# Policies, Norms and Actions: Groundwork for a Framework

Robert Craven
robert.craven@doc.ic.ac.uk

Department of Computing,
Imperial College London,
SW7 2AZ

**Abstract**

Constraints on computational agents' behaviour are studied both in work on policy-governed systems—usually as part of work on security or policy-based management in distributed software engineering—and also in multi-agent systems research, where the terminology is generally one of 'norms' and concepts drawn from deontic logic. Interaction between these treatments, and the research communities that study them, has not been as thorough as it might, for though the perpectives, methods and interests are sometimes different, there is a great deal of shared ground. In the current research report, we present a language and tools which can be used for reasoning about and studying the operation of both norms and policies on a multi-agent, or distributed, system. The language is based on one member, $\mathcal{C}+$, of a family of knowledge representation formalisms studied in AI. We describe the types of domains that can be represented, the kinds of analysis tasks that are possible, and describe our current implementation (which is freely available for download). Future directions for this work are described.

**Acknowledgements**

The work on norms in $\mathcal{C}+$ presented in Section 5.1 is joint with Marek Sergot, as is the implementation of iCCalc and the 'rooms' example. I would like to thank Srdjan Marinovic for reading a draft and for providing detailed and useful criticism.

Research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

## Contents

**1 Introduction** **2**

**2 Background: labelled transition systems and $\mathcal{C}+$** **4**
2.1 Signatures and Causal Laws . . . . . 4
2.2 Action Descriptions to Transition Systems . . . . . 7
2.3 Causal theories . . . . . 8
2.4 Action Descriptions to Causal Theories . . . . . 9
2.5 Definiteness and Completion . . . . . 11

**3 Basic formalism** **12**
3.1 Signature . . . . . 12
3.2 System Models . . . . . 13
3.3 Policies . . . . . 14

3.4 Semantics for policy descriptions . . . . . 19

**4 Policy composition** **21**
4.1 Language and semantics . . . . . 23
4.2 Example composition . . . . . 27

**5 Norms** **29**
5.1 Syntax and Semantics . . . . . 29
5.2 Example: Rooms . . . . . 30
5.3 Extracting the System Component . . . . . 34

**6 Analysis Queries and Examples** **35**
6.1 Implementation . . . . . 35
6.2 Query and Analysis Types . . . . . 37

**7 Related work** **43**

**8 Concluding Remarks and Future Work** **44**

**A Sample code for a 'rooms' example** **46**

# 1 Introduction

In this paper we explore several ways in which controls over agents' behaviour can be represented and reasoned about. There are three main streams of influence that here come together: work on action languages, from artificial intelligence; work on norms, from multi-agent systems; and research on policy algebras, from policy management. The aim is to study the relationship between these different techniques and paradigms, and to suggest the benefits of combining them. To that end, we include many examples, and describe the uses to which a combined system can be put in the analysis of various forms of directives governing an agent's behaviour in a changing environment. Examples are illustrated with the use of our implementation, which covers all ideas presented in the paper. In the rest of this introduction, we expand on the various components of work, and the motivations behind it.

The norms we represent in the current framework are typically used to represent the high-level requirements of a system, which do not take into account the way the domain is implemented at the lower levels of detail. Norms are here understood to classify different behaviours and combinations of the properties of a system as good or bad. Policies, by contrast, concern the micro-management of the specific actions performed by the devices, agents and components of a system; the policies are intended to achieve, at a practical level, what the more abstract norms recommend. We combine norms and policies in a common framework, in which the components, devices, and agents that the norms and policies refer to have their behaviour modelled by a logical formalism from Artificial Intelligence, amenable to temporal reasoning, model-checking, and event-based descriptions.

Several more detailed motivations resulted in the current work. One was to provide a perspicuous and intuitive representation for the combination of default policy decisions and exceptions, in a context where there could be named sets of policies. In our previous work [Craven et al., 2008, Craven et al., 2009], we have used normal logic programs, and specifically negation-as-failure, to represent exceptions to policies and default policy rules. However, that work did not have a clear distinction between the rules that state how policies override each other and how the PDP reaches a decision given the policy rules, and the policy rules themselves; this meant that it was frequently unclear how defaults and exceptions should be expressed. It also seems desirable that authorization policies should be able to be grouped together, and that one should be able to state, for instance, that one policy should apply in the case that no rule whatsoever in a given state applies. It became clear that papers that have recently been published in the policy literature on an algebraic approach to policy composition in fact offer a better solution to this problem.

(Model-checking and enriched concepts from deontic logic have been used before in research on policies, but there is no unifying framework that combines them with policy algebra and with a powerful representation of the domain that the policies govern. This is what we attempt in the current report.)

There is also increasing interest from the policy community in the possibilities of using various techniques from AI in reasoning about policies and their interactions with the systems they govern. Techniques that have been applied to policy reasoning include first-order logic theorem proving, reasoning using description logic, and abductive and inductive logic programming. However, there are many other kinds of techniques that could be used; in the context of the current work, we have in mind particularly model-checking, and reasoning using enriched deontic and agentive concepts. These typically take some form of graphical structure—often, transition systems or labelled transition systems—as their basic semantics. It therefore seemed attractive to let the policies be interpreted over similar structures.

There are many sorts of language for defining labelled transition systems. We have chosen to use $\mathcal{C}+$, which one of the most recent in a series of 'action languages' [Gelfond and Lifschitz, 1998] used to represent the effects of actions and events on a changing environment; these langauges are most widely used in knowledge representation. $\mathcal{C}+$ can be used to depict the states and behaviour of different agents, acting together or independently, and includes built-in features for the treatment of various kinds of default, inertia, ramifications, and so on.

In the present work, we add constructions to $\mathcal{C}+$ for policies, and for policy composition. We wish to reason about the behaviour of policies, and of agents governed by policies, in an environment on which policy depends, and which can be altered by the actions performed in accordance with the policies. Both authorization policies, positive and negative, and obligation policies are covered. Positive and negative authorization policies are grouped together into sets, and these, through the logic of policy algebra, determine whether or not a request by a *subject* to perform an *action* on a *target* is allowed. If the request is allowed, then the effects of the requested action are caused, and this determines the way the system evolves over time. Obligation policies are, for us, encoded as triggers in much the same way as event-condition-action rules. When the condition of a rule is true, then a request is triggered automatically—again, for a *subject* to perform an *action* on a *target*. The difference is that we do not insist that events are present: the triggers can be any kind of combination of current conditions. This request is then subject to the authorization policies in place. The combination of action languages, labelled transition systems, and policies, allows reasoning about the effects of policy in a changing environment; this is very important for policy analysis, as policy evaluation can then be seen to depend on how the system has evolved over time.

In recent years there has been increasing interest in the use of norms to regulate the interactions of multi-agent systems. Norms can partly be understood as accepted standards of behaviour to which certain objects capable of doing things should adhere (this being a near-paraphrase of a widely-used dictionary's definition of the term). In a more general way, we understand norms as the expression of what is good and bad in a domain—where the generalization allows features of an agent's state, of any other object's state, of the environment, of histories of objects and actions, and so on, to be considered as bound by the norms. The norms need not only register what is accepted or acceptable, but what is desired, right, legal, immoral, conformant with protocol, etc. We focus on the expression of a standard, rather than the standard itself. Examples of systems of norms in this wider sense are: legal systems; Robert's rules of order; bidding protocols in auctions; Kantian ethics; and preferences over items on a menu. It is of course a stretch to call the last of these 'norms', but that does not matter.

One way in which norms are used in multi-agent systems is to specify, from the perspective of a system designer or other sort of system-wide authority, how the system should be implemented: how it should behave, what the desirable system configurations are. This is the way in which we use norms in the current work. The actual system may, in fact, not adhere to the given norms: this is one way in which our treatment of norms differs from that of policies, for whether or not a policy determines a request as allowed makes all the difference to whether the action requested will be performed. Norms regulate, policies regiment. The presence of these different ways of

holding a standard to the behaviour of artificial systems in the same formalism, to be evaluated over the same models, is fruitful: we can reason about whether the actual policies in place satisfy the (various kinds of) norms, and if they do not satisfy them, why not.

All work presented here has been implemented in freely-available software; see Section 6.1 for details and the URL. (This implementation is joint work with Marek Sergot.)

The structure of this report is as follows. First, background to the action language $\mathcal{C}+$. Then a preliminary version of how to add policy structures to the language, followed by how to do this for policies combined by a policy algebra. Then, description of a 'normative' addition to the language, orthogonal to the policy bit. Then examples, and discussion of the possibilities for various kinds of analysis query, together with details of an implementation. Finally, related work and concluding remarks, including on further work.

# 2 Background: labelled transition systems and $\mathcal{C}+$

Action languages are logics for describing how a system behaves over time, as a consequence of actions performed within the system. $\mathcal{C}+$ is the most recent member of a family of action languages which began with $\mathcal{A}$ [Gelfond and Lifschitz, 1993], and whose early history is surveyed in [Gelfond and Lifschitz, 1998]. $\mathcal{C}+$ has a number of very appealing features, which recommended it as a starting-point for the kinds of development we pursue in the current paper.

- It provides a very natural treatment of inertia (default persistence), which gives intuitively desired results.
- It is easy to say many things in $\mathcal{C}+$: concurrent actions, default effects, ramifications, non-determinism, are all accommodated.
- After only a brief acquaintance with the language, one can write action descriptions which capture the intended behaviour.
- The underlying formalism that can be provided—that of causal theories—is simple: so in cases where there are complex interactions between causal laws, reference to this underlying formalism quickly resolves confusion.
- There is a semantics of labelled transition systems already in place. Since we are interested in making use of the bridge this affords to methods in other areas of AI, this is a significant advantage.

We will give an overview of the syntax of $\mathcal{C}+$; show how labelled transition systems are defined; describe the underlying framework of causal theories and the alternative route to transition systems this affords; present the algorithm used to find models of an important subclass of causal theories; and give an outline of $n\mathcal{C}+$ [Craven and Sergot, 2008], an extension to $\mathcal{C}+$ incorporating deontic concepts. The presentation of $\mathcal{C}+$ we give will not include all features of that language, but only those which are relevant for our work. For the whole, see [Giunchiglia et al., 2004]. Our terminology does not precisely match that of the original authors.

## 2.1 Signatures and Causal Laws

First, the syntax of $\mathcal{C}+$. We begin with $\sigma$, a multi-valued, propositional signature. Members of $\sigma$ are known as *constants*. $\sigma$ is assumed to be partitioned into a set $\sigma^f$ of *fluent constants* and a set $\sigma^a$ of *action constants*. Further, the fluent constants are partitioned into those which are *simple* and those which are *statically determined*. We sometimes use $\sigma^{smpl}$ to stand for the simple fluent constants, and $\sigma^{stat}$ to stand for the statically determined fluent constants. And so:

$$\sigma = \sigma^f \cup \sigma^a = \sigma^{smpl} \cup \sigma^{stat} \cup \sigma^a.$$

Statically determined fluent constants depend only on features of the current state: the values they take may not directly depend on the past history of the domain. In the representation of many domains, the set of statically determined fluent constants is empty.

For each constant $c \in \sigma$ there is a finite, non-empty set $dom(c)$, disjoint from $\sigma$ and known as the *domain* of $c$. An *atom* of the signature is an expression $c{=}v$, where $c \in \sigma$ and $v \in dom(c)$. *Formulas* are constructed from the atoms using propositional connectives and a familiar syntax, with a *literal* as an expression $A$ or $\neg A$, for atomic $A$. The expressions $\top$ and $\bot$ are connectives of zero arity, with the usual interpretation. A *Boolean* constant is one whose domain is the set of truth-values $\{\mathbf{t}, \mathbf{f}\}$, and a *Boolean* signature is, by extension, one all of whose constants are Boolean. If $c$ is a Boolean constant, we often write $c$ for $c{=}\mathbf{t}$, so that where our propositional signatures are restricted to be Boolean and we deal with no formula containing $\mathbf{f}$, we may reduce our syntax to that of standard propositional logic.

A *fluent formula* is a formula whose constants all belong to $\sigma^f$; an action formula is a formula which contains at least one action constant, and no fluent constants. Note that according to these definitions, $\bot$ and $\top$ are fluent formulas but not action formulas. Disjointness of $\sigma^f$ and $\sigma^a$ forces disjointness of the set of fluent formulas (which we abbreviate to $fmla^f$) and the set of action formulas (abbreviated to $fmla^a$). The disparity in the definitions of $fmla^f$ and $fmla^a$ makes later definitions more compact.

An *interpretation* of a multi-valued propositional signature $\sigma$ is a function mapping every constant $c$ to some $v \in dom(c)$; an interpretation $X$ is said to *satisfy* an atom $c{=}v$ if $X(c) = v$, and in this case one may write $X \models c{=}v$. Standard structural recursions over the propositional connectives apply, and where $\Gamma$ is a set of formulas of our propositional signature, $X \models \Gamma$ expresses that $X \models c{=}v$, for every $c{=}v$ in $\Gamma$. We let the expression $\mathrm{I}(\sigma)$ stand for the set of interpretations of $\sigma$.

A *static law* is an expression of the form

$$F \textbf{ if } F' \tag{1}$$

where $F$ and $F'$ are fluent formulas. These laws are similar to the *state constraints* which appear elsewhere in computer science; their meaning is that when the formula $F'$ is true in a state, then the formula $F$ *is caused* to be true. An *action constraint* is an expression of the form

$$A \textbf{ if } G \tag{2}$$

where $A$ is an action formula and $G$ is a formula. An action constraint $A$ **if** $G$ means that when $G$ is true in a state, then when the system evolves from that state, it must do so in a way which makes $A$ true. If $G$ is a conjunction $A' \wedge F$, where $A'$ is an action formula and $F$ a fluent formula, this means that if $F$ is true in a state, and the actions $A'$ then occur, then the actions $A$ must also occur. A *fluent dynamic law* has the form

$$F \textbf{ after } G \tag{3}$$

where $F$ is a fluent formula and $G$ a formula, the restriction that $F$ must not contain statically determined fluents. Fluent dynamic laws can be understood as stating that where $G$ is true in a state, then $F$ must be true in the *next* state. In particular, as with action constraints, if $G$ is a conjunction of an action formula $A$ and a fluent formula $F'$, then (3) would mean that where $F'$ is true in a state, and $A$ occurs, then $F$ is true in the following state. *Causal laws* are static laws or dynamic laws, and an *action description* is a set of causal laws.

An action description $D$ is said to be *definite* when

- the head of every causal law of $D$ is either an atom or $\bot$, and
- no atom is the head of infinitely many causal laws of $D$.

When an action description of $\mathcal{C}+$ is definite in this sense, then there is a straightforward method of finding runs through the transition system it defines, which we will present in Section 2.5. All of the examples we will mention in the current paper are definite action descriptions.

For the purpose of illustration, consider the very simple action description having as its Boolean signature

$$\sigma^{smpl} = \{p\} \qquad \sigma^{stat} = \{q\} \qquad \sigma^{a} = \{a\}$$

Thus, $p$ and $q$ are intended to represent properties of states, and $a$ to represent an action, the performance of which may affect those properties. Let the laws of the action description be:

$$\begin{aligned} &q \textbf{ if } p, \\ &\neg q \textbf{ if } \neg p, \\ &p \textbf{ after } a, \\ &\neg p \textbf{ after } \neg a. \\ &\textbf{exogenous } a. \end{aligned} \tag{4}$$

We will soon see that action descriptions of $\mathcal{C}+$ can be rendered graphically; action description (4) can be depicted as in Figure 1. The first two (static) laws make $q$'s value dependent on that of $p$.

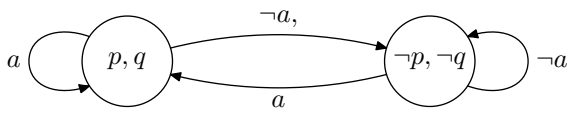


Figure 1: Transition system for action description (4)

The second two laws make the value of $p$ change, depending on whether $a$ is performed or not. The fifth law states that $a$ can occur, or not occur, in any situation, other things being equal (for the meaning of the keyword **exogenous**, see the end of this subsection). We see that static laws can be used to describe how the effects of actions ramify. Transitions between states are depicted as arrows, and the labels on the arrows represent what actions and events occur, to move the domain from one state to another. For instance, if $p$ and $q$ are true of the system being represented, then if the event or action $a$ does not occur (i.e. $\neg a$ is true), then the system will move to a state in which both $p$ and $q$ are false—one where $\neg p$ and $\neg q$ are true.

A number of abbreviations of causal laws are useful, making the meaning of specific structures of law more clear. Fluent dynamic laws of the form $\bot$ **after** $A$, where $A$ is an action formula, may also be written

$$A \textbf{ is non-executable}$$

and fluent dynamic laws $\bot$ **after** $A \wedge F$, where $A$ is an action formula and $F$ a fluent formula, may be written

$$A \textbf{ is non-executable if } F$$

A static law of the form $F$ **if** $\top$ can simply be written as

$$F$$

in which case $F$ must be true in all states. Further, the law

$$\textbf{inertial } c$$

where $c$ is a simple fluent constant, abbreviates the set of causal laws

$$\{c{=}v \textbf{ if } c{=}v \textbf{ after } c{=}v \mid v \in dom(c)\}$$

The effect of this is to enable the value of the fluent constant $c$ to persist by default into the next state (unless $c$ is caused to have some other value). The law

$$\textbf{exogenous } c$$

where $c$ is an action constant abbreviates the set of action dynamic laws

$$\{c{=}v \textbf{ if } c{=}v \mid v \in \mathit{dom}(c)\}$$

These have the effect of allowing $a$ to take any value in a transition (except if it is caused to have some value by other causal laws).

## 2.2 Action Descriptions to Transition Systems

Every action description of $\mathcal{C}+$ defines a *labelled transition system*. In general, transition systems are graphs, whose vertices represent the states of some system, and where an edge between two vertices represents that the system may evolve from the one state to the other. The edges are typically called *transitions*, and where the transitions are labelled, the labels usually denote the performance of some action, or the execution of some computation, which effects the given transition between states. This sort of graphical structure is, of course, ubiquitous in computing and artificial intelligence.

The precise form of the transition systems with which we will deal in the current thesis will vary, usually as the language used to define them accrues special features. It is therefore difficult even to define an underlying template to which all labelled transition systems must conform. So we relativize to $\mathcal{C}+$, as follows.

**Definition 1** A *transition system for* $\mathcal{C}+$ is a triple $(S, L, R)$, where

- $S$ is a set of *states*, sometimes called *vertices*;
- $L$ is a set of *labels*;
- $R$ is a set of *transitions*, $R \subseteq S \times L \times S$, sometimes called *edges*. ⌟

The purpose of this section is to describe how each action description $D$ of $\mathcal{C}+$—with signature $\sigma^f \cup \sigma^a$—defines a labelled transition system $\mathcal{L}_D$, of the form defined above. But we will first note that states will turn out to be interpretations of $\sigma^f$ which satisfy certain constraints, and the set of labels will be the set $\mathrm{I}(\sigma^a)$. The definitions and theorems in this section are mostly taken from [Sergot, 2004].

Suppose we are given an action description $D$ of $\mathcal{C}+$, with signature $\sigma^f \cup \sigma^a$.

**Definition 2** We define:

$$\begin{aligned}
T_{\mathrm{static}}(s) &= \{F \mid F \textbf{ if } F' \in D,\ F \in \mathit{fmla}^f,\ s \models F'\},\\
E(s, e, s') &= \{F \mid F \textbf{ after } G \in D,\ s \cup e \models G\},\\
A(s, e) &= \{A \mid A \textbf{ if } G \in D,\ A \in \mathit{fmla}^a,\ s \cup e \models G\},\\
\mathit{Simple}(s) &= \{c{=}v \mid c \in \sigma^{\mathit{smpl}},\ s \models c{=}v\},
\end{aligned}$$ ⌟

With these preliminary definitions, we can now define the transition systems defined by $\mathcal{C}+$ action descriptions. We first need to define our states.

**Definition 3** Let $D$ be an action description of $\mathcal{C}+$. An interpretation $s$ of $\sigma^f$ is a *state* of $D$ iff

$$\{s\} = \{s' \in \mathrm{I}(\sigma^f) \mid s' \models T_{\mathrm{static}}(s) \cup \mathit{Simple}(s)\}.$$ ⌟

In words, $s$ is a state when $s$ satisfies the set of forumlas $T_{\mathrm{static}}(s) \cup \mathit{Simple}(s)$, and there is no other interpretation of $\sigma^f$ which satisfies this set. We move on to the transitions defined by $\mathcal{C}+$ action descriptions.

**Definition 4** Let $D$ be an action description of $\mathcal{C}+$, with signature $\sigma$. Suppose $s, s' \in \mathrm{I}(\sigma^f)$, and $e \in \mathrm{I}(\sigma^a)$. We say that $(s, e, s')$ is a *transition* of $D$ iff $s$ is a state (according to the preceding definition) and:

- $\{s'\} = \{s'' \in \mathrm{I}(\sigma^f) \mid s'' \models T_{\mathrm{static}}(s') \cup E(s, e, s')\}$;
- $\{e\} = \{e' \in \mathrm{I}(\sigma^a) \mid e' \models A(s, e)\}$. ⌟

Again, in words: $(s, e, s')$ is a transition when $s$ is a state; where $s'$ satisfies $T_{\mathrm{static}}(s') \cup E(s, e, s')$ and no other interpretation does so; and where $e$ satisfies $A(s, e)$ and no other interpretation in $\sigma^a$ does so.

The transition systems defined by a $\mathcal{C}+$ action description have the form given above in Definition 1: the component $S$ is the set of states defined by the action description $D$; the possible labels $L$ are found in $\mathrm{I}(\sigma^a)$; and $R$ is the set of transitions, as given in Definition 4.

**Theorem 5** If $(s, e, s')$ is a transition of an action description $D$, then $s'$ is a state.

*Proof:* This is Proposition 4 of [Sergot, 2004]. ⌟

At a first glance the reason that these definitions take precisely the form they do is far from obvious, and they are apt to seem somewhat arbitrary. Matters become clearer when we examine the relationship of action descriptions of $\mathcal{C}+$ to causal theories.

## 2.3 Causal theories

The language of causal theories [McCain and Turner, 1997, Giunchiglia et al., 2004] is a more general-purpose, non-monotonic formalism which can be seen as underlying the action language $\mathcal{C}+$. (The manner in which it is non-monotonic is not entirely straightforward: for details, see [Sergot and Craven, 2005a].) Causal theories are very closely related to Reiter's Default Logic [Reiter, 1980], as Section 7 of [Giunchiglia et al., 2004] shows, and it is partly their scope for a highly nuanced representation of default behaviour which brings causal theories close to $\mathcal{C}+$. It will be seen that $\mathcal{C}+$ action descriptions correspond to families of certain forms of causal theories; but this correspondence does not use the full expressivity of causal theories.

In causal theories we start, as with $\mathcal{C}+$, with a multi-valued propositional signature $\sigma$. In the language of causal theories, however, there is no distinction between fluent constants and action constants—members of $\sigma$ are undifferentiated. Formulas are built up from atoms $c{=}v$ using standard propositional connectives, again including $\top$ and $\bot$ as connectives of zero arity.

A *causal rule* is an expression of the form

$$F \Leftarrow G,$$

where $F$ and $G$ are formulas of the underlying, multi-valued propositional signature. Such expressions are related to the (almost) natural language statement "if $G$, then the fact that $F$ is caused"; perhaps they could better be paraphrased as "if $G$, then there is a reason for $F$ to be true (and $F$ *is* true)". A *causal theory* is a set of causal rules.

Now let $\Gamma$ be a causal theory, and take $X$ to be an interpretation of its underlying propositional signature. The *reduct* of $\Gamma$ with respect to $X$ is defined as

$$\Gamma^X = \{F \mid F \Leftarrow G \in \Gamma \text{ and } X \models G\}.$$

$X$ is a model of the causal theory $\Gamma$, written $X \models_{\mathsf{C}} \Gamma$, if $X$ is the unique model of $\Gamma^X$. (This uniqueness constraint is related to the role of singletons in Definition 4.)

For an illustration of the preceding definitions, consider the causal theory $T_1$, with underlying Boolean signature $\{p, q\}$:

$$p \Leftarrow q$$
$$q \Leftarrow q$$
$$\neg q \Leftarrow \neg q$$

There are clearly four possible interpretations of the signature:

$$X_1\colon\ p \mapsto \mathbf{t},\ q \mapsto \mathbf{t}$$
$$X_2\colon\ p \mapsto \mathbf{t},\ q \mapsto \mathbf{f}$$
$$X_3\colon\ p \mapsto \mathbf{f},\ q \mapsto \mathbf{t}$$
$$X_4\colon\ p \mapsto \mathbf{f},\ q \mapsto \mathbf{f}$$

and it is clear that

$$T_1^{X_1} = \{p, q\} \text{ whose only model is } X_1$$
$$T_1^{X_2} = \{\neg q\} \text{ which has two models}$$
$$T_1^{X_3} = \{p, q\} \text{ whose only model is } X_1 \neq X_3$$
$$T_1^{X_4} = \{\neg q\} \text{ which has two models}$$

In only one of these cases—that of $X_1$—is it true that the reduct of the causal theory with respect to the interpretation has that interpretation as its unique model. Thus $X_1 \models_{\mathsf{C}} T_1$ and $X_1$ is the only model of $T_1$.

See [Sergot and Craven, 2005b] (or its expanded version [Sergot and Craven, 2005a]) for logical properties of causal theories, and relations to modal logic. Also see [Erdogan and Lifschitz, 2006].

## 2.4 Action Descriptions to Causal Theories

Action descriptions of $\mathcal{C}+$ can be seen as shorthand for families of causal theories. The index set is the non-negative integers, which represents the time for which the system runs.

Thus to every action description $D$ of $\mathcal{C}+$—with signature $\sigma$—and non-negative integer $t$, there corresponds a causal theory $\Gamma_t^D$. The signature of $\Gamma_t^D$ contains the constants $c[i]$,[1] such that

- $i \in \{0, \ldots, t\}$ and $c \in \sigma^f$, or
- $i \in \{0, \ldots, t-1\}$ and $c \in \sigma^a$,

and the domains of such constants $c[i]$ are kept identical to those of their constituents $c$ in the signature of the action description. Where $\sigma$ is the signature of the $\mathcal{C}+$ action description $D$, we will let $\sigma_m$ denote the signature of the causal theory $\Gamma_m^D$. The expression $F[i]$, where $F$ is a formula, denotes the result of inserting $[i]$ after every occurrence of a constant in $F$. The causal rules of $\Gamma_t^D$ are:

$$F[i] \Leftarrow F'[i],$$

for every static law $F$ **if** $F'$ in $D$ and every $i \in \{0, \ldots, t\}$;

$$A[i] \Leftarrow G'[i],$$

and for every action dynamic law $A$ **if** $G$ in $D$ and every $i \in \{0, \ldots, t-1\}$;

$$F[i+1] \Leftarrow G[i],$$

for every fluent dynamic law $F$ **after** $G$ in $D$ and every $i \in \{0, \ldots, t-1\}$;

$$c[i+1]{=}v \Leftarrow c[i+1]{=}v \wedge c[i]{=}v$$

for every simple fluent constant $c$, $v \in dom(c)$ and $i \in \{0, \ldots, t-1\}$;

$$a[i]{=}v \Leftarrow a[i]{=}v$$

[1] These are written as $i : k$ in [Giunchiglia et al., 2004]; we find the current notation easier to read.

for every action constant $a$, $v \in dom(a)$, and $i \in \{0, \ldots, t-1\}$; and

$$c[0]{=}v \Leftarrow c[0]{=}v,$$

for every simple fluent constant $c$ and $v \in dom(c)$.

We have already defined the labelled transition systems which are determined by $\mathcal{C}+$ action descriptions, in a way which did not depend at all on the formalism of causal theories. The same transition systems can be defined much more succinctly using causal theories, however, and it is often much more easy when trying to imagine the systems defined by action descriptions, to think in terms of the causal-theoretic definitions rather than those given in Section 2.2. In the current context we will identify interpretations of the underlying propositional signature of $D$ with the sets of atoms they satisfy. Where $i$ is a non-negative integer and $s$ an interpretation, we can write $s[i]$ for the result of suffixing $[i]$ to every constant of an atom made true by the interpretation (in symbols, $\{c[i]{=}v \mid s \models c{=}v\}$).

We trespass on our previous definitions:

**Definition 6** Let $D$ be an action description of $\mathcal{C}+$, with signature $\sigma$.

- A *state* is any $s \in \mathrm{I}(\sigma^f)$, such that $s[0] \models_{\mathsf{C}} \Gamma_0^D$;
- a *transition* is any triple $(s, e, s') \in \mathrm{I}(\sigma^f) \times \mathrm{I}(\sigma^a) \times \mathrm{I}(\sigma^f)$ such that

$$s[0] \cup e[0] \cup s'[1] \models_{\mathsf{C}} \Gamma_1^D. \qquad \lrcorner$$

According to this definition, the component $S$ of the labelled transition systems defined by $\mathcal{C}+$ action descriptions is the set

$$\{s \mid s \in \mathrm{I}(\sigma^f),\ s[0] \models \Gamma_0^D\};$$

the set of labels $L$ is $\mathrm{I}(\sigma^a)$ as before; and the set of edges is the set

$$\{(s, e, s') \mid s, s' \in \mathrm{I}(\sigma^f),\ e \in \mathrm{I}(\sigma^a),\ s[0] \cup e[0] \cup s'[1] \models \Gamma_1^D\}.$$

**Theorem 7** Let $D$ be an action description. Then for any transition $(s, e, s')$, $s'$ is a state, where transitions and states are both defined according to Definition 6.

*Proof:* This is Proposition 7 of [Giunchiglia et al., 2004]. $\lrcorner$

We now have two alternative definitions for the labelled transition systems defined by our action descriptions. The following theorem shows they coincide.

**Theorem 8** Let $D$ be an action description of $\mathcal{C}+$, with signature $\sigma$. Then:

(i) $s$ is a state of $D$ according to Definition 3 iff $s$ is a state of $D$ according to Definition 6;

(ii) $(s, e, s')$ is a transition of $D$ according to Definition 4 iff $(s, e, s')$ is a transition of $D$ according to Definition 6.

In other words, Definitions 3, 4 and 6 coincide.

*Proof:* This is essentially Theorem 9 of [Sergot, 2004]. $\lrcorner$

Let $\Gamma_t^D$ be the causal theory generated from the action description $D$ and non-negative integer $t$ as described above. Let $s_0, \ldots, s_t$ be interpretations of $\sigma^f$ and $e_0, \ldots, e_{t-1}$ be interpretations of $\sigma^a$. Then using the notation above, we can represent interpretations of the signature of $\Gamma_t^D$ in the form

$$s_0[0] \cup e_0[0] \cup s_1[1] \cup e_1[1] \cup \cdots \cup e_{t-1}[t-1] \cup s_t[t] \tag{5}$$

The following result holds.

**Theorem 9** An interpretation of the signature of $\Gamma_t^D$ is a model of $\Gamma_t^D$ iff each triple $(s_i, e_i, s_{i+1})$, for $0 \leqslant i < t$, is a transition.

*Proof:* Proposition 8 of [Giunchiglia et al., 2004]. ⌟

Let $D$ be an action description of $\mathcal{C}+$. A *run* of length $t$ through this transition system is defined to be a sequence

$$(s_0, e_0, s_1, e_1, \ldots, e_{t-1}, s_t) \tag{6}$$

such that all triples $(s_i, e_i, s_{i+1})$, for $0 \leqslant i < t$, are members of the transition system.

**Theorem 10** Let $D$ be an action description and $t$ any non-negative integer. Then the sequence (6) is a run of the transition system iff the interpretation (5) is a model of the causal theory $\Gamma_t^D$.

*Proof:* First, assume we have a run of the transition system of length $t$. Then every triple $(s_i, e_i, s_{i+1})$, for $0 \leqslant i < t$, is a transition, and so by Theorem 9 the interpretation (5) is a model of $\Gamma_t^D$.
Alternately, suppose that (5) is a model of the causal theory $\Gamma_t^D$, Then clearly, each triple $(s_i, e_i, s_{i+1})$, for $0 \leqslant i < t$, is a transition, and so the sequence (6) is a run of the transition system defined by $D$. ⌟

## 2.5 Definiteness and Completion

A causal theory $\Gamma$ is said to be *definite* when:

- the head of every causal rule is either an atom or $\bot$, and
- no atom is the head of infinitely many causal rules.

Clearly this definition is related to that of the definiteness of an action description of $\mathcal{C}+$, given in Section 2.1. Indeed, it turns out that when an action description $D$ is definite, then the causal theories $\Gamma_t^D$, for $0 \leqslant t$, are also definite.

Models of a definite causal theory can be found by first 'completing' the causal theory, transforming it to a set of formulas of propositional logic. The models of the propositional logic formulas are the same as the models of the causal theory. (This process is structurally similar to what happens in the Clark completion used in the semantics of logic programs [Clark, 1978]—though the Clark completion is used to *define* a kind of semantics; we make no reference to completion in the definition of models.) Once the completion of the causal theory has been found, the models of the resulting propositional theory can

The latter problem can be shipped out to a propositional satisfaction solver. So, in the light of Theorem 10, we have a means of finding runs of length $t \leqslant 0$ through the transition system defined by a definite action description of $\mathcal{C}+$.

Let $\Gamma$ be a definite causal theory, of which we wish to find the completion, and let $\Gamma$ have signature $\sigma$. An atom $c{=}v$ of $\sigma$ is said to be *trivial* when $dom(c) = \{v\}$: in this case any interpretation of $\sigma$ must assign to $c$ the value $v$. (We also call the constant $c$ trivial in this case.) We let *Trivial*($\sigma$) be the set of atoms $c{=}v$ of $\sigma$ such that $c$ is trivial. For each non-trivial atom $c{=}v$ of $\sigma$, the *completion formula* is:

$$c{=}v \equiv G_1 \vee \cdots \vee G_n,$$

where

$$c{=}v \Leftarrow G_1, \ldots, c{=}v \Leftarrow G_n$$

are all the rules in $\Gamma$ with head $c{=}v$. The *completion* of $\Gamma$ is the set of all completion formulas of all non-trivial atoms in $\sigma$, together with all formulas $\neg F$ for each rule

$$\bot \Leftarrow F$$

in $\Gamma$. We let $comp(\Gamma)$ be the completion of $\Gamma$; this is:

$$\{c{=}v \equiv G_1 \vee \cdots \vee G_n \mid c{=}v \in \mathit{Trivial}(\sigma), \forall G(c{=}v \Leftarrow G \in \Gamma \leftrightarrow \exists i(0 \leqslant i \leqslant n \wedge G = G_i))\}$$
$$\cup \{\neg F \mid \bot \Leftarrow F \in \Gamma\}$$

**Theorem 11** The models of a definite causal theory $\Gamma$ are precisely those of its completion.

*Proof:* Proposition 6 of [Giunchiglia et al., 2004]. ⌟

# 3 Basic formalism

We first cover the case of a centralized set of policies: where a single set of policies holds for the entire agent society or distributed system. There may be—indeed, in all examples, there will be—different agents, but they are all subject to the same set of policies. Later sections will consider the case where these multiple agents are subject to different policy sets.

## 3.1 Signature

In this section we add more structure to the signatures of $\mathcal{C}+$ that we will use in representing policies. The language presented here will later be further enriched to accommodate named sets of policies, and expressions for defining an algebra over such named policies; for the moment, we keep to the essentials.

Sets, may be empty, except where stated to be otherwise. Let $\mathsf{Ag}$ be a non-empty set of *agents*; this set will normally be written $\{ag_1, \ldots, ag_n\}$. For each $ag_i \in \mathsf{Ag}$, there is a set of multi-valued *fluents*, $\mathsf{flu}_i$. We define, for each $i$ such that $1 \leqslant i \leqslant n$, a set $\mathsf{Flu}_i = \{ag_i{:}f \mid f \in \mathsf{flu}_i\}$ of *fluent constants*; each member $ag_i{:}f$ has the same domain as its component $f$. There is also a set $\mathsf{Flu}$ of multi-valued fluent constants which can be considered to represent properties of the environment, which is not an agent. The set $\mathsf{Fluents} = \mathsf{Flu} \cup \mathsf{Flu}_1 \cup \cdots \cup \mathsf{Flu}_n$ includes all fluent constants. We use letters $f$ to refer both to members of $\mathsf{flu}_i$ and $\mathsf{Fluents}$. For each $f \in \mathsf{Fluents}$ there is a non-empty set $dom(f)$, the *domain* of $f$. If a fluent constant $f$ has domain $\{\mathbf{t}, \mathbf{f}\}$, it is *Boolean*. If $f \in \mathsf{Fluents}$ and $v \in dom(f)$, the expression $f{=}v$ is known as a *fluent atom*. For a Boolean fluent constant $c$, we usually abbreviate $c{=}\mathbf{t}$ to $c$, and $c{=}\mathbf{f}$ to $\neg c$ ($\neg c$ is therefore an atom). Some members of $\mathsf{Fluents}$ are simple fluent constants, and some are statically determined fluent constants; we often refer to members of $\mathsf{Flu}$ or $\mathsf{Flu}_i$, or $\mathsf{flu}_i$ as 'simple' or 'statically determined' if the members of $\mathsf{Fluents}$ they give rise to are simple or statically determined.

$\mathsf{Sub} \subseteq \mathsf{Ag}$ is the set of *subjects*, $\mathsf{Tar} \subseteq \mathsf{Ag}$ is the set of *targets*; these need not be disjoint, but $\mathsf{Sub}$ must be non-empty. There is also a set $\mathsf{Act}$ of *actions*. The function $subjects : \mathsf{Act} \to \wp(\mathsf{Sub})$ gives, for each action, the set of subjects that can perform it—the subjects that have the capacity or ability, not those who are *permitted*. Analgously, $targets : \mathsf{Act} \to \wp(\mathsf{Tar} \cup \{\bot_{tar}\})$ gives the set of possible targets. (We allow the possibility that it may be easier to model some actions as not having a target: this is the role of $\bot_{tar}$.) For $act \in \mathsf{Act}$, if $sub \in subjects(act)$ and $tar \in targets(act)$, then *sub:act:tar* is a Boolean action constant—we often call these simply *actions*, where that does not lead to confusion with the members of $\mathsf{Act}$. These actions *sub:act:tar* are the equivalent of what is often represented, in work on policies, as $done(X)$, where $X$ is some appropriate access or action in the system being modelled. Where *tar* is $\bot_{tar}$, then *sub:act:tar* may be abbreviated to *sub:act*. The set of all these is $\mathsf{Actions}$: this is, to stress the point, a set of *Boolean* action constants.[2] $\mathsf{Actions}$ is partitioned into two sets $\mathsf{Actions}_{eca}$ and $\mathsf{Actions}_{reg}$; this will be of relevance later in the discussion of obligations—we do not explain it more here. There is also a set $\mathsf{Events}$ of *events*, constants which represent events in the system being modelled which are not controlled by policies. The

[2]It is entirely possible to use *multi-valued* action constants instead, with a (very minor) complication of syntax. However, in this case, defaults for which values the members of $\mathsf{Actions}$ take—to deal with the case where there are no requests for that member, or all requests are denied—need to be stated explicitly in the action description. In the current version of the language, we assume that the negations of members *Sub:Act:Tar* of $\mathsf{Actions}$ are true by default: which can be glossed as an assumption that actions not requested, by default do not happen.

members of Events are multi-valued $\mathcal{C}+$ action constants. We let Happenings = Actions ∪ Events. As have fluent constants, each member $a \in$ Happenings has a non-empty domain $dom(a)$, and the same conventions are observed for happening constants as for fluent constants; there are therefore *happening atoms*. The set Fluents ∪ Happenings is known as that of *system constants*, and abbreviated to Sys.

The constants introduced so far are enough to represent domains in which subjects perform actions on targets, these have certain effects on the subjects and targets and environment, and events in the environment play their part. However, where we are modelling the effects of *policies* on systems (which will be our objective in the present section), then additional constants are used, to represent the policies and the way they interact with the subjects and targets.

All new policy constants we will add are Boolean action constants. First, for all *Sub:Act:Tar* ∈ Actions, there are action constants *permitted*(*Sub:Act:Tar*) and *denied*(*Sub:Act:Tar*); if we have *Sub:Act:Tar* ∈ $\mathsf{Actions}_{eca}$, then we will also add action constants *obligation*(*Sub:Act:Tar*). These three types of constants are used in the specification of positive and negative authorization policies, and obligation policies. Secondly, to represent the fulfilment and violation of obligations, we have for each *Sub:Act:Tar* ∈ $\mathsf{Actions}_{eca}$ a constant *fulfilled*(*Sub:Act:Tar*) and a constant *violated*(*Sub:Act:Tar*). Thirdly, for all *Sub:Act:Tar*, a constant [*Sub* **requests** *Act* **on** *Tar*], which represents that *Sub* has sent a request to the policy enforcement point (PEP) or reference manager to perform *Act* on *Tar*. We let the set of all [*Sub* **requests** *Act* **on** *Tar*] constants be Requests. Fourthly, again for all *Sub:Act:Tar* ∈ Actions, there is a constant [*Sub:Act:Tar* **is allowed**], which is used to mean that a request to the PEP from *Sub* to perform *Act* on *Tar* is granted. Let the set of all these new constants be Pol. We refer to the members of Sys and Pol collectively as *constants*.

As $\mathcal{C}+$ is our base language, we make it clear that using the notation normally employed for $\mathcal{C}+$ action descriptions, $\sigma^f$ = Fluents and $\sigma^a$ = Happenings ∪ Pol. Signatures $\sigma$ of our language are given by $\sigma = \sigma^f \cup \sigma^a$, as usually for $\mathcal{C}+$.

## 3.2 System Models

In order to model the effects of actions on state variables and properties of systems, we write laws of $\mathcal{C}+$. The system representations are kept, as far as possible, distinct from the representations of policies. For a signature $\sigma$ (as given in Section 3.1), a *system description* is defined to be an action description of the restricted version of $\mathcal{C}+$ defined in Section 2, using as its signature only $\sigma^s$ = Fluents ∪ Happenings: the policy action constants Pol are therefore excluded. System descriptions are action descriptions of $\mathcal{C}+$, with a little additional structuring of the underlying signatures.

System descriptions, by themselves, represent domains in which there is no policy apparatus in place to control the performance of actions. Accordingly, there is no gap between the request for an action and that action's performance, and any action immediately produces its effects as described in the causal laws of the action description.

For example, consider a very simple system, in which two user accounts *left* and *right* can read a file *file*; when an account $x$ reads the file, *x:hasRead*(*file*) becomes true. The system description, $A_1^S$, can be represented as in Table 1. Its transition system is shown in Figure 2. In the figure, we have abbreviated fluents *left:hasRead*(*file*) and *right:hasRead*(*file*) by *l:hr* and *r:hr*, and abbreviated the actions (*left:read:file*) and (*right:read:file*) by $l$ and $r$, all for the sake of readability.

It is often useful to have a logically separable representation of the way a system evolves, as this enables one to reason more easily about the sets of configuration states the system supports. Adding policies, and the representation of how they are implemented and enforced, does not affect the set of possible states the system can be in, but only the transitions between those states, and thus the possible histories of the system (where histories are conceived as runs through the resultant labelled transition system). Thus if a reasoning task requires the ability to look merely at state properties, then doing so on the bare system description and the transition system it defines could be more clear. Further, bare system descriptions of the sort depicted in Figure 2 provide a starting point for the design of policies: in they can be viewed as showing the 'most permissive' policy (without explicitly having the policy action constants), which allows every action requested, and

| | | |
|---|---|---|
| $\mathsf{Sub} = \{\mathit{left}, \mathit{right}\}$ | | |
| $\mathsf{Act} = \{\mathit{read}\}$ | | |
| $\mathsf{Tar} = \{\mathit{file}\}$ | | |
| $\mathsf{Fluents} = \{\mathit{Ag}{:}\mathit{hasRead}(\mathit{file}) \mid \mathit{Ag} \in \mathsf{Sub}\}$ | | |
| $\mathsf{Actions} = \{(\mathit{Ag}{:}\mathit{read}{:}\mathit{file}) \mid \mathit{Ag} \in \mathsf{Sub}\}$ | | |
| $\mathsf{Events} = \{\}$ | | |
| System Laws: | $\mathit{Ag}{:}\mathit{hasRead}(\mathit{file})$ **after** $(\mathit{Ag}{:}\mathit{read}{:}\mathit{file})$ | $(\forall \mathit{Ag} \in \mathsf{Sub})$ |
| | **inertial** $F$ | $(\forall F \in \mathsf{Fluents})$ |
| | **exogenous** $A$ | $(\forall A \in \mathsf{Actions})$ |

Table 1: Simple action description $A_1^S$

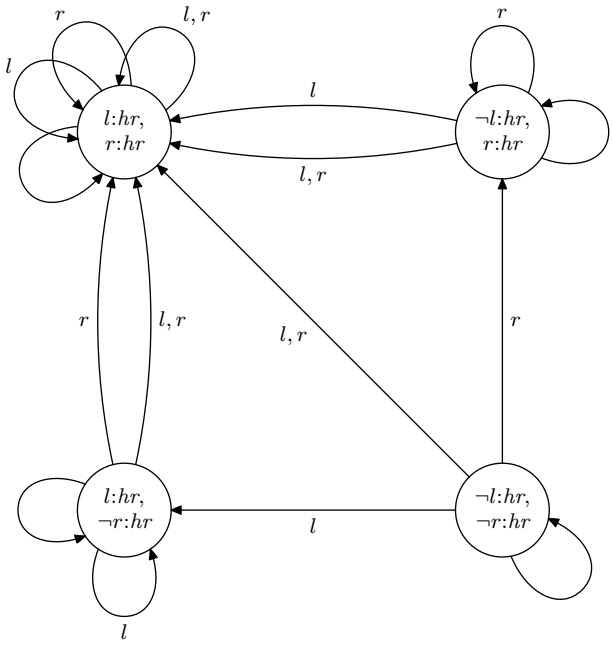


Figure 2: Transition system for system description $A_1^S$.

permits requests at any time. Policy design then amounts to identifying the transitions considered undesirable, and writing policies to exclude them.[3]

## 3.3 Policies

Policies govern a system, consisting of the subjects and targets referred to in the policy, other objects, and the properties of all. The properties of the system vary over time, as a consequence both of the actions that are performed according to policies, and events occuring which do not

[3]This idea is used later, when norms are brought in and their relations to policies studied.

fall within the policy's scope. These system properties themselves determine the applicability of a policy in a given state. Policies partially determine the way the system behaves: a request by a subject to perform an action will be allowed or denied depending on what the authorization policies imply, and thus certain system transitions are made possible, others not, by the authorization policy being used. Obligation policies trigger requests for actions under stated conditions.

In order to represent the operation of policies, we try and stay close, in the structure of our action descriptions, to the nature of the policy architecture we presuppose. This will perhaps come more to the fore in Section 4, where we consider policy composition and enrich our language for it, but the effects of staying close to policy architecture are felt here too.

A *positive authorization rule* has the form

$$\mathit{permitted}(\mathit{Sub{:}Act{:}Tar}) \textbf{ if } G \tag{7}$$

where $\mathit{Sub{:}Act{:}Tar} \in \mathsf{Actions}$ and $G$ is a formula. The meaning is that where $G$ is true, then *Sub* is permitted, according to the policy, to perform *Act* on *Tar*. In the case where $G$ is a conjunction $A \wedge F$, with $A$ a happening formula and $F$ a fluent formula, then the meaning is that if $F$ is true in a state, and the actions and events $A$ occur, then *Sub:Act:Tar* is permitted. If $G$ is $\top$, then the rule may be written simply as

$$\mathit{permitted}(\mathit{Sub{:}Act{:}Tar})$$

A *negative authorization rule* has the form

$$\mathit{denied}(\mathit{Sub{:}Act{:}Tar}) \textbf{ if } G \tag{8}$$

In other respects, the same things are true of negative as of positive authorization rules. Such rules are intended to mean that when $G$ is true, then the action *Sub:Act:Tar* is explicitly denied by the policy.

An *obligation rule* has the form

$$\mathit{obligation}(\mathit{Sub{:}Act{:}Tar}) \textbf{ if } G \tag{9}$$

Conditions on abbreviations are as before; the meaning is, of course, that if $F$ is true in a state and $A$ occur, then *Sub:Act:Tar* is obligatory: *Sub* must perform *Act* on *Tar*.

A set of positive authorization rules is a *positive authorization policy*; a set of negative authorization rules is a *negative authorization policy*; a set of obligation rules is an *obligation policy*. If a set has both positive and negative authorization rules, it is an *authorization policy*. A *policy* can contain all three types of rule. Where confusion is unlikely to result, a single rule is also sometimes referred to as a policy.

Policies are of little interest to us without an underlying description of the system they govern; we therefore join policies to the system descriptions presented in the previous section. In doing this, the assumption we made for system descriptions, that actions can be performed in any state that the system supports them in, regardless of any other controls, must be abandoned: it is precisely the function of a policy to provide additional control over the behaviour of the system.

This means that the relationship between actions, requests for actions, and the effects of actions must be represented. We introduce the action constants [*Sub:Act:Tar* **is allowed**] and [*Sub* **requests** *Act* **on** *Tar*], for all $\mathit{Sub{:}Act{:}Tar} \in \mathsf{Actions}$, for this purpose. We also alter the nature of system descriptions when they are yoked to policies: in action constraints $A$ **if** $G$, $A$ may no longer contain members of Actions; it *may* contain, however, constants of the form [*Sub* **requests** *Act* **on** *Tar*]—and $A$ may still contain members of Events. The removal of members of Actions signals that whether or not an action, having form *Sub:Act:Tar*, is to be performed, will now depend on the policy point; the allowance of [*Sub* **requests** *Act* **on** *Tar*] constants in the heads of action constraints signals that whether or not a request for an action is made will *not* depend directly on the policy system.[4]

[4]In some policy systems, such as Ponder2 [Damianou et al., 2001], making a request must also be allowed by a policy. We could model such complication here—it is quite straightforward—but have chosen not to.

Policies need one other kind of rule, aside from the authorization and obligation rules we have introduced so far. This is to express the interface between PDP and PEP, and for this we use the action constants [*Sub:Act:Tar* **is allowed**]. A *policy description* is a policy supplemented with a set of *allowance rules*, which are action constraints of the form

$$[\textit{Sub:Act:Tar}\ \textbf{is allowed}]\ \textbf{if}\ F \tag{10}$$

where $F$ conjoins atoms of constants *permitted*(*Sub:Act:Tar*) and *denied*(*Sub:Act:Tar*). For example, a specific allowance rule might be

$$[\textit{Sub:Act:Tar}\ \textbf{is allowed}]\ \textbf{if}\ \textit{permitted}(\textit{Sub:Act:Tar}) \wedge \neg \textit{denied}(\textit{Sub:Act:Tar})$$

The effect of this will be to allow any requests for an action *Sub:Act:Tar* that are both permitted and not denied by the policy. When the semantics of policy descriptions is given, it will also be clear that the logic of the system is such that the law

$$(\textit{Sub:Act:Tar})\ \textbf{if}\ [\textit{Sub:Act:Tar}\ \textbf{is allowed}] \wedge [\textit{Sub}\ \textbf{requests}\ \textit{Act}\ \textbf{on}\ \textit{Tar}] \tag{11}$$

is implied. However, we do not include this explicitly in policies.

So far, we have been considering only the relationship of authorizations to the system; obligations are treated somewhat differently. We believe that there two principle ways of using obligations in policies. The first is as statements of event-condition-action (ECA) rules, and the second is as the cue for a monitoring system that takes action in response to the obligation's fulfilment or violation. In the first case, we suppose the way the policy system enforces the obligation is by triggering a request for the action; this request must then be authorized in just the same way as an unprompted request by a subject within the system. In the second case, the action itself is not prompted by the policy system, but depends on a request by a subject. However, laws can be written that express the response to the violation or fulfilment of the obligation.

To allow for this in our policy descriptions, we do two things. The first is to make a division, in our signatures, between those actions which may be within the scope of an ECA rule, and those which, when they are the subject of obligations, may only be monitored (the division does not affect the expression of authorization policies). The intuition here is that some actions are within the control of the policy system, and others are not, and may merely be monitored, with the policy system taking action in response to the obligations' status. This is the prupose of the partioning of our set $\mathsf{Actions}$ as $\mathsf{Actions}_{eca} \cup \mathsf{Actions}_{reg}$, where $\mathsf{Actions}_{eca}$ and $\mathsf{Actions}_{reg}$ are disjoint. The semantics of our policy descriptions will be such that for all $\textit{Sub:Act:Tar} \in \mathsf{Actions}_{eca}$, it is as though the action constraint

$$[\textit{Sub}\ \textbf{requests}\ \textit{Act}\ \textbf{on}\ \textit{Tar}]\ \textbf{if}\ \textit{obligation}(\textit{Sub:Act:Tar}) \tag{12}$$

were present in the system description.

To model the fulfilment and violation of obligations, we have the following laws, whose presence is felt semantically, but which are not included explicitly:

$$\textit{fulfilled}(\textit{Sub:Act:Tar})\ \textbf{if}\ \textit{obligation}(\textit{Sub:Act:Tar}) \wedge (\textit{Sub:Act:Tar}) \tag{13}$$

$$\textit{violated}(\textit{Sub:Act:Tar})\ \textbf{if}\ \textit{obligation}(\textit{Sub:Act:Tar}) \wedge \neg(\textit{Sub:Act:Tar}) \tag{14}$$

For a $\textit{Sub:Act:Tar} \in \mathsf{Actions}_{reg}$, the usual procedure will be to include obligation policies that respond to its violation and fulfilment, where the obligation policy refers to actions $\textit{Sub:Act:Tar} \in \mathsf{Actions}_{eca}$. For instance, suppose there is an obligation on a user to release a lock on a password file if there is an emergency status, but that the system cannot control whether the user does this or not. If the user does not release the lock, the system will impose a punishment by putting the user's home directory into a 'red' category. This can be represented as follows (we use abbreviations of

Boolean constants):

$$\mathit{obligation}(U{:}\mathit{release}{:}F) \textbf{ if } U{:}\mathit{isUser} \wedge F{:}\mathit{fileType}(\mathit{passwd}) \wedge F{:}\mathit{hasLock}{=}U \wedge \mathit{emergency}$$

$$\mathit{obligation}(\mathit{system}{:}\mathit{putCat}(\mathit{red}){:}D) \textbf{ if } \mathit{violated}(U{:}\mathit{release}{:}F) \wedge U{:}\mathit{isUser} \wedge F{:}\mathit{fileType}(\mathit{passwd}) \wedge F{:}\mathit{hasLock}{=}U \wedge \mathit{emergency} \wedge U{:}\mathit{homeDir}{=}D \quad (15)$$

$$\mathit{permitted}(\mathit{system}{:}\mathit{putCat}(\mathit{red}){:}D) \textbf{ if } U{:}\mathit{isUser} \wedge U{:}\mathit{homeDir}{=}D \quad (16)$$

$$[\mathit{Sub}{:}\mathit{Act}{:}\mathit{Tar} \textbf{ is allowed}] \textbf{ if } \mathit{permitted}(\mathit{Sub}{:}\mathit{Act}{:}\mathit{Tar})$$

$$\neg F{:}\mathit{hasLock}{=}\textbf{f after } (U{:}\mathit{release}{:}F)$$

$$D{:}\mathit{hasCat}(\mathit{red}) \textbf{ after } (\mathit{Sub}{:}\mathit{putCat}(\mathit{red}){:}D)$$

The first four laws are policies; the fifth states that when a user releases the lock on a file, this causes the user no longer to have the lock; the sixth states that a user's home directory is put into a 'red' category when the system assigns it to that category. *emergency* is a member of Events. Where $U$ is the name of an account, and $F$ a filename, we suppose $(U{:}\mathit{release}{:}F) \in \mathsf{Actions}_{reg}$, but that $(\mathit{system}{:}\mathit{Act}{:}\mathit{Tar}) \in \mathsf{Actions}_{eca}$, for all *Act* and *Tar* such that $(\mathit{system}{:}\mathit{Act}{:}\mathit{Tar}) \in \mathsf{Actions}$. If, in a given state $s$, some user has a lock on a password file and an emergency is declared, then that user is obliged to release the lock—so that all transitions $(s, e, s')$ from $s$ have $e \models \mathit{obligation}(U{:}\mathit{release}{:}F)$. Whether or not the user makes the release will depend on two things: a request's being made, and that request's being authorized, by an authorization policy also present (obligations do not automatically include the authorization to satisfy them). Let us suppose the user does not fulfil the obligation: it is violated, and so $e \models \mathit{violated}(U{:}\mathit{release}{:}F)$. In that case, the other obligation is activated: that the system should put the user's home directory in the 'red' category. Across the same transition $(s, e, s')$, we therefore have for $U$'s directory $D$, that $e \models \mathit{obligation}(\mathit{system}{:}\mathit{putCat}(\mathit{red}){:}D)$. As this action is a member of $\mathsf{Actions}_{eca}$, the law

$$[\mathit{system} \textbf{ requests } \mathit{putCat}(\mathit{red}) \textbf{ on } D] \textbf{ if } \mathit{obligation}(\mathit{system}{:}\mathit{putCat}(\mathit{red}){:}D)$$

will be present, and a request for change in categories will automatically be fired, in the manner of ECA rules, so that $e \models [\mathit{system} \textbf{ requests } \mathit{putCat}(\mathit{red}) \textbf{ on } D]$. Whether the change in categories is made then depends on the presence of an authorization policy allowing it; in our case, we have such a policy (16). The semantics therefore means that in the successor state $s'$ in the transition $(s, e, s')$, we must have $s' \models D{:}\mathit{hasCat}(\mathit{red})$.

Note that, as $(\mathit{system}{:}\mathit{Act}{:}\mathit{Tar}) \in \mathsf{Actions}_{eca}$, and $\mathit{emergency} \in \mathsf{Events}$, the second policy (15) has a structure that corresponds directly to the ECA rule (in a stereotypical syntax):

$$\begin{aligned} \textsc{On}: &\ \mathit{emergency} \\ \textsc{If}: &\ U{:}\mathit{isUser} \wedge F{:}\mathit{fileType}(\mathit{passwd}) \wedge F{:}\mathit{hasLock}{=}U \\ &\ \wedge \mathit{violated}(U{:}\mathit{release}{:}F) \wedge U{:}\mathit{homeDir}{=}U \\ \textsc{Do}: &\ (\mathit{system}{:}\mathit{putCat}(\mathit{red})){:}D) \end{aligned}$$

In the domain being modelled, the sequence of events described above is shown in Figure 3. The two states between which the system moves, $s$ and $s'$, are shown at either end of a sequence of actions and events. In our model, this sequence is compressed, and modelled as occurring over a single transition in the labelled transition system defined by the policy description. This greater abstraction and concision in the graphical depiction of systems does not alter the logic of policy decisions or the effects of actions.

It is worth stressing again that a policy decision—a positive or negative authorization, or an obligation—in a given state can depend on the other actions that are performed in that state. This is somewhat unusual. For example, consider a computer system where two user accounts exist, *left* and *right*. Files (to keep it simple, we suppose there is just one, *file*) may be accessed at a given time by one or other user, but not simultaneously by both. This can be expressed in the action description shown in Table 2, the transition system for which is shown in Figure 4. Certain abbreviations have been made when representing elements of the signature: *left* and *right* are $l$

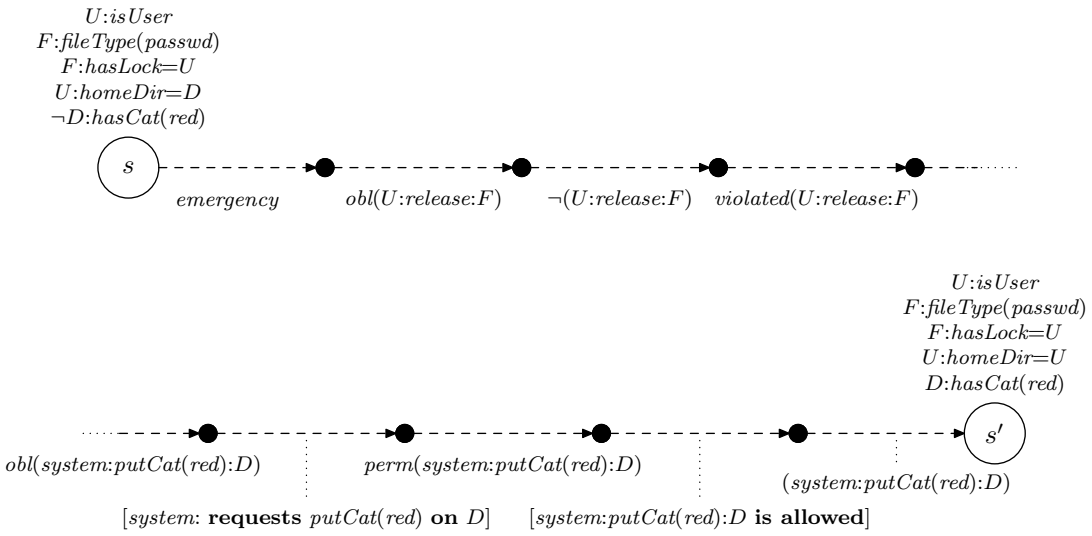


Figure 3: Domain flow diagram for 'release lock' example

$$\begin{aligned}
\mathsf{Sub} &= \{\mathit{left}, \mathit{right}\} \\
\mathsf{Act} &= \{\mathit{read}\} \\
\mathsf{Tar} &= \{\mathit{file}\} \\
\mathsf{Fluents} &= \{\mathit{Ag}{:}\mathit{hasRead}(\mathit{file}) \mid \mathit{Ag} \in \mathsf{Sub}\}
\end{aligned}$$

| | | |
|---|---|---|
| System Laws: | $\mathit{Ag}{:}\mathit{hasRead}(\mathit{file})$ **after** $(\mathit{Ag}{:}\mathit{read}{:}\mathit{file})$ | $(\forall \mathit{Ag} \in \mathsf{Sub})$ |
| | **inertial** $F$ | $(\forall F \in \mathsf{Fluents})$ |
| Policies: | $\mathit{permitted}(\mathit{Ag}{:}\mathit{read}{:}\mathit{file})$ **if** $\neg(\mathit{Ag}'{:}\mathit{read}{:}\mathit{file})$ | $(\forall \mathit{Ag}, \mathit{Ag}' \in \mathsf{Sub} : \mathit{Ag} \neq \mathit{Ag}')$ |
| | $[\mathit{Sub}{:}\mathit{Act}{:}\mathit{Tar}$ **is allowed**$]$ **if** $\mathit{permitted}(\mathit{Sub}{:}\mathit{Act}{:}\mathit{Tar}) \wedge [\mathit{Sub}$ **requests** $\mathit{Act}$ **on** $\mathit{Tar}]$ | |

Table 2: Simple policy description $A_1$ for our language

and $r$; *file* has been removed, as there is only one such file; *read* has also been removed, as it is the only act; requests by *left* to perform a *read* on *file* are depicted as $l{:}req$—and similarly for the other policy-related actions and the other agent. Note that we have not yet described how to generate the sort of transition systems depicted in Figure 4; that will be shown soon.

Policy description $A_1$ in effect exhibits a kind of circular dependency in the logic of its policy: for *left* to be permitted, *right* must not be doing; *right* does if it is allowed; *right* is allowed if it is permitted; it is permitted if *left* does not do; *left* does if it is permitted. It is a particular feature of the semantics of $\mathcal{C}+$ and of the semantics of the causal theories on which $\mathcal{C}+$ depends, that such circular dependencies in causal laws can exist without removing the possibility of finding models for them. This fact enables us to write policies that have much more direct circular dependencies: a policy that permits Romulus to go to Rome if Remus is not permitted to go to Rome, and permits Remus to go to Rome if Romulus is not permitted, is easy to support. In many policy languages, and evaluation frameworks, policies that have such cycles through negation cannot be

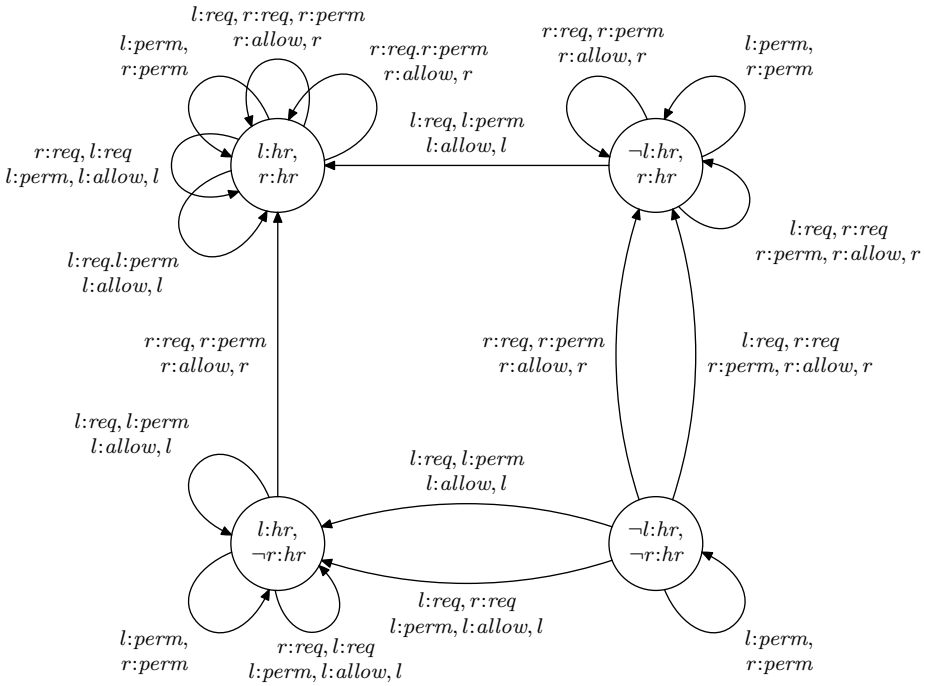


Figure 4: Transition system for action description in Table 2

supported, and result in failure.

In many treatments of deontic logic, there is a close relationship between permission and obligation; one is frequently defined in terms of the other, so that an obligation to do $a$ is defined as the lack of permission to do $\neg a$—or a permission to do $a$ as the lack of an obligation to do $\neg a$. Research on norm-governed systems of agents, drawing as it does more on deontic logic as a branch of modal logic, has often inherited this relationship. In research into the policy-based management of distributed systems, in contrast, the modalities are usually more separate. This is partly because of the particular operational focus of that work. Authorization policies then guide whether or not a request results in the execution of an action. Obligation policies can be understood as event-condition-action rules, in the presence and applicability of which an action is automatically attempted. (Alternatively, obligations can be seen as conditions attached to the award of authorizations.) Where obligations are interpreted as event-condition-action rules, it may be necessary for the action that is fired to have an associated permission.

In the current paper, the modalities are also kept distinct—we do not follow the tradition of deontic logic that strongly relates them. We take them to be similar to event-condition-action rules, in that where the body of an obligation policy rule is true, this automatically fires a request for the action referred to in the head to be performed.

## 3.4 Semantics for policy descriptions

The semantics for policy descriptions is given in terms of labelled transition systems. This can be done directly, by defining how to each policy action description a labelled transition system corresponds, or indirectly, by showing how to translate policy action descriptions of our language into $\mathcal{C}+$ proper, and then noting that, of course, every action description of $\mathcal{C}+$ defines a labelled transition system. Theorems then demonstrate that the transition systems defined by the indirect and the direct methods are the same.

We first show how to construct labelled transition systems for our policy descriptions.

Recall that for policy descriptions $P$ with signature $\sigma$,

$$\sigma = \sigma^f \cup \sigma^a \qquad \sigma^f = \mathsf{Fluents}$$
$$\sigma^a = \mathsf{Happenings} \cup \mathsf{Pol}_a \qquad \mathsf{Happenings} = \mathsf{Actions} \cup \mathsf{Events}$$

We need a preliminary definition, for a modified version $A^*(s,e)$ of $A(s,e)$ first found in Definition 2.

**Definition 12** Let $P$ be a policy description, with signature $\sigma$. Then

$$\begin{aligned}
&A^*_P(s,e) = \\
&\quad A_P(s,e) \\
&\quad \cup \{c{=}\mathbf{f} \mid c \in \mathsf{Actions},\ e \models c{=}\mathbf{f}\} \\
&\quad \cup \{c{=}v \mid c \in \mathsf{Requests},\ v \in \mathit{dom}(c),\ e \models c{=}v\} \\
&\quad \cup \{(\mathit{Sub{:}Act{:}Tar}) \mid e \models [\mathit{Sub{:}Act{:}Tar}\ \mathbf{is\ allowed}] \wedge [\mathit{Sub}\ \mathbf{requests}\ \mathit{Act}\ \mathbf{on}\ \mathit{Tar}]\} \\
&\quad \cup \{(\mathit{Sub{:}Act{:}Tar}){=}\mathbf{f} \mid e \not\models [\mathit{Sub{:}Act{:}Tar}\ \mathbf{is\ allowed}] \wedge [\mathit{Sub}\ \mathbf{requests}\ \mathit{Act}\ \mathbf{on}\ \mathit{Tar}]\} \\
&\quad \cup \{[\mathit{Sub}\ \mathbf{requests}\ \mathit{Act}\ \mathbf{on}\ \mathit{Tar}] \mid e \models \mathit{obligation}(\mathit{Sub{:}Act{:}Tar}) \wedge \mathit{Sub{:}Act{:}Tar} \in \mathsf{Actions}_{eca}\} \\
&\quad \cup \{[\mathit{Sub}\ \mathbf{requests}\ \mathit{Act}\ \mathbf{on}\ \mathit{Tar}]{=}\mathbf{f} \mid e \not\models \mathit{obligation}(\mathit{Sub{:}Act{:}Tar}) \wedge \mathit{Sub{:}Act{:}Tar} \in \mathsf{Actions}_{eca}\} \\
&\quad \cup \{\mathit{fulfilled}(\mathit{Sub{:}Act{:}Tar}) \mid e \models \mathit{obligation}(\mathit{Sub{:}Act{:}Tar}) \wedge \mathit{Sub{:}Act{:}Tar}\} \\
&\quad \cup \{\mathit{fulfilled}(\mathit{Sub{:}Act{:}Tar}){=}\mathbf{f} \mid e \not\models \mathit{obligation}(\mathit{Sub{:}Act{:}Tar}) \wedge \mathit{Sub{:}Act{:}Tar}\} \\
&\quad \cup \{\mathit{violated}(\mathit{Sub{:}Act{:}Tar}) \mid e \models \mathit{obligation}(\mathit{Sub{:}Act{:}Tar}) \wedge \neg \mathit{Sub{:}Act{:}Tar}\} \\
&\quad \cup \{\mathit{violated}(\mathit{Sub{:}Act{:}Tar}){=}\mathbf{f} \mid e \not\models \mathit{obligation}(\mathit{Sub{:}Act{:}Tar}) \wedge \neg \mathit{Sub{:}Act{:}Tar}\}
\end{aligned}$$ ⌟

This provides the set of action constants that are caused to be true over the transition $(s,e,s')$. We can now define the labelled transitions formed from policy descriptions.

**Definition 13** Let $P$ be a policy description with signature $\sigma$. The *labelled transition system* defined by $P$ is a triple $(S,L,R)$. $S$, the set of states, is given as in Definition 3: it is the set of interpretations $s$ of $\mathsf{Fluents}$ (i.e., $\sigma^f$) such that

$$\{s\} = \{s' \in \mathrm{I}(\sigma^f) \mid s' \models T_{\mathrm{static}}(s) \cup \mathit{Simple}(s)\}.$$

$L$, the set of labels, is simply $\mathrm{I}(\sigma^a)$, the set of interpretations of the action constants—including the policy constants. Finally, $R$ is the set of transitions $(s,e,s')$, where we define this anew. $(s,e,s')$ is a *transition* of $P$ iff $s$ is a state, and

- $\{e\} = \{e' \in \mathrm{I}(\sigma^a) \mid e' \models A^*_P(s,e)\}$;
- $\{s'\} = \{s'' \in \mathrm{I}(\sigma^f) \mid s'' \models T_{\mathrm{static}}(s') \cup E(s,e,s')\}$. ⌟

As well as this direct semantics for policy descriptions, which shows how labelled transition systems are defined by sets of policies and causal laws, it is possible to translate policy descriptions into $\mathcal{C}+$ action descriptions. This makes it possible to use existing implementations that calculate models of $\mathcal{C}+$ action descriptions, such as iCCalc, to work with our policy descriptions, and answer queries about them.

**Definition 14** Let $P$ be a policy description with signature $\sigma$. The *corresponding $\mathcal{C}+$ action description* $D_P$ has the same signature as $P$, and all its causal laws. It also has the additional laws:

$$\mathbf{exogenous}\ [\mathit{Sub}\ \mathbf{requests}\ \mathit{Act}\ \mathbf{on}\ \mathit{Tar}]$$

for all *Sub:Act:Tar* $\in$ Actions;

**default** $\neg$(*Sub:Act:Tar*)
**default** $\neg$*permitted*(*Sub:Act:Tar*)
**default** $\neg$*denied*(*Sub:Act:Tar*)
**default** $\neg$*obligation*(*Sub:Act:Tar*)
**default** $\neg$*fulfilled*(*Sub:Act:Tar*)
**default** $\neg$*violated*(*Sub:Act:Tar*)
**default** $\neg$[*Sub:Act:Tar* **is allowed**]

for all *Sub:Act:Tar* $\in$ Actions;

(*Sub:Act:Tar*) **if** [*Sub:Act:Tar* **is allowed**] $\wedge$ [*Sub* **requests** *Act* **on** *Tar*]
*fulfilled*(*Sub:Act:Tar*) **if** *obligation*(*Sub:Act:Tar*) $\wedge$ (*Sub:Act:Tar*)
*violated*(*Sub:Act:Tar*) **if** *obligation*(*Sub:Act:Tar*) $\wedge$ $\neg$(*Sub:Act:Tar*)

for all *Sub:Act:Tar* $\in$ Actions; and

[*Sub* **requests** *Act* **on** *Tar*] **if** *obligation*(*Sub:Act:Tar*)

for all *Sub:Act:Tar* $\in$ $\mathsf{Actions}_{eca}$. ⌟

This definition corresponds closely to the augmentations of the set $A_P(s,e)$ introduced in Definition 12. The following proposition shows that the direct labelled transition system semantics of that definition match those determined by the $\mathcal{C}+$ translation given above.

**Proposition 15** Let $P$ be a policy description, with signature $\sigma$. The labelled transition system defined by $P$ according to Definition 13 is the same as the labelled transition system defined by the $\mathcal{C}+$ action description $D_P$, where $D_P$ is as given in Definition 14.

*Proof:* Let $(S_P, L_P, R_P)$ and $(S_{D_P}, L_{D_P}, R_{D_P})$ be the two transition systems. We clearly have $S_P = S_{D_P}$ and $L_P = L_{D_P}$; we need to check whether the transitions are the same. We need to show that for all $(s, e, s')$, $A^*_P(s,e) = A_{D_P}(s,e)$. First, we show $A^*_P(s,e) \subseteq A_{D_P}(s,e)$. Clearly $A_P(s,e) \subseteq A_{D_P}(s,e)$, so assume $c{=}v$ is in one of the additional sets added in $A^*_P(s,e)$, i.e. $c{=}v$ is in $A^*_P(s,e) - A_{D_P}(s,e)$. A case analysis based on the additional laws of $D_P$ and the sets of $A^*_P(s,e) - A_{D_P}(s,e)$ then shows that $c{=}v \in A_{D_P}(s,e)$. The proof in the other direction proceeds similarly. ⌟

# 4 Policy composition

Thus far, we have grouped authorization and obligation rules together into a single set, conceived of as existing at a single, centralized policy point. The set of policies has no internal structure. Many common policy frameworks, however, allow multiple set of rules, the decisions of which are combined in reaching an overall decision for the family of sets. The ways in which the sub-decisions are combined can either be user-defined, or are fixed by the specific framework being used to evaluate the policies. XACML, for example, has a small number of simple *policy combination rules*, which can state whether a 'deny' overrides a 'permit'. Another way of putting this is to say that the positive authorization rules form a set $P^+$, the negative rules form $P^-$, and $P^-$ always takes precedence over $P^+$. In our previous work [Craven et al., 2009], we allowed such precedence rules to be much more context-dependent—with a rich notion of context embracing policy and system history, constraints, and properties of subjects, actions and targets—but without making any kind of logical separation of policies into different sets.

Recent work on policy composition has taken an algebraic approach, allowing names for sets of policies in a policy language, and composition operators for their combination. We have been

particularly interested in the papers [Bruns et al., 2007], [Bruns and Huth, 2008], [Ni et al., 2009] and [Li et al., 2009]. The semantics of policy composition is greatly clarified in this research, and we wish to make use of it, in several ways, here. The main difference between the approaches of the first two papers, from one set of authors, and the second two papers, from another, is in the multiplicity of policy decisions their frameworks allow, and the resultant expressivity of the languages. In the first two papers, a four-valued logic, that of [Belnap, 1977], is used to express policy evaluations. A decision is a member of the set $\{p, d, u, c\}$,[5] where $p$ represents a permitted access, $d$ a denial, $u$ an undefined decision (neither $p$ or $d$), and $c$ a conflict (both $p$ and $d$). The value $u$, can be taken to apply in several different situations: first, where there is no rule in a given policy set that applies to a request, and secondly, if there is an error in the evaluation of a condition of a policy, so that it is unknown what the decision would be. The value $c$ is used to expression the result of using both of two sub-policies, one of which evaluates an access request to $p$ and the other to $d$.

That the $u$ value serves this dual purpose just mentioned is one of the motivations behind the work of the two related papers [Ni et al., 2009] and [Li et al., 2009]; they enrich the values of the policy logic in slightly different, though related ways. They use non-singleton, non-empty subsets of policy evaluation values for representing indeterminacy resulting from errors in the results of a policy evaluation. The underlying policy values of [Li et al., 2009] are $\mathsf{p}$, $\mathsf{d}$, $\mathsf{n/a}$ and $\mathsf{in}$. However, policy evaluations in the algebra are formed of subsets of these values. With respect to a given policy, $\{\mathsf{p}\}$ represents an access request which is permitted by that policy, $\{\mathsf{d}\}$ an access that is denied, and $\{\mathsf{n/a}\}$ a request to which the policy is not applicable. We will discuss $\{\mathsf{in}\}$ in a moment.

A value such as $\{\mathsf{p}, \mathsf{n/a}\}$ is used by the authors in the presence of an evaluation error during a request. For example, if a policy $p_1$ has only the rule

$$\mathit{permitted}(\mathit{Sub}{:}\mathit{read}{:}F) \textbf{ if } \mathit{Sub}{:}\mathit{isUser} \wedge F{:}\mathit{fileType}(\mathit{passwd})$$

then if the database holding information about which groups *Sub* belongs to is corrupt, or inaccessible owing to network problems, at the time of the request of some specific *Sub*, then the PDP does not know whether $\mathit{Sub}{:}\mathit{isUser}$ is true or not. This means that the PDP does not know whether the rule is inapplicable ($\mathsf{n/a}$), or would grant access ($\mathsf{p}$). The uncertainty, a consequence of error, can be modelled as the value $\{\mathsf{p}, \mathsf{n/a}\}$. The set $\{\mathsf{d}, \mathsf{n/a}\}$ represents the analagous situation for a negative authorization rule.

Algebraic operators on policies are defined by pointwise extension of operators over the values; for instance, if a policy $p_1$ evaluates an access request [*Sub* **requests** *Act* **on** *Tar*] to $\{\mathsf{p}\}$, and $p_2$ evaluates it to $\{\mathsf{d}\}$, then a 'deny overrides' policy combination would evaluate to $\{\mathsf{d}\}$, whereas a 'permit overrides' would evaluate to $\{\mathsf{p}\}$. In some situations, however, it is desirable that conflict between two such sub-policies $p_1$ and $p_2$ should not be removed on combination. This is particularly the case when the policy combining them is not the 'top level' of policy in the PDP, whose values are used by the PEP in deciding whether to grant an access. Where one wishes to retain a representation of conflict, the value $\{\mathsf{in}\}$ can be used; this is one of its functions. The other is in the case where a policy evaluation error has occured in a policy $p$ (such as $\{\mathsf{p}, \mathsf{n/a}\}$), but one does not wish to expose the occurrence of this error to a policy $r$ defined using $p$. In such a case, it would be possible to define an intermediate policy $q$, which takes an such error value of $p$ and transforms it to $\{\mathsf{in}\}$.

The value $\{\}$ is used where there are no policy rules in a policy; this is distinguished from $\{\mathsf{n/a}\}$, in which policy rules exist but none is applicable. The values not mentioned—$\{\mathsf{p}, \mathsf{d}, \mathsf{in}\}$, for instance—are used when combinations are made of policies whose evaluations are uncertain, propagating and adding to the uncertainty.

It is clear that, if one were to use the value $\mathsf{in}$ only in the first of the two functions assigned to it in [Li et al., 2009]—that of representing conflict—and use $\mathsf{p}$, $\mathsf{d}$, $\mathsf{n/a}$ and $\mathsf{in}$ as the values for policies rather than the subsets of them, then the result would be in all essentials the same as the approach, based on Belnap logic, of [Bruns et al., 2007]. We therefore see the main expressive difference between the two approaches as being whether or not they allow the explicit depiction

[5] We use $p$ for "permit", instead of their $g$ for "grant".

of failures and errors in the evaluation process. Which policy composition framework we should adopt will then depend on whether such errors are ones we want to represent, and reason about, during policy analysis and refinement. In the current work, we are interested in such errors only when they spring from policy-related processes. For example, if policies are distributed between different organizations, and the information relevant to policy evaluation spread between different sites, then if an organization denies access to retrieve a given credential, this is something we would wish to model. A communications failure in contacting the distant site, though it does affect policy decisions, is not itself policy-based, and we would not wish to model it. This means that a four-valued approach is appropriate in the centralized case, where all policies and information is presumed to be accessible; but in the distributed case, if it is deemed important to model communication and policy decision failures at different sites, then the richer composition semantics using subsets can be adopted. At the moment, we are only considering the centralized case, and so we will use Belnap logic, and the composition operators of [Bruns et al., 2007]. Later on, we will change to the richer framework.

For the moment, we deal with policies consisting only of authorization rules; remarks on the composition of obligation policies are made in Section 8. The language that we introduce in the following section is called $p\mathcal{C}+$.

## 4.1 Language and semantics

In PBel, the policy algebra based on Belnap logic introduced in [Bruns et al., 2007], single policy rules are the 'atoms' of the composition process. We will alter the approach to allow for sets, composed of only positive authorization policies, or only of negative authorization policies. (This can be translated into their approach straightforwardly.) So, we assume a set $\mathsf{PolicyNames} = \{p_1, \ldots, p_n\}$ of *policy names*. The syntax of authorization rules is changed, to show that each such rule is a member of a named set of policies. Thus, a *positive authorization rule* has the form

$$\mathit{permitted}(p, \mathit{Sub{:}Act{:}Tar}) \textbf{ if } G \tag{17}$$

and a negative authorization rule has the form

$$\mathit{denied}(p, \mathit{Sub{:}Act{:}Tar}) \textbf{ if } G \tag{18}$$

where $p \in \mathsf{PolicyNames}$.

We will wish to be able to make use of the basic operations over the Belnap space defined in [Bruns et al., 2007]. To be clear about the nature of these operations, it is useful to visualize the bilattice of values. The relationship between them can be shown as in Figure 5. The bilattice has two orderings. The *truth ordering*, which in its application to policies is better thought of as a *permission ordering*, is marked $t$ in Figure 5. Its least element is $\mathsf{d}$ and its greatest is $\mathsf{p}$, with $\mathsf{n/a}$ and $\mathsf{in}$ being non-comparable. The *information ordering*, marked $k$ in the diagram, has $\mathsf{n/a}$ as its least element, with $\mathsf{in}$ its greatest.

Basic operations over these values are defined; the binary operations are familiar from lattice theory. The operators are $\wedge$, $\vee$, $\neg$ for the permission ordering, and $\otimes$, $\oplus$ and $-$ for the knowledge ordering; in both cases, they are respectively a meet, join, and complement operation. We provide the meanings in Table 3, in order to make later definitions clearer. The order of operands is given alphabetically; the operators are commutative. The operations are lifted from their application to policy values to policies; this means, for example, that if two of our policy names are $p_1$ and $p_2$ (i.e. $p_1, p_2 \in \mathsf{PolicyNames}$), and a given access request is evaluated to $\mathsf{p}$ by $p_1$, but to $\mathsf{n/a}$ by $p_2$, then the policy $p_1 \otimes p_2$ would evaluate to $\mathsf{n/a}$; $p_1 \vee p_2$ would evaluate to $p$; and so on.

In addition to the positive and negative authorization rules as defined above in (17) and (18), two other sorts of expression are added to our language. A *policy composition rule* has the form

$$\textbf{policy } P \textbf{ is } P' \tag{19}$$

where $P$ is a policy index, and $P'$ is a *policy composition*. Policy compositions have their syntax defined by the Backus-Naur form:

$$P ::= p \mid P_1 \wedge P_2 \mid P_1 \vee P_2 \mid \neg P \mid P_1 \oplus P_2 \mid P_1 \otimes P_2 \mid -P \tag{20}$$

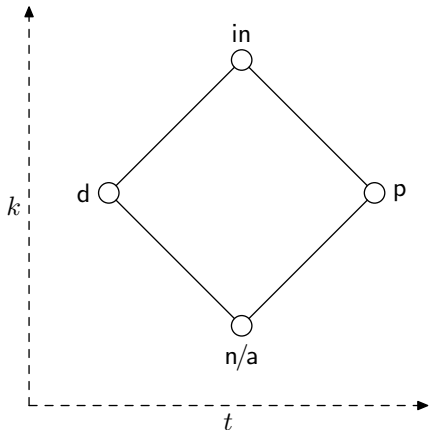


Figure 5: Belnap bilattice for PBel policy decisions

| | | |
|---|---|---|
| $\mathsf{d} \wedge \mathsf{in} = \mathsf{d}$ | $\neg\mathsf{d} = \mathsf{p}$ | $\mathsf{d} \otimes \mathsf{in} = \mathsf{d}$ |
| $\mathsf{d} \wedge \mathsf{n/a} = \mathsf{d}$ | $\neg\mathsf{in} = \mathsf{in}$ | $\mathsf{d} \otimes \mathsf{n/a} = \mathsf{n/a}$ |
| $\mathsf{d} \wedge \mathsf{p} = \mathsf{d}$ | $\neg\mathsf{n/a} = \mathsf{n/a}$ | $\mathsf{d} \otimes \mathsf{p} = \mathsf{n/a}$ |
| $\mathsf{in} \wedge \mathsf{n/a} = \mathsf{d}$ | $\neg\mathsf{p} = \mathsf{d}$ | $\mathsf{in} \otimes \mathsf{n/a} = \mathsf{n/a}$ |
| $\mathsf{in} \wedge \mathsf{p} = \mathsf{in}$ | $-\mathsf{d} = \mathsf{d}$ | $\mathsf{in} \otimes \mathsf{p} = \mathsf{p}$ |
| $\mathsf{n/a} \wedge \mathsf{p} = \mathsf{n/a}$ | $-\mathsf{in} = \mathsf{n/a}$ | $\mathsf{n/a} \otimes \mathsf{p} = \mathsf{n/a}$ |
| $X \wedge X = X$ | $-\mathsf{n/a} = \mathsf{in}$ | $X \otimes X = X$ |
| $\mathsf{d} \vee \mathsf{in} = \mathsf{in}$ | $-\mathsf{p} = \mathsf{p}$ | $\mathsf{d} \oplus \mathsf{in} = \mathsf{in}$ |
| $\mathsf{d} \vee \mathsf{n/a} = \mathsf{n/a}$ | | $\mathsf{d} \oplus \mathsf{n/a} = \mathsf{d}$ |
| $\mathsf{d} \vee \mathsf{p} = \mathsf{p}$ | | $\mathsf{d} \oplus \mathsf{p} = \mathsf{in}$ |
| $\mathsf{in} \vee \mathsf{n/a} = \mathsf{p}$ | | $\mathsf{in} \oplus \mathsf{n/a} = \mathsf{in}$ |
| $\mathsf{in} \vee \mathsf{p} = \mathsf{p}$ | | $\mathsf{in} \oplus \mathsf{p} = \mathsf{in}$ |
| $\mathsf{n/a} \vee \mathsf{p} = \mathsf{p}$ | | $\mathsf{n/a} \oplus \mathsf{p} = \mathsf{p}$ |
| $X \vee X = X$ | | $X \oplus X = X$ |

Table 3: Meanings of basic policy operators

where $p$ ranges over policy indices. Finally, for each policy description a single *top policy designator* is required, which is a statement of the form

$$\textbf{policy } P \textbf{ is top} \tag{21}$$

where $P$ is a policy index. This selects $P$ as the top policy: the one whose decisions are used by the PDP to respond to requests passed from the PEP.

For the semantics, we show how policy descriptions of the new form define labelled transition systems, and also how they can be translated directly into $\mathcal{C}+$. This mirrors exactly the two routes, represented by Definitions 13 and 14, that we earlier gave for semantics of policy descriptions without composition. We first define the new policy descriptions.

**Definition 16** A *composed policy description* $P$ of signature $\sigma$ has the same form of signature as defined in Section 3.1, with the following exceptions. A fixed set of *policy indices* is added; these are divided into *basic* and *non-basic* indices. The basic policy indices are further divided into those which are *positive* and those which are *negative*. *permitted*(*Sub:Act:Tar*) and *denied*(*Sub:Act:Tar*) constants are removed, and replaced with action constants *permitted*($p$, *Sub:Act:Tar*) for $p$ which are positive basic, and *denied*($p$, *Sub:Act:Tar*) for $p$ which are negative basic, for all *Sub:Act:Tar* $\in$ Actions. $P$ has the same form of system model as (non-composed) policy descriptions. In addition, it has authorization rules, policy composition rules, and a top policy designator, as described in the current section. We insist that if some policy index $p'$ appears in the head of a *permitted* (respectively *denied*) authorization rule, then all rules having $p'$ in their head are *permitted* (respectively *denied*) rules, and $p'$ must be a positive (respectively, negative) basic policy index. Basic policy indices may not appear in the left-hand-sides of policy composition rules, i.e. in the $P$ position of a rule **policy** $P$ **is** $P'$. Further, any non-basic policy index appearing in a composed policy description $P$ must appear on the left-hand-side of precisely one policy composition rule. Obligation policy rules have the same structure as previously. ⌟

In order to say how labelled transition systems are defined by such composed policy descriptions, we need a subsidiary definition, updating that for $A_P(s, e)$ (Definition 12).

**Definition 17** Let $P$ be a composed policy description, with siganture $\sigma$. Let the policy $p_\top$ be desiganted as top in $P$—so the rule

$$\textbf{policy } p_\top \textbf{ is top}$$

is in $P$. Where $s$ is an interpretation of $\sigma^f$ and $e$ is an interpretation of $\sigma^a$, then

$$\begin{aligned} A^*_P(s,e) = A_P(s,e) &\cup \{[\textit{Sub:Act:Tar} \textbf{ is allowed}] \mid (\textit{Sub:Act:Tar}), (s,e) \models_{p_\top} \mathsf{p}\} \\ &\cup \{[\textit{Sub:Act:Tar} \textbf{ is allowed}]{=}\mathbf{f} \mid (\textit{Sub:Act:Tar}), (s,e) \not\models_{p_\top} \mathsf{p}\} \end{aligned}$$

It remains to define the relationship $\models_p$, where $p$ is a policy index. This is done recursively, depending on which $p$ is used.

| | |
|---|---|
| $(\textit{Sub:Act:Tar}), (s,e) \models_p \mathsf{p}$ | if $p$ is basic and $e \models \textit{permitted}(\textit{Sub:Act:Tar})$ |
| $(\textit{Sub:Act:Tar}), (s,e) \models_p \mathsf{d}$ | if $p$ is basic and $e \models \textit{denied}(\textit{Sub:Act:Tar})$ |
| $(\textit{Sub:Act:Tar}), (s,e) \models_p \mathsf{n/a}$ | if $p$ is basic and $e \not\models \textit{permitted}(\textit{Sub:Act:Tar}) \wedge \textit{denied}(\textit{Sub:Act:Tar})$ |
| $(\textit{Sub:Act:Tar}), (s,e) \models_p x$ | if $p$ is not basic and $p$ **is** $f(p_1, \ldots, p_n) \in P$ |
| | and $\forall i (1 \leqslant i \leqslant n \rightarrow (\textit{Sub:Act:Tar}), (s,e) \models_{p_i} x_i)$ |
| | and $x = f(x_1, \ldots, x_n)$. ⌟ |

In the final clause of this definition, the definitions of the operators $\wedge$, $\vee$, $\neg$, $\otimes$, $\oplus$ and $-$ as they apply to policy values are applied recursively, to give the value $f(x_1, \ldots, x_n)$. Constraints placed on the occurrence of policy indices in composed policy descriptions, according to Definition 16, ensure that Definition 17 is well-formed.

**Definition 18** Let $P$ be a composed policy description of signature $\sigma$. The *labelled transition system defined by* $P$ is the triple $(S, L, R)$. $S$, the set of *states*, is given as usually, as the interpretations $s$ of Fluents such that:

$$\{s\} = \{s' \in \mathrm{I}(\sigma^f) \mid s' \models T_{\text{static}}(s) \cup \textit{Simple}(s)\}. \tag{22}$$

$L$, the set of labels, is simply $\mathrm{I}(\sigma^a)$, the set of interpretations of the action constants—including the policy constants. Finally, $R$ is the set of transitions $(s, e, s')$, where we define this anew. $(s, e, s')$ is a *transition* of $P$ iff $s$ is a state, and

- $\{e\} = \{e' \in \mathrm{I}(\sigma^a) \mid e' \models A^*_P(s, e)\}$;

- $\{s'\} = \{s'' \in \mathrm{I}(\sigma^f) \mid s'' \models T_{\mathrm{static}}(s') \cup E(s, e, s')\}$. ⌟

Translating composed policy descriptions into $\mathcal{C}+$ requires that we alter the signatures, adding constants that are used to represent the values of decisions for each policy index.

**Definition 19** Let $P$ be a composed policy description, with signature $\sigma$. For each $p$ which is a policy index in $P$ and each action constant of the form (*Sub:Act:Tar*), add a multi-valued action constant *pdp*($p$, *Sub:Act:Tar*) to the signature $\sigma$; the domain of each new constant is the set $\{\mathsf{p}, \mathsf{d}, \mathsf{in}, \mathsf{n/a}\}$. The new constants will represent what values a request for *Sub:Act:Tar* would take, according to policy $p$, as defined by the authorization rules and policy composition rules of $P$. Let the new signature be $\sigma'$. The $\mathcal{C}+$ *action description corresponding to* $P$, $C_P$, has all causal laws as defined in Definition 14 for the action descriptions corresponding to (non-composed) policy descriptions, with the exception of the laws

**default** ¬*permitted*(*Sub:Act:Tar*) and **default** ¬*denied*(*Sub:Act:Tar*)

for *Sub:Act:Tar* ∈ Actions; these are no longer needed as the form of authorization rules is different. Instead of them, we include

**default** ¬*permitted*($p$, *Sub:Act:Tar*)

for all positive basic $p$ and *Sub:Act:Tar* ∈ Actions, and

**default** ¬*denied*($p$, *Sub:Act:Tar*)

for all negative basic $p$ and *Sub:Act:Tar* ∈ Actions. For any positive authorization rule

*permitted*($p$, *Sub:Act:Tar*) **if** $G$

in $P$, we include a rule

*pdp*($p$, *Sub:Act:Tar*)=p **if** *permitted*($p$, *Sub:Act:Tar*)

in $C_P$; for all negative authorization rules

*denied*($p$, *Sub:Act:Tar*) **if** $G$

in $P$, we include a rule

*pdp*($p$, *Sub:Act:Tar*)=d **if** *denied*($p$, *Sub:Act:Tar*)

in $C_P$. For any basic index $p$ and *Sub:Act:Tar* ∈ Actions we also include the rule

**default** *pdp*($p$, *Sub:Act:Tar*)=n/a

in $C_P$. This models the semantics of authorization rules, and the policies which contain them, in $\mathcal{C}+$. We must now consider the policies defined as compositions of other policies. Each composition rule

$p$ **is** $P'$

in $P$ has as its right-hand-side a function of other policy indices; $P'$ can be written $f(p_1, \ldots, p_n)$, where $p_1, \ldots, p_n$ are the policy indices appearing in $P'$. For any composition rule $p$ **is** $f(p_1, \ldots, p_n)$, we add to $C_P$ the set of laws

*pdp*($p$, *Sub:Act:Tar*)=$x$ **if** *pdp*($p_1$, *Sub:Act:Tar*)=$x_1 \wedge \cdots \wedge$ *pdp*($p_n$, *Sub:Act:Tar*)=$x_n$

such that $x_1, \ldots, x_n \in \{\mathsf{p}, \mathsf{d}, \mathsf{in}, \mathsf{n/a}\}$, and $x = f(x_1, \ldots, x_n)$. Where

**policy** $p$ **is top**

is in $P$, we also include the laws

$$[\mathit{Sub{:}Act{:}Tar}\ \textbf{is allowed}]\ \textbf{if}\ \mathit{pdp}(p, \mathit{Sub{:}Act{:}Tar}){=}\mathsf{p}$$
$$[\mathit{Sub{:}Act{:}Tar}\ \textbf{is allowed}]{=}\textbf{f if}\ \mathit{pdp}(p, \mathit{Sub{:}Act{:}Tar}){=}\mathsf{d}$$
$$[\mathit{Sub{:}Act{:}Tar}\ \textbf{is allowed}]{=}\textbf{f if}\ \mathit{pdp}(p, \mathit{Sub{:}Act{:}Tar}){=}\mathsf{in}$$
$$[\mathit{Sub{:}Act{:}Tar}\ \textbf{is allowed}]{=}\textbf{f if}\ \mathit{pdp}(p, \mathit{Sub{:}Act{:}Tar}){=}\mathsf{n/a}$$

in $C_P$. ⌟

A similar proposition to 15 may now be stated.

**Proposition 20** Let $P$ be a composed policy description. Let the labelled transition system given by Definition 18 be $(S, L, R)$. Let the LTS given by by the $\mathcal{C}+$ action description corresponding to $P$, according to Definition 19, be $(S', L', R')$. Define

$$L'' = \{e' \mid \exists e \in L',\ e' \text{ is } e, \text{ restricted to the signature of } P\}$$
$$R'' = \{(s, e', s') \mid \exists (s, e, s') \in R',\ e' \text{ is } e \text{ restricted to the signature of } P\}$$

(The restrictions to the signature of $P$ remove reference to the new constants $\mathit{pdp}(p, \mathit{Sub{:}Act{:}Tar})$.) Then, the labelled transition systems $(S, L, R)$ and $(S', L'', R'')$ are equal.

*Proof:* We clearly have $S = S'$ and $L = L''$; we need to show that $R = R''$. This essence of this is a straightforward proof by structural induction over the definition of the 'top' policy. (Details omitted.) ⌟

## 4.2 Example composition

We now consider a very simple example, showing how access control policies can be composed and how the formulation in $\mathcal{C}+$ allows us to answer queries about them, using the implementation in iCCalc. The example domain is that of a library which has two categories of user: *reader* and *librarian*. For the moment, we consider the case where only one user exists, *borges*, but make no restrictions on the categories of user *borges* belongs to: there are therefore four possibilities. Users of the library can read, or write to, the library catalogue; authorization policies will govern the circumstances under which different categories of user may do this. We also model an action by the library *system* of altering the contents of the catalogue; for the purposes of this example, we do not treat this action as being governed by the policies. The action, *system:alter:catalogue*, can be thought of as representing any alteration of the library catalogue: say, when a book's ISBN number is updated automatically from a central database. The consequence of this action is that *borges* is no longer treated as having read the contents of the catalogue. We model the domain as shown below in Table 4. The set PolicyNames contains four policy names: $p_\top$ is the top policy used by the PDP; $l$ contains the rules governing access for librarians; $r_w$ contains a negative authorization policy stating that readers may not write to the catalogue; and $r_r$ is the positive authorization policy that readers may read the library catalogue.

What is missing from Table 4 is the policy-algebraic expression that defines $p_\top$ in terms of the non-basic policy names, $l$, $r_w$ and $r_r$. A context for this is easily imagined: perhaps those responsible for writing the policy for the library staff are different from those that set the policy for its users, and the work of each has to be combined; or it may simply be natural to group the rules together into such sub-policies. In either case, the sets must be combined.

Let us first consider the following composition:

$$\textbf{policy}\ p_\top\ \textbf{is}\ l \wedge (r_w \wedge r_r) \tag{23}$$

This is a simple-minded combination: as Table 3 makes clear, the result of $\wedge$-ing a 'permitted' decision with a 'not-applicable' decision is 'not-permitted'—so that where *borges* is a librarian but not a reader, then because $r_r$ and $r_w$ are not applicable, $p_\top$ is evaluated to $\mathsf{n/a}$, in spite of the permission that, one feels, policy $l$ should contribute to be overriding.

$$
\begin{aligned}
\mathsf{Sub} &= \{borges\} \\
\mathsf{Act} &= \{read, write\} \\
\mathsf{Tar} &= \{catalogue\} \\
\mathsf{Fluents} &= \{borges{:}hasRead(catalogue), borges{:}hasWritten(catalogue), \\
&\quad borges{:}userType(reader), borges{:}userType(librarian)\} \\
\mathsf{Actions}_{eca} &= \{\} \\
\mathsf{Actions}_{reg} &= \{(borges{:}Act{:}catalogue) \mid Act \in \mathsf{Act}\} \\
\mathsf{Events} &= \{system{:}alter{:}catalogue\} \\
\mathsf{PolicyNames} &= \{p_\top, l, r_w, r_r\}
\end{aligned}
$$

System Laws:

$borges{:}hasRead(catalogue)$ **after** $borges{:}read{:}catalogue$
$borges{:}hasWritten(catalogue)$ **after** $borges{:}write{:}catalogue$
$\neg(borges{:}hasRead(catalogue))$ **after** $system{:}alter{:}catalogue$
$\{$**inertial** $F \mid F \in \mathsf{Fluents}\}$
**exogenous** $system{:}alter{:}catalogue$

Policies:

$permitted(l, (borges{:}write{:}catalogue))$ **if** $borges{:}userType(librarian)$
$permitted(l, (borges{:}read{:}catalogue))$ **if** $borges{:}userType(librarian)$
$denied(r_w, (borges{:}write{:}catalogue))$ **if** $borges{:}userType(reader)$
$permitted(r_r, (borges{:}read{:}catalogue))$ **if** $borges{:}userType(reader)$

**policy** $p_\top$ **is top**

Table 4: Basic elements of the policy description for 'library' example.

This idea of policies being overriding promotes the introduction of another policy operator, $>$, taken from [Bruns et al., 2007]. Where $q_1$ and $q_2$ are policy names, then $q_1 > q_2$ is the policy which evaluates to $q_1$ if $q_1$ is the value of $q_1$, unless $q_1$ evaluates to n/a, in which case the value of $q_1 > q_2$ is $q_2$. $X > Y$ is defined as

$$X \oplus ((-(X \oplus (\neg X))) \otimes Y).$$

As a correction of the first proposal, the top policy could be defined in the following way:

$$\textbf{policy } p_\top \textbf{ is } l > (r_w > r_r) \tag{24}$$

This means that $l$ trumps all other policies: whenever *borges* is a librarian, then requests both to write to the catalogue and to read it are allowed. If $l$ does not apply (*borges* is not a librarian), then $(r_w > r_r)$ is used to evaluate requests. This means that, if *borges* is a reader, then a request to write to the catalogue would be denied; a request to read the catalogue would be allowed.

All other combinations of requests and 'circumstances' (in this case, user possessions by *borges*) of user categories result in a n/a decision by $p_\top$; the semantics of how the PDP interacts with the top policy means that such requests are not allowed. In particular, if *borges* is neither librarian nor user, then he may neither write to, or read, the catalogue.

The analysis of composed policies will be treated more fully in Section 6, where we give more detailed examples and show how the tool iCCalc is designed to support queries.

# 5 Norms

## 5.1 Syntax and Semantics

In [Craven and Sergot, 2008], changes were made to the language and semantics of $\mathcal{C}+$ in order to allow the representation of 'normative features' of agents and institutions; the resulting language was called $n\mathcal{C}+$. Semantically, $n\mathcal{C}+$ introduces two sorts of classification on states or transitions. First, states and transitions are either 'green' or 'red': where a green state (or transition) is one considered acceptable, or compliant with the norms governing the system being modelled; and a red state (or transition) is one which violates those norms, by being immoral, unethical, illegal, rude, bad, or against the norms in some other way. The green states might be thought of as those where the system is functioning well, the red states where something has gone wrong, in the sense of contravening norms. According to this coarse-grained classification, states and transitions *as a whole* are coloured: there is no scope for singling out the individual normative status of a given agent's contribution to the state or transition. There need be no relationship between the ways states and transitions are coloured, with the excpetion of what [Craven and Sergot, 2008] calls the 'green-green-green' constraint (*ggg*): if the system is in a green state, and performs a green transition, then all resulting states must be green. This constraint makes concrete the intuition that if things are going well, and nothing bad happens, then everything is still going well.

Recall Definitions 1 and 4: a transition system for $\mathcal{C}+$ has the structure $(S, A, R)$. To classify states and transitions as red and green, we add $S_{\text{grn}}$ and $R_{\text{grn}}$, where $S_{\text{grn}} \subseteq S$ is the set of green system states, and $R_{\text{grn}} \subseteq R$ is the set of green transitions. Dependently, we also define $S_{\text{red}} = S - S_{\text{grn}}$ and $R_{\text{red}} = R - R_{\text{grn}}$. Two new categories of causal law are then added to $\mathcal{C}+$, to determine which category states and transitions shall belong to. A *state permission law* has the form

$$\textbf{not-permitted } F \tag{25}$$

where $F$ is a fluent formula; and an *action permission law* has the form

$$\textbf{not-permitted } A \textbf{ if } G \tag{26}$$

where $A$ is an action formula and $G$ is a formula. The meaning of (25) is that where $s \in S$, then if $s \models F$, then $s \in S_{\text{red}}$; but states are green ($s \in S_{\text{grn}}$) by default. Similarly, the meaning of a law (26) is that if some transition $(s, e, s') \in R$ is such that $s \cup e \models G \wedge A$, then $(s, e, s') \in R_{\text{red}}$; transitions too are green ($(s, e, s') \in R_{\text{grn}}$) by default. Note that state and action permission laws, if present in an action description $D$, do not alter the structure or nature of the labelled transition systems: they are not used in the definision of $S$ and $R$, only in classifying the states and transitions normatively. Using this formal apparatus, the green-green-green constraint (henceforth, $\ggg$), requires that when $s \in S_{\text{grn}}$ and $e \in R_{\text{red}}$, then $s' \in S_{\text{red}}$.

(There is a certain regrettable overlap in terminology here, that might be confusing to somebody new to the various languages and concepts presented. Specifically, the use of **not-permitted** to indicate normative status might be confused with the idea of 'permission' as this features in policies, and the policy language we introduced in Sections 3 and 4. They are distinct. We stress again that norms are higher-level, abstract classifications of what is deemed to be good or bad behaviour and properties in a system, but which do not in themselves determine how the bits of system can act. Policies are typically lower-level, and through the presence of a PDP and PEP—a policy management component—they do control what can be done in a system. Policies are modelled in such a way that changes in policies mean a change in the structure of the labelled transition systems used to depict the domain's behaviour.)

As mentioned above, this classification is of states and transitions as a whole: there is no sense of whether an individual agent's contribution to a state or transition is good or bad—whether some specific agent conforms to the norms governing it, or not. Yet clearly this is desirable, so $n\mathcal{C}+$ also adds features that pick out an individual agent's contribution to a transition, and allow that contribution also to be classified. The full form of labelled transition systems for $n\mathcal{C}+$ can now be defined.

**Definition 21** A *coloured agent-stranded transition system* is a tuple

$$(S, A, R, S_{\text{grn}}, R_{\text{grn}}, \mathsf{Ag}, \mathit{strand}, \mathit{green}).$$

$(S, A, R)$ is as given in Definition 1, and $S_{\text{grn}} \subseteq S$ and $R_{\text{grn}} \subseteq R$ are the sets of green states and transitions. $\mathsf{Ag}$ is a set of *agents*. For $ag \in \mathsf{Ag}$ and $e \in A$, $\mathit{strand}(ag, e)$, which we also write as $e_{ag}$, picks out $ag$'s contribution to the label $e$—precisely what this 'contribution' is, we leave open. Finally, $\mathit{green}(ag)$, for each $ag \in \mathsf{Ag}$, is the set of transitions that are green for $ag$ (so, $\mathit{green} : \mathsf{Ag} \to \wp(R)$). ⌟

As well as the state and action permission laws we have already introduced, $n\mathcal{C}+$ also has agent-specific permission laws, which have the form:

$$\textbf{not-permitted}(ag)\ A\ \textbf{if}\ G \tag{27}$$

Here, $ag$ must be in $\mathsf{Ag}$, and as usual $A$ is an action formula and $G$ a formula. An action description of $n\mathcal{C}+$ is an action description of $\mathcal{C}+$, which also may include laws of the form (25), (26) and (27).

**Definition 22** Let $D$ be an action description of $n\mathcal{C}+$. The coloured agent-stranded transition system $(S, A, R, S_{\text{grn}}, R_{\text{grn}}, \mathsf{Ag}, \mathit{strand}, \mathit{green})$ determined by $D$ has:

- $(S, A, R)$ is the $\mathcal{C}+$ labelled transition system defined by $D^-$, where $D^-$ is $D$ without any permission laws.
- $S_{\text{grn}} = S - S_{\text{red}}$, where $S_{\text{red}}$ is $\{s \in S \mid s \models F \text{ for some law (25) in } D\}$
- $R_{\text{grn}} = R - R_{\text{red}}$, where $R_{\text{red}}$ is

  $$\{(s, e, s') \in R \mid s \cup e \models A \wedge G \text{ for some (26) in } D\} \cup \{(s, e, s') \in R \mid s \in S_{\text{grn}},\ s' \in S_{\text{red}}\}$$

- $\mathit{strand}$ has domain $\mathsf{Ag} \times A$.
- $\mathit{green} : \mathsf{Ag} \to \wp(R)$, and $\mathit{green}(ag) = R - \mathit{red}(ag)$, where $\mathit{red}(x)$ is defined as

  $$\{(s, e, s') \in R \mid s \cup e \models A \wedge G \text{ for some law 27) in } D\}$$ ⌟

The paper [Craven and Sergot, 2008] provides fuller discussion of these definitions, and also contains a reduction to the formalism of causal theories, similar to that for regular $\mathcal{C}+$ action descriptions given in Section 2.4.

$n\mathcal{C}+$ and the language for algebraic combinations of policies in $\mathcal{C}+$ we presented in Section 4 are different extensions of the basic language, and they can be combined. We call $\mathcal{C}+$ with cconstructions for policy composition $p\mathcal{C}+$; if $n\mathcal{C}+$ is also present, the result is $np\mathcal{C}+$.

## 5.2 Example: Rooms

We here consider a variation of the 'rooms' example presented in [Craven and Sergot, 2008]. The domain is a suite of four rooms, arranged in a square; agents, who are male or female, can move between rooms that adjoin. We will represent states in diagrams; Figure 6 shows a state in which a man, $m$, is in the top-left room, and a woman, $f$, is in the lower-right room. As the arrangement of rooms and doorways makes clear, agents can either move clockwise or anti-clockwise around the suite. The example domain was originally designed with norm-governed agent systems in mind, but it is easy to imagine that the agents in it are controlled centrally, and have their access to rooms governed by a policy.

Before we present the siganture and system laws used to describe the domain, it is useful to introduce a number of convenient expressions for representing the details of a particular arrangement of rooms; this makes the expression of the causal laws simpler. So, for the four-room suite

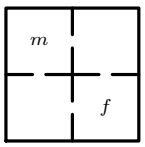


Figure 6: State from the 'rooms' example.

treated in the current example, we let $Rooms = \{top\_left, top\_right, bot\_left, bot\_right\}$; the variable $R$, possibly with sub- and superscripts, will range over this set. Similarly, $D$ and variations on it will range over the two directions, *anti* and *clock*. When it is possible to exit $R_1$ in direction $D$ and arrive in $R_2$, we denote this $adj(D, R_1, R_2)$—so that in our example, the following are true:

$$adj(anti, top\_right, top\_left) \qquad adj(clock, top\_right, bot\_right)$$
$$adj(anti, top\_left, bot\_left) \qquad \ldots$$

The basic action description of system laws is shown in Table 5. A simple fluent constant $A{:}loc$, for each agent $A$, is used to describe the agent's location. We also use statically-determined fluent constants $alone(A, A')$ and $alone(A)$, for agents $A$ and $A'$, where $A \neq A'$, to represent whether two agents are alone together in a room, or a single agent is alone in a room. These statically determined fluent constants are of use in the expression of policies and norms, later. The actions are for an agent $A$ to move in a given direction $D$, which we represent as $A{:}move{:}D$.

Causal law (28) describes the effects of a successful move by an agent. Law (29) means that agents cannot request to move to non-adjacent rooms. Law (30) means that agents may not simulatneously request to move to two different rooms. Law (31) means that two different agents cannot move in the same direction through a doorway, and law (32) means that agents cannot move in *opposite* directions through doorways: strait are the gates.

Let us suppose that a system designer wishes to impose some order on this non-normative, ungoverned free-for-all. The basic norm to be adhered to is that men and women ought not to be alone together in a room. In $n\mathcal{C}+$, we can represent this using the state permission law

$$\textbf{not-permitted}\ alone(m, f) \tag{33}$$

This has the desired effect of colouring red all states where the man $m$ is in the same room as the woman $f$; all other states are coloured green by the default which is standard in $n\mathcal{C}+$. As there are no action permission laws, the transitions are coloured green by default, with the exception of all transitions from green states to states which, according to (33), are red—the *ggg* constraint colours those transitions red. A fragment of the resulting transition system is shown in Figure 7. The colour of the agents in a state represents the colouring determined by the state permission

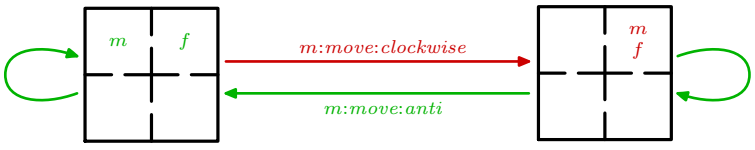


Figure 7: Fragment of transition system for the 'rooms' example.

laws; transition colours are shown by the colour of the arrows and their labels.

$$
\begin{aligned}
\mathsf{Sub} &= \{m, f\} \\
\mathsf{Act} &= \{move{:}anti, move{:}clock\} \\
\mathsf{Tar} &= \{\bot_{tar}\} \\
\mathsf{Fluents} &= \{A{:}loc \mid A \in \mathsf{Sub}\} \qquad (dom(A{:}loc) = Rooms) \\
\mathsf{SDFluents} &= \{alone(A), alone(A, A') \mid A, A' \in \mathsf{Sub},\ A \neq A\} \\
\mathsf{Actions}_{eca} &= \{A{:}move{:}anti, A{:}move{:}clock \mid A \in \mathsf{Sub}\} \\
\mathsf{Actions}_{reg} &= \{\} \\
\mathsf{Events} &= \{\}
\end{aligned}
$$

System Laws:

$$\{A{:}loc = R' \textbf{ after } A{:}move{:}D \wedge A{:}loc{=}R \mid A \in \mathsf{Sub},\ adj(D, R, R')\} \tag{28}$$

$$\{req(A{:}move{:}D) \textbf{ is non-executable if } A{:}loc{=}R \mid A \in \mathsf{Sub},\ \neg\exists X\, adj(D, R, X)\} \tag{29}$$

$$\{req(A{:}move{:}D) \wedge req(A{:}move{:}D') \textbf{ is non-executable} \mid A \in \mathsf{Sub},\ D \neq D'\} \tag{30}$$

$$\{A{:}move{:}D \wedge A'{:}move{:}D \textbf{ is non-executable if } A{:}loc{=}R \wedge A'{:}loc{=}R \mid A \neq A'\} \tag{31}$$

$$\{A{:}move{:}D \wedge A'move{:}D' \textbf{ is non-executable if} A{:}loc = R \wedge A'{:}loc{=}R' \mid A \neq A',\ D \neq D', adj(D, R, R')\} \tag{32}$$

$$\{\textbf{default } alone(A, A') \textbf{ if } A{:}loc{=}R \wedge A'{:}loc{=}R \mid A \neq A'\}$$

$$\{\neg alone(A, A') \textbf{ if } A{:}loc{=}R \wedge A'{:}loc{=}R' \mid A \neq A',\ R \neq R'\}$$

$$\{\neg alone(A, A') \textbf{ if } A{:}loc{=}R \wedge A'{:}loc{=}R \wedge A''{:}loc{=}R \mid A \neq A',\ A \neq A'', A' \neq A''\}$$

$$\textbf{default } alone(A)$$

$$\{\neg alone(A) \textbf{ if } A{:}loc{=}R \wedge A'{:}loc{=}R \mid A \neq A'\}$$

$$\{\textbf{inertial } F \mid F \in \mathsf{Fluents}\}$$

Table 5: Signature and system laws for the 'rooms' example.

The norms that have coloured the states and transitions of this system in different ways represent a system-level authority's perspective on what is desirable or undesirable in the system. The policies are used to change the various components', agents', or subjects' behaviour, and it is natural that the objective should be to have policies in place that guarantee the achievement of the norms—where achievement means that nothing bad ever happens: no redness is present. The policies can be imagined as being derived by the various agents themselves; or, in the case where the agents are less autonomous, the policies might also be derived by a system-wide authority, either the same as or different from the one who sets the norms. There are various different possibilities. (The current work does not take into account the epistemic complications of derivation, or indeed, how derivation of policies from norms is achieved: it only considers the ways of representing both, and reasoning about their interactions.)

For a first attempt at policies that are intended to achieve norm-compliance, consider those set out in Table 6. The first, positive authorization policy $p_m$ states that men are allowed to go anywhere. The second states that men are not allowed to move clockwise into the top-left room—forbidding transitions where men move from the bottom-left to top-left. The third policy states that women are allowed to go into any room where there is no man. The policy combination for $p_\top$ means that $p_{m\text{-}restrict}$ overrides $p_m$.

Do these policies eliminate redness? It is first important to note that there are several different interpretations of that question for which the answer might be required. The most obvious is: do

$\mathsf{PolicyNames} = \{p_\top, p_m, p_{m\text{-}restrict}, p_f\}$

| | | |
|---|---|---|
| Policies: | $permitted(p_m, m{:}move{:}D)$ **if** $m{:}loc{=}R$ | $(\exists R'\, adj(D, R, R'))$ |
| | $denied(p_{m\text{-}restrict}, m{:}move{:}clock)$ **if** $m{:}loc{=}R$ | $(adj(clock, R, top\_left))$ |
| | $permitted(p_f, f{:}move{:}D)$ **if** $f{:}loc{=}R \wedge \neg m{:}loc{=}R'$ | $(adj(D, R, R'))$ |
| | **policy** $p_\top$ **is top** | |
| | **policy** $p_\top$ **is** $(p_{m\text{-}restrict} > p_m) \oplus p_f$ | |

Table 6: Policy set (1) for the 'rooms'.

the policies eliminate red transitions? Yet this is not the best way of understanding the question of our interest, in policy authors, is in writing policies which prevent the system from entering bad states and doing bad things, and also ensure that if the system *does* ever go wrong, then it recovers. For, there may be green transitions from a red state to a redstate, and so the elimination of red transitions may not guarantee that a system *must* always recover and move out of a red state. So, a better interpretation of the question is: are there any transitions in the system to a red state? This is what we shall answer.

The first set of policies, in Table 6, clearly do a poor job. The transition shown in Figure 7 from left to right, is one to a red state; but this is not eliminated by the policies. To see this, consider the fate of a request $req(m{:}move{:}clock)$ when the system is in the state on the left. $p_f$ evaluates to $\mathsf{n/a}$ for this, $p_m$ of course evaluates to $\mathsf{p}$, and $p_{m\text{-}restrict}$ evaluates to $\mathsf{n/a}$, because its condition amounts to **if** $m{:}loc{=}bot\_right$ given the arrangement of the rooms in our example. $p_\top$ is therefore whatever $(\mathsf{n/a} > \mathsf{p}) \oplus \mathsf{n/a}$ is evaluated to, and this is $\mathsf{p}$. The request is thus allowed. In fact, there are 26 transitions to red states under this policy set; if all requests were allowed, then this number would only increase to 28, which shows that the policies are very unsuccessful job in eliminating them. (The two which are eliminated are those where the man moves clockwise into the top-left room: the first where the woman is already there, the second where she moves in simultaneously from the top-right.)

Let us have another attempt—shown in Table 7. In set (2), we will add policies that mean the

$\mathsf{PolicyNames} = \{p_\top, p_m, p_{m\text{-}restrict}, p_f, p_{m\text{-}bar}, p_{f\text{-}stay}\}$

| | | |
|---|---|---|
| Policies: | $permitted(p_m, m{:}move{:}D)$ **if** $m{:}loc{=}R$ | $(\exists R'\, adj(D, R, R'))$ |
| | $denied(p_{m\text{-}restrict}, m{:}move{:}clock)$ **if** $m{:}loc{=}R$ | $(adj(clock, R, top\_left))$ |
| | $permitted(p_f, f{:}move{:}D)$ **if** $f{:}loc{=}R \wedge \neg m{:}loc{=}R'$ | $(adj(D, R, R'))$ |
| | $denied(p_{m\text{-}bar}, m{:}move{:}D)$ **if** $m{:}loc{=}R \wedge f{:}loc{=}R'$ | $(adj(D, R, R'))$ |
| | $denied(p_{f\text{-}stay}, f{:}move{:}D)$ **if** $alone(f) \wedge f{:}loc{=}R$ $\wedge \neg m{:}loc{=}R_x \wedge \neg m{:}loc{=}R_y$ | $(adj(anti, R, R_x), adj(clock, R, R_y))$ |
| | **policy** $p_\top$ **is top** | |
| | **policy** $p_\top$ **is** $(p_{m\text{-}bar} > p_{m\text{-}restrict} > p_m) \oplus (p_{f\text{-}stay} > p_f)$ | |

Table 7: Policy set (2) for the 'rooms'.

man may not enter a room if there is somebody there, and also a policy requiring that if a woman is alone and there is nobody in a room either side of her, then she must stay put. The intention is that the second of these policies should guarantee that there are no meetings where a man and woman enter the same room from different directions.

This attempt works much better. It eliminates all red transitions, and leaves only 5 transitions to red states. When these transitions are examined, using a tool such as iCCalc, they are seen all to be self-transitions. The policies have ensured that no red state is never entered from a green state, but failed to ensure recovery. To do that, we can augment the policies to include several obligations (the authorization policies remain the same). They require that, if a man and woman are alone together, the man must move clockwise if he is allowed to; if they are alone and they man is not allowed to move clockwise, then the woman must move anti-clockwise. The new policy set in full is in Table 8. As is easy to check (and intuitively clear), this policy achieves what we

---

$\mathsf{PolicyNames} = \{p_\top, p_m, p_{m\text{-}restrict}, p_f, p_{m\text{-}bar}, p_{f\text{-}stay}\}$

| | | |
|---|---|---|
| Policies: | $permitted(p_m, m{:}move{:}D)$ **if** $m{:}loc{=}R$ | $(\exists R'\, adj(D,R,R'))$ |
| | $denied(p_{m\text{-}restrict}, m{:}move{:}clock)$ **if** $m{:}loc{=}R$ | $(adj(clock,R,top\_left))$ |
| | $permitted(p_f, f{:}move{:}D)$ **if** $f{:}loc{=}R \wedge \neg m{:}loc{=}R'$ | $(adj(D,R,R'))$ |
| | $denied(p_{m\text{-}bar}, m{:}move{:}D)$ **if** $m{:}loc{=}R \wedge f{:}loc{=}R'$ | $(adj(D,R,R'))$ |
| | $denied(p_{f\text{-}stay}, f{:}move{:}D)$ **if** $alone(f) \wedge f{:}loc{=}R$ $\wedge \neg m{:}loc{=}R_x \wedge \neg m{:}loc{=}R_y$ | $(adj(anti,R,R_x), adj(clock,R,R_y))$ |
| | $obligation(m{:}move{:}clock)$ **if** $alone(m,f) \wedge allow(m{:}move{:}clock)$ | |
| | $obligation(f{:}move{:}anti)$ **if** $alone(m,f) \wedge \neg allow(m{:}move{:}clock)$ | |
| | **policy** $p_\top$ **is top** | |
| | **policy** $p_\top$ **is** $(p_{m\text{-}bar} > p_{m\text{-}restrict} > p_m) \oplus (p_{f\text{-}stay} > p_f)$ | |

Table 8: Policy set (3) for the 'rooms'.

---

wanted: there are no transitions to red states. It does not rule out red *states*: but the only way a red state can be part of a simulated run through the system is if such a state is the *initial* state of the run. The actions of the various components of the system governed by policies—in our case, the man and woman—will always ensure that any red initial state is immediately left behind, and none is subsequently entered.

## 5.3 Extracting the System Component

We would like to be able to speak of the transition system defined by the 'system component' of an action description of $p\mathcal{C}+$, $n\mathcal{C}+$ or $np\mathcal{C}+$—this would model the effects of actions on non-normative, non-policy features of the system, if policies and norms are removed. If we simply remove from an action description $D$ all policy laws and permission laws, however, we will still be left with policy action constants, and some system laws may refer to those action constants. In the 'rooms' action description shown in Table 5, for example, the causal laws (29) and (30) both use the action constant $req(A{:}move{:}D)$. However, we can make two changes to action descriptions involving policies or norms, which reveals the underlying system structure. ($i$) first remove any causal laws that use policy-related constants such as $req(\cdot)$, $permitted(\cdot)$ and so on. Then make action constants exogenous, by adding a causal law

**exogenous** $a$

for every $a \in \sigma^a$. iCCALC can then be used to find the transitions of the system, and visualize them (see Section 6 for how this is done, and examples). Where $D$ is an action description of $p\mathcal{C}+$, $n\mathcal{C}+$ or $np\mathcal{C}+$, let $D^-$ be the result of removing normative and policy elements, in this way.

An alternative way to see the basic underlying structure of the system is to retain the policy constants, not add the 'exogeneity' laws, but to replace any policies with the 'most permissive policy' $\hat{p}$, which simply permit any request. This policy is shown in Table 9. Where $D$ is an action

| |
|---|
| $\mathsf{PolicyNames} = \{\hat{p}\}$ |
| $\mathsf{Policies}: \quad \{\mathit{permitted}(\hat{p}, A) \mid A \in \mathsf{Actions}\}$ |
| **policy** $\hat{p}$ **is top** |

Table 9: The most permissive policy, $\hat{p}$.

description of $p\mathcal{C}+$, $n\mathcal{C}+$ or $np\mathcal{C}+$, let $\hat{D}$ be the result of replacing the policy of $D$ by $\hat{p}$. This means that all requests for action are allowed, and so the only constraints placed on which actions may be performed are the 'physical' constraints of the underlying system, together with causal laws that determine when requests are made.

In the case of the action description in Table 5, for the 'rooms' example, there is a desired isomorphism between the transition system $(S^-, A^-, R^-)$ defined by $D^-$ and the transition system defined by $\hat{D}$, which we can call $(\hat{S}, \hat{A}, \hat{R})$. They are not the *same* transition system, because $(\hat{S}, \hat{A}, \hat{R})$ is defined over a signature that also includes policy action constants; but if one starts with $(\hat{S}, \hat{A}, \hat{R})$ and removes all policy action constants, and bleaches states and transitions of their norm-determined colourings, the result is $(S^-, A^-, R^-)$. We can now as two questions. Is this a general result—does it hold for every transition system $D$ of $p\mathcal{C}+$, $n\mathcal{C}+$ or $np\mathcal{C}+$? Further, if it *is* a general result, does that mean that $\hat{D}$ and $D^-$ always reveals the underlying system's structure?

The answer to both questions is negative. First, let us consider the relationship between $(S^-, A^-, R^-)$ and $(\hat{S}, \hat{A}, \hat{R})$, and consider the very simple action description shown in Table 10. In this domain, because a request is automatically triggered when $p$ is true, then removing all causal laws in which policy-related constants occur will remove information that is essential to the determination of the structure of $(\hat{S}, \hat{A}, \hat{R})$. In fact, $(\hat{S}, \hat{A}, \hat{R})$ has no transition from the state $\{p\}$ where $a$ is not performed, whereas $(S^-, A^-, R^-)$ does have such a transition: all action constants are made exogenous in it.

On the whole, it seems to us that the use of the most permissive policy $\hat{p}$ will enable a labelled transition system that corresponds more closely to the intended underlying system behaviour to be derived; but we leave this matter for further investigation.

# 6 Analysis Queries and Examples

In this section, we describe the sorts of analysis and simulation currently possible on our language—whether $p\mathcal{C}+$ or $np\mathcal{C}+$. First, the implementation we work with, iCCALC, is described.

## 6.1 Implementation

$\mathcal{C}+$ was originally supported by $\mathcal{C}$CALC,[6] written in Prolog. $\mathcal{C}$CALC supports a wide range of query tasks relating to $\mathcal{C}+$ and the language of causal theories. A $\mathcal{C}+$ action description $D$ is written as a Prolog source file, and $\mathcal{C}$CALC then finds the completion of this, $\mathit{comp}(\Gamma_t^D)$ for some run-length $t$. A query $Q$ is added, and the result is converted to conjunctive normal form (cnf), and sent to

[6] See `http://www.cs.utexas.edu/users/tag/cc/`.

| | |
|---|---|
| $\mathsf{Sub} = \{x\}$ | |
| $\mathsf{Act} = \{a\}$ | |
| $\mathsf{Tar} = \{\bot_{tar}\}$ | |
| $\mathsf{Fluents} = \{p\}$ | |
| $\mathsf{Actions}_{reg} = \{x{:}a\}$ | |
| $\mathsf{Events} = \{\}$ | |
| System Laws: | **inertial** $p$ |
| | $\neg p$ **after** $x{:}a$ |
| | $req(x{:}a)$ **if** $p$ |
| $\mathsf{PolicyNames} = \{p_\top\}$ | |
| Policies: | **policy** $p_\top$ **is top** |
| | $permitted(p_\top, x{:}a)$ |

Table 10: Simple policy description.

an externally implemented SAT-solver. Solutions represent runs of length $t$ through the transition system defined by $D$, that are consistent with the information of the query $Q$.

With Marek Sergot of Imperial College, the current author reimplemented $\mathcal{C}$CALC, to add support for a number of additional features. The result, iCCALC, some of the same core algorithms as $\mathcal{C}$CALC, but there are substantial additions and alterations, as follows.

- The syntax for source files is changed: full Prolog can now be used in the definitions of signatures and action descriptions, with features to use named variables as a shorthand. This makes the specification of large domains much more convenient. $\mathcal{C}$CALC's input syntax was subject to an overly strict type-check: this has been removed.
- There is a choice between using the SAT-solver `clasp`,[7] a state-of-the-art SAT-solver, or, if SICStus-Prolog[8] is being used, of employing its built-in Boolean constraint satisfaction library, `clp(b)`, for query solution. Initial experiments indicate that `clasp` is much the best option.
- Several additions to the basic $\mathcal{C}+$ language are fully-supported, including $n\mathcal{C}+$, $p\mathcal{C}+$ (the language for policies presented in the current paper), and $\mathcal{C}+_{timed}$ [Craven and Sergot, 2005]. Queries in the agentive modal logic LUCA [Sergot, 2008] are supported by an addition to the code, such queries being evaluated over transition systems defined using $\mathcal{C}+$ or one of its extensions.
- Full transition systems, and fragments of transition systems, can be written to an external file and visualized using `graphviz`,[9] freely-available graph visuzlization software.
- The code runs on many common variants of Prolog, including SICStus Prolog (versions 3 and 4), Yap Prolog[10] and SWI-Prolog[11].

[7] See `http://www.cs.uni-potsdam.de/clasp/`.
[8] See `http://www.sics.se/sicstus/`.
[9] See `http://www.graphviz.org/`.
[10] See `http://www.dcc.fc.up.pt/~vsc/Yap/`.
[11] See `http://www.swi-prolog.org/`.

The code for iCCALC is freely available from `http://www.doc.ic.ac.uk/~rac101/iccalc/`.

## 6.2 Query and Analysis Types

Recall the 'rooms' example presented in Section 5.2. We will use this domain, with more variations of its norms and policies, to illustrate the various kinds of analysis and reasoning task possible over domains of $\mathcal{C}+$ and extensions of that language for norms and policies. We will only consider the sorts of queries currently supported within iCCALC; but as the action descriptions of members of the family of langauges all define labelled transition systems (sometimes coloured, sometimes stranded), then of course many other kinds of query can be imagined. It is possible, for instance, to use model-checkers to check various kinds of property over the transition system. Though these possibilities are very interesting, we do not discuss them further in detail here.

In order to illustrate the kinds of query possible, we will use the action description for the 'rooms' example with a slight variations on policy set (2). The relevant system specifications are shown in Table 5. However, we will also add some agent-specific norms to the example, so that the full strength of the representation, and all forms of query, can be shown. The full compliment of norms and policies is shown in Table 11. There are two agents, and the state permission law

---

$\mathsf{Agents} = \{m, f\}$

Norms: **not-permitted** $alone(m, f)$
**not-permitted**$(m)$ $\neg m{:}move{:}clock \wedge \neg m{:}move{:}anti$ **if** $m{:}loc{=}R \wedge f{:}loc{=}R$

$\mathsf{PolicyNames} = \{p_\top, p_m, p_{m\text{-}restrict}, p_f, p_{f\text{-}stay}\}$

Policies:

| | |
|---|---|
| $permitted(p_m, m{:}move{:}D)$ **if** $m{:}loc{=}R$ | $(\exists R'\, adj(D, R, R'))$ |
| $denied(p_{m\text{-}restrict}, m{:}move{:}clock)$ **if** $m{:}loc{=}R$ | $(adj(clock, R, top\_left))$ |
| $permitted(p_f, f{:}move{:}D)$ **if** $f{:}loc{=}R \wedge \neg m{:}loc{=}R'$ | $(adj(D, R, R'))$ |
| $denied(p_{m\text{-}bar}, m{:}move{:}D)$ **if** $m{:}loc{=}R \wedge f{:}loc{=}R'$ | $(adj(D, R, R'))$ |
| $denied(p_{f\text{-}stay}, f{:}move{:}D)$ **if** $alone(f) \wedge f{:}loc{=}R$ $\wedge \neg m{:}loc{=}R_x \wedge \neg m{:}loc{=}R_y$ | $(adj(anti, R, R_x), adj(clock, R, R_y))$ |

**policy** $p_\top$ **is top**
**policy** $p_\top$ **is** $(p_{m\text{-}restrict} > p_m) \oplus (p_{f\text{-}stay} > p_f)$

$obligation(f{:}move{:}anti)$ **if** $f{:}loc{=}top\_right$

Table 11: Norms and policies for the 'rooms' example.

---

from before stating that the man and woman may not be alone together. There is a single agent-specific permission law, which states that it is not permitted for $m$, the male, to stand still, if $f$ is in the room. Note that this is different from an obligation on $m$ to move if $f$ is in the room: the obligation would force $m$ to make a request to move, and thus effects the system's behaviour; the norm merely colours $m$'s strand red should he be with a woman and fail to move. The absence of agent-specific laws for $f$ means that $f$'s strand in a transition will always be green. We will also impose the 'local-global' constraint, which means that a transition is (globally) red, whenever any agent strand of that transition is red. The policies are those from (2), except that $p_{m\text{-}bar}$, which

stated that the man is not permitted to enter a room if the woman is there, has been omitted. We have also added a new obligation, requiring $f$ to move anti-clockwise when she is in the top-right room.

We now describe a sample of the kinds of analysis query possible. Given the number of types and examples, we do not list the solutions to all, but merely a representative.

**Normative status** The colourings of the transition system and its strands mean we can ask lots of questions about the combinations of colourings with properties of states and systems; to do so is to ask about the circumstances in which norms are adhered to, or contravened. For example, (*i*) Is there a red state where $m$ is in the top-left room? (*ii*) Is there a transition where $m$ transgresses the norms—his strand is red—but the resulting state is green? (*iii*) If $m$ moves anti-clockwise four times in a row, is one of the states he passes through necessarily red?—This last can be queried by asking iCCalc for a run of length 4, where each state is green, and where $m$:*move*:*anti* is true across each transition. Any solution disproves the statement that a red state must be passed through. iCCalc finds 36 solutions to this query, with groups divded depending on which room $m$ starts from. The solutions all have $f$ chase $m$ round the suite of rooms (or $m$ chase $f$).

**Red transitions, agent-redness** We may also be interested in the relationship of the agent-specific norms to the global norms. Of course, where the local-global deontic constraint is enforced, any transition where an agent acts badly is automatically forced to break the global norms, and is coloured red. Yet we may still want to know whether there are transitions that are globally red without being red for any specific agent. Such transitions may of particular interest, in that they signify either a failure to design agent-specific norms that match the global norms, in which case the agent-specific norms may need modification; or else, they might represent transitions which are known to be bad because of factors the agents cannot control—some agent whose norms and behaviour we cannot direct did something wrong, or the system malfunctioned in a way that nobody was responsible for. There are 14 transitions of this sort in our new 'rooms' example; they are all instances where $m$ and $f$ are initially in adjoining rooms, the woman doesn't move, and the man moves into the woman's room.
If the optional local-global constraint is *not* enforced, we may be interested in seeing whether the property it enforces is, nevertheless, true. Are all states coloured red for some particular agent also globally red? If we remove the constraint from our domain, the answer is that there are 13 transitions where $m$ breaks his agent-specific norms, but the transition is not considered globally red. These might be instances of a too-strict development of agent norms.

**Planning under norms** One form of planning query that considers the normative status of the world asks whether it is possible to go through from an initial state in which $F$ is true, to a final state in which $G$ is true, with a particular pattern of adherence to norms along the way. That adherence may be total: no agent does anything wrong, meaning that there are no intermediate states between those in which $F$ and $G$ are true that are red, no red transitions between any of these states, and there is no transition that has a red strand for any agent. Yet there are also weakings of this total adherence to norms that one might be willing to accept: runs, for instance, where every strand in every transition is green, but there are some transitions which are globally red (this situation can only arise when the optional 'global-local' constraint is not imposed). Such runs might be considered to be those in which external factors are responsible for the system's badness, not any of the agents whose actions the plan is to dictate. For example, suppose we wanted to find the shortest sequence of actions of $m$ and $f$ from a state where $m$ is in the bottom-left room, $f$ is in the bottom-right, to a state where $m$ is in the top-left, and $f$ in the bottom-left; none of the intermediate states brought about by the actions in the plan should be red, and all agent-specific norms should be adhered to. Is this possible? iCCalc confirms that it is, and that the shortest plan involves three steps—sets of concurrent actions by $m$ and $f$. The sequence where $m$ and $f$ request to move anti-clockwise at each stage results, after three steps, in both agents being where they are required to be. The agent-specific norms are kept—and,

in fact, all transitions are globally green too.[12]

**Obligation violations** Given the possibility of including obligations in our policies of $p\mathcal{C}+$, we can ask what the interaction is between these obligations and the authorization policies. The effect of an obligation on a subject is to cause that subject to make a request; are such requests permitted or denied? If permitted, the obligation is fulfilled; otherwise, it is violated. We may wish to know whether obligations are ever violated given specific system constraints, or whether, when an obligation is violated by an agent $ag$, the previous actions of $ag$ were in some way responsible for putting him in a situation that made such a violation ineluctable. Our example policy contains a single obligation, on $f$: she must go anti-clockwise if ever in the top-right room. As iCCALC shows, this obligation is violated in 6 transitions, which fall into two groups depending on the reason for the violation. The first group of 3 violations occurs when $m$ is in the diagonally opposite, bottom-left room. In this situation, $p_{f\text{-}stay}$ denies $f$ permission to move in any direction, so that the request forced by $f$'s obligation is not allowed. There are three violations in this sub-group because $m$ can either do nothing, request to go clockwise, or request to move anti-clockwise. In the other group of three, the man is in the top-left room, and $f$ is then denied permission to move anti-clockwise into it by $p_f$; the three members again match $m$'s possibilities for requests. The fragment of the transition system showing violations is depicted in Figure 8. Those action and policy

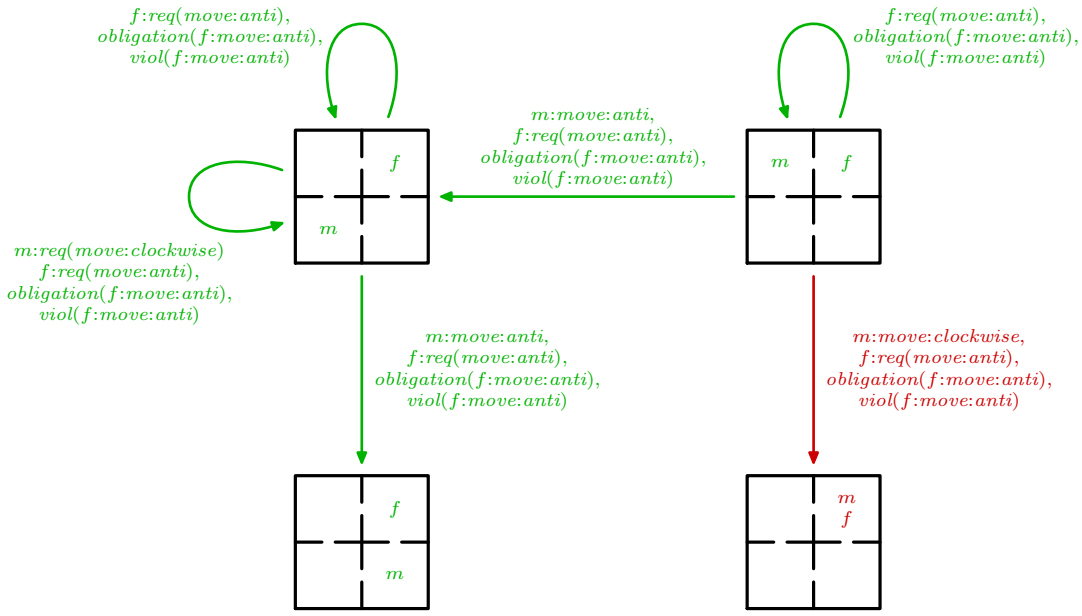


Figure 8: Violations in the 'rooms' example.

constants that are most relevant to the violations have been marked: the requests made by the woman, her obligations, whether or not the obligation was violated, and any actions

[12]In iCCALC, finding the shortest plan from a state where $F$ is true to a state where $G$ is true is done as follows. The length $n$ of plans is gradually increased, from 0 upwards. For length $n$, we query whether there is a run through the transition system defined by the action description, such that $F$ is true in the initial state and $G$ in the final state. If there is, we then need to check that *all* runs of length $n$, beginning in a state which satisfies $F$, in which the same actions are performed by the agents for which we are planning, results in $G$ being true in the final state. (This is to rule out $G$'s holding by luck, given the possibility of non-deterministic action descriptions in $\mathcal{C}+$ and its extensions.) If this is true, then the plan is the sequence of actions performed by our agents. If not, we increment the value of $n$ and repeat the process.

performed by the man. The various authorization policy decisions, and requests of the man, have not been shown. (These can be viewed in full using the visualization feature in iCCalc.)

**Policy conflict** Violated obligations represent one form of policy conflict analysis: for an obligation to be violated, the authorization policy must imply that a request to perform the action is not allowed. There are also other forms of policy conflict, where named sub-policies used in the definition of the top policy are found to give conflicting decisions. Whether or not this is a problem will depend on the application area, and the specific way in which policies are combined. For instance, if there are two policies $p$ and $p_e$, where $p_e$ is an exception to $p$, then $p_e$ may contain negative authorizations that override the positive decisions of $p$ under certain circumstances; we have made use of such structures in examples in the current paper, using the $>$ operator. But there may be other situations, where the way that policies are combined requires that they never issue contradictory decisions. Suppose, instead of $p_e > p$, the policy author had written $p_e \oplus p$; then, when both $p_e$ and $p$ apply, $p_e \oplus p$ would be evaluated to $\mathsf{in}$, and this may propogate undesired effects through the evaluation of the rest of the composition. We can use analysis to discover when sub-policies give conflicting results: for iCCalc includes an option to make the internal workings of the PDP visible; action constants are then included for the PDP's evaluation of requests against all named policies in the action description. Making that query for the 'rooms' example shows there are several examples of named policies reaching 'conflicting' decisions—one $\mathsf{p}$ and the other $\mathsf{d}$—although, since the policy for our example has been well designed with good uses of $>$, these do not signal a flaw in the policy. (It is also possible, of course, to test for the presence of an $\mathsf{in}$ evaluation against a named policy directly; if we do this for our example, no solutions are found.)

**Coverage gap** A coverage gap exists with respect to a policy $p$ when there is a request to which no rule in $p$ applies. If policies are defined as compositions using a policy algebra, this may not be a problem: for different named sub-policies may be intended to apply to disjoint sets of requests, with no single named basic policy covering all. However, if it is known that all requests of a given type should be jointly covered by a given set $p_1, \ldots, p_n$ of policies, then we may wish to prove that this is so; this can be done by showing whether there is any transition such that a request *sub*:*req*(*act*:*tar*) will be evaluated to $\mathsf{n/a}$ by all policies $p_1, \ldots, p_n$. If there any transitions of this sort, then iCCalc will find them, discovering all requests that are not covered, as well as the various circumstances in which the gaps occur.

**Policy comparison** One kind of policy subsumption was defined in [Craven et al., 2009] as holding between two policies, in the context of a given domain description, when all positive and negative authorizations and all obligations implied by one policy in a given context, are also implied by the other policy; the second policy is then said to subsume the first, for the given domain description. One purpose of determining whether one policy subsumes another is when updates to a policy are being considered: if the new policy to be added already has all its decisions matched by an existing policy, then there may be no reason to add it.
Where $p_1$ and $p_2$ are two basic policies (hence sets of rules, rather than algebraic expressions in terms of other policies), then a similar notion of policy subsumption is useful in the current work. Suppose $p_1$ and $p_2$ are positive—analagous remarks will hold for negative basic policies. Then, if all contexts in which $p_1$ determines an action as permitted are also contexts in which $p_2$ determines the action as permitted, there can be no reason to add the rules of $p_1$ to those of $p_2$: $p_1 \cup p_2$ is the same, for the system being considered, as $p_2$. We can define this as follows.

**Definition 23** Let $p_1$ and $p_2$ be basic positive authorization policies, where $p_2$ features in some action description $D$ of $p\mathcal{C}+$ or $np\mathcal{C}+$. Let $(S, A, R)$ be the LTS defined by $D$. Then, if for all *Sub:Act:Tar* $\in \mathsf{Actions}$ and all $(s, e, s') \in R$

$$s \cup e \models \mathit{permitted}(p_1, \mathit{Sub{:}Act{:}Tar}) \qquad \text{implies} \qquad s \cup e \models \mathit{permitted}(p_2, \mathit{Sub{:}Act{:}Tar})$$

we say that $p_2$ *D-subsumes* $p_1$, and write this as $p_1 \subseteq_D p_2$. If $p_1$ and $p_2$ are negative basic authorization policies, the definition is the same but with *permitted* replaced by *denied*. ⌟

Note that if $p_2$ $D$-subsumes $p_1$, then replacing $p_2$ by $p_1 \cup p_2$ has no effect on the transition system defined, and so no effect on the policy decisions made, however $p_2$ features in the definition of the top policy. That this property holds is clearly one reason why this definition of subsumption is useful.

What if $p_1$ and $p_2$ are not both basic policies? In general, there is then the possibility that for a given request they may take any of the four values p, d, n/a and in. Two natural definitions of subsumption would then be that $p_2$ $D$-subsumes $p_1$ when, for all $(s, e, s')$ in the transition system,

$$s \cup e \models \mathit{pdp}(p_1, \mathit{Sub{:}Act{:}Tar}){=}X \qquad \text{implies} \qquad s \cup e \models \mathit{pdp}(p_2, \mathit{Sub{:}Act{:}Tar}){=}X$$

for $(i)$ $X \in \{\mathsf{p}, \mathsf{d}\}$ or $(ii)$ $X \in \{\mathsf{p}, \mathsf{d}, \mathsf{n/a}, \mathsf{in}\}$. However, note that where $p_1$ and $p_2$ are non-basic, the expression $p_1 \cup p_2$ is not meaningful, because $p_1$ and $p_2$ are no longer sets of rules. Instead of considering adding the rules of $p_1$ to those of $p_2$, and thus comparing $p_2$ with $p_1 \cup p_2$, we might consider *replacing* $p_2$ by $p_1$; but if $p_1$ and $p_2$ are non-basic, then it is not only the p and d values that may make a difference. The way $p_2$ features in the algebraic definition of the top policy may mean that the values of n/a and in are taken account in whether the top policy allows an action or not. This means that, if we are interested in replacing $p_2$ by $p_1$—a form of update—then $(ii)$, above, would be the correct definition of subsumption. Yet this is very strong: in the context of the given system, $p_1$ and $p_2$ should give identical results to all possible requests.

What if we consider replacing $p_2$ by $p_1 \mathit{Op} p_2$, where $\mathit{Op}$ is a (possibly non-basic) composition operation which is intended to be as close as possible to set union, and thus to mirror the situation with basic $p_1$ and $p_2$? The natural candidate here is probably $\oplus$. What definition of policy subsumption for non-basic $p_1$ and $p_2$ would make it true that where $p_2$ $D$-subsumes $p_1$, then replacing $p_2$ by $p_1 \oplus p_2$ has no effect on what the top policy allows? The meanings of the operators, as shown in Figure 5 and Table 3, make this clear. It is that, for all $(s, e, s')$ in the transition system and all *Sub:Act:Tar*,

$$s \cup e \models \mathit{pdp}(p_1, \mathit{Sub{:}Act{:}Tar}){=}X \quad \text{implies} \quad \exists X'[X' \leqslant_k X \wedge s \cup e \models \mathit{pdp}(p_1, \mathit{Sub{:}Act{:}Tar}){=}X']$$

Testing for policy subsumption is easy using iCCalc. In our 'rooms' example, there are no two basic policies between which a relation of subsumption holds.

**Simulation** "Simulation" is understood here to be the modelling of how the policy-governed system behaves as a response to various changes in the environment; in this way simulation does not involve finding out whether some specific policy-related, or norm-related property holds. One sort of simulation that can be performed using iCCalc is to specify what is known to be true in an initial state of the system, and then to include hypothetical requests for actions and non-policy events that occur at various times after the initial state. We can then see what follows about the state properties of the system under these assumptions, or how the policy decisions turn out. iCCalc can show all runs consistent with the input information. It is often instructive to view how a system will behave under a number of predicted conditions and inputs; this often reveals properties of the policies that are in play which are unexpected—for which it did not occur to an analyst to query explicitly.

We can also include, under the current heading, more general 'system enquiries', such as whether a given system state is possible if certain policies are in place. In our 'rooms' example, for instance, we may want to know whether the man and the woman can be in the top-left room together—whether the policies allow this, rather than whether it is acceptable that they be so. Queries like this are the bread-and-butter of iCCalc usage: in the current instance the answer is of course that the state is possible, and the further information is supplied that such a state is not permitted (red).

**Modular analysis** In the case where the different policy names are used to hold the policies in sway over different components of a distributed system, it is of course possible to analyse the behaviour of those components' policies separately. In this context, with $n$ components of a distributed system governed by policies $p_1, \ldots, p_n$, each policy $p_i$ would govern different actions from a $p_j$ for $i \neq j$. For any given access request, the $p_1, \ldots, p_n$ can be defined in such a way that either all of them evaluate to n/a, or precisely one of them evaluates to p or d. The top policy can then be defined as the join $p_1 \oplus \cdots \oplus p_n$ of the various components' policies, and queries can be made of the behaviour of one of the $p_i$ in isolation, or of the effect of decisions of the policy $p_i$, and the behaviour of the $i^{\text{th}}$ component it controls. (In the rooms example, we can view the man and the woman as being different components, with the policy $(p_{m\text{-}restrict} > p_m)$ as being the module of the total policy governing the man's behaviour, and $(p_{f\text{-}stay} > p_f)$ as the sub-policy governing the woman's behaviour.)

**Planning under policies** Where an action description has policies, these will be taken account in any planning query made, as the effects of a request are determined by the policies in place. Where there are obligations on an agent, these also influence the solutions to a planning problem: as the agent will automatically—given our understanding of obligation policies as ECA rules—request to try and fulfil any obligation under which it finds itself. Incorporating the constraints of existing policies into the solution of queries is performed automatically by iCCalc, which builds only those transitions consistent with the policies i attempting to solve a planning problem.

There is another way, however, in which we might study the interaction between policies and plans. Suppose a successful plan has been generated to solve a planning problem in the presence of given policies. (Perhaps, also, in certain kinds of accordance with norms, as described above in 'Planning under norms'.) Such a plan will be a sequence of actions which, if performed, guarantees the state will move from that initially specified to one where the planning goal is true. The policy engineer may then want to derive a set of policies from the plan that can be added to the policies already in place, in order to ensure the plan is followed. The new policies would be obligations: in certain circumstances, and in the right order, request to perform the actions contained in the plan.

There are complications here which we do not probe further at this time. One concerns the possibility of adapting the current set of authorization policies in order to allow different obligation policies, implementing a different plan. Another concerns the way in which a group of agents can, with policies governing the various members of the group, jointly achieve the plan's goal. These are subjects for further work.

**Policies and Norms** Finally, one family of queries concerns the relationship between norms and policies. If both are in place, and the action description is therefore in $np\mathcal{C}+$, we can ask about the interaction between the various named policies and the normative status of the system and its agents. Given the designation of a top policy, then clearly where some agent $\alpha$'s strand of a transition $(s, e, s')$ is red, there must be an agent-specific permission law

$$\textbf{not-permitted}(\alpha)\ F\ \textbf{if}\ G$$

such that $s \cup e \models F \wedge G$. The top policy must make whatever actions are in $F$ permitted: the requests for them must be evaluated as p according to the top policy. It is therefore immediate that, if this redness is one we wish to eliminate, the top policy is not doing its job: and we can see this only from a query about *norms*. However, we may expect that one of the various named policies in terms of which the top policy is defined should cover this case of redness: should deny the requests for actions comprising $F$ in these circumstances. If there *is* such a denial, then the top policy has been badly defined in terms of the other policies; if there is no such denial, then the policy in question has itself been badly defined. So, a combined analysis of when red strands arise in the presence of denials according to the specific policy will indicate how the policies can be repaired in order to eliminate the redness.

Code for the 'rooms' example, together with code for all queries mentioned above, is available from the iCCalc website, `http://www.doc.ic.ac.uk/~rac101/iccalc/`.

# 7 Related work

Some related work—that which is very closely related—is discussed in the relevant sections of this report. The background of action languages is presented in Section 2. Work on policy algebras is discussed in Section 4. The specific formalism we use for the representation of norms is presented in Section 5.1. In the remainder of the current section we discuss other relevant work.

The closest precursor to the current work is [Gelfond and Lobo, 2008], where a language $\mathcal{AOPL}(\Sigma)$ is presented. This is used to specify positive and negative authorization and obligation policies, where some policies are defeasible and some not, and there is a priority ordering over the former. As with our work there is a background semantic structure of labelled transition systems for depicting the changing behaviour of a system. (The authors use the action language $\mathcal{AL}$ of [Baral and Gelfond, 2000] for defining the transition systems, but as the policy component is not tightly coupled to their system-definition component, many other formalisms could be used.) The main analytic interest of the authors, at least as represented in the paper, is in checking various forms of policy compliance with actions that can be performed in the system being modelled, and for this reason, policies are not presumed to be enforced: whether or not an action is permitted or not permitted, or required by an obligation, makes no *automatic* difference to whether the action *is* performed. In contrast, in our work, policies do automatically determine whether an action is possible in a given state: policies affect the structure of the transition system. The reason for the difference of approaches is probably to be found in the fact that $\mathcal{AOPL}(\Sigma)$ appears to be used for the deliberation of a single agent about which of its behaviours would be compliant with a policy that is externally set but not enforced: so that policies here represent laws which the agent may have a good reason to obey, but is not forced to obey. In this respect, policies are for [Gelfond and Lobo, 2008] what norms are for us, and the reasoning of [Gelfond and Lobo, 2008]'s agent about whether its actions are policy-compliant can be thought to match the kinds of query we explored in Section 6.2 about the normative status of various actions. One advantage of our work is then that it enables reasoning about the interaction between policies in this 'normative' sense, and the policies which are actually in force, or might be, over an agent. Further, our work is used to represent the norms, policies, and actions of multiple agents in a system; in its current state, the language of [Gelfond and Lobo, 2008] is designed for policies governing only a single agent—as befits their purpose of considering an agent's own reasoning about whether to adhere to policies. We also have an existing implementation supporting various kinds of query task, in the form of iCCalc. This implementation can perform the types of analysis mentioned in [Gelfond and Lobo, 2008].

Two more technical differences concern the way in which defaults are dealt with and the treatment of obligations. Defaults in $\mathcal{AOPL}(\Sigma)$ are ordered by a priority or preference relation, and this priority determines which of a set of 'triggered' defaults should be used in generating the answer sets for a given policy. For us, ordered defaults can be expressed both at the level of policy composition (using operators such as the $>$ defined in Section 4.2) and within the basic policies themselves—though there is no built-in way of expressing ordered defaults in the basic rules: this can be done in whatever way the policy-designer wishes, using familiar techniques. We regard it largely as a matter of personal preference which of the two approaches (ordered defaults in the algebra, or a preference relation as in [Gelfond and Lobo, 2008]) is considered to be more clear: the expressive power of each seems comparable, as our success in formulating examples from each approach in the other makes clear.

The second difference concerns the treatment of obligations. For us, these are not treated algebraically, and are understood as ECA rules. For [Gelfond and Lobo, 2008], they are treated in just the same way as authorization policies. This does raise the question of what the essential difference between authorizations and obligations is for them. In many systems of modal deontic logic, an obligation not to do something is equivalent to the negation of permission to do it, and

the two concepts are thus interdefinable. It is unclear to us whether [Gelfond and Lobo, 2008] should take this approach too. For us, authorization and obligation policies have orthogonal interpretations, and the question of the their logical relationship, raised in deontic logic, does not therefore not arise. That said, further work will examine whether an algebraic approach for our obligations is appropriate; see Section 8 for more details.

In [Irwin et al., 2006], the authors present a language and semantics that represents both authorization and obligation policies. However, the obligations are explicitly tied to authorizations, as actions that must be fulfilled if an access is granted. We can model such obligations in our work, either as norms of the form

$$\textbf{oblig}(Sub)\ A\ \textbf{if}\ allow(Sub{:}Act{:}Tar) \wedge Sub{:}Act{:}Tar \tag{34}$$

or else as policies

$$obligation(A)\ \textbf{if}\ allow(Sub{:}Act{:}Tar) \wedge Sub{:}Act{:}Tar \tag{35}$$

The choice between them depends on whether the subject being modelled automatically generates a request in the presence of the given obligation. Finally, in [Craven et al., 2009] the authors present a formal framework for the analysis of authorization and obligation policies in the context of dynamic systems. The language has slightly more control over some aspects of the specification of policies than the current work: time is reified, and so constraints can be used to express the dependence of current policy logic on historical conditions of the system. However, there is no normative dimension, and defaults and exceptions for policy decisions are expressed in a much less structured, more ad-hoc manner than in the current approach, where policy algebra is used to impose a coherent logic on the combination of different (sets of) policy rules. The underlying formalism of [Craven et al., 2009] is abductive constraint logic programming; one advantage of using $\mathcal{C}+$ is the ready determination of transition system semantics.

# 8 Concluding Remarks and Future Work

We have presented a way of introducing the representation of authorization and obligation policies into an action language, $\mathcal{C}+$. Authorization policies are defined using the policy-algebraic approach of [Bruns et al., 2007], and obligation policies are treated as event-condition-action rules whose fulfilment depends on the right authorizations being in place. Using $\mathcal{C}+$ as a base language allows the dynamic properties of systems to be studied in relation to policies, and gives an attractive and widely-used semantics, in the form of labelled transition systems. This provides a potential bridge to other techniques for system analysis, notably model checking.

The policy component of the language can also be combined with another extension, $n\mathcal{C}+$, which is used to represent and reason about norms which are binding on multiple subjects or agents. When both are combined, norms can be viewed as specifications which determine when the states or behaviour of a system and its agents are bad, undesirable, good, ideal, and so on; and policies can be seen as the concrete attempt to function as the norms dictate. We showed how a number of properties which relate to these interactions can be expressed and analysed using our languages. We discussed our implementation of these languages, iCCalc, which can be used for all the forms of analytic task we mentioned.

There are several directions for future work. First, we are interested in examining how the fine-grained approach to the analysis of individual and collective agency presented in [Sergot, 2008] and [Sergot, 2007] might be used in work on policies. One way in which this might help is in the derivation of policies from norms: if a state or transition violates norms, then an agentive analysis of the type [Sergot, 2008] shows how to perform could indicate which of a group of subjects was more responsible for the violation; it is those agents whose policies must be adapted. Another application of the logic of agency could be in the expression of policies describing what to do when an agent violates its explicit obligation policies. If that violation occurs as a result of the agent being forced to do something by a group of other agents, then sanctions imposed by system

policies may be different than if the agent's violation was entirely down to its own actions. These two applications of what [Sergot, 2008] calls the 'logic of unwitting collective agency' embody two points at which more refined agentive constructions can enter the current work: the first, at the level of queries over the LTS defined; the second, within the object language of policies itself. The second, as one might expect, would be much more challenging research. We have begun experiments with an extension of iCCalc, written by Sergot, that model-checks agentive constructions of [Sergot, 2008] over transition systems defined using $p\mathcal{C}+$ and $n\mathcal{C}+$.

Secondly, we are interested in exploring the connections between the current work and abductive logic programming. A subclass of the policies and action descriptions it is possible to represent in $p\mathcal{C}+$ could be given an equivalent formulation as normal logic programs: the main constraint determining the class is that there be no circular dependencies through causal laws between constants. This suggests that ACLP can be used to perform analytic tasks for theories of $p\mathcal{C}+$ and, it seems likely, also $np\mathcal{C}+$. We wish to exploit these relationships and possibilities.

Thirdly, we wish to examine the automated derivation of agent-specific norms from norms, and of policies from agent-specific norms. If the system norms—those that are not agent-specific—are conceived of as set by a system-wide authority or designer, then the various subjects or agents that form the system may be interested in deriving policies which guarantee that they will adhere to the norms. (The intermediate step in this derivation is agent-specific norms.) We expect that techniques from automated planning could be used in creating policies of this sort. One aspect of this problem is where agents are grouped together, and where a group derives its policies together in such a way that all members of the group are norm-compliant. It may be that analysis of the various agents' actions, in terms of the rich set of agentive operators of [Sergot, 2008] referred to above, allows a better derivation of policies for groups. For example, if $x$ and $y$ are in the same group of agents, and it can be shown that in a given transition, $x$ brings it about that $y$ violates the norms where $y$ would not otherwise have done so, then $x$ is the agent for which restraining policies should be derived, rather than $y$.

Fourthly, we are interested in the addition of constructions for the representation of knowledge to the current formalism. In the derivation of policies by which an individual subject should govern its behaviour, account should presumably be taken of those aspects of an agent's environment of which it is able to be aware. The evaluation of policies by an agent's PDP ought to be possible: there should be no conditions in the body of the policy rules which the agent cannot verify. One way in which knowledge can be taken into account is suggested by [Gelfond and Lobo, 2008]: on the basis of a division of fluents into those whose values the agent can know, and those it cannot know, states are divided into equivalence classes, where each class represents how the world might be, given what an agent knows. However, there is no explicit modelling of the perceptual capabilities, or the effects of actions on knowledge, at the level of the transition system, and it is precisely this we are interested in. Here, they may be further entanglements with policies and norms, in that the policies and norms may themselves state what an agent should come to know.

Fifthly, the composition of obligations. It may be that the sort of approach we use for composing authorization policies can profitably be extended to obligation policies. However, the fact that obligation policies are, for us, ECA rules means that the interpretation of the meaning fo composition would be different. For authorization policies, it is a way of deciding what the result of a request is, given the relations of a request to various named sets of policy rules; the different decisions of the various sets are combined to produce the total decision. Yet there is no comparable role for a request in the context of obligation policies, and it is not clear what useful meaning could be given to a 'combination' of an obligation to do $\phi$ with an obligation to do $\psi$. We are currently examing such questions.

Sixthly, in the current work requests themselves are not governed by authorization policies: a subject does not permission to make a request. The only point to which policies apply is in the decision whether a request eventuates in the relevant action. This contrasts with a system such as Ponder [Damianou et al., 2001], where whether or not a request is made is itself subject to policy. We have modelled, at this stage, at a higher level of abstraction—but we are interested in delving further in order to represent policies applying to requests. This is the most straightforward of the various extensions.

# A Sample code for a 'rooms' example

We here present the input code to iCCalc for a simplified version of the 'rooms' example discussed throughout the paper, with comments on features of the input. The code is general enough to support easy adaptations to different numbers of agents, rooms, and different room topologies; for this reason the code is more complicated than would be the case if it were suited to the specific domain only. The language is Prolog.

First, variables are declared which enabe the concise representation of features of the $\mathcal{C}+$ signature, $\mathcal{C}+$ laws, policies and norms:

```
Ag,OtherAg :: agent(Ag).
M :: male(M).
F :: female(F).
Room,OtherRoom,RoomTo,RoomFrom :: room(Room).
Dir,OtherDir :: direction(Dir).
```

Whenever one of the variables on the LHS of the `::` appears elsewhere in the source code, it is ground to all values of the *first* variable in the list such that the Prolog code on the RHS is true. For example, if `M` appears elsewhere in the source code, the clause `A if C` containing it will be replaced by `A if male(M),C`.

Next, some details of objects and other aspects of the domain.

```
agent(M).
agent(F).

male(m1).
male(m2).
female(f).

room(left).
room(right).

direction(left).
direction(right).
```

That is: there are two men and one woman; two rooms, named `left` and `right`, and two directions in which to travel between rooms. An agent is a male or a female.

Next, predicates used to make life easier in writing causal laws.

```
adj(Direction, RoomF, RoomT) :-
 ord_basic_adj(Direction, RoomF, RoomT).
adj(Direction, RoomF, RoomT) :-
 opp_dir(OppDir, Direction),
 ord_basic_adj(OppDir, RoomT, RoomF).

ord_basic_adj(right, left, right).

opp_dir(right, left).
```

`adj/2` defines whether it is possible to go from `RoomF` to `RoomT` in direction `Direction`. The other predicates are subsidiary definitions.

We can now define the signature of our action description.

```
fc(Ag^loc).
sdfc(alone(Ag)).
sdfc(alone(M,F)).
pac(Ag^move^Dir).
```

```
domain(Ag^loc, Room).

inertial FC :-
 fc(FC).
```

There is one family of simple fluent constants, used to record an agent's location; it is non-Boolean. Two kinds of statically-determined fluent constants record whether an agent is alone in a room, or a man is alone in a room with a woman. There is one group of policy-controlled action constant, used to represent an agent's moving in a given direction. (The variable declarations at the beginning of the source file are used to fill out the signature.) All simple fluent constants are inertial.

Now, the causal laws describing movement. Moving somewhere puts an agent there:

```
Ag^move^Direction causes Ag^loc=RoomT if Ag^loc=RoomF :-
 adj(Direction, RoomF, RoomT).
```

(The body ensures that only topologically possible movement occurs.) We stipulate that it is also not possible to *request* to move to non-adjacent rooms. This is not essential, but means that irrelevant requests are not included. In the same way we require that it is not possible to request to move in two different directions at once.

```
nonexecutable Ag^req(move^Dir) if Ag^loc=Room :-
 \+ adj(Dir, Room, _).

nonexecutable Ag^req(move^Dir) & Ag^req(move^OtherDir) :-
 Dir @< OtherDir.
```

It is not possible for two agents to move concurrently through the same doorway—either in the same direction, in or in opposite directions:

```
nonexecutable Ag^move^Dir & OtherAg^move^Dir
if Ag^loc=Room & OtherAg^loc=Room :-
 Ag @< OtherAg.

nonexecutable Ag^move^D & OtherAg^move^OD
if Ag^loc=OR & OtherAg^loc=R :-
 Ag \= OtherAg,
 adj(D, OR, R),
 opp_dir(D, OD).
```

That is all that is needed for the definition of the effects of movement, and the various constraints on what is physically possible.

Next, a series of laws defining what it means for an agent to be alone, or a man and a woman to be alone together.

```
default alone(Ag).
caused -alone(Ag) if Ag^loc=Room & OtherAg^loc=Room :-
 Ag \= OtherAg.

default alone(M, F) if M^loc=Room & F^loc=Room.
caused -alone(M, F) if M^loc=Room & F^loc=OtherRoom :-
 Room \= OtherRoom.
caused -alone(M, F) if M^loc=Room & F^loc=Room & Ag^loc=Room :-
 M \= Ag,
 F \= Ag.
```

The first law states that people are usually alone. The second states that someone is not alone if there is someone else in the same room. The third law states that a man is, by default, alone with a woman if they are in the same room. The fourth states that men and women in different rooms are not alone together, and the fifth states that gooseberries get in the way.

We now cover policies. The first states a user-defined policy operator: this is the $>$ we referred to and defined in Section 4.2.

```
pol_abbrev(X > Y, X join ((neg (X join (not X))) meet Y) ).
```

The first argument is the operator expression being defined; the second is the definition. Next, we state and define the top policy:

```
top pol is (pmen join (pfemrestrict > pfem)).
```

This law expresses that `pol` is the name of the top policy, and that this is defined in terms of three other policy names: `pmen`, `pfem` and `pfemrestrict`. These three now need defining.

First, `pmen`:

```
pmen @ {
 perm(M^move^Dir) if M^loc=R :-
  adj(Dir, R, _)
}.
```

The braces enclose a set of policies; the `@` is used to prefix the set's name. `pmen` therefore a positive authorization policy, with a single rule, stating that men may move in any direction physically possible. A similar policy, `pfem`, covers women:

```
pfem @ {
 perm(F^move^Dir) if F^loc=R :-
  adj(Dir, R, _)
}.
```

However, women are subject to the additional policy, `pfemrestrict`:

```
pfemrestrict @ {
 denied(F^move^Dir) if F^loc=R & M^loc=left :-
  adj(Dir, R, left)
}.
```

This means that women will be denied access to the left room if there is a man inside.

That is all the definition of policies needs; the single (non-agent-specific) norm is:

```
notpermitted alone(M, F).
```

Any state where a man and woman are alone in a room will be coloured red; all other states will be green. The red transitions are then determined in accordance with the *ggg* constraint.

# References


[Baral and Gelfond, 2000] Baral, C. and Gelfond, M. (2000). Reasoning agents in dynamic domains. In *In Workshop on Logic-Based Artificial Intelligence*, pages 257–279. Kluwer Academic Publishers.

[Belnap, 1977] Belnap, N. (1977). A useful four-valued logic. In Dunn, J. and Epstein, G., editors, *Modern Uses of Multiple-Valued Logic*, pages 8–37. D. Reidel, Dordrecht.

[Bruns et al., 2007] Bruns, G., Dantas, D. S., and Huth, M. (2007). A simple and expressive semantic framework for policy composition in access control. In Ning, P., Atluri, V., Gligor, V. D., and Mantel, H., editors, *FMSE*, pages 12–21. ACM.

[Bruns and Huth, 2008] Bruns, G. and Huth, M. (2008). Access-control policies via belnap logic: Effective and efficient composition and analysis. In *CSF*, pages 163–176. IEEE Computer Society.

[Clark, 1978] Clark, K. (1978). Negation as failure. In Gallaire, H. and Minker, J., editors, *Logic and Databases*, pages 293–322, New York. Plenum Press.

[Craven et al., 2009] Craven, R., Lobo, J., Ma, J., Russo, A., Lupu, E., Bandara, A., Calo, S., and Sloman, M. (2009). Expressive policy analysis with enhanced system dynamicity. In *ASIACCS*, pages 239–250. ACM.

[Craven et al., 2008] Craven, R., Lupu, E., Lobo, J., Bandara, A., Calo, S., Ma, J., Russo, A., and Sloman, M. (2008). An expressive policy analysis framework with enhanced system dynamicity. Technical Report, Department of Computing, Imperial College London.

[Craven and Sergot, 2005] Craven, R. and Sergot, M. J. (2005). Distant Causation in $\mathcal{C}+$. *Studia Logica*, 79(1):73–96.

[Craven and Sergot, 2008] Craven, R. and Sergot, M. J. (2008). Agent strands in the action language $n\mathcal{C}+$. *J. Applied Logic*, 6(2):172–191.

[Damianou et al., 2001] Damianou, N., Dulay, N., Lupu, E., and Sloman, M. (2001). The ponder policy specification language. In Sloman, M., Lobo, J., and Lupu, E., editors, *POLICY*, volume 1995 of *LNCS*, pages 18–38. Springer.

[Erdogan and Lifschitz, 2006] Erdogan, S. T. and Lifschitz, V. (2006). Actions as special cases. In Doherty, P., Mylopoulos, J., and Welty, C. A., editors, *KR*, pages 377–388. AAAI Press.

[Gelfond and Lifschitz, 1993] Gelfond, M. and Lifschitz, V. (1993). Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–321.

[Gelfond and Lifschitz, 1998] Gelfond, M. and Lifschitz, V. (1998). Action languages. *Electronic Transactions on AI*, 3.

[Gelfond and Lobo, 2008] Gelfond, M. and Lobo, J. (2008). Authorization and obligation policies in dynamic systems. In de la Banda, M. G. and Pontelli, E., editors, *ICLP*, volume 5366 of *Lecture Notes in Computer Science*, pages 22–36. Springer.

[Giunchiglia et al., 2004] Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., and Turner, H. (2004). Nonmonotonic causal theories. *Artificial Intelligence*, 153:49–104.

[Irwin et al., 2006] Irwin, K., Yu, T., and Winsborough, W. H. (2006). On the modeling and analysis of obligations. In Juels, A., Wright, R. N., and di Vimercati, S. D. C., editors, *ACM Conference on Computer and Communications Security*, pages 134–143. ACM.

[Li et al., 2009] Li, N., Wang, Q., Qardaji, W. H., Bertino, E., Rao, P., Lobo, J., and Lin, D. (2009). Access control policy combining: theory meets practice. In Carminati, B. and Joshi, J., editors, *SACMAT*, pages 135–144. ACM.

[McCain and Turner, 1997] McCain, N. and Turner, H. (1997). Causal theories of action and change. In Shrobe, H. and Senator, T., editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 460–465, Menlo Park, California. AAAI Press.

[Ni et al., 2009] Ni, Q., Bertino, E., and Lobo, J. (2009). D-algebra for composing access control policy decisions. In Li, W., Susilo, W., Tupakula, U. K., Safavi-Naini, R., and Varadharajan, V., editors, *ASIACCS*, pages 298–309. ACM.

[Reiter, 1980] Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13:81–132.

[Sergot, 2004] Sergot, M. (2004). $(\mathcal{C}/\mathcal{C}+)^{++}$: An action language for modelling norms and institutions. Technical Report 2004/8, Department of Computing, Imperial College London.

[Sergot, 2007] Sergot, M. (2007). Action and agency in norm-governed multi-agent systems. In Artikis, A., O'Hare, G. M. P., Stathis, K., and Vouros, G. A., editors, *ESAW*, volume 4995 of *Lecture Notes in Computer Science*, pages 1–54. Springer.

[Sergot, 2008] Sergot, M. (2008). The logic of unwitting collective agency. Technical Report 2008/6, Department of Computing, Imperial College London.

[Sergot and Craven, 2005a] Sergot, M. and Craven, R. (2005a). Logical Properties of Nonmonotonic Causal Theories and the Action Language $\mathcal{C}+$. Technical Report 2005/5, Department of Computing, Imperial College London.

[Sergot and Craven, 2005b] Sergot, M. and Craven, R. (2005b). Some logical properties of nonmonotonic causal theories. In Baral, C., Greco, G., Leone, N., and Terracina, G., editors, *Logic Programming and Nonmonotonic Reasoning*, volume 3662 of *Lecture Notes in Computer Science*, pages 198–210. Springer.