamma)$

Then $\Gamma \cup \Delta$ *is a conservative extension of* Γ.

The restriction to rules of form (17) can be relaxed to allow disjunctive heads in Γ, but not in Δ.

Theorem 2. *Let* Γ *be a be the ground instantiation of the encoding* $ASP(A, n)$ *and* Δ *the ground instantiation of the encoding* $ASP(D, n)$ *of a deontic specification* D. *If* $head(\Delta)$ *is consistent and* $head(\Delta_1) \cap lit(\Gamma) = \emptyset$, *then* $\Gamma \cup \Delta$ *is a conservative extension of* Γ.

Proof. It is necessary first to verify that all rules of $\Gamma \cup \Delta$ conform to the schema (17). This is immediate for rules of form (17), (21) and (23) in Γ and rules of form (40–44) in Δ. This leaves only rules of forms (22) and (24) which have strongly negated heads. It suffices to note that a strongly negated atom is a literal, so (22) and (24) are also of form (17).

Next, split the rules in Δ into the following sets:

- Δ_1 contains state norms (40), action norms (41) and the ggg-constraint.
- Δ_2 contains the two closure conditions (42, 43)

The set $head(\Delta_1)$ consists of atoms of form $\{\textit{status}(\textit{red}, m), \textit{trans}(\textit{red}, a, m)\}$ and is clearly consistent. Moreover, for $r \in \Delta_1$

1. $head(r) \subseteq head(\Delta_1)$ trivially since $r \in \Delta_1$, and
2. $naf(r) \subseteq lit(\Gamma)$ vacuously, since no $r \in \Delta_1$ contains *naf*-literals.

Suppose the condition $head(\Delta_1) \cap lit(\Gamma) = \emptyset$ of the theorem is true. Since it has already been established that all rules in $\Gamma \cup \Delta_1$ conform to the schema (17) it follows that Γ, Δ_1, and $head(\Delta_1)$ satisfy all the conditions of Theorem 1 (substituting $head(\Delta_1)$ for D and Δ_1 for Δ). Hence $\Gamma \cup \Delta_1$ is a conservative extension of Γ.

The set $head(\Delta_2)$ consists of atoms of form $\{\textit{status}(\textit{green}, m), \textit{trans}(\textit{green}, a, m)\}$ also clearly consistent. For every $r \in \Delta_2$

1. $head(r) \subseteq head(\Delta_2)$ trivially since $r \in \Delta_2$, and
2. $naf(r) \subseteq lit(\Gamma \cup \Delta_1)$ since the *naf*-literals in the rules (42–43) consists of atoms of form $\{\textit{trans}(\textit{red}, a, m), \textit{status}(\textit{red}, m)\}$ and all of these are in Δ_1.

Suppose the condition $head(\Delta_2) \cap lit(\Gamma \cup \Delta_1) = \emptyset$ of the theorem is true. Since each of the rules conform to the schema (17) it follows that $\Gamma \cup \Delta_1$, Δ_2, and $head(\Delta_2)$ satisfy all the conditions of Theorem 1 (substituting $\Gamma \cup \Delta_1$ for Γ, $head(\Delta_2)$ for D and Δ_2 for Δ) and hence that $\Gamma \cup \Delta_1 \cup \Delta_2$ is a conservative extension of $\Gamma \cup \Delta_1$.

Lemma 1 can now be applied to conclude that $\Gamma \cup \Delta_1 \cup \Delta_2$ is a conservative extension of Γ. Since $\Delta = \Delta_1 \cup \Delta_2$ this completes the proof.

The provisos in Theorem 1 must be managed carefully, and in particular is very sensitive to the form of rules. *A fortiori* this also goes for Theorem 2. If the causal theory Γ conforms to the format of schema (17) then the conservativeness of the deontic extension is guaranteed. The ASP encoding of $\mathcal{AL}$ action descriptions stays within this language fragment, but it is not obvious that the planning module, with its choice rule (25), does. However, there is a faithful translation of choice rules into normal non-disjunctive rules of form (17).[60] Hence, the deontic overlay remains conservative when the causal theory is used for planning, or in our case, for extracting plays of a training scenario.

As an example of a non-conservative extension, suppose one wishes to ensure that no states or transitions are labelled both green and red. This appears to be a sensible requirement, and one that is not implicit in the logical deontic norms. The logical norms do ensure that every state and transition is coloured; i.e., either red or green, but it does not preclude the possibility of 'chromatic superposition'. It is tempting to require this to be so by adding a rule to that effect, viz:

$$T1 \mathrel{!}= T2 \leftarrow \textit{status}(\textit{red}, T1), \textit{status}(\textit{green}, T2). \tag{47}$$

Recall that the restriction of Δ (the deontic overlay) to rules of form 17 in Theorem 1 is essential. Rule (47), however, is equivalent to a *constraint*:

$$\leftarrow \textit{status}(\textit{red}, T1), \textit{status}(\textit{green}, T2), T1 = T2. \tag{48}$$

Formula (48) is *not* of the form (17), and therefore violates the provisos of Theorem 1. It follows that it may interfere with a causal theory via its deontic extension. Consider for instance a causal theory

consisting of the two state constraints

$$\Gamma = \left\{ \begin{array}{ll} \mathit{holds}(f_2, 1) \leftarrow & \mathit{not}\ \mathit{holds}(f_1, 1). \\ \mathit{holds}(f_1, 1) \leftarrow & \mathit{not}\ \mathit{holds}(f_2, 1). \end{array} \right.$$

and the deontic specification

$$\Delta = \left\{ \begin{array}{ll} \mathit{status}(\mathit{red}, 1) \leftarrow & \mathit{holds}(f_2, 1). \\ \mathit{status}(\mathit{green}, 1) \leftarrow & \mathit{holds}(f_2, 1). \end{array} \right.$$

Admittedly, this deontic specification does not make much intuitive sense, but it is well-formed according to the syntactic restrictions of Theorem 1, and since the labelling of states and transitions are in general computed from arbitrarily complex conditions, one can easily imagine situations such as this arising from design errors. Now, the program $\Gamma \cup \Delta$ has two answer sets, viz. $A = \{\mathit{holds}(f_2, 1), \mathit{status}(\mathit{red}, 1), \mathit{status}(\mathit{green}, 1)\}$ and $A' = \{\mathit{holds}(f_1, 1)\}$, whereas the program $\Gamma \cup \Delta \cup \{47\}$ only has one, namely $\{\mathit{holds}(f_1, 1)\}$. Clearly $A' \cap \mathit{lit}(\Gamma \cup \Delta) = A'$ which since A' and A are different entails that the addition of rule (47) to $\Gamma \cup \Delta$ is non-conservative.

The moral of this story is that it is not possible to ensure programmatically and conservatively that a deontic overlay is well-defined, if well-definedness is taken to mean that a unique colour is assigned to every state and transition. True, models superimposing two colours onto the same node or transition will be eliminated by the constraint (47), but as these models may capture valid interpretations of the causal theory, that theory would then have been changed.

6 Scoring

Intuitively, a path in the transition diagram for a causal description of a simulation scenario corresponds to a play of that scenario. It is one possible sequence of events unfolding from the initial situation ensuant to the trainee's choices.

In this section we turn to the problem of developing a quantitative performance metric from the red/green labelling of states and actions. Behind this approach is the two-fold assumption that competency in means-ends reasoning is most naturally expressed in *qualitative* terms* whereas the concept of *overall performance* is most useful as a numeric indicator. The problem addressed in the present section is therefore that of translating a qualitative description of *skilful action* into a numeric measure of the quality of a particular *course of action*.

For any scoring function, certain demands should be placed on it to make it useful. We shall focus on the following two:

i) a scoring function should output interpretable values, and
ii) different outputs should be instructively comparable; that is, the relative performance of an agent in different plays of possibly different scenarios should be comparable.

*It should be stressed that this assumption is not true in general, a simple counterexample would be measuring time to completion.

Let $s(g, n)$ denote the stipulated scoring function where g is the number of green actions and states (henceforth the *green number*) and n is the length of a play, i.e. the number of actions (edges) in the corresponding path in a transition diagram. Moving towards a formal regimentation of requirement (i) it seems reasonable to require scores to increase with the green number—the more the trainee gets right the higher his performance should be rated. However, this measure should be tempered by the length of a play, since shorter plays are intuitively better than longer ones (as long as they are successful). The assumption here is that if a task can be completed to satisfaction quickly, then it is always better to do so.

The picture that emerges is of a scoring function that balances the green number against the length of a play to reach a verdict on the performance of an agent. To capture this formally, we require the scoring function s to satisfy the following formal conditions:

- Monotony in the green number:

$$\text{If } g_i \leq g_j \text{ then } s(g_i, n) \leq s(g_j, n) \tag{49}$$

- Antitony in the length of a play:

$$\text{If } n_i \leq n_j \text{ then } s(g, n_j) \leq s(g, n_i) \tag{50}$$

As regards point (ii), the demand that values of s be comparable can plausibly be taken to mean that they all fall on the same scale within a fixed numerical range. A candidate function meeting both (i) and (ii), as so understood, would be the ratio of *actual* green actions and states to the number of *possible* green actions and states in a play:

$$s(g, n) = \frac{g}{2n + 1}, \qquad g \leq 2n + 1 \tag{51}$$

This function fulfils the listed desiderata; the output grows with increasing values of g and shrinks with increasing values of n. Moreover, since g lies in the integer interval $[0, 2n + 1]$, values of s are rational numbers between 0 and 1.

6.1 Factoring in task complexity

Although it gets a number of things right, the function s has the rather serious limitation that it does not factor in the *complexity* of a play. In the limiting case where an agent only performs green actions leading to green states, the green number will be equal to $2n + 1$ and hence the score to 1, irrespective of n. In other words it does not matter how efficiently or inefficiently the agent completes a task. As long as he meets his objective and does not make a blunder he gets the highest score.

This feature can lead to anomalies. Consider a slight modification of the industrial kitchen scenario: suppose only states in which there is a fire are deemed unacceptable. In other words, we are supposing that a smoking or overheated pan is not in and of itself considered a problem. As long as a fire does not break out the agent is free from blame in Management's eyes. A matching modification of the kitchen theory can be made by removing the action norms (34) and (35) as well as the state norms (36) and (37) from the deontic specification. The colouring of the associated transition diagram changes to that of Fig. 4.

The most effective way to prevent a pan fire is still to turn off the stove immediately (by the conservativeness of the deontic specification, changing it does not change the causal theory). This is

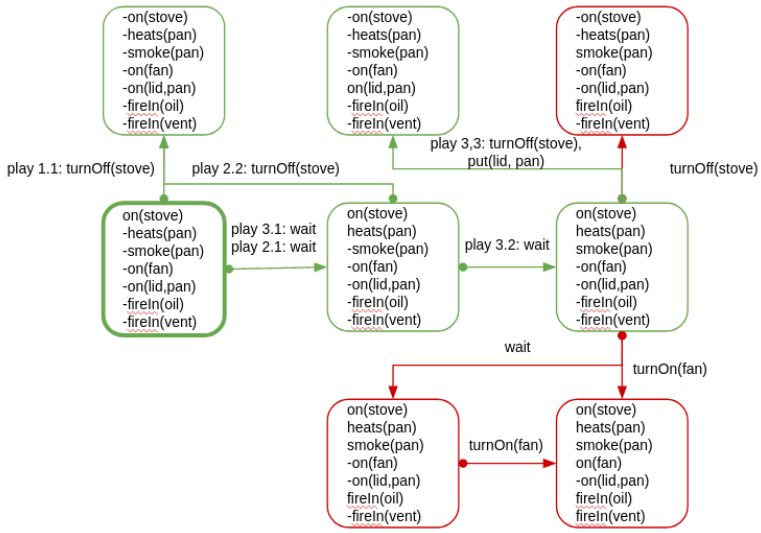


Figure 4. The kitchen scenario with only fires disallowed.

the play of length 1 with green number 3 indicated by the label 'play 1.1' (meaning play 1, action 1) in Fig. 4 which is scored at $s(3,1) = 1$. A less informed, or insouciant, agent might idle for one or more steps before he decides what to do. The longer he waits, the more precarious the situation becomes, so less waiting is intuitively better. Consider the plays labelled 2.1–2.2 and 3.1–3.3 which are both alternative courses of action to avert the pan fire, but ones in which the agent does not act expeditiously. In play 2.1–2.2 the agent turns off the stove in the second time-step. It suffices to defuse the hazard, but the wait is unnecessary. Nevertheless, it is intuitively better than the play 3.1–3.3 in which the agent idles for two time-steps before he places a lid on the pan and turns off the stove in one go. This is in the nick of time, according to the kitchen theory, since the fire could not have been prevented any later. Still, it *is* a successful play, as measured by the goal of averting fire, although the agent dawdles and performs two actions where one would have sufficed. The difference in quality is not visible in the score, unfortunately, since all plays have a green number of $2n + 1$ where n is its length. Thus, play 1.1 has a green number of 3 and length 1, play 2.1–2.2 has green number 5 and length 2 and play 3.1–3.3 has green number 7 and length 3. The scores are therefore

$$\begin{aligned} s(3,1) &= s(5,2) \\ &= s(7,3) \\ &= 1 \end{aligned}$$

In other words, the scoring function s is not able to distinguish between efficient and inefficient plays. The play in which an agent does nothing for as long as he can do so with impunity is rated equally to a simpler and more efficient solution.

This goes to show that the *inherent complexity* of the task at hand needs to be factored into the assessment of a play. In the kitchen scenario, fire can be prevented in one step by immediately turning off the stove. We shall say that its *task complexity* is 1. The reason why the play 1.1 is intuitively the optimal play is because it stays within this bound. Similarly 2.1–2.2 is intuitively a better play than 3.1–3.3 since the latter exceeds the task complexity with more steps than the former does. We need to work out the formal ramifications of this intuition.

What is needed for present purposes is a simple and operationalizable concept of minimum length of a causal chain necessary to complete a task successfully. For training, success can plausibly be taken to entail conformity to the standard of propriety expressed by the deontic specification. Hence task complexity, we propose, should be understood as the simplest way to complete the task in an *acceptable* manner. This concept comes close to what Wood calls *component complexity*,[72] which he construes as a direct function of the number of distinct acts that need to be executed in the performance of the task. There is a difference though; we require the acts in questions to be *acceptable means* of achieving the goal, that is, that they be green transitions:

Definition 5. *The complexity of a scenario is the shortest* green *path from the initial state to a goal state.*[*]

This simple concept can be factored into the score function (51) by multiplying the function s (the ratio of the actual green number to the possible green number) with the ratio of the task complexity to the actual length of a play. Letting c denote task complexity, this becomes

$$\begin{aligned} s^*(g,n,c) &:= s(g,n) \times \frac{c}{n} \\ &= \frac{g}{2n+1} \times \frac{c}{n} \\ &= \frac{gc}{2n^2+n} \end{aligned}$$

Function s^* is a generalization of s since $s^*(g,n,c) = s(g,n)$ whenever $c = n$. Since ASP does not have real numbers, only integers, it is a good idea to scale the result and round it. Adding the appropriate domain restrictions on the arguments, our definition becomes:

Definition 6.

$$s^*(g,n,c) =_{df} \lceil 100\frac{gc}{2n^2+n} \rceil, n \geq c, g \leq 2n+1, c \geq 1$$

The domain restrictions are summarized as follows:

- $n \geq c$ since the length of a play cannot be shorter than the task complexity,

[*]A path's being green does not entail that every state on it is green as well. It is entirely possible, according to the proposed definition of a deontic overlay, to have a play consisting entirely of red states with all transitions green. This is as it should be. Think of any action that reduces loss without eliminating it. Picture a fireman evacuating a burning building. Every person he rescues out of the inferno represents an acceptable action, obviously. Nevertheless as long as there are still people in the building, the current state of affairs is red all the same. This pattern of green transitions linking red states is typical of gradual recovery from sub-optimal situations. It is a large part of what training is about. Since the only constraint we have placed on the colouring of a transition diagram is the ggg-constraint, this possibility is left open.

- $g \leq 2n + 1$ since the green number is upper bounded by the length of a play, and
- $c \geq 1$ since no non-trivial task can be solved in no steps at all

Definition 6 presupposes the existence of a green path from the initial state to a goal. This is an example of a constraint one may consider adding to the deontic specification (it is not implied as it is). It can be verified by checking the satisfiability of the theory after adding the following constraint:

$$\leftarrow \mathit{not}\ \mathit{trans}(\mathit{green}, A, T), \mathit{action}(A), T < n. \tag{52}$$

Of course (52) does not extend a theory *conservatively* since *only* green paths will satisfy (52). In general, however, models of a deontically annotated causal theory include plays in which the agent makes mistakes and acts with imprudence. We do not wish to exclude these plays as they represent real strategies a trainee may pursue.

A practical solution to this is to run the answer set solver in planning mode twice, once with and once without the constraint (52). When run with (52), the solver returns models of green paths. The solver can be instructed to optimize over length and select the shortest of these. The task complexity of the scenario is then set to that length. When the solver is run without the constraint, on the other hand, it generates all possible plays. Having the task complexity available from the first step, one can use the function s^* to assign a score to each of these. This is how it is implemented in the ExManSim architecture.

Returning to the example that motivated the need for generalizing s to s^*, we note that Definition 6 eliminates the particular anomaly in that example. When task complexity is factored into the assessment, play 1.1 scores $s^*(3, 1, 1) = 100$, play 2.1–2.2 scores $s^*(5, 2, 1) = 50$ and play 3.1–3.3 scores $s^*(7, 3, 1) = 34$. This linear ranking of the three plays accords well with intuition. Of course, further benchmarking is necessary. Theorem 3 collects some properties of s^* for easy reference, all of them are straightforward to verify.

Theorem 3.

1. s^* *is monotone in its first argument,*
2. s^* *is monotone in its third argument,*
3. s^* *is antitone in its second argument,*
4. s^* *is antitone in the* difference *between the second and third argument, i.e.*

$$s^*(g, n', c') \leq s^*(g, n, c) \textit{ whenever } n - c \leq n' - c'$$

5. *the range of* s^* *is the integer interval* $[0, 100]$*,*
6. $s^*(g, n, n) = s(g, n)$*,*
7. $s^*(g, n, c) = 100$ *whenever* $g = 2n + 1$ *and* $n = c$
8. $s^*(g, n, c) = 0$ *whenever* $g = 0$

To analyze the behaviour of s^* one must examine projections onto lower dimensional spaces, which means fixing one or more of the function variables to a constant value. Setting task complexity to 5 gives the plot in Fig. 5, from which some of the properties of Theorem 3 can be confirmed by inspection: A green number of zero always gives score zero, the length of a play impacts scores negatively but can be balanced by green numbers, and so on. Fixing complexity and length gives the plot in Fig. 6a, which shows that the score is a linear function of the green number for the same values of length and complexity.

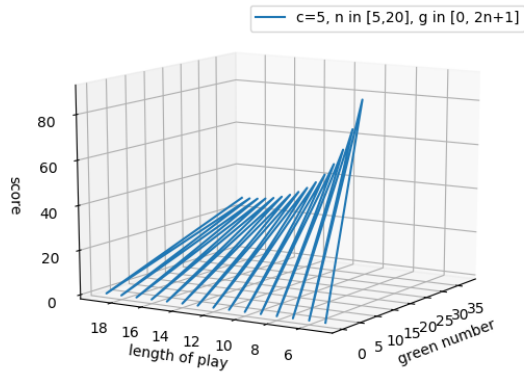
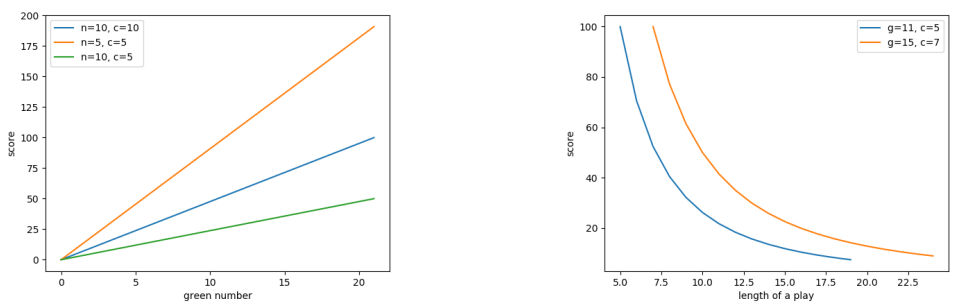


Figure 5. Plot of score s^* with constant complexity.

The slope of the line is influenced both by complexity and length. Fixing the green number of a play and complexity gives the curves in Fig. 6b. Values decrease logarithmically due to the quadratic denominator of s^*.



(a) Score s^* as a function of the green number

(b) Score s^* as the function of the length of a play

Figure 6. Behaviour of score s^*

By superimposing the function that takes green numbers as inputs (Fig. 6a) with the function that takes length as input (Fig. 6b), an example of the trade-off between length and green numbers can be read of from the intersection, cf. Fig. 7. The plot shows that $s^*(11, 6, 5) \geq s^*(7, 5, 5)$. Informally, a minimal-length play with about a 60/40 ratio of green to red transitions and states is dominated by a play of the same complexity in which the trainee performs one unnecessary action but raises the green to red ratio to approx. 85. In other words, a play that exceeds task complexity is not automatically inferior to a play that

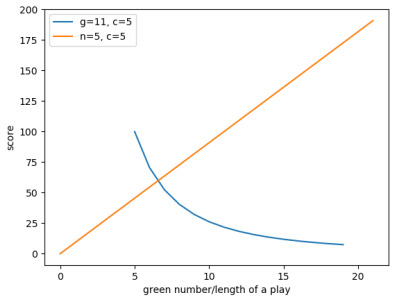


Figure 7. The green number vs. the length of a play

does not. It depends on how the goal is reached. If the minimal-length play includes many unacceptable means and intermediate states whereas the longer play largely avoids them, the green number of the latter may overtake the efficiency of the former as the main influence on the score.

7 Sample plays

In this section we compare a selection of plays and explain discrepancies in their score by reference to the causal laws and deontic norms. These examples serve to illustrate how the causal theory, the deontic specificaton, and the scoring function combine to form a single theory. The plays are shown in an abbreviated form as alternating sequences of actions and fluents: for each time-step the chosen action is displayed together with a selected fact that explains the colour of the state at that time instant. A green state or transition is indicated by the sign ✓ in the right margin and a red one with ✗. A goal state is marked 🐰. We consider two goal descriptions $\neg$*holds(fireIn(oil), T)* in Fig. 8 and $T = n$ in Fig. 9.

Starting with Fig. 8a, the first thing to note is that the initial state is green even though the supposition is that the stove is turned on too hot. This is due to the closure condition (42) and the absence of a deontic norm outlawing hot stoves. They combine to make the state at step 1 green *by default*. In this particular play the agent perceives the hazard immediately and turns off the stove. The *turnOff* action causes $\neg$*holds(on, 2)*, by causal law (1) and since the pan is not smoking, state constraint (9) is triggered, and state 2 becomes a goal state. This is an optimal play with a score of a 100.

In Fig. 8b the agent turns on the fan at step 1. This is not directly harmful, but by casual law (5) it is not an action that prevents the pan from overheating. The pan consequently overheats in step 2 colouring that state red. Since the initial state is green, the *ggg*-constraint kicks in and colours the act of turning on the fan red. This contrasts instructively with the act of turning off the stove in step 2 of the same play. Since the stove is already overheated at this moment, turning it off does not suffice to prevent smoke from developing at the next instant. Hence state 3 is red. Yet, since state 2 is also red, the *ggg*-constraint is not triggered, and the action at that step (turning off the stove) is coloured green by default. Now, by the axiom of inertia (23) the effect of turning off the stove in step 2 persists in state 3 where it combines with the action of putting the lid on the pan. Therefore causal law (5) causes state 4 to become a goal state thus ending the play. This play exceeds the task complexity of the kitchen scenario by two steps and has a red to green ratio of 3/4. It is scored at 23.

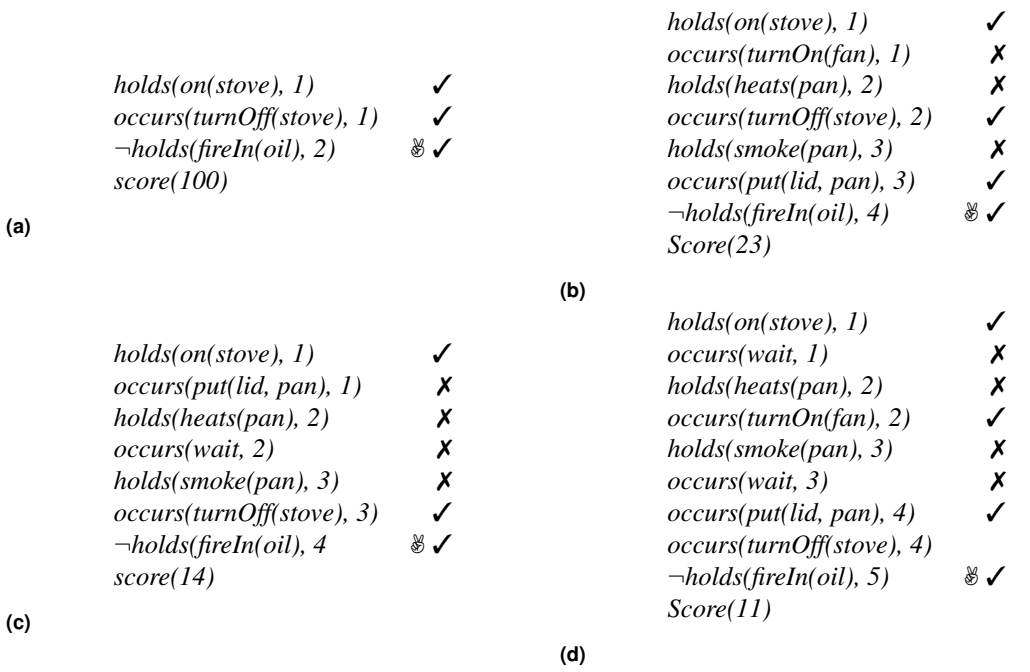


Figure 8. Sample plays with goal ¬*holds(fireIn(oil), T)*.

Step 1 of the play in Fig. 8c is similar to step 1 of Fig. 8b insofar as the agent chooses the wrong action. He puts the lid on the pan when he should have turned the stove off first. In step 2, he waits. Since the pan overheats at this point, waiting is expressly forbidden by the norm (34). Waiting also causes the pan to start smoking in state 3, at which point the agent turns off the stove just in time to prevent the fire. Compared to play (8a), the forbidden wait in step 2 is the negative difference that explains why play (8c) scores considerably lower at 14.

Turning now to the play in Fig. 8d, the agent procrastinates in step 1. Again, since this is not an action that prevents the stove from overheating, it is coloured red by the norm (34) (and would be anyway by the *ggg*-constraint). Next, the agent makes the rather uninformed choice of turning on the fan in step 2. However, there is no rule that expressly prohibits this; turning on the fan is acceptable even though it allows smoke to develop at the next step. Contrast this with the wait action which is covered by the norm (35). Intuitively, the meaning of (35) is to prohibit an action if it does contribute to suppressing the smoke from the pan. The lack of generality of norms (34–35) may therefore be considered a flaw in this particular deontic specification, and one may consider replacing them with

unacceptable ¬*turnOff(stove)* **if** *heats(pan)* (53)

unacceptable ¬*put(lid, pan)* **if** *smoke(pan)* (54)

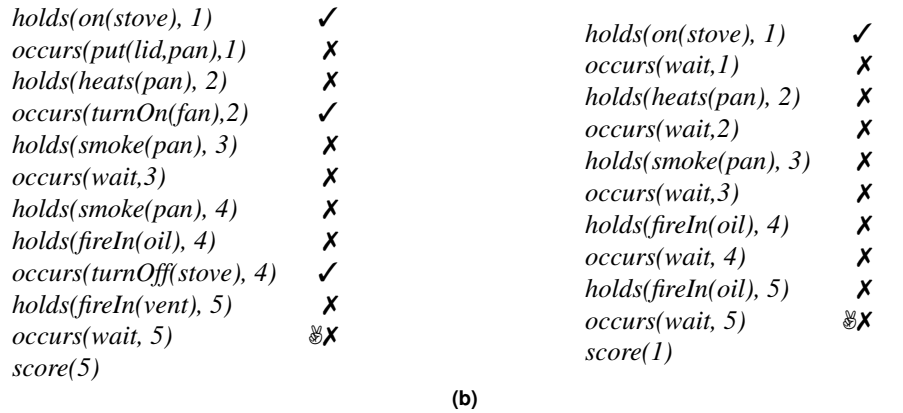


Figure 9. Sample plays with goal $T = n$.

Norms (53–54) follow the same representational strategy as employed in causal laws (5–6). It draws on the closed world assumption for actions expressed by rule (24) to conclude that the lid has not been put on the pan (resp. the stove turned off) if that action has not been explicitly recorded or deduced. Now, moving forward one time-step, the agent also chooses to do nothing in step 3, which forces him to put on the lid and turn off the stove in one go in step 4. This is the very last time-point at which the fire can be prevented, which is why the play scores very low at 11.

Now, changing the goal from $\neg$*holds(fireIn(oil), T)* to $T = n$ means that we are allowing the agent to fail. The new goal is simply to extract sequences of actions that are as long as the planning horizon, irrespective of the outcome. This way, poor plays in which a fire does break out, in the pan or in the vent shaft etc., come into scope for the scoring function. Fig. 9 shows two such plays.

The play in Fig. 9a displays features that we have already discussed; the action in the first time-step is the wrong one, so state 2 becomes red, which in turns entails that the action in question is coloured red by the ggg-constraint. Turning on the fan in step 2 is deemed acceptable, even though it allows smoke to develop—another illustration that the norm (35) should probably be generalized. The agent woefully does nothing in step 3, closing his window of opportunity to prevent fire in the cooking oil which consequently breaks out in state 4. The agent then turns off the stove. Although that is an acceptable action at this point, it is too late to prevent the fire. It also does not prevent the fire from spreading to the vent shaft in step 5, which it does because the fan is on. Needless, to say this is a poor play. It gets a score of 5.

An even worse performance is the play in Fig. 9b. Indeed this is as bad as it gets according to this particular deontic specification. The agent does nothing at all, allowing the pan fire to break out and rage on. One thing to note though, is that this scenario scores less than the play 9a although the outcome is less severe. More specifically, the fire does not spread to the vent shaft since the fan is not on. This suggests that a binary scoring scheme where states and actions are either good or bad may be a bit too simplistic. An interesting generalization would be to subdivide the class of unacceptable actions into those that do not have positive consequences (such as waiting) and those that may have negative consequences (such

as turning on the fan). These actions could then be scored differently. We leave this topic of degrees of goodness and badness for future research.

8 Discussion and Final Remarks

The scoring scheme we have proposed is most aptly viewed as expressing a concept of instrumental rationality, or more precisely the efficacy of an agent to achieve the desired ends given the means provided. A next step would be to develop scoring functions for other skills. As an example, consider the skill of situation awareness.[73] A trainee can exhibit various degrees of effective means-ends reasoning in a scenario, but one would not know if this was due to excellent situation awareness, knowledge of routines, other skills or pure good fortune. The type of scoring function presented above may be extendable to situation awareness, either by carefully designing tasks that eliminate routine and familiarity effects, or by randomizing the selection and location of key objects in the scene. This is a topic we are actively researching at the time of writing. An expected gain is to have some way of assessing situation awareness, that unlike other techniques like e.g. the SAGAT method,[73] does not require plays to be suspended whilst trainees are queried.

To make our work of use to exercise managers and domain experts, we are also developing graphical tools for scenario design.[74] Specifically, we are investigating block-based programming* for selecting ontology-defined objects and actions and composing scenarios from them. The Blockly design space is combined with a sunburst diagram† that displays all possible plays of a scenario, including colours and scores for each play. A scenario designer can thus assess whether a particular scenario design has the appropriate complexity and the relevant events to train the desired skills, and redesign if needed. Such a design space will present a major improvement on current practices. Note also that the designer will be oblivious to all the ASP computing going on behind the scene.

The framework as described above, has a binary scoring scheme: states and transitions are either good or bad. This scheme is easily extendable to ordinal or interval scoring scales, to reflect further degrees of goodness or badness.

We expect the development in this article to have a significantly positive impact on planning and analyzing simulation-based training. Once the framework has been implemented in an exercise design and analysis tool, empirical validation of usability can commence. Our approach is instrumental in acquiring structured training data. When large amounts of such data have been secured, machine learning techniques can be employed on that data to give input to scenario design as described in this article. In this way, both machine reasoning and machine learning may work together for better design and analysis of simulation-based training. However, the main contribution of the present article is its theoretical elaboration of AI planning for simulation-based training and the resulting stringent definition of the scenario concept, including plays, goals and scoring.

Funding

This research was by the Research Council of Norway under project no. 282081 “MixStrEx”.

*https://developers.google.com/blockly
†https://datavizproject.com/data-type/sunburst-diagram/

References

1. Salas E, Wildman JL and Piccolo RF. Using simulation-based training to enhance management education. *Academy of Management Learning & Education* 2009; 8(4): 559–573.
2. Ericsson KA. The influence of experience and deliberate practice on the development of superior expert performance. In Ericsson KA, Charness N, Feltovich PJ et al. (eds.) *The Cambridge Handbook of Expertise and Expert Performance*, chapter 38. Cambridge Univ. Press, 2006. pp. 683–703.
3. Beaubien JM, Shadrick SB, Paley MJ et al. Using deliberate practice to train military-civilian interagency coordination. In *Proc. Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2006*. National Training and Simulation Association, pp. 151–158.
4. Grunnan T and Fridheim H. Planning and conducting crisis management exercises for decision-making: the do's and don'ts. *EURO Journal on Decision Processes* 2017; 5: 79–95.
5. Hannay JE and Kikke Y. Structured crisis training with mixed reality simulations. In *Proc. 16th Int'l Conf. Information Systems for Crisis Response and Management (ISCRAM)*. pp. 1310–1319.
6. Simulation Interoperability Standards Organization, Orlando. *Guideline on Scenario Development for Simulation Environments*, 2016.
7. Lifschitz V. *Answer set programming*. Springer Heidelberg, 2019.
8. Gelfond M and Kahl Y. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press, 2014.
9. Baral C. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
10. Russell S and Norvig P. *Artificial Intelligence: A Modern Approach*. Third ed. Series in Artificial Intelligence, Prentice Hall, 2010.
11. Sergot M. (C+)++ : An action language for modelling norms and institutions. Technical report, Imperial College, 2005.
12. Gelfond M and Lifschitz V. Action languages. *Electronic Transactions on Artificial Intelligence* 1998; 3: 195–210.
13. Rouwendal van Schijndel DK, Hannay JE and Stolpe A. Simulation vignette generation from Answer Set specifications. In *Proc. 17th Int'l Conf. Information Systems for Crisis Response and Management (ISCRAM)*. pp. 110–121.
14. Rouwendal DK, Stolpe A and Hannay JE. Toward an AI-based external scenario event controller for crisis response simulations. In *Proc. 18th Int'l Conf. Information Systems for Crisis Response and Management (ISCRAM)*. pp. 106–117.
15. Ericsson KA. An introduction to Cambridge Handbook of Expertise and Expert Performance: Its development, organization, and content. In Ericsson K, Charness N, Feltovich P et al. (eds.) *The Cambridge Handbook of Expertise and Expert Performance*, chapter 1. Cambridge Univ. Press, 2006. pp. 3–20.
16. Shadish WR, Cook TD and Campbell DT. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Houghton Mifflin, 2002.
17. Cronbach LJ and Meehl PE. Construct validity in psychological tests. *Psychological Bulletin* 1955; 52(4): 281–302.
18. Schraagen JM. Designing training for professionals based on subject matter experts and cognitive task analysis. In Ericsson KA (ed.) *Development of Professional Expertise*, chapter 7. Cambridge University Press, 2009. pp. 157–179.

19. Hærem T and Rau D. The influence of degree of expertise and objective task complexity on perceived task complexity and performance. *Journal of Applied Psychology* 2007; 92(5): 1320–1331.
20. Shanteau J. Competence in experts: The role of task characteristics. *Organizational Behavior and Human Decision Processes* 1992; 53: 252–266.
21. Chi MTH, Glaser R and Rees E. Expertise in problem solving. In Sternberg R (ed.) *Advances in the Psychology of Human Intelligence*. Lawrence Erlbaum Associates, Inc., 1982. pp. 17–76.
22. Ford DN and Sterman JD. Expert knowledge elicitation to improve formal and mental models. *System Dynamics Review* 1998; 14(4): 509–340.
23. Sternberg RJ. Cognitive conceptions of expertise. *International Journal of Expert Systems* 1994; 7(1): 1–12.
24. Bereiter C and Scardamalia M. *Surpassing Ourselves: An Inquiry into the Nature and Implications of Expertise*. Open Court, 1993.
25. Johnson EJ. Expertise and decision under uncertainty: Performance and process. In Chi MTH, Glaser R and Farr MJ (eds.) *The Nature of Expertise*. Lawrence Erlbaum Associates, Inc., 1988. pp. 209–228.
26. Papineau D. The evolution of means-end reasoning. *Royal Institute of Philosophy Supplements* 2001; 49: 145–178.
27. Tversky A and Kahneman D. Judgement under uncertainty: Heuristics and biases. *Science* 1974; 185(27): 1124–1131.
28. Hammond KR, Brehmer TR and Steinmann DO. Social judgement theory. *Human Judgment and Decision Processes* 1975; : 271–312.
29. Hogarth R. Beyond discrete biases: Functional and dysfunctional aspects of judgmental heuristics. *Psychological Bulletin* 1981; 90(2): 197–217.
30. Gigerenzer G and Todd PM (eds.) *Simple Heuristics that Make Us Smart*. Oxford University Press, 1999.
31. Simon HA. *The Sciences of the Artificial*. Third ed. MIT Press, 1996.
32. IEEE Standards Association. *1516-2010 – IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)*, 2010.
33. Simulation Interoperability Standards Organization. *SISO-STD-003-2006 – Standard for Base Object Model (BOM) Template Specification*, 2006.
34. Gruber TR. A translation approach to portable ontology specifications. *Knowledge Acquisition* 1993; 5(2): 199–220.
35. Mojtahed V, Andersson B, Kabilan V et al. BOM++, a semantically enriched BOM. In *Proc. 2008 Spring Simulation Interoperability Workshop (SIW)*. Simulation Interoperability Standards Organization, pp. 315–327.
36. De Nicola A, Melchiori M and Villani ML. Creative design of emergency management scenarios driven by semantics: An application to smart cities. *Information Systems* 2019; 81: 21–48.
37. Steel J, Iannella R and Lam HP. Using ontologies for decision support in resource messaging. In *Proc. 5th Int'l Conf. Information Systems for Crisis Response and Management (ISCRAM)*. pp. 189–196.
38. Bénaben F, Hanachi C, Lauras M et al. A metamodel and its ontology to guide crisis characterization and its collaborative management. In *Proc. 5th Int'l Conf. Information Systems for Crisis Response and Management (ISCRAM)*. pp. 189–196.
39. Durak U, Oğuztüzün H, Köksal Algin C et al. Towards interoperable and composable trajectory simulations: an ontology-based approach. *Journal of Simulations* 2011; 5(3): 217–229.
40. Singapogu SS, Gupton K and Schade U. The role of ontology in C2SIM. In *Proc. 21st International Command and Control Research and Technology Symposium (ICCRTS 2016)*. The International Command and Control Institute, pp. 1–21.

41. Simulation Interoperability Standards Organization. *The Command and Control Systems – Simulation Systems Interoperation (C2SIM) Product Development Group (PDG) and Product Support Group (PSG)*, 2019. Accessed December 19, 2019.
42. van den Berg TW, Huiskamp W, Siegfried R et al. Modelling and Simulation as a Service: Rapid deployment of interoperable and credible simulation environments – an overview of NATO MSG-136. In *Proc. 2018 Winter Simulation Innovation Workshop*. pp. 149–166.
43. Hannay JE, van den Berg T, Gallant S et al. Modeling and Simulation as a Service infrastructure capabilities for discovery, composition and execution of simulation services. *J Defense Modeling and Simulation: Applications, Methodology, Technology* 2020; 14(2): 139–158.
44. Hannay JE and van den Berg TW. The NATO MSG-136 Reference Architecture for M&S as a Service. In *Proc. NATO Modelling and Simulation Group Symp. on M&S Technologies and Standards for Enabling Alliance Interoperability and Pervasive M&S Applications (STO-MP-MSG-149)*. NATO Science and Technology Organization, pp. 1–18.
45. Sergot M and Craven R. The deontic component of action language nc+. In *Proceedings of the 8th International Conference on Deontic Logic and Artificial Normative Systems*. DEON'06, Berlin, Heidelberg: Springer-Verlag, p. 222–237.
46. Giunchiglia E, Lee J, Lifschitz V et al. Nonmonotonic causal theories. *Artificial Intelligence* 2004; 153(1–2): 49–104.
47. Lee J, Lifschitz V and Yang F. Action language bc: preliminary report. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*. pp. 983–989.
48. Gelfond M and Lifschitz V. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 1991; 9: 365–385.
49. Brewka G, Delgrande J, Romero J et al. asprin: Customizing answer set preferences without a headache. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*. pp. 1467–1474.
50. Babb J and Lee J. Cplus2asp: Computing action language $\mathcal{C}$+ in answer set programming. In Cabalar P and Son TC (eds.) *Logic Programming and Nonmonotonic Reasoning*. Springer Berlin Heidelberg, pp. 122–134.
51. Zeigler BP, Praehofer H and Kim TG. *Theory of Modeling and Simulation*. 2nd ed. Academic Press, 2000.
52. Foo N and Peppas P. System properties of action theories. In Kim TG (ed.) *Artificial Intelligence and Simulation*. Springer Berlin Heidelberg, pp. 416–427.
53. Wainer GA, Goldstein R and Khan A. Introduction to the discrete event system specification formalism and its application for modeling and simulating cyber-physical systems. In *2018 Winter Simulation Conference (WSC)*. pp. 177–191.
54. Foo N and Peppas P. Systems theory: Melding the ai and simulation perspectives. In Kim TG (ed.) *Artificial Intelligence and Simulation*. Springer Berlin Heidelberg, pp. 14–23.
55. McCarthy JW. Situations, actions, and causal laws. In *Semantic Information Processing*. MIT Press, p. 410–41.
56. Kowalski R and Sergot M. A logic-based calculus of events. In Schmidt JW and Thanos C (eds.) *Foundations of Knowledge Base Management: Contributions from Logic, Databases, and Artificial Intelligence Applications*. Springer Berlin Heidelberg, pp. 23–55.
57. Gelfond M and Lifschitz V. The stable model semantics for logic programming. In Kowalski R, Bowen and Kenneth (eds.) *Proceedings of International Logic Programming Conference and Symposium*. MIT Press, pp. 1070–1080.
58. Baral C and Gelfond M. Reasoning agents in dynamic domains. In *Logic-based artificial intelligence*. Springer, 2000. pp. 257–279.

59. Gelfond M. Answer sets. *Foundations of Artificial Intelligence* 2008; 3: 285–316.
60. Gebser M, Kaminski R, Kaufmann B et al. *Answer Set Solving in Practice*. Morgan & Claypool Publishers, 2012.
61. Fikes RE and Nilsson NJ. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 1971; 2(3): 189 – 208.
62. McCarthy J and Hayes PJ. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer B and Michie D (eds.) *Machine Intelligence 4*. Edinburgh University Press, 1969. pp. 463–502.
63. Siegfried R, Laux A, Rother M et al. Scenarios in military (distributed) simulation environments. In *Proc. 2012 Spring Simulation Interoperability Workshop*. 12S-SIW-014, pp. 1–12.
64. Gebser M, Leone N, Maratea M et al. Evaluation techniques and systems for answer set programming: a survey. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, pp. 5450–5456.
65. Gebser M, Jost H, Kaminski R et al. Ricochet robots: A transverse asp benchmark. In Cabalar P and Son TC (eds.) *Logic Programming and Nonmonotonic Reasoning*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 348–360.
66. Gelfond M and Lobo J. Authorization and obligation policies in dynamic systems. In Garcia de la Banda M and Pontelli E (eds.) *Logic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-89982-2, pp. 22–36.
67. Stolpe A. A theory of permission based on the notion of derogation. *Journal of Applied Logic* 2010; 8(1): 97–113.
68. Stolpe A. Relevance, derogation and permission: A case for a normal form for codes of norms. In *Proceedings of the 10th International Conference on Deontic Logic in Computer Science*. DEON'10, Berlin, Heidelberg: Springer-Verlag, p. 98–115.
69. Carmo J and Jones AJI. Deontic database constraints, violation and recovery. *Studia Logica: An International Journal for Symbolic Logic* 1996; 57(1): 139–165.
70. Craven R and Sergot M. Agent strands in the action language nC+. *Journal of Applied Logic* 2008; 6(2): 172 – 191. Selected papers from the 8th Int'l Workshop on Deontic Logic in Computer Science.
71. Lifschitz V and Turner H. Splitting a logic program. In Hentenryck V and Pascal (eds.) *Proceedings of International Conference on Logic Programming (ICLP)*. pp. 23–37.
72. Wood RE. Task complexity: Definition of the construct. *Behaviour and Human Decision Processes* 1986; 37: 60–82.
73. Endsley MR. Theoretical underpinnings of situation awareness: A critical review. In Endsley MR and Garland DJ (eds.) *Situation awareness analysis and measurement*. Lawrence Erlbaum Associates Publishers, 2000. pp. 13–32.
74. Rouwendal van Schijndel DK, Stolpe A and Hannay JE. Using block-based programming and sunburst branching to plan and generate crisis training simulations. In Stephanidis C and Antona M (eds.) *Proc. 22nd International Conference on Human Comupter Interaction (HCI International 2020)*, *Communications in Computer and Information Science*, volume 1226. Springer International Publishing, pp. 463–471.