A Clausal Logic for Deontic Action Specification (Extended Abstract)

Ron van der Meyden
Department of Computer Science,
Rutgers University*

August 17, 1998

Abstract

This paper investigates the theoretical foundations of a clausal action specification language designed to support artificial intelligence models of legal reasoning. We focus on two issues. First, action compositionality, which we achieve by treating actions as propositions, so that we may admit logical relations between actions and treat conjunction of actions as parallel composition. Second, we consider deontic actions, which change the interpretation of modal operators on actions, specifically the modality of permission. Each specification in the language is guaranteed to have a unique model, a Kripke structure which describes the possible actions, the effects of these actions and how the permissible actions change from state to state. This model is obtained as the fixpoint of an operator on a class of Kripke structures. We establish completeness of a top-down proof theory for query answering in this model.

1 Introduction

The representation of action has been a central concern for artificial intelligence since the beginnings of the discipline [7, 16]. By and large, however, the field has concentrated on simple 'blocks-world' examples [11, 15], and does not yet seem to have progressed to realistically complex problems. One of the consequences of this is that logical formalisms for the specification of actions have failed to address the 'software engineering' issue of how to construct and maintain large and complex action specifications. This paper is a preliminary theoretical investigation of an approach to action

*This paper appears in Proc. International Logic Programming Symposium, MIT Press, 1991. Author's current address: Computing Sciences, University of Technology, Sydney, PO Box 123 Broadway NSW 2007, Australia. Email: ron@socs.uts.edu.au, WWW: http://www-staff.socs.uts.edu.au

specification designed with these issues in mind. One of the purposes of the paper is to advocate a logic programming approach to modal logics of action.

Our work originates in a research effort to construct a logical language designed to support artificial intelligence models of legal reasoning [18, 22, 23]. Specifically, this work is motivated by the need for a logical framework in which to describe the facts of corporate taxation cases, i.e., facts about the world of business transactions, stocks, bonds, dividends, liquidations, corporate reorganizations etc. The representation of this domain of corporate activity poses several problems for standard AI representations of action.

First, the space of actions in this domain is large and complex. However, actions are not unrelated: many actions are complex composites of simpler actions. This makes it desirable that logical relationships between actions be expressible, so that we may build up definitions of actions in the same way we build up definitions of static concepts. AI representations of action have generally treated actions as atomic and unrelated, and have not provided ways to express logical relationships between actions.

Second, an adequate description of the world of corporate transactions requires representations of the *deontic* modalities on actions: permission, obligation and prohibition. Corporate securities are viewed by the law as a "bundle of rights" of the holder to perform certain actions, such as the right to vote at general meetings of stockholders, the right to transfer the security to others etc. Rulings on the tax consequences of a series of transactions involving individuals and corporations are typically based on considerations involving the way in which the transactions affect the rights and obligations of the participants. The actions agents may take in this world include "deontic actions" which change not only the state of the world but also the actions permitted, obligatory or prohibited at the next and future states. For example, the sale of a security involves the transfer of the rights associated with the security from the seller to the buyer. More complicated situations arise: in a proxy arrangement, a stockholder transfers to another his right to vote, i.e., he performs an action such as "permit Jones to vote at the meeting". In case Jones himself cannot attend the meeting he may also wish to "permit Jones to permit Smith to vote at the meeting". We are therefore faced with the need to represent complicated interactions between actions and modalities on actions. We restrict our attention in this paper to the modality of permission. (Our formal treatment of this modality can be given other interpretations: the reader may prefer to read "action a is physically possible" instead of our "action a is permitted".)

Our approach to these issues is as follows. Actions and permission of actions will be represented using modal operators with Kripke semantics. We present a Horn-clause-like language for specifying the meaning of actions. By use of a minimization policy in the style of the minimal Herbrand model semantics for logic programs [32], each specification in the language has a unique model, a Kripke structure which describes the possible actions, the effects of these actions, and how the permissible actions change from state to state. We show that this minimal model is the least fixpoint of an operator on a class of Kripke structures. The main technical contribution of the paper is to provide a complete proof theory for query answering in this structure.

The logic programming style formulation gives our style of action representation several advantages over approaches prevalent in the AI literature. One of the useful aspects of logic programming is that it builds-in, through negation-as-failure, a 'closed-world-assumption', which states that all facts not derivable from those explicitly stated are false. This obviates the need to represent a large number of 'negative facts', and allows us to concentrate on specifying the positive aspects of the domain we are trying to model. One of the contributions of this paper is to show how to apply a closed world assumption to action semantics and modal operators on actions.

Closely related is the fact that negation as failure provides a way to handle exceptions and other structural complexities of real world domains related to non-monotonicity, such as the frame problem and the qualification problem [30]. In the AI literature, proposals to handle these complexities are often couched as theories in a first-order logic of arbitrary structure, together with a second-order 'circumscription' condition. While this may solve the representational problems that are the focus of these studies, there is a serious drawback. Being second-order, the circumscription condition has no complete proof theory. There are some cases in which circumscription may be shown to be equivalent to a first-order theory, but it is frequently not shown that the theories of action proposed fall in these classes.

We take a more disciplined approach to action representation. Rather than circumscribe an arbitrary logical theory we describe our minimization policy model-theoretically, and then identify a class of sentences with respect to which we show the existence of a unique minimal model, and the completeness of a top-down proof theory. The next section sketches the major aspects of our proposal.

2 Overview

Let us first consider action compositionality. Typically, AI representations of action have represented actions as atomic objects, each of which is defined independently of other actions, either by "add-delete lists" (STRIPS) [7] or by axioms stating the effects of actions, as in the situation calculus [16]. However, in complex domains one often finds that there are logical relationships between actions. For example, it is desirable to view the complex action "grant a stock option to all preferred stock holders" as entailing the actions "grant a stock option to x" for *each* preferred stock holder x. Having defined the action $grant{-}option(x)$, it is wasteful to have to define $preferred{-}option{-}grant$ from scratch. Failing to recognize the logical relationship may also lead to inconsistencies if at some later stage the action $grant{-}option$ has its definition updated. All the usual software engineering considerations dictate that we be allowed to compose previously defined actions to define new actions.

Part of our approach to handling action compositionality will be to treat actions as propositions, so that logical relationships between actions may be treated as such. (We also intend to allow such action composition operators as sequential composition and non-deterministic choice, but exclude these from the present paper for simplicity.) Thus, action a's necessarily involving action b is represented by the implication $a \Rightarrow b$. This approach has the added benefit that simultaneous performance of actions may be represented by logical conjunction.

With each state we associate the set of propositional atoms true at the state, plus a set of action atoms. The latter are interpreted as the actions that were involved in the composite action that produced the state from its predecessor. Thus, we may write

$$\begin{gathered} \mathit{preferred{-}stock}(s) \wedge \mathit{owns}(x, s) \Rightarrow \\ \circ(\mathit{preferred{-}option{-}grant} \Rightarrow \mathit{grant{-}option}(x)) \end{gathered}$$

to express the example above. Here the operator $\circ$ is an 'at-all-next-time' operator, and when a and b are action constants the clause $\circ(a \Rightarrow b)$ may be read "if the next action involves doing a then it involves doing b also".

Since actions are treated as propositions holding at the state they bring about, we may also write $a \Rightarrow p$ where a is an action and p is a proposition to express the fact that p is one of the effects of a. For example,

$$\mathit{transfer}(y, x) \Rightarrow \mathit{owns}(x, y)$$

expresses that one result of a transfer of y to x is that x owns y. In order to ensure that each specification has a unique model, we apply a 'closed world assumption' [29]: a proposition is a consequence of an action just in case there is a proof to that effect.

Of course we must also have some 'frame axioms' [17] to ensure persistence of certain propositions. One way this could be done is to associate to each propositional atom p an action atom $\mathit{delete{-}p}$ and state the frame axioms in the form

$$p \Rightarrow \circ(\neg \mathit{delete{-}p} \Rightarrow p)$$

for the persistent propositions. The negation symbol ' $\neg$ ' in this formula is intended to be a 'negation-as-failure', i.e., $\neg q$ is to hold just when q is not derivable. This allows us to give succinct definitions of actions in the STRIPS style by writing axioms only for the atoms added or deleted by an action. For example, the definition of *transfer* is now completed by adding the axiom

$$\mathit{owns}(z, y) \Rightarrow \circ(\mathit{transfer}(y, x) \Rightarrow \mathit{delete{-}owns}(z, y)).$$

It is not our intention to force this solution of the frame problem on the user, however, merely to make it available by providing negation-as-failure as a feature of the language.

The discussion so far has left open the question of what compounds of simultaneously performed actions are to be possible at a given state in our unique model. Another 'minimization principle' is introduced for this purpose: at each state we take the maximal set of action combinations allowed by the specification, and let each of these correspond to a successor of the state. Thus, if we have just the actions a, b and no rules to relate these actions at a state s, then there is a successor of s corresponding to each of the compound actions $\emptyset, \{a\}, \{b\}, \{a, b\}$. However, if there is a constraint $a \Rightarrow b$ active at s then we have only successors for each of $\emptyset, \{b\}, \{a, b\}$. Note that the set of active constraints can vary from state to state. In the ownership example, if $owns(z, y)$ holds in a state then the constraint

$$transfer(y, x) \Rightarrow delete\!-\!owns(z, y)$$

holds for the successors of the state. As the ownership relations change so does the set of such active constraints. It follows from this that the branching structure of the minimal model may also vary from state to state.

One of the consequences of this minimization policy is that the set of successors of a state may be viewed as a lattice, where the order is that derived from inclusion of the action sets. This allows us to give a semantic interpretation of "doing just a", where a is a conjunction of action atoms. Since doing a may involve other actions, this will not in general correspond semantically to the successor with action set $\{a\}$. Instead we take the minimal element of the lattice containing a. For example, in case we have only a constraint $a \Rightarrow b$, "doing just a" corresponds to the successor with action set $\{a, b\}$. Proof theoretically, the action "just a" corresponds to the compound action consisting of a and all the actions that can be derived from a.

Having defined the action "just a", we may define two modalities taking "doing just a" as an argument. First, $\pi(a)$ expresses "doing just a is permitted". Deontic actions like *pa*: "permit a" and *ppa*: "permit the action permit a" may then be specified by rules of the form $pa \Rightarrow \pi(a)$ and $ppa \Rightarrow \pi(pa)$. The closed world assumption on consequences of actions also applies to these deontic consequences: an action will be permitted in the minimal model only if it can be proved to be permitted. That is, we apply a 'principle of least privilege'. We also have the dynamic logic [28] modality 'after': $\langle a \rangle p$ expresses that "doing just a" results in a state satisfying p. Proof-theoretically, this will correspond to assuming that a holds in the next state and showing that p follows. This modality allows us to formulate hypothetical queries of the form "would p become true if I did just a?"

It is also important for legal applications to have a modality that analyzes the legal powers of agents by asking "is there any permitted action that has consequences p?" To handle this we introduce a modality P so that we may write such clauses as

$$P(owns(y, z)) \Rightarrow transferable(z, y)$$

to define the predicate "z is legally transferable to y", which holds just when some action resulting in y's ownership of z is permitted.

The language developed in this paper falls short of the programme just sketched in two ways. First, we consider only a propositional logic, so as to be able to focus on the theoretical problems raised by the minimization policy. This is not a serious deficiency, however, as we construct our theory in such a way that the proof theory in the predicate case may be obtained by a 'lifting lemma' from the propositional case. Second, we disallow negation in the body of rules. This restriction is more serious, and its removal will involve considerable extra work. However, the present paper lays the foundations for such a generalization by showing that a unique model may be associated to every specification in the negation-free language. In this we follow standard methodology in the theory of logic programming. (We do show in the full version of the paper that we can already handle negation in goals.)

In Section 3 we provide a formal Kripke semantics for the modal operators of our language, formulated on a fully general class of structures. Section 4 then defines tree models, a subclass of this set of models which plays a role in our theory analogous to the role of Herbrand structures for the usual theory of logic programs. We construct a partial order on the class of tree models which captures the three types of closed world assumption we have mentioned (on propositions, action definitions and permissions) and show that the class forms a complete lattice with respect to this order. In section 5 we introduce a clausal fragment of the language and show that every program in this fragment has a least model which may be obtained as the least fixpoint of an operator on the class of tree models. Section 6 then exploits this result to show completeness for a top-down proof procedure. Section 7 discusses related work.

3 The Modal Operators: Syntax and Semantics

In this section we introduce a modal language and provide a general model theoretic semantics. This language will be restricted to a clausal subset in Section 5. We suppose that $\mathcal{ACT}$ and $\mathcal{PROP}$ are disjoint *infinite* sets of constants (infinite because we have in mind a generalization of the propositional language of this paper to first-order). We will refer to the elements of $\mathcal{ACT}$ as *action constants* and to the elements of $\mathcal{PROP}$ as *propositional constants*. Besides these the language contains the logical symbols $\wedge, \Rightarrow, \neg$, the unary modal operators $\Box, \circ, P, \pi$ and the binary modal operator $\langle\rangle$.

The set of formulae of the language is the smallest set satisfying

1. Any propositional constant or action constant is a formula.
2. If $A_1 \dots A_n$ are action constants then $\pi(A_1 \wedge \dots \wedge A_n)$ is a formula.
3. If φ is a formula then $\Box\varphi, \circ\varphi$ and $P\varphi$ are formulae.
4. If φ_1, φ_2 are formulae then $\varphi_1 \Rightarrow \varphi_2$, $\varphi_1 \wedge \varphi_2$ and $\neg\varphi_1$ are formulae.
5. If $A_1 \dots A_n$ are action constants and φ is a formula then $\langle A_1 \wedge \dots \wedge A_n\rangle\varphi$ is a formula.

A *model structure* for the language is a Kripke structure of the form $K = \langle W, \tau, \Phi, R, P\rangle$ where W is a set of *worlds* (also called *states*), the functions $\tau : W \rightarrow \mathcal{P}(\mathcal{ACT})$ and $\Phi : W \rightarrow \mathcal{P}(\mathcal{PROP})$ associate a set of action constants and propositional constants with each world, respectively, and R and P are access relations over the set of worlds, i.e., subsets of $W \times W$. If $w \in W$ is a world then we define the set of possible successors of w by $succ(w) = \{v \in W \mid wRv\}$ and the set of actions possible by $actions(w) = \{\tau(v) \mid v \in W, wRv\}$. If *act* is a subset of $\mathcal{P}(\mathcal{ACT})$ then we will write $act{\Uparrow}s = \{s' \in act \mid s \subset s'\}$. We will say that *act* is an *action lattice* if $\mathcal{ACT} \in act$ and for all $s \subset \mathcal{ACT}$ we have $\cap act{\Uparrow}s \in act$, i.e., the intersection of the sets in *act* which contain s is itself a member of *act*. This means that *act* contains a least element greater than any $s \subset \mathcal{ACT}$. A model structure is a *model* provided

1. $P \subset R$.
2. For every world w the set $actions(w)$ is an action lattice.
3. For all worlds w, v_1, v_2 , if wRv_1, wRv_2 and $\tau(v_1) = \tau(v_2)$ then $v_1 = v_2$.

Intuitively, the relation R represents the *possible* state evolutions in the model. The access relation P indicates the set of permitted transitions. The first constraint ensures that permitted transitions are in fact possible. Constraint 3 states that action is deterministic: performing a particular compound action always leads to the same state. Because of constraints 2 and 3, if $s \subseteq \mathcal{ACT}$ we may define $\sqcap succ(w){\Uparrow}s$ to be the unique successor v of w with $\tau(v) = \cap actions(w){\Uparrow}s$. That is, $\sqcap succ(w){\Uparrow}s$ is the successor of w whose action set is the minimal set containing s. The action "doing just s" performed at the state w results in a transition to the state $\sqcap succ(w){\Uparrow}s$.

The following are the important clauses of the truth definition. We slightly abuse notation by writing $\{A\}$ for $\{A_1, \ldots, A_n\}$ when $A = A_1 \wedge \ldots \wedge A_n$.

$$
\begin{array}{lll}
K, w \models A & \text{iff} & A \in \Phi(w), \text{ when } A \in \mathcal{PROP} \\
K, w \models A & \text{iff} & A \in \tau(w), \text{ when } A \in \mathcal{ACT} \\
K, w \models \Box\varphi & \text{iff} & K, v \models \varphi \text{for all } v \in W \text{with } wR^*v \\
K, w \models \circ\varphi & \text{iff} & K, v \models \varphi \text{for all } v \in succ(w) \\
K, w \models P\varphi & \text{iff} & K, v \models \varphi \text{for some } v \in W \text{with } wPv \\
K, w \models \pi(A) & \text{iff} & wP \sqcap succ(w){\Uparrow}\{A\} \\
K, w \models \langle A\rangle\varphi & \text{iff} & K, \sqcap succ(w){\Uparrow}\{A\} \models \varphi
\end{array}
$$

The intuitive interpretation of the modal operators is as follows. The modalities $\Box$ and $\circ$ refer to the *possible* futures. The formula $\Box\varphi$ holds at a state when φ holds at all states accessible via some sequence of actions. The formula $\circ\varphi$ holds at a state when φ holds at all possible successor states, i.e., φ always holds at the 'next time'. The modalities P and π refer to the *permitted* futures. The formula $P\varphi$ holds when φ holds at some permitted successor. Finally, $\pi(A)$ holds when the action "just A " results in a permitted successor state and $\langle A\rangle\varphi$ holds when the action "just A" results in a state satisfying φ .

4 Minimization

In this section we introduce the minimization policy which captures the three types of closed world assumption. Let $K_1 = \langle W_1, \tau_1, \Phi_1, R_1, P_1 \rangle$ and $K_2 = \langle W_2, \tau_2, \Phi_2, R_2, P_2 \rangle$ be two models. A mapping $h : W_1 \rightarrow W_2$ is a *homomorphism* if

1. If wR_1v then $h(w)R_2h(v)$, for all w, v .

2. If wP_1v then $h(w)P_2h(v)$, for all w, v .

3. For all w , $\tau_1(w) \subset \tau_2(h(w))$ and $\Phi_1(w) \subset \Phi_2(h(w))$.

4. For all $s \subset \mathcal{ACT}$ and all w , $h(\sqcap succ_1(w) \Uparrow s) = \sqcap succ_2(h(w)) \Uparrow s$.

Thus, homomorphisms map states to states satisfying more facts and allowing more possible transitions and more permitted transitions. The last constraint ensures that the interpretation of "doing just s" in the first model is mapped to the interpretation of "doing just s" in the second. It can be shown that we must have $actions_2(h(w)) \subset actions_1(w)$ for this to be possible, i.e., the model K_2 satisfies more constraints action constants.

In order to provide a fixpoint semantics we let $\mathcal{TREE}$ be the set of models which have the structure of an infinite tree. Note that we allow the branching structure of the tree to vary from state to state, i.e., the logical relations between action constants can vary from state to state. The class $\mathcal{TREE}$ is semantically complete in the sense that the consequence relation is unaltered by restricting to tree models. This is a consequence of the next lemma, a straightforward generalization of a well-known unfolding construction for modal logic [5].

Theorem 4.1: Lemma: Let K be a model and w a state of K. There exists a tree model K' and a homomorphism from K' to K, mapping the root to w, such that for all formulae φ and states v of K', we have $K', v \models \varphi$ if and only if $K, h(v) \models \varphi$.

Because of constraint 4, given two tree models, if there exists a homomorphism preserving the root, it is unique. Define a partial order $\leq$ on $\mathcal{TREE}$ by $K_1 \leq K_2$ if there exists a homomorphism from K_1 to K_2 preserving the root.

Theorem 4.2: Lemma: The set $\mathcal{TREE}$ is a complete lattice under the order $\leq$.

Let $\mathcal{SEQ} = \mathcal{P}(\mathcal{ACT})^*$ be the set of all finite sequences of sets of action constants. We represent tree models as follows. A *tree structure* is a tuple of the form $K = \langle r, W, \Phi, P \rangle$ where

1. r is a subset of $\mathcal{ACT}$, called the *root* of K ,

2. W is a prefix-closed subset of $\mathcal{SEQ}$,

3. for all $w \in W$, $\Phi(w) \subset \mathcal{PROP}$, and

4. P is a subset of W .

Note that because W is prefix-closed it contains the null sequence Λ , which represents the root of the tree.

Each tree structure corresponds to a model structure $K' = \langle W', \tau', \Phi', R', P' \rangle$ where $W' = W$, $\Phi' = \Phi$ and

$$\begin{aligned} \tau'(\Lambda) &= r \\ \tau'(ws) &= s \text{for all } ws \in W \\ R' &= \{(w, ws) \mid ws \in W\} \\ P' &= \{(w, ws) \mid ws \in P\}, \end{aligned}$$

using the notational convention that s stands for a subset of $\mathcal{ACT}$ and w stands for an element of $\mathcal{SEQ}$. Observe that, because of the prefix-closure constraint, the relation R' gives W the structure of a tree. Using this identification, we note that constraints 1 and 3 of the definition of a model are immediately satisfied by a tree structure. Thus, a tree structure is a *tree model* just when for all $w \in W$ the set $actions(w) = \{s \mid ws \in W\}$ is an action lattice. Note that since an action lattice is by definition non-empty (always $\mathcal{ACT} \in act$), each branch of a tree model is infinite.

Given a tree model K with states W, it is possible to associate with any element $w = s_1 \ldots s_k \in \mathcal{SEQ}$ a unique state $\sqcap W{\uparrow}w$ of the model, which is intuitively the state reached by performing the sequence of actions "just s_1 " then "just s_2 " then ... "just s_k " from the root Λ. Formally, $\sqcap W{\Uparrow}\Lambda = \Lambda$ and $\sqcap W{\Uparrow}ws = \sqcap succ(\sqcap W{\Uparrow}w){\Uparrow}s$. With the appropriate definitions this is in fact a greatest lower bound, related to Shoham's notion of chronological minimization [30], hence the notation.

Example 1: Figure 1 below illustrates a tree model. A colon separates the action constants from the propositional constants at each state. Action lattices are indicated by hexagons. For example,

$$actions(\{b\}) = \{\emptyset, \{b\}, \{j\}, \{b, j\}, \{j, p\}, \{b, j, p\}\}$$

Broken lines indicate the Hasse diagram of the order on action lattices. An arc from a state to an action lattice indicates that there are transitions to all states in the lattice. States to which transitions are permitted are contained in rectangles. For example, the transition from state $\{b\}$ to state $\{b\}\{j, p\}$ is permitted. If w is the sequence $\{b\}\{p\}$ then $\sqcap W{\Uparrow}w$ is the state $\{b\}\{j, p\}$.

5 The Clausal Language: Syntax and Semantics

The class of *programs* is defined as follows. A formula will be called a *goal* if it is a propositional atom or an action atom, or has one of the forms $G_1 \wedge G_2$, $G_1 \vee G_2$, PG , $\langle A \rangle G$ or $\pi(A)$ where the G are goals and A is a conjunction of action constants. An *action-goal* will be any action constant, $true$, or a formula of the form $G_1 \wedge G_2, G_1 \vee G_2$ where the G_i are action goals. Let G_1, G_2 be arbitrary goals.

1. If A is an atomic proposition then a formula $G_1 \Rightarrow A$ or $G_2 \Rightarrow \circ(G_1 \Rightarrow A)$ is a *propositional clause.*

2. If A is an action atom then a formula $G_1 \Rightarrow A$ or $G_2 \Rightarrow \circ(G_1 \Rightarrow A)$ is an *action clause* provided G_1 is an action goal. (The goal G_2 may be arbitrary.)

3. If A is a conjunction of action constants, a formula $G \Rightarrow \pi(A)$ where G is an arbitrary goal is a *pi-clause.*

4. If C is a propositional (action, pi) clause of any of the above forms, then $\Box C$ is also a propositional (action, pi) clause.

A *program* will be a set of propositional, action and pi clauses. Clauses of the form C are used to initialize the root; clauses of the form $\Box C$ are intended for action and predicate definitions: these hold at all states. Note the operators $\langle\rangle, P$ are not allowed to occur in the heads of clauses.

If $w \in \mathcal{SEQ}$ and Σ is a program we will write Σ^w for the set of clauses $\{C \mid C \in \Sigma \text{or } \Box C \in \Sigma\}$ when w is the root Λ , and $\{C \mid \Box C \in \Sigma\}$ otherwise. The set Σ^w is just the set of $\Box$ -free "instances" of clauses from Σ which apply at w . In addition, given a program Σ and a tree model K, we will write $\Gamma_{K,w}$ for the set of constraints of the form $G \Rightarrow A$ where $A \in \mathcal{ACT}$ and either

1. $\Box G \Rightarrow A \in \Sigma$, or

2. $\Box G' \Rightarrow \circ(G \Rightarrow A) \in \Sigma$ and $K, w \models G'$, or

3. $w = \Lambda$ and $G' \Rightarrow \circ(G \Rightarrow A) \in \Sigma$ and $K, w \models G'$.

This is intuitively the set of constraints on the action constants that must hold for each element of $actions(w)$ if the model is to satisfy the program.

We now define an operator $T : \mathcal{TREE} \rightarrow \mathcal{TREE}$ for each program Σ . Let $K = \langle r, W, \Phi, P\rangle$ be a tree model. The model $K' = T(K)$ is defined in two stages: first we compute the set of states W', then the rest of the structure. Note that if $w \in \mathcal{SEQ}$ then the result $\sqcap W \Uparrow w$ of 'executing w' depends only on the elements of W of length no greater than the length of w. We use this fact to give an inductive construction of the set W' by $\Lambda \in W'$ and for each $w \in W'$,

$$actions'(w) = \{s \mid \forall v \in W (w = \sqcap W' \Uparrow v \Rightarrow [s \in actions(v) \wedge s \models \Gamma_{K,v}])\}.$$

That is, we specify the sequences ws in W' of length $n + 1$ by reference to the sequences w of length n. The intent of this definition is to enforce constraints on the action constants. For example, the definition of $actions'(\Lambda)$ simplifies to $\{s \in actions(\Lambda) \mid s \models \Gamma_{K,\Lambda}\}$. Here we simply delete those branches which do not satisfy all the necessary constraints. In general, W' will be a strict subset of W. This means that the interpretation of actions changes in K': an action resulting in state v in K will result in state $\sqcap W' \uparrow v$ in K'. Several elements of W may map to a single element of W' under the mapping $v \rightarrow \sqcap W' \uparrow v$. The definition takes this into account by enforcing at a state $w \in W'$ all those constraints $\Gamma_{K,v}$ which must hold at any state $v \in W$ which maps to w. In addition, we also take all constraints which already hold at v, represented by $actions(v)$.

The intuition underlying the rest of the definition is very similar to that for the classical operator of van Emden and Kowalski. For r' and $\Phi'(\Lambda)$ we simply enforce those facts which follow in one deductive step from some rule in the program:

$$r' = \Big\{A \in \mathcal{ACT} \mid G \Rightarrow A \in \Sigma^{\Lambda} \text{and } K, \Lambda \models G\Big\}$$
$$\Phi'(\Lambda) = \Big\{A \in \mathcal{PROP} \mid G \Rightarrow A \in \Sigma^{\Lambda} \text{and } K, \Lambda \models G\Big\}$$

For $w' \neq \Lambda$ the definition of $\Phi'(w)$ is similar, but slightly complicated by the fact that we need to collect inferences from all states of K which map to w:

$$\Phi'(w) = \{A \in \mathcal{PROP} \mid \text{for some } vs \in W \text{with } w = \sqcap W' \uparrow vs, \text{either}\}$$
$$G \Rightarrow A \in \Sigma^{vs} \text{and } K, vs \models G \text{or}$$
$$G' \Rightarrow \circ(G \Rightarrow A) \in \Sigma^{v} \text{and } K, v \models G' \text{and } K, vs \models G\}$$

Finally, for the permitted states we again enforce all the 'single-step-inferences' in K, and then map to K':

$$P' = \{\sqcap W' \uparrow w\{A\} \mid w \in W \text{and } G \Rightarrow \pi(A) \in \Sigma^{w} \text{and } K, w \models G\}.\}$$

Put $T(K) = \langle r', W', \Phi', P' \rangle$. Our operator T can be shown to satisfy the two crucial lemmas:

Theorem 5.1: **Lemma:** For each program the operator T is monotonic.

Theorem 5.2: **Lemma:** If K is a tree model then $K, \Lambda \models \Sigma$ if and only if $T(K) \leq K$.

It follows from these lemmas that each program has a minimal model. Because of the way we defined the order on the lattice of tree models, this model corresponds to minimizing the set of propositions true at each state, minimizing the set of permitted transitions and *maximizing* the branching at each state. We encourage the reader to understand the minimal model in terms of the minimization policy and the proof theory to follow rather than via the operator T.

In the classical logic programming theory, the minimal model contains just those atoms that are consequences of the program. A similar property holds for our construction, because of the following lemma:

Theorem 5.3: **Lemma:** If h is a homomorphism from a model K_1 to a model K_2 then $K_1, w \models G$ implies $K_2, h(w) \models G$, for every goal G and state w.

Figure 1: Part of a minimal model

Since there exists a homomorphism from the minimal tree model to any model of the program, our minimal model satisfies exactly those goals true in all models. (Of course, this will no longer hold once we introduce negation into the theory.)

Example 2: The following is a propositional version of the stock option example of Section 2. We suppose that $\mathcal{ACT}$ contains three actions b: "Jones buys preferred stock in Acme Co.", j: "Acme Co. grants a stock option to Jones", and p: "Acme Co. grants a stock option to all its preferred stock holders". We work with a single proposition h: "Jones holds preferred stock in Acme Co." Consider the program Σ consisting of the following rules:

$$\begin{array}{l} \Box\pi(b) \\ \Box\pi(p) \\ \Box(b \Rightarrow h) \\ \Box(h \Rightarrow \circ(p \Rightarrow j)) \end{array}$$

The first two rules concern the permitted actions. The first rule states that Jones may always buy preferred stock in Acme Co, the second that Acme Co. may always grant a stock option to all its preferred stock holders simultaneously. The third rule describes the effects of the action b: if Jones buys preferred stock then he becomes a preferred stock holder. The final rule gives the semantics of the action p: if Jones is a preferred stock holder and Acme grants a stock option to all preferred stock holders then Acme grants a stock option to Jones.

Figure 1 illustrates a small part of the minimal model of this program. ction lattices are indicated by hexagons, and an arc from a state in an action lattice indicates that there are transitions to all states in the lattice. Broken lines indicate the Hasse diagram of the order on action lattices. States to which transitions are permitted are contained in rectangles. The remainder of the model has a structure similar to that shown. For each state w satisfying h the constraint $p \Rightarrow j$ must hold at the next state, so we have $succ(w) \approx \mathrm{L}_2$. Successors of states not satisfying h are subject to no constraints, so we have $succ(w) \approx \mathrm{L}_1$. $\Box$

In order to prove completeness we use the following characterization of the minimal model.

Theorem 5.4: **Theorem:** Let $K_0 = \bot$ be the least tree model and put $K_{i+1} = T(K_i)$. Then the least fixpoint of T is $\bigsqcup_{i=1}^{\infty} K_i$

Unfortunately, the operator T is not continuous (this has to do with the infiniteness of the set of action constants) so this result requires some effort. We identify a notion of continuity of *models*, which T preserves. Since T also commutes with least upper bounds of continuous models, this gives us the desired characterization. The counter-example to the continuity of the operator T involves discontinuous models.

6 Proof Theory

In this section we introduce a tableaux proof theory for the clausal fragment. By inverting the rules of this proof theory one obtains a proof procedure which is suitable for use as a top-down interpreter in the style of SLD-resolution for Horn clause logic programming. Our proof theory consists of the following set of rules defining the relation $\Sigma, N \vdash G$ between programs Σ , *finite* elements N of $\mathcal{SEQ}$ and goals G . If W is the set of states of the minimal model then N will be the proof-theoretic object corresponding to the state $\sqcap W \uparrow N$.

Since an action atom holds at a state if it was part of the compound action producing the state from its predecessor, we have the rule

1. If $s \subset \mathcal{ACT}$ is finite and $A \in s$ then $\Sigma, Ns \vdash A$.

Clearly this rule does not tell us what *derived* action atoms must hold. The next two rules, which are generalizations of *Modus Ponens*, deal with these. These rules apply to both action constants A and propositional constants A. Recall from Section 5 that Σ^N denotes the set of $\Box$ -free instances of rules from Σ which apply at N.

1. If $G \Rightarrow A \in \Sigma^N$ and $\Sigma, N \vdash G$ then $\Sigma, N \vdash A$.

2. If $\Sigma, N \vdash G_1$ and $\Sigma, Ns \vdash G_2$ and $G_1 \Rightarrow \circ(G_2 \Rightarrow A) \in \Sigma^N$ then $\Sigma, Ns \vdash A$.

Next, we have one rule for each of the modal operators. Proof-theoretically, performing the action A corresponds to asserting A in the next state and deriving its consequences. Thus, for the dynamic modality 'after' we have the rule

1. If $\Sigma, N\{A\} \vdash G$ then $\Sigma, N \vdash \langle A \rangle G$.

The rules for the permission operators are

1. If $\Sigma, N \vdash G_1$ and $\Sigma, N\{A\} \vdash G_2$ and $G_1 \Rightarrow \pi(A) \in \Sigma^N$ then $\Sigma, N \vdash PG_2$.

2. Suppose A, B satisfy $\Sigma, N\{A\} \vdash B$ and $\Sigma, N\{B\} \vdash A$. If $\Sigma, N \vdash G$ and $G \Rightarrow \pi(A) \in \Sigma^N$ then $\Sigma, N \vdash \pi(B)$.

Note that the precondition of rule 6 is necessary because we may have $\sqcap W \uparrow N\{A\} = \sqcap W \uparrow N\{B\}$. We omit the obvious rules for conjunction, disjunction and the constant *true*.

Theorem 6.1: Theorem: (Soundness & Completeness) Let K be the minimal tree model of Σ . Then $\Sigma, N \vdash G$ if and only if $K, \sqcap W \uparrow N \models G$.

The proof requires a rather delicate argument. By the convergence result K is the least upper bound of the countable collection of models $K_i = T^i(\bot)$. The idea of the completeness half of the proof is to show that $K_i, \sqcap W_i \uparrow N \models G$ implies $\Sigma, N \vdash G$. This is done by induction on i, with a nested induction on the structure of G.

Figure 2: A top-down proof

$$
\begin{array}{ccc}
 & \Sigma, \Lambda \vdash \mathbf{P}(\langle p \rangle j) & \\
 & \mid \quad 5, \Box\pi(b) & \\
 & \Sigma, \{b\} \vdash \langle p \rangle j & \\
 & \mid \quad 4 & \\
 & \Sigma, \{b\}\{p\} \vdash j & \\
 & \mid \quad 3, \Box(h \Rightarrow \circ(p \Rightarrow j)) & \\
 & \bigwedge & \\
\Sigma, \{b\}\{p\} \vdash p & & \Sigma, \{b\} \vdash h \\
\mid \quad 1 & & \mid \quad 2, \Box(b \Rightarrow h) \\
\phi & & \Sigma, \{b\} \vdash b \\
 & & \mid \quad 1 \\
 & & \phi
\end{array}
$$

However, because the deletion aspect of the operator T detects inferences between action constants faster than inferences between propositional constants, the obvious ideas for the inductive step require that $K_i, \sqcap W_{i+1} \uparrow N \models G$ implies $\Sigma, N \vdash G$. A further nested induction is required to show that the latter in fact follows from the induction hypothesis. It may be possible to give a semantics that operates at the same speed as the proof procedure, but it appears this would require passing through structures which are not models because they violate the action lattice conditions.

Different top-down procedures can be constructed which invert the rules we have given, with varying degrees of intelligence. We will not attempt to go into details, but present an example of how the proofs might look.

Example 3: Let Σ be the program of Example 1. Figure 2 shows a top-down proof that the goal $\mathbf{P}(\langle p \rangle j)$ ("After some permitted action a stock option grant to all preferred shareholders involves a stock option grant to Jones") holds at the root. Each step is labelled with the number of the proof rule used and the clause of the program to which this rule is applied. Note that the branching is conjunctive. $\Box$

7 Related Work

Previous extensions of logic programming to modal languages include temporal logics [1, 3, 8, 9, 27] and epistemic logic [14]. Farinas del Cerro [6] describes a general framework, called MOLOG, for logic programming in a class of different modal languages. Also related is the work of Gabbay [10], McCarty [20, 21] and Miller [26] on intuitionistic logic programming. In particular, our fixpoint operator bears some similarity to that used by McCarty, which also operates by deleting states.

The literature on the semantics of modal logic programming is sparse. For the case of linear time temporal logic programs, Baudinet [3] provides a fixpoint semantics and some expressibility results. Balbiani et al. [2] provide a semantics for the class of languages MOLOG based on the skolemization of the existential quantifiers implicit in occurrences of the serial modal operator $\Diamond$ in the heads of rules. This semantics involves first transforming programs as follows. Each occurrence of $\Diamond$ in the head of an instance of a clause is replaced by a distinct modal operator $\langle f_i \rangle$, related to the modality $\Diamond$ by the containment of access relations $R_{\langle f_i \rangle} \subset R_{\Diamond}$. The translated program is interpreted with respect to tree-like structures in which every node has a single distinct successor associated with each operator $\langle f_i \rangle$. It is then possible to show that these structures form a complete lattice and to provide an operator over this lattice whose least fixpoint yields the 'minimal model' of the program. Although this work has some similarity to ours, in that tree models are used, a crucial difference is that we allow the branching structure of the tree to vary from state to state, whereas the trees in the MOLOG semantics have a fixed structure. We also note that our semantics is somewhat more elegant because the tree structure in the MOLOG semantics depends on the particular program, while we use the same class of trees for all programs. Note that we finesse the problems associated with existential operators in the heads of rules by defining the operator π in a way that strips it of existential import.

Georgeff [11, 12] comes closest to our treatment of the logical relations between actions with his version of the situation calculus which reifies actions and introduces a binary relation "causes" on actions. Although he has several interesting suggestions for keeping the presentation first-order, he eventually makes the usual appeal to circumscriptive mechanisms.

The order inherent in the notation $\sqcap W \!\uparrow\! N$ bears some similarity to the notion of 'chronological minimization' of Shoham [30]. Shoham's work is concerned with inadequacies, pointed out by Hanks and McDermott [13], of nonmonotonic logics for the representation of common-sense reasoning about time. Shoham presents an order over linear temporal models which sequentially minimizes a subset of the propositional constants, much in the way we sequentially minimize over the set $\mathcal{ACT}$. He then defines a nonmonotonic consequence relation by taking minimal models with respect to this order. However, he rejects this solution and goes on to define much more complicated and less intuitive semantics, for two reasons. First, he is dissatisfied with the need to distinguish certain propositions for minimization, claiming that there are no principled grounds on which such distinctions can be drawn in applications. Second, he also wishes to minimize the remaining propositions (our set $\mathcal{PROP}$). With regard to the second, we point out that the combination of our minimal model construction and the minimization in $\sqcap W \!\uparrow\! N$ results in the minimization of both sets of propositions. As to the first, we can only state that we find the distinction between actions and propositions perfectly acceptable. We also note that our clausal language is more powerful than Shoham's 'causal theories' in that we permit 'forward looking operators' in the body of rules.

Most philosophical work in deontic logic assumes that the deontic modalities are standard modal operators, applying to propositions, but this leads to unintuitive inferences known as the 'deontic paradoxes'. Castaneda [4] has long urged that these paradoxes will be resolved by taking the deontic modalities to apply to actions rather than propositions. In this paper we have taken somewhat of a hybrid view: the modality π applies to actions only, so can be read "it is permitted to do". On the other hand, we have treated actions as a special sort of proposition, and the modality P applies to any proposition, and can be read "it is not prohibited to bring it about that". A first attempt to construct a deontic logic based on dynamic logic was made by McCarty [19], but had no proof theory. Later work includes a reduction of deontic logic to dynamic logic by Meyer [24, 25]. We have argued elsewhere [31] that this reduction is not appropriate for legal applications, but the difficulties emerge only when sequential composition of actions is considered, which we have avoided here in order to concentrate on logical relations between actions. Our notion of "doing just a" is related to McCarty's 'strict satisfaction'; no similar notion appears in Meyer's work.

Acknowledgments Thanks to Thorne McCarty for his comments on the manuscript. This work was supported by a Rutgers University Graduate School Excellence Fellowship.

References

[1] M. Abadi and Z. Manna. Temporal logic programming. *Journal of Symbolic Computation*, 8:277–295, 1989.

[2] Ph. Balbiani, L. Farinas del Cerro, and A.Herzig. Declarative semantics for modal logic programs. In *Proceedings of the International Conference on Fifth Generation Computing Systems*, pages 507–514, 1988.

[3] M. Baudinet. Temporal logic programming is complete and expressive. In *Proceedings of the 16th Symposium on Principles of Programming languages*, pages 267–280, 1989.

[4] H.-N. Castaneda. The paradoxes of deontic logic. In R. Hilpinen, editor, *New Studies in Deontic Logic*. Reidel, Dordrecht/Boston, 1981.

[5] B.F. Chellas. *Modal Logic: an introduction*. Cambridge University Press, 1980.

[6] L. Farinas del Cerro. MOLOG: A system that extends PROLOG with modal logic. *New Generation Computing*, 4:35–50, 1986.

[7] R.E. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[8] M. Fujita, S. Kono, H. Tanaka, and T. Moto-oka. Tokio: Logic programming based on temporal logic and its compilation to prolog. In *Third International Conference on Logic Programming. LNCS 225*, pages 695–709. Springer-Verlag, 1986.

[9] D.M. Gabbay. Modal and temporal logic programming. In Anthony Galton, editor, *Temporal Logics and their Applications*, pages 197–237. Academic Press, 1987.

[10] D.M. Gabbay and U. Reyle. N-PROLOG: An extension of PROLOG with hypothetical implications I. *Journal of Logic Programming*, 1:319–355, 1984.

[11] M.P. Georgeff. Actions, processes, and causality. In M.P. Georgeff and A. Lansky, editors, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 99–122. Morgan Kaufman, 1987.

[12] M.P. Georgeff. Many agents are better than one. In Frank M. Brown, editor, *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, pages 59–75. Morgan Kaufman, 1987.

[13] S. Hanks and D. McDermott. Nonmonotonic logics and temporal projection. *Artificial Intelligence*, 33:379–412, 1987.

[14] Y.J. Jiang. Epistemic logic programming. In *Proceedings of the International Conference on Fifth Generation Computing Systems*, pages 515–523, 1988.

[15] Z. Manna and R. Waldinger. A theory of plans. In M.P. Georgeff and A. Lansky, editors, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 99–122. Morgan Kaufman, 1987.

[16] J. McCarthy. Situations, actions and causal laws. In Marvin Minsky, editor, *Semantic Information Processing*, pages 410–418. 1968.

[17] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, pages 463–502. 1969.

[18] L.T. McCarty. Reflections on TAXMAN: An experiment in artificial intelligence and legal reasoning. *Harvard Law Review*, 90:837–893, 1977.

[19] L.T. McCarty. Permissions and obligations. In *Proceedings IJCAI-83*, pages 287–294, August 1983.

[20] L.T. McCarty. Clausal intuitionistic logic. I. Fixed-point semantics. *Journal of Logic Programming*, 5(1):1–31, 1988.

[21] L.T. McCarty. Clausal intuitionistic logic. II. Tableau proof procedures. *Journal of Logic Programming*, 5(2):93–132, 1988.

[22] L.T. McCarty. A language for legal discourse. I. Basic features. In *Proceedings of the Second International Conference on Artificial Intelligence and Law*, pages 180–189. ACM Press, June 1989.

[23] L.T. McCarty and N.S. Sridharan. The representation of an evolving system of legal concepts. II. Prototypes and deformations. In *Proceedings IJCAI-81*, pages 246–253, 1981.

[24] J.-J. Meyer, H. Weigand, and R. Wieringa. A specification language for static, dynamic and deontic integrity constraints. In J. Demetrovics and B.Thalheim, editors, *Mathematical Fundamentals of Database Systems*. Springer LNCS No. 346, 1989.

[25] J.-J. Ch. Meyer. A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic*, 29(1), 1988.

[26] D. Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, 6(1):79–108, 1989.

[27] B.C. Moszkowski. *Executing Temporal Logic Programs*. Cambridge University Press, 1986.

[28] V. Pratt. Semantical considerations on Floyd-Hoare logic. In *Proceedings 17th IEEE Symposium on Foundations of Computer Science*, pages 109–121, October 1976.

[29] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Morgan Kaufman, 1978.

[30] Y. Shoham. *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence*. The MIT Press, 1988.

[31] R. van der Meyden. The dynamic logic of permission. In *Proceedings of the Fifth Symposium on Logic in Computer Science*, pages 72–78, 1990.

[32] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.