Default Reasoning about Actions via Abstract Argumentation

Ringo BAUMANN, Hannes STRASS
Computer Science Institute, University of Leipzig, Germany
{baumann, strass}@informatik.uni-leipzig.de

Abstract. Reasoning about actions is a subfield of artificial intelligence that is concerned with representing and reasoning about dynamic domains. We propose to employ abstract argumentation for this purpose. Specifically, we present a translation of action domains from a specification language into Dung-style argumentation frameworks (AFs). As the key advantage of our approach, we use existing semantics for argumentation to make predictions about the domain in various manners and utilise existing results about argumentation to show that the approach can be efficiently implemented. This demonstrates the practical value not only of its theoretical results, but also abstract argumentation itself.

Keywords. Reasoning about actions, default reasoning, argumentation

1. Introduction

In the past decade, approaches to abstract argumentation, in particular Dung-style AFs [5], have received considerable attention from the artificial intelligence community. More or less two directions in this research area were investigated exhaustively: first, exploration of theoretical properties of argumentation semantics as well as advancements of them which satisfy certain desired features; and second, extending and/or revising former argumentation systems to handle preferences or values for example. Alas, actual applications for abstract argumentation regarding real-world problems are very rare (see [12] for an excellent summary).

In this paper, we take a first step into this direction and propose to employ Dung-style AFs for reasoning about actions. The ability to make predictions about how a dynamically changing domain will evolve is of vital importance to intelligent systems, be they embodied or software-based. We show that argumentation can be elegantly used to fulfil this task. Our work rests on various theoretical results about argumentation that have been obtained in the last years and thus shows that these results are not only of academic interest but also practical value.

Specifically our contributions in this paper are as follows. We present the – to the best of our knowledge – first approach using abstract argumentation to reason about actions that is independent of a specific time structure and can also do default reasoning in dynamic domains.[1] We generalise the concept of *weak expansions* [2] to the infinite case, leading to a concept of stratification for argumentation frameworks that is much like the one for logic programs. Stratification enables us to demonstrate that our approach is efficiently implementable. Furthermore, we show how different argumentation semantics can be used for different types of reasoning (well-founded v. hypothetical) and knowledge (complete v. incomplete) – notably with *one and the same encoding.* That way, we can emulate conceptually different action languages within a single uniform framework.

[1]Michael and Kakas [?] proposed an argumentation-based approach to combine reasoning about actions with default reasoning. They however use a specific form of *assumption-based* argumentation, while in this paper we use standard *abstract* argumentation.

The paper is organised as follows. In the next section, we provide the necessary formal background on argumentation and a gentle introduction to action theories. The section thereafter formalises how we specify dynamic domains and translate them into abstract argumentation frameworks. Finally, we prove several beneficial properties of the resulting frameworks, discuss related work and conclude.

2. Background

2.1. Argumentation Theory

An *argumentation framework* $\mathcal{F}$ is a pair (A, R), where A is a non-empty (possibly infinite) set whose elements are called *arguments* and $R \subseteq A \times A$ a binary relation, called the *attack relation.* If $(a, b) \in R$ holds we say that a *attacks* b, or b is *defeated* by a in $\mathcal{F}$. We will slightly abuse notations, and write $(A, b) \in R$ for $\exists a \in A : (a, b) \in R$; likewise we use $(b, A) \in R$ and $(A, A') \in R$. An AF is called *finitary* if all arguments have a finite number of defeaters. An argument $a \in A$ is *defended* by a set $A' \subseteq A$ in $\mathcal{F}$ if for each $b \in A$ with $(b, a) \in R$, $(A', b) \in R$. Furthermore, we say that a set $A' \subseteq A$ is *conflict-free* in $\mathcal{F}$ if there are no arguments $a, b \in A'$ such that a attacks b. The set of all conflict-free sets of an AF $\mathcal{F}$ is denoted by $cf(\mathcal{F})$. For an AF $\mathcal{F} = (B, S)$ we use $A(\mathcal{F})$ to refer to B and $R(\mathcal{F})$ to refer to S.

Extension-based semantics Semantics determine acceptable sets of arguments for a given AF $\mathcal{F}$, so-called *extensions.* The set of all extensions of $\mathcal{F}$ under semantics σ is denoted by $\mathcal{E}_\sigma(\mathcal{F})$. For two semantics σ, τ we use $\sigma \subseteq \tau$ to indicate that for any $\mathcal{F} \in \mathscr{A}$, $\mathcal{E}_\sigma(\mathcal{F}) \subseteq \mathcal{E}_\tau(\mathcal{F})$. We consider here $\sigma \in \{st, ad, pr, co, gr\}$ for stable, admissible, preferred, complete and grounded semantics [5].

Definition 1. Given an AF $\mathcal{F} = (A, R)$ and $E \subseteq A$. Then,

1. $E \in \mathcal{E}_{st}(\mathcal{F})$ iff $E \in cf(\mathcal{F})$ and for each $a \in A \backslash E$, $(E, a) \in R$,
2. $E \in \mathcal{E}_{ad}(\mathcal{F})$ iff $E \in cf(\mathcal{F})$ and each $e \in E$ is defended by E in $\mathcal{F}$,
3. $E \in \mathcal{E}_{pr}(\mathcal{F})$ iff $E \in \mathcal{E}_{ad}(\mathcal{F})$ and for no $E' \in \mathcal{E}_{ad}(\mathcal{F})$, $E \subsetneq E'$,
4. $E \in \mathcal{E}_{co}(\mathcal{F})$ iff $E \in \mathcal{E}_{ad}(\mathcal{F})$ and for each $a \in A$ defended by E in $\mathcal{A}$, $a \in E$,
5. $E \in \mathcal{E}_{gr}(\mathcal{F})$ iff $E \in \mathcal{E}_{co}(\mathcal{F})$ and for no $E' \in \mathcal{E}_{co}(\mathcal{F})$, $E' \subsetneq E$.

There are several relations between these semantics: $st \subseteq pr \subseteq co$, $gr \subseteq co$ and the unique grounded extension is contained in every complete extension. Furthermore, each AF has a grounded and at least one preferred and complete extension. Given a finitary AF $\mathcal{F}$ the grounded extension of $\mathcal{F}$ can be obtained by iteratively applying its *characteristic function* $C_\mathcal{F}$ on the empty set. For any set $S \subseteq A(\mathcal{F})$, $C_\mathcal{F}(S) = \{a \mid a \text{ is defended by } S \text{ in } \mathcal{F}\}$, thus $C_\mathcal{F}$ is monotonic.

Weak expansions and chains We recapitulate several definitions firstly introduced by [2]. The lemma at the end of this section will play a crucial role for the evaluation of a given action domain. For short, weak expansions add new arguments and possibly new attacks, but no attacks between previous arguments; additionally, the added arguments never attack former ones.

Definition 2. An AF $\mathcal{F}^*$ is a *weak expansion* of AF $\mathcal{F} = (A, R)$ – written as $\mathcal{F} \prec_W^N \mathcal{F}^*$ – iff $\mathcal{F}^*$ has a representation as $(A \cup A^*, R \cup R^*)$, such that $A^* \neq \emptyset$, $A^* \cap A = R^* \cap R = \emptyset$ and $\forall a, b \in A\, ((a, b) \in R^* \rightarrow \neg(a \in A^* \wedge b \in A))$.

Definition 3. Let $C = \langle \mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_n \rangle$ be a sequence of AFs and $\mathcal{F}$ an AF. C is a *weak expansion chain of* $\mathcal{F}$ iff $\mathcal{F} = \mathcal{F}_n$ and $\mathcal{F}_i \prec_W^N \mathcal{F}_{i+1}$ for all $1 \leq i \leq n-1$.

The following lemma states that under certain conditions detecting whether an argument is accepted can be considerably simplified.

Lemma 1 (Corollary 7,[2]). *Let* $\langle \mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_n \rangle$ *be a weak expansion chain of* $\mathcal{F}$ *and let* i *the smallest index with* $\mathsf{a} \in A(\mathcal{F}_i)$*. For any semantics satisfying directionality,* a *is in some/all extensions of* $\mathcal{F}$ *iff* a *is in some/all extensions of* $\mathcal{F}_i$.

2.2. Action Theories

Action theories are used to represent agents' knowledge about dynamic domains. Envisioned by pioneer John McCarthy as early as 1959 [8], one of their tasks is to predict how the world will evolve over time. Such predictions are highly relevant in artificial intelligence, for example to enable an agent to plan ahead the course of actions suitable to meet its goals. A major representational and inferential problem any action theory must solve is the so-called *frame problem* of specifying the world properties that do *not* change when an action is performed [?]. Having been discovered in 1969, it took until 1991 that the frame problem was solved in a generally accepted way [13]. Today, the field of reasoning about actions and change has brought forth quite a number of logic-based formalisms, each with their own solution to the frame problem [14].

Empowered by these advances in classical action theories, researchers began to recognise that agents typically have only incomplete knowledge about their environment, and started to address this issue. Mueller [10] described a general method for default reasoning about (linear) time based on the circumscription of abnormality predicates. Kakas et al. [?] sketched an integration of temporal and default reasoning and in subsequent works developed an argumentation-based semantics for the approach using linear time [?,9]. Lakemeyer and Levesque [?] gave a definition of progression [13] in the presence of state defaults for a modal fragment of the Situation Calculus that uses branching time. Baumann et al. [3] approached the state default problem of inferring what usually holds in an abstract formalism that is independent of a particular time structure [14].

In parallel to the development of the highly expressive, logic-based formalisms for reasoning about changing worlds, researchers have proposed so-called *action languages* [6] for describing domains. They are simpler, much closer to natural language and usually have a semantics based on state transition systems. The approach presented in this paper is in effect also based on this paradigm since we define an action language with argumentation-based semantics.

3. Using Argumentation Frameworks to Reason about Actions

This section describes our approach for reasoning about actions and change via abstract argumentation. Roughly, the approach works as follows. The user specifies an action domain in an action language that we define next. Our definitions construct an argumentation framework from a description of a domain in this action language. The obtained argumentation framework can be used to answer queries about the domain using various semantics.

3.1. Specifying Action Domains

The vocabulary for speaking about dynamic domains consists of three components. First fluents, properties that may change over time. Second actions (also called events), that happen and initiate those changes. Third a time structure, that specifies the time points at which we are interested in the state of the world and how these time points relate to each other. The first two can be viewed as constant symbols, the third element is given by a directed tree whose nodes are time points and whose edges induce a reachability relation among the time points.

Definition 4. A *domain vocabulary* is a tuple $(\mathbf{F}, \mathbf{A}, \mathbf{T}, \prec)$, where

- $\mathbf{F}$ is a set of fluents. A *fluent literal* is of the form f or $\neg f$ for some $f \in \mathbf{F}$. Define $\overline{f} \stackrel{\text{def}}{=} \neg f$ and $\overline{\neg f} \stackrel{\text{def}}{=} f$, and for a set L of fluent literals set $\overline{L} \stackrel{\text{def}}{=} \{\overline{l} \mid l \in L\}$. The set of all fluent literals is then $\mathbf{F}^{\pm} \stackrel{\text{def}}{=} \mathbf{F} \cup \overline{\mathbf{F}}$.
- $\mathbf{A}$ is a set of actions.
- The pair $(\mathbf{T}, \prec)$ is a *time structure*, a directed graph with the properties
  - $*$ $\mathbf{T}$ is a countable set of time points, each with a finite degree,
  - $*$ $\mathbf{0} \in \mathbf{T}$ is the root of the tree (called the *least time point*),
  - $*$ for any $t \in \mathbf{T}$ there is a unique finite, directed path from $\mathbf{0}$ to t.

The intuition behind trees as time structures is that edges lead to direct successor time points and the direction of the edges express the flow of time. Hence our time structures may be branching into the future, but they are always linear with respect to the past. This is a very abstract view of time that can accommodate different notions of time that are used in the literature: For example, the pair $(\mathbb{N}, \{(n, n+1) \mid n \in \mathbb{N}\})$ of the natural numbers with the usual successor relation defines a discrete linear time structure [10]. The second major time structure used in the reasoning about actions community, the branching time of *situations* [13], can be modelled using terms. There, a special constant $\mathbf{0}$ denotes the initial situation and new situations are inductively defined from actions a and given situations s by the terms $do(a, s)$ denoting the result of executing a at s.

The state of the world at a time point of any time structure is described by providing the truth values of the relevant aspects of the environment represented by fluents. Since some world aspects might be unknown, we allow the representation of incomplete knowledge about a time point.

Actions are formalised by stating their *action preconditions* – world properties that must hold in order for the action to be executable – and direct effects, that express how actions change the state of the world. Since world states are modelled using fluents, the changes initiated by actions are modelled by fluent literals that become true whenever certain fluent literals – the *effect preconditions* – hold. Finally, to formalise how the world normally behaves we use *defaults* – which say that a fluent literal normally holds whenever all literals in a set of *prerequisites* hold. Several action languages offer additional expressiveness, for example indirect action effects. We however want to keep it simple here since our main goal is to show how abstract argumentation can be used to reason about dynamic domains.

A specification of an action domain in our action language consists of two parts: the first part contains such general knowledge about the domain – action preconditions, action effects, state defaults –, the second part contains information about what holds and happens at various time points of a specific domain instance. We begin with how to express knowledge about the general workings of a domain.

Definition 5. Consider a fixed domain vocabulary $(\mathbf{F}, \mathbf{A}, \mathbf{T}, \prec)$, and let $a \in \mathbf{A}$ be an action, $l \in \mathbf{F}^{\pm}$ be a fluent literal and $C \subseteq \mathbf{F}^{\pm}$ be a finite set of fluent literals. A *statement* can be:

- a *precondition statement*: $\underline{\texttt{possible}}$ a $\underline{\texttt{if}}$ C
- a *direct effect statement*: $\underline{\texttt{action}}$ a $\underline{\texttt{causes}}$ l $\underline{\texttt{if}}$ C
- a *default statement*: $\underline{\texttt{normally}}$ l $\underline{\texttt{if}}$ C

In statements of the above form, we will refer to literal l as the *consequent*. If $C = \emptyset$ for a statement, we omit the $\underline{\texttt{if}}$ part in writing. For illustration, we use the following running example throughout the paper.

Example 1 (Machine supervision)**.** In this simplified domain, the agent's task is to supervise the operation of a machine. If the temperature of the machine becomes too high, it has to be shut down. Normally, however, the machine operates within temperature. To make statements about this domain, we use the set of fluents $\mathbf{F} = \{\mathsf{On}, \mathsf{Cool}\}$ to express that the machine is on and within acceptable temperature, and the set of actions $\mathbf{A} = \{\mathsf{Switch}\}$ for toggling the machine's power button.

How fluents and actions interrelate is now given through the following statements. When switching the machine on/off, the status of fluent On flips – $\underline{\texttt{action}}$ Switch $\underline{\texttt{causes}}$ On $\underline{\texttt{if}}$ $\{\neg\mathsf{On}\}$, $\underline{\texttt{action}}$ Switch $\underline{\texttt{causes}}$ $\neg\mathsf{On}$ $\underline{\texttt{if}}$ $\{\mathsf{On}\}$ – and turning it off furthermore causes the machine to cool down, formalised as $\underline{\texttt{action}}$ Switch $\underline{\texttt{causes}}$ Cool $\underline{\texttt{if}}$ $\{\mathsf{On}\}$. The usual state of affairs in normal operation mode is expressed by the default statement $\underline{\texttt{normally}}$ Cool $\underline{\texttt{if}}$ $\{\mathsf{On}\}$.

For a specific domain instance, we will also assume given a *narrative* consisting of observations of the status of fluents and occurrences of actions at various time points. Although we introduce a more general notation, in this paper, we restrict our attention to actions that end in the direct successor time point.

Definition 6. For a fixed domain vocabulary $(\mathbf{F}, \mathbf{A}, \mathbf{T}, \prec)$, let $a \in \mathbf{A}$ be an action, $l \in \mathbf{F}^{\pm}$ a fluent literal and $s, t \in \mathbf{T}$ time points with $s \prec t$. An *axiom* can be:

- an *observation axiom*: $\underline{\texttt{observed}}$ l $\underline{\texttt{at}}$ t
- an *occurrence axiom*: $\underline{\texttt{happens}}$ a $\underline{\texttt{from}}$ s $\underline{\texttt{to}}$ t

So when we use the linear time structure shown earlier, action occurrences are always of the form $\underline{\texttt{happens}}$ a $\underline{\texttt{from}}$ n $\underline{\texttt{to}}$ $n+1$ for some $n \in \mathbb{N}$, which means that a has a fixed duration of one time-step. However, we allow the possibility of concurrency, that is, multiple actions happening at the same time.

The combination of general information about the domain and information about a specific instance is now called an action domain specification.

Definition 7. An *action domain specification*, or *domain* for short, is a set $\Sigma = \Upsilon \cup \Omega$ where Υ is a finite set of statements and Ω is a set of axioms.

Example 2 (Continued)**.** We extend the partial domain signature of the machine supervision domain by the time structure $(\mathbf{T}, \prec)$ where $\mathbf{T} = \{0, 1, 2, 3\}$ and $\prec$ is given by $\{(0, 1), (1, 2), (2, 3)\}$. (Note that in this case $\mathbf{0} = 0$.) Now we can express a narrative where the machine is initially off, then switched on at time point 1 and nothing further happens: $\Omega = \{\underline{\texttt{observed}}\ \neg\mathsf{On}\ \underline{\texttt{at}}\ 0, \underline{\texttt{happens}}\ \mathsf{Switch}\ \underline{\texttt{from}}\ 1\ \underline{\texttt{to}}\ 2\}$.

Whenever using situations as underlying time structure of a domain, we tacitly assume included the set $\{\underline{\texttt{happens}}\ a\ \underline{\texttt{from}}\ s\ \underline{\texttt{to}}\ do(a, s) \mid a \in \mathbf{A}, s \in \mathbf{T}\}$ of occurrence axioms (expressing the meaning of situations as hypothetical future time points) and restrict the user specification to observation axioms.

For any action domain specification – be it linear-time or branching-time – we now want to make predictions about how the domain will normally evolve over time. We do this by translating the specifications into argumentation frameworks and using argumentation semantics to reason about the domains.

3.2. From Action Domain Specifications to Abstract Argumentation Frameworks

In the previous section, we introduced the syntax of a language for describing dynamic domains. Now we present the argumentation-based semantics for that language. To this end, we define a translation function from action domain specifications into abstract argumentation frameworks.

This translation function will be mostly *modular*, which means that most of the constituents of a domain description can be translated in isolation, that is, without considering other parts of the domain. It is *mostly* modular because there will be one exception for modularity: to correctly express the effects of actions, we need access to effect statements when translating occurrence axioms.

The basic intuition underlying our translation is the following. Each piece of knowledge about the domain is modelled as an argument [?,9]. The most important arguments will express whether a fluent holds at a time point. There will be other arguments, that state various causes for fluents to have a certain truth value. For example, there will be arguments stating persistence as a reason for a fluent being true or false, arguments about direct action effects, default conclusions and lastly observations. To resolve conflicts between these causes – e.g. persistence says a fluent should be false while a direct effect says it should be true – we use a fixed natural priority ordering that prefers observations over action effects, which are in turn preferred over defaults, that on their part trump persistence. This priority ordering is expressed in the defined framework via attacks, that (along with the remaining attacks) encode the relation between different pieces of knowledge in the domain and eventually determine the semantics of the given specification.

For the rest of this subsection, we assume a given action domain specification Σ over a domain vocabulary $(\mathbf{F}, \mathbf{A}, \mathbf{T}, \prec)$. Although the translation can be defined in a strictly formal way, we have chosen a less rigorous presentation that we hope is much easier to read. In the paragraphs below, we define arguments and attacks that are created from elements of Σ. To express that argument a attacks argument b, we will write $a \twoheadrightarrow b$. The resulting argumentation framework $\mathcal{F}_\Sigma$ is understood to contain all arguments and attacks that we define below. Along the way, we will illustrate most of the definitions with the relevant parts of the argumentation framework of our running example domain.

Fluents and time points. First of all, we create arguments that express whether a fluent is true or false at a time point, or alternatively whether a fluent literal holds at a time point. For a fluent literal $l \in \mathbf{F}^{\pm}$ and time point $t \in \mathbf{T}$ they are of the form $\mathit{holds}(l, t)$. Obviously, a fluent cannot be both true and false at any one time point, so for all $f \in \mathbf{F}$ and $t \in \mathbf{T}$, we create the attacks $\mathit{holds}(f, t) \rightarrow \mathit{holds}(\neg f, t)$ and $\mathit{holds}(\neg f, t) \rightarrow \mathit{holds}(f, t)$.

Persistence. To solve the frame problem, we define arguments $\mathit{frame}(l, t_1, t_2)$ for all $l \in \mathbf{F}^{\pm}$ and $t_1 \prec t_2$. These arguments say "the truth value of fluent literal l persists from time point t_1 to its direct successor t_2." First, we express that l holding at t_1 is evidence against its negation $\bar{l}$ persisting from t_1 to t_2: $\mathit{holds}(l, t_1) \rightarrow \mathit{frame}(\bar{l}, t_1, t_2)$. Also, l persisting from t_1 to t_2 is evidence against it being false at t_2: $\mathit{frame}(l, t_1, t_2) \rightarrow \mathit{holds}(\bar{l}, t_2)$.

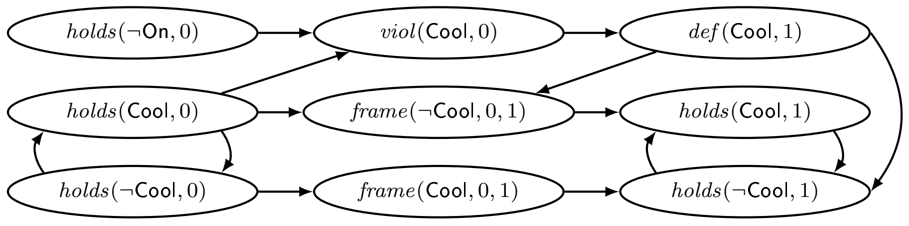
Example 2 (Continued)**.** The AF about fluent On and time points $0, 1$ looks thus:

$$\begin{tikzcd}
\mathit{holds}(\mathsf{On}, 0) \arrow[r] \arrow[d, bend left] & \mathit{frame}(\neg\mathsf{On}, 0, 1) \arrow[r] & \mathit{holds}(\mathsf{On}, 1) \arrow[d, bend left] \\
\mathit{holds}(\neg\mathsf{On}, 0) \arrow[r] \arrow[u, bend left] & \mathit{frame}(\mathsf{On}, 0, 1) \arrow[r] & \mathit{holds}(\neg\mathsf{On}, 1) \arrow[u, bend left]
\end{tikzcd}$$

Defaults. Now we encode default conclusions: for $\underline{\texttt{normally}}\ l\ \underline{\texttt{if}}\ C \in \Sigma$ and $t \in \mathbf{T}$ we create arguments $\mathit{def}(l, t)$ and $\mathit{def}(\bar{l}, t)$ with the intended meaning that l normally holds (normally does not hold) at t. The argument for l being normally true at t attacks the arguments for l being false or normally false at t: We add $\mathit{def}(l, t) \rightarrow \mathit{holds}(\bar{l}, t)$ and $\mathit{def}(l, t) \rightarrow \mathit{def}(\bar{l}, t)$. A default is inapplicable if some prerequisite $c \in C$ is false at t, expressed by $\mathit{holds}(\bar{c}, t) \rightarrow \mathit{def}(l, t)$ for each $c \in C$. Defaults generally override persistence, so a default $\underline{\texttt{normally}}\ l\ \underline{\texttt{if}}\ C$ will attack persistence of literal $\bar{l}$ to time point t. If there is a time point $s \prec t$ we add the attack $\mathit{def}(l, t) \rightarrow \mathit{frame}(\bar{l}, s, t)$.

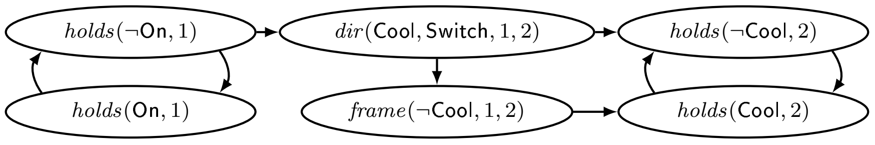
We also create special arguments that express whether the world is abnormal with regard to a specific default, that is, whether the default was violated at the time point. A state default $\underline{\texttt{normally}}\ l\ \underline{\texttt{if}}\ C$ is *violated* whenever all literals in C hold but l does not hold, which hints at an abnormality of the world. For a default $\underline{\texttt{normally}}\ l\ \underline{\texttt{if}}\ C \in \Sigma$ and time points $s, t \in \mathbf{T}$ with $s \prec t$ we create the argument $\mathit{viol}(l, s)$ and the attacks detailed below. First, we require that abnormal situations do not go away by default, therefore a violated default blocks its own application at the successor time point by the attack $\mathit{viol}(l, s) \rightarrow \mathit{def}(l, t)$. Conversely, a default is not violated at s iff one of: (1) it was applied, hence the attack $\mathit{def}(l, s) \rightarrow \mathit{viol}(l, s)$; (2) its consequent holds (the world is normal), thus we add $\mathit{holds}(l, s) \rightarrow \mathit{viol}(l, s)$; or (3) one of its prerequisites is false, hence we include the attack $\mathit{holds}(\bar{c}, s) \rightarrow \mathit{viol}(l, s)$ for each $c \in C$.

Example 2 (Continued)**.** Here is a part of the argumentation (sub-)framework expressing that the machine is usually cool when on. (The lower part about persistence of Cool is isomorphic to the above graph for persistence of On.) Below we can see that $\mathit{holds}(\neg\mathsf{On}, 0)$ defends $\mathit{def}(\mathsf{Cool}, 1)$, which in turn defends $\mathit{holds}(\mathsf{Cool}, 1)$.



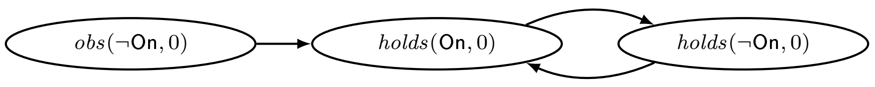
Action effects. Now for modelling the direct effects of actions. Let action occurrence $\underline{\texttt{happens}}\ a\ \underline{\texttt{from}}\ s\ \underline{\texttt{to}}\ t \in \Sigma$ and effect statement $\underline{\texttt{action}}\ a\ \underline{\texttt{causes}}\ l\ \underline{\texttt{if}}\ C \in \Sigma$. We devise an argument $dir(l, a, s, t)$ that encodes occurrence of effect l through a from s to t. First, the action effect l can never materialise if some precondition in C is false: we add $holds(\overline{c}, s) \twoheadrightarrow dir(l, a, s, t)$ for all $c \in C$. If there is a precondition statement $\underline{\texttt{possible}}\ a\ \underline{\texttt{if}}\ C_a \in \Sigma$, we add the attacks$holds(\overline{c}, s) \twoheadrightarrow dir(l, a, s, t)$ for all $c \in C_a$. As usual, to derive effect l at t we attack its negation $\bar{l}$ at t, $dir(l, a, s, t) \twoheadrightarrow holds(\bar{l}, t)$. Since effects override both defaults and persistence, a direct effect l also attacks persistence of and default conclusions about its negation $\bar{l}$, as well as possible conflicting effects. We add $dir(l, a, s, t) \twoheadrightarrow frame(\bar{l}, s, t)$, $dir(l, a, s, t) \twoheadrightarrow def(\bar{l}, t)$ and finally $dir(l, a, s, t) \twoheadrightarrow dir(\bar{l}, a', s, t)$ for all $a' \in \mathbf{A}$.

Example 2 (Continued)**.** If the machine was turned off, it would cool down.



Observations. For observations $\underline{\texttt{observed}}\ l\ \underline{\texttt{at}}\ t \in \Sigma$ we create arguments $obs(l, t)$ and $obs(\bar{l}, t)$ saying that l (resp. $\bar{l}$) has been observed. Observations are the uber-causes and attack all other causes, including observations to the contrary: $obs(l, t) \twoheadrightarrow holds(\bar{l}, t)$, $obs(l, t) \twoheadrightarrow frame(\bar{l}, s, t)$ for $s \prec t$, $obs(l, t) \twoheadrightarrow def(\bar{l}, t)$, $obs(l, t) \twoheadrightarrow dir(\bar{l}, a, s', t)$ for $s' \prec t$ and all $a \in \mathbf{A}$, finally $obs(l, t) \twoheadrightarrow obs(\bar{l}, t)$.

Example 2 (Continued)**.** Initially, the machine is observed to be off.



This concludes our definition of the argumentation framework $\mathcal{F}_\Sigma$ associated with an action domain specification Σ.

Example 2 (Continued)**.** The full argumentation framework $\mathcal{F}_\Sigma$ constructed from the machine supervision domain is too large to show here, we can however have a look at its extensions and what they predict about the domain. The grounded extension of $\mathcal{F}_\Sigma$ is given by

$obs(\neg\mathsf{On}, 0)$	$holds(\neg\mathsf{On}, 0)$		
$frame(\neg\mathsf{On}, 0, 1)$	$holds(\neg\mathsf{On}, 1)$		
$dir(\mathsf{On}, \mathsf{Switch}, 1, 2)$	$holds(\mathsf{On}, 2)$	$def(\mathsf{Cool}, 2)$	$holds(\mathsf{Cool}, 2)$
$frame(\mathsf{On}, 2, 3)$	$holds(\mathsf{On}, 3)$	$frame(\mathsf{Cool}, 2, 3)$	$holds(\mathsf{Cool}, 3)$

In words, the observation that the machine is off lets us conclude it is indeed off; this persists to time point 1. Then the machine is switched on, fluent On becomes true and persists that way. Meanwhile, there is no information about the temperature of the machine; however, after it is switched on, the default can be applied and the machine is henceforth assumed to be cool.

There are two stable extensions, which contain everything that is grounded and additionally make a commitment toward the initial status of Cool, where the first one contains $holds(\mathsf{Cool}, 0)$ and the second one $holds(\neg\mathsf{Cool}, 0)$. There are three complete extensions, the grounded one and the two stable extensions.

4. Properties of the Constructed Framework

The example framework just seen illustrated the workings of our approach, yet allows no general conclusions about beneficiary properties of the constructed frameworks. We will now proceed to prove several nice properties that are highly relevant for knowledge representation and ultimately pave the way for an efficient implementation of our approach.

We start with some general observations. First, the AFs we construct in Section 3.2 are always finitary, although they may of course be infinite due to an infinite set of time points. Second, it follows from the definition that the argumentation frameworks are free of self-loops. According to results about strong equivalence [11], this means that all attacks encoded therein are actually meaningful. In the remainder, we explore some more properties in greater detail.

4.1. Encoded Priorities

A fluent f can hold, respectively not hold, at a time point t for different reasons: because an observation says so, it is a direct effect of an action, it is the conclusion of an applicable state default or simply because of persistence. Whenever two of these reasons are in conflict, for instance persistence says f should be true at t while an action effect says f should be false, we somehow have to decide for a truth value of f. We now want to show that the extensions of the translated AFs $\mathcal{F}_\Sigma$ satisfy a certain priority ordering among these reasons.

Theorem 2. *Let Σ be a domain and $\mathcal{F}_\Sigma$ the associated AF, and consider reasoning in the stable, complete, preferred and grounded semantics. Concerning the causes that influence whether fluents hold at a time point, we have the following:*

(1) observations override action effects, (2) action effects override default conclusions, (3) default conclusions override persistence and (4) the frame problem is solved.

Hence when, for example, persistence says f should be true and a direct effect says f should be false, then the preferred cause "direct effect" takes precedence and f will indeed not hold. Above, default conclusions refers to the consequents of applicable state defaults, where we recall that violated state defaults are inapplicable at the next time point. So if, in our running example, the machine is On and $\neg\mathsf{Cool}$ at one time point, the default statement $\underline{\texttt{normally}}$ Cool $\underline{\texttt{if}}$ $\{\mathsf{On}\}$ is violated and $\neg\mathsf{Cool}$ will persist. We have to note that Theorem 2 is not obvious and the proofs of the propositions about each of the items above are quite lengthy.

4.2. Extensions

The results of the previous section show that the argumentation semantics considered in this paper respect a suitable priority ordering among causes. If this ordering is the same across the semantics, the reader may ask, then where lies the difference between them? Roughly, the different semantics are used to model different types of knowledge (complete v. incomplete) and modes of reasoning (well-founded v. hypothetical). The grounded semantics, for example, accepts only conclusions that are well-founded with respect to definite knowledge. However, this may lead to the agent having only incomplete knowledge about the domain. The stable semantics, on the other hand, may make unproven assumptions about the world, but provides complete knowledge about the domain in each extension:

Proposition 3. *Let Σ be a domain with associated AF $\mathcal{F}_\Sigma = (A, R)$. For each $E \in \mathcal{E}_{st}(\mathcal{F}_\Sigma)$, $t \in \mathbf{T}$ and $f \in \mathbf{F}$ we have either $holds(f, t) \in E$ or $holds(\neg f, t) \in E$.*

While this result hints at the benefits of using the stable semantics for reasoning about actions with our frameworks, this conclusion has to be qualified: the existence of stable extensions cannot be guaranteed in general.

4.3. Computing Extensions

We already observed that the constructed AF $\mathcal{F}_\Sigma$ may be infinite but is in any case finitary. Hence, using a result of Dung, we may obtain the unique grounded extension of $\mathcal{F}_\Sigma$ by iteratively applying the characteristic function on the empty set [5]. Unfortunately, there is no similar (constructive) method for the other semantics we are interested in. Nevertheless we will show that the evaluation of an argument from $\mathcal{F}_\Sigma$ can be implemented in principle since we only have to compute extensions in finite subframeworks. Here, the concept of *stratification* plays the lead. A stratification divides the arguments of an AF into layers that satisfy a simple syntactic dependency criterion: a layer of an argumentation framework is any subset of its arguments that is not attacked from the outside. If there is an increasing sequence of layers whose union is the set of all arguments, then the argumentation framework is called stratified.

Definition 8. Let $\mathcal{F} = (A, R)$ be an argumentation framework. A set $L \subseteq A$ is a *layer of $\mathcal{F}$* iff for all $a \in L$ and $b \in A \setminus L$ we have $(b, a) \notin R$. A strictly increasing sequence of layers $L_0 \subsetneq L_1 \subsetneq \ldots$ is a *stratification of $\mathcal{F}$* iff $L_0 \cup L_1 \cup \ldots = A$.

We call an stratification *infinite (finite)* iff it possesses infinitely (finitely) many layers. For a layer L_i, the argumentation framework *associated to layer L_i* is $\mathcal{F}_i \stackrel{\text{def}}{=} (L_i, R \cap (L_i \times L_i))$. An AF $\mathcal{F}_\Sigma$ automatically constructed from a domain Σ allows for a fairly straightforward definition of a stratification:

Definition 9. Let Σ be a domain over vocabulary $(\mathbf{F}, \mathbf{A}, \mathbf{T}, \prec)$ and $\mathcal{F}_\Sigma = (A, R)$ be the argumentation framework obtained from it by the encoding specified in Section 3.2. For time points $s, t \in \mathbf{T}$ define

$$L(t) \stackrel{\text{def}}{=} \left\{ holds(l,t), obs(l,t), viol(l,t), def(l,t) \;\middle|\; l \in \mathbf{F}^\pm \right\} \cap A$$

$$L(s,t) \stackrel{\text{def}}{=} \left\{ frame(l,s,t), dir(l,a,s,t) \;\middle|\; l \in \mathbf{F}^\pm, a \in \mathbf{A} \right\} \cap A$$

For a set $B \subseteq A$ of arguments denote by $\mathbf{T}(B)$ the time points occurring in B. Now define by induction on natural numbers the sets $L_0 \stackrel{\text{def}}{=} L(\mathbf{0})$ and for $n \in \mathbb{N}$,

$$L_{n+1} \stackrel{\text{def}}{=} \bigcup_{s \in \mathbf{T}(L_n), s \prec t} (L(s) \cup L(s,t) \cup L(t))$$

For the stratification, we construct the bottom layer by taking all arguments about the least time point $\mathbf{0}$. The following layers are then defined inductively according to the time structure: in each step, we add the arguments about direct successors of the time points in the arguments in the previous layer. It is straightforward to prove that the construction as a matter of fact yields a stratification:

Proposition 4. *Let Σ be a domain over vocabulary $(\mathbf{F}, \mathbf{A}, \mathbf{T}, \prec)$; let $\mathcal{F}_\Sigma = (A, R)$ be the AF constructed from Σ by the encoding from Section 3.2. Then the sets $L_0, L_1, \ldots$ obtained according to Definition 9 are a stratification for $\mathcal{F}_\Sigma$.*

Of course the same holds for finite stratifications, that end in some layer $L_m = A$. The real power of stratifications lies however in being able to decide the status of an argument after discarding a possibly infinite part of the framework:

Proposition 5. *Let Σ be a domain and $\mathcal{F}_\Sigma = (A, R)$ its associated argumentation framework with stratification $L_0, L_1, \ldots$ according to Definition 9. For any semantics $\sigma \in \{gr, co, pr\}$ we have: an argument a is credulously/sceptically accepted in $\mathcal{F}_\Sigma$ w.r.t. σ iff a is credulously/sceptically accepted in $\mathcal{F}_n$ w.r.t. σ where $\mathcal{F}_n$ is the AF associated to the least layer L_n such that $\mathsf{a} \in L_n$.*

It is crucial to note that this result makes implementing our approach *feasible in principle*, since it reduces the relevant decision problems about infinite AFs to equivalent problems about finite AFs. What is more, translating action domains to stratified argumentation frameworks also provides an important step towards an *efficient* implementation: An intelligent agent that constantly executes actions has to decide upon future actions in a timely manner. Consider an agent that has executed m actions since having been switched on. To plan n time points into the future, it needs to consider an argumentation framework of a size that grows in $m + n$. As time passes, n (the lookahead into the future) might be kept constant but m (the history of past actions) will surely grow up to the point where despite only involving a linear blowup the associated AFs are too large for the agent to handle. The solution to this problem is known as *progression* in the reasoning about actions community: every once in a while, the agent replaces its knowledge base about the whole past by a smaller but equivalent one about the present, thereby effectively resetting m [13]. That way, the size of the knowledge base the agent handles can be significantly reduced while keeping the information it contains. For grounded semantics, the obvious technical approach to progression is then to replace the sub-framework $\mathcal{F}'_\Sigma$ speaking about the past until time point t by its unique grounded extension E', thereby obtaining a splitting [1]. Empirical evidence about extensions of AFs admitting a splitting [4] shows that this provides a promising basis for implementing the approach presented in this paper.

5. Related Work and Conclusions

We presented an encoding of action domains into abstract argumentation frameworks. In being independent of a particular notion of time and able to do default reasoning in dynamic domains, the approach goes well beyond the capabilities of current action languages. We used theoretical results from argumentation to show how our approach can be put to use and argue how it can be implemented.

In continuation of their preliminary earlier work [?], Michael and Kakas developed an approach that combines default reasoning with temporal reasoning and is based on *assumption-based* argumentation [?,9]. The approach uses a fixed linear time structure and a tailor-made definition of argumentation semantics. The work presented in this paper uses a more general notion of time, furthermore we employ *abstract* argumentation with the standard definitions of its semantics, and can immediately use existing results from this area, in particular existing solvers. Earlier, Kakas et al. [7] translated domains in an action language based on linear time into the argumentation framework of logic programming without negation as failure, a contribution much closer to the original work on action languages [6].

Acknowledgement. The authors are grateful to Gerhard Brewka for providing valuable feedback on earlier versions of this paper.

References

[1] Ringo Baumann. Splitting an argumentation framework. In *LPNMR*, pages 40–53, 2011.

[2] Ringo Baumann and Gerhard Brewka. Expanding argumentation frameworks: Enforcing and monotonicity results. In *COMMA*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 75–86. IOS Press, 2010.

[3] Ringo Baumann, Gerhard Brewka, Hannes Strass, Michael Thielscher, and Vadim Zaslawski. State Defaults and Ramifications in the Unifying Action Calculus. In *KR*, pages 435–444, 2010.

[4] Ringo Baumann, Gerhard Brewka, and Renata Wong. Splitting argumentation frameworks: An empirical evaluation. In *TAFA*, pages 17–31, 2011.

[5] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *AIJ*, 77:321–358, 1995.

[6] Michael Gelfond and Vladimir Lifschitz. Action Languages. *ETAI*, 3, 1998.

[7] Antonis C. Kakas, Rob Miller, and Francesca Toni. An Argumentation Framework of Reasoning about Actions and Change. In *LPNMR*, pages 78–91, 1999.

[8] John McCarthy. Programs with Common Sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, 1959.

[9] Loizos Michael and Antonis Kakas. A Unified Argumentation-Based Framework for Knowledge Qualification. In E. Davis, P. Doherty, and E. Erdem, editors, *Proceedings of the Tenth International Symposium on Logical Formalizations of Commonsense Reasoning*, Stanford, CA, March 2011.

[10] Erik Mueller. *Commonsense Reasoning.* Morgan Kaufmann, 2006.

[11] Emilia Oikarinen and Stefan Woltran. Characterizing Strong Equivalence for Argumentation Frameworks. In *KR*, 2010.

[12] I. Rahwan and G. R. Simari, editors. *Argumentation in Artificial Intelligence.* Springer, 2009.

[13] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* The MIT Press, September 2001.

[14] Michael Thielscher. A Unifying Action Calculus. *AIJ*, 175(1):120–141, 2011.