REPRESENTING ACTIONS IN LOGIC PROGRAMS AND DEFAULT THEORIES

A SITUATION CALCULUS APPROACH

HUDSON TURNER

▷ We address the problem of representing commonsense knowledge about action domains in the formalisms of logic programming and default logic. We employ a methodology proposed by Gelfond and Lifschitz which involves first defining a high-level language for representing knowledge about actions, and then specifying a translation from the high-level action language into a general-purpose formalism, such as logic programming. Accordingly, we define a high-level action language $\mathcal{AC}$, and specify sound and complete translations of portions of $\mathcal{AC}$ into logic programming and default logic. The language $\mathcal{AC}$ includes propositions that represent "static causal laws" of the following kind: a fluent formula ψ can be made true by making a fluent formula ϕ true (or, more precisely, ψ is a caused whenever ϕ is). Such propositions are more expressive than the state constraints traditionally used to represent background knowledge. Our translations of $\mathcal{AC}$ domain descriptions into logic programming and default logic are simple, in part because the noncontrapositive nature of causal laws is easily reflected in such rule-based formalisms. ◁

1. INTRODUCTION

In this paper we address the problem of representing commonsense knowledge about action domains in the formalisms of logic programming and default logic. We employ a methodology proposed by Gelfond and Lifschitz [GL93] which involves first defining a high-level language for representing commonsense knowledge about actions, and then specifying a translation from the high-level action language into a general-purpose formalism, such as logic programming. The advantage of starting with a high-level action language is that it can utilize a restricted syntax and a relatively simple semantics which is nonetheless adequate to capture our commonsense

Address correspondence to Hudson Turner, Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712, USA, hudson@cs.utexas.edu

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/96/$7.00

intuitions about the whole family of action domains expressible in the language. Moreover, given a suitable action language and a translation from that language into a general-purpose formalism, the correctness of our representations of knowledge about actions in the general-purpose formalism becomes a precise mathematical question, to be settled by proving the translation sound and complete.

Accordingly, in this paper we define a high-level action language $\mathcal{AC}$, and specify sound and complete translations of portions of $\mathcal{AC}$ into logic programming and default logic.

The action language $\mathcal{AC}$ is based closely upon the language $\mathcal{AR}$ of Giunchiglia, Kartha and Lifschitz [GKL95] and its predecessor $\mathcal{AR}_0$ [KL94]. The language $\mathcal{AC}$ improves on $\mathcal{AR}$ by adopting the method for representing causal background knowledge that was introduced by McCain and Turner in [MT95b]. In $\mathcal{AR}$, background knowledge is represented in the form of state constraints, which have been traditionally used for this purpose. Recently, many authors have argued that state constraints are inadequate for representing background knowledge in action domains, because they do not adequately represent causal relations [Gef90, Elk92, Bar95, Lin95, MT95b, Thi95]. Accordingly, $\mathcal{AC}$ includes propositions that express "static causal laws" of the following kind: if a fluent formula ϕ is caused to be true, then a fluent formula ψ is caused to be true. From such a causal law it follows that, in the action domain being described, one can make ψ true by making ϕ true. It also follows that in every possible state, ψ is true if ϕ is. In [MT95b] we argued for the usefulness of such propositions, and showed that they are much more expressive than state constraints.

The portion of the language $\mathcal{AC}$ that we correctly embed in logic programming subsumes the action language $\mathcal{A}$ [GL93]. By way of comparison, the original translation in [GL93] of $\mathcal{A}$ into logic programming was partial — it was sound only for a restricted subset of the $\mathcal{A}$ domain descriptions — and incomplete. Later sound and complete translations of $\mathcal{A}$ utilized extensions or variants of logic programming: abductive logic programming [DD93, Dun93], equational logic programming [Thi94], or disjunctive logic programming [Tur94]. For the portion of $\mathcal{AC}$ that corresponds to $\mathcal{A}$, we specify in this paper a translation into logic programming that is arguably simpler than any of those previously published, and does not require abduction or disjunction. Furthermore, the portion of $\mathcal{AC}$ that we embed in logic programming is considerably more expressive than $\mathcal{A}$.

We employ in this paper the answer set semantics for (nondisjunctive) logic programs with classical negation [GL91].[1] From a declarative point of view, nondisjunctive logic programming under the answer set semantics is essentially a special case of Reiter's default logic [Rei80]. In what follows, we will devote considerable attention to translations of $\mathcal{AC}$ into the more expressive formalism of default logic. In fact, our translation into logic programming is, for the most part, simply a special case of this more general translation.

In the default theories obtained by our translation from $\mathcal{AC}$, every default rule is either "justification-free" or "normal." In this way our approach is simpler than that of Morris [Mor88], which uses non-normal defaults. Moreover, our method has the considerable advantage of guaranteeing correct representations of commonsense knowledge for a wide variety of action domains.

[1] Recall (from [GL90]) that such programs are easily reducible to normal logic programs under the stable model semantics [GL88].

In Section 2 we illustrate the range of applicability of the definitions introduced in this paper, by considering four example action domains. We provide for each a brief informal description, a formalization in the action language $\mathcal{AC}$, and a corresponding formalization in logic programming, default logic, or both. After some preliminary definitions (Section 3), we define in Section 4 the high-level action language $\mathcal{AC}$. In Section 5 we specify a translation from a subset of $\mathcal{AC}$ into default logic, and state the relevant soundness and completeness theorems. We also specify a second, simpler translation which is sound and complete for a smaller subset of $\mathcal{AC}$. Section 5 also includes a comparison between the default theory we obtain for the classic Yale Shooting domain [HM87] and the default theories considered by Hanks and McDermott [HM87] and by Morris [Mor88]. In Section 6 we show that, under a simple syntactic restriction on the form of $\mathcal{AC}$ domain descriptions, the translations into default logic can be adapted to generate logic programs that correctly represent commonsense knowledge about actions. In Appendix A we present the Splitting Theorems for default logic from [Tur96b]. These are technical results of a general nature, which are used in proving the soundness and completeness of the translation from $\mathcal{AC}$ into default logic. We defer until Appendix B the proofs of theorems, most notably the proof of the correctness of the embedding of $\mathcal{AC}$ in default logic.

2. FOUR EXAMPLES

In order to illustrate the range of applicability of the definitions introduced in this paper, we next consider four example action domains, providing for each an informal description, a formalization in the high-level action language $\mathcal{AC}$, and a sound and complete translation into default logic, logic programming, or both.

Example 1

We begin with yet another variant of the Yale Shooting domain [HM87]. There is a pilgrim and a turkey. The pilgrim has two guns. If the pilgrim fires a loaded gun, the turkey will be caused to be not alive in the resulting situation. Furthermore, one can make the turkey be not trotting by making it not alive, because any causal explanation for the turkey being not alive is also a causal explanation for the turkey not trotting. Initially the turkey is trotting and at least one of the two guns is loaded.

Based on this informal description, we can conclude, for instance, that the turkey is not trotting in the situation that would result if the pilgrim were to shoot his two guns, one after the other, in the initial situation.

This is an example of a "temporal projection" action domain, in which we are told only about the values of fluents in the initial situation. Furthermore, this is an "incomplete" temporal projection domain, since the information we are given about the initial situation does not completely describe it.

This action domain includes a "static causal law": whenever not alive is caused to be true, not trotting is also caused to be true. It follows from this static causal law that one can make the turkey be not trotting by making it be not alive. Therefore, shooting a loaded gun when the turkey is trotting has not only the "direct effect" of killing the turkey, but also the "indirect effect," or "ramification," of making it stop trotting.

$$\frac{\neg Holds(Trotting, S_0)}{False} \quad \frac{\neg(Holds(Loaded(Gun_1), S_0) \vee Holds(Loaded(Gun_2), S_0))}{False}$$

$$\frac{\neg Holds(Alive, s)}{\neg Holds(Trotting, s)} \quad \frac{Holds(Loaded(x), s)}{\neg Holds(Alive, Result(Shoot(x), s))}$$

$$\frac{: Holds(f, S_0)}{Holds(f, S_0)} \quad \frac{: \neg Holds(f, S_0)}{\neg Holds(f, S_0)}$$

$$\frac{Holds(f, s) : Holds(f, Result(a, s))}{Holds(f, Result(a, s))} \quad \frac{\neg Holds(f, s) : \neg Holds(f, Result(a, s))}{\neg Holds(f, Result(a, s))}$$

FIGURE 2.1. Default theory for Example 1.

In the action language $\mathcal{AC}$, this action domain can be formalized as follows.[2]

initially *Trotting*

initially $Loaded(Gun_1) \vee Loaded(Gun_2)$

$\neg Alive$ **suffices for** $\neg$ *Trotting*

$Shoot(x)$ **causes** $\neg Alive$ **if** $Loaded(x)$

This $\mathcal{AC}$ domain description entails, for instance, the $\mathcal{AC}$ proposition

$$\neg \mathit{Trotting} \textbf{ after } Shoot(Gun_1); Shoot(Gun_2)$$

which says that $\neg$ *Trotting* holds in the situation that would result from performing the action sequence $Shoot(Gun_1); Shoot(Gun_2)$ in the initial situation.

The domain description includes the proposition

$$\neg Alive \textbf{ suffices for } \neg \mathit{Trotting}$$

which describes the static causal law: it says that, in the action domain we are describing, whenever $\neg Alive$ is caused, $\neg$ *Trotting* is also caused. Because of this static causal law, it is impossible in this action domain for trotting to be true when alive is false. Intuitively, this can be explained as follows. In every situation, (we require that) every fact is caused. In particular then, whenever alive is false in a situation, the fact that alive is false must be caused. And since not alive is caused, it follows by the static causal law that not trotting is also caused; and consequently not trotting must be true as well. Accordingly, the semantics of $\mathcal{AC}$ guarantees that no model of the domain description includes a situation in which both *Trotting* and $\neg Alive$ hold. On the other hand, we emphasize that in the semantics of $\mathcal{AC}$ it does not follow from this proposition that *Alive* can be made true by making *Trotting* true! This failure of contraposition reflects the fact that one cannot make a turkey be alive just by making it trot. (On the contrary, common sense tells us that one simply cannot make a turkey trot unless it is alive.)

The action domain in this example is correctly formalized by the default theory shown in Figure 2.1, which can be obtained from the above domain description by a translation defined in Section 5 of this paper.

[2] Although $\mathcal{AC}$ domain descriptions do not include variables, we sometimes use meta-variables in our representations of them. For instance, the meta-variable x in the fourth expression in the domain description ranges over $\{Gun_1, Gun_2\}$.

The first rule in this default theory reflects the assertion that the turkey is initially trotting, by ensuring that there can be no consistent extension of the default theory in which the turkey is initially not trotting. In a similar fashion, the second rule says that at least one of the pilgrim's guns is initially loaded. The form of these two rules may be surprising. For instance, one may wonder why the first rule isn't simply

$$\frac{True}{Holds(Trotting, S_0)}$$

instead. This can be explained as follows.

Consider the $\mathcal{AC}$ domain description obtained by deleting the first two propositions from the above domain description.

$\neg Alive$ **suffices for** $\neg Trotting$

$Shoot(x)$ **causes** $\neg Alive$ **if** $Loaded(x)$

This reduced domain description has exactly twelve models, one for each possible initial situation. (Recall that the $\mathcal{AC}$ proposition $\neg Alive$ **suffices for** $\neg Trotting$ rules out any situation in which *Alive* is false and *Trotting* is true.) In the semantics of $\mathcal{AC}$, the role of the proposition

initially *Trotting*

is simply to eliminate those models in which trotting is initially false. Similarly, the $\mathcal{AC}$ proposition

initially $Loaded(Gun_1) \vee Loaded(Gun_2)$

simply eliminates those models in which both guns are initially unloaded. Thus the full domain description has exactly three models.

Now, the translation into default logic has the property that there is a one-to-one correspondence between $\mathcal{AC}$ models of the domain description and consistent extensions of the corresponding default theory. Thus, the default theory for the reduced domain description has twelve consistent extensions. Adding the rule

$$\frac{\neg Holds(Trotting, S_0)}{False}$$

simply eliminates those extensions that include the literal $\neg Holds(Trotting, S_0)$. Similarly, adding the rule

$$\frac{\neg(Holds(Loaded(Gun_1), S_0) \vee Holds(Loaded(Gun_2), S_0))}{False}$$

simply eliminates those extensions in which, roughly speaking, both guns are initially unloaded. Adding both of these rules eliminates nine extensions in all, and leaves us with exactly the three extensions that correspond to the three models of the original domain description. Because of the simple, monotonic behavior of such default rules, the correctness of this aspect of the translation is relatively transparent.

The third rule in the default theory says that the turkey can be made to stop trotting by making it not alive. Notice that this default rule does not allow one to derive $Holds(Alive, S_0)$ from $Holds(Trotting, S_0)$, for instance. This reflects the

1. $False \leftarrow not\, Holds(\, Trotting, S_0)$
2. $False \leftarrow not\, Holds(Loaded(Gun_1), S_0), not\, Holds(Loaded(Gun_2), S_0)$
3. $\neg Holds(\, Trotting, s) \leftarrow \neg Holds(Alive, s)$
4. $\neg Holds(Alive, Result(Shoot(x), s)) \leftarrow Holds(Loaded(x), s)$
5. $Holds(f, S_0) \leftarrow not\, \neg Holds(f, S_0)$
6. $\neg Holds(f, S_0) \leftarrow not\, Holds(f, S_0)$
7. $Holds(f, Result(a, s)) \leftarrow Holds(f, s), not\, \neg Holds(f, Result(a, s))$
8. $\neg Holds(f, Result(a, s)) \leftarrow \neg Holds(f, s), not\, Holds(f, Result(a, s))$

FIGURE 2.2. Logic program for Example 1.

fact that the static causal law is noncontrapositive. Nonetheless, in the context of the default theory as a whole, this default rule guarantees, for instance, that no consistent extension can include both $Holds(\, Trotting, S_0)$ and $\neg Holds(Alive, S_0)$.

The fourth rule in the default theory says that the turkey is not alive after the pilgrim fires a gun, if that gun is loaded.[3]Notice that this default rule does not allow one to derive the literal $\neg Holds(Loaded(Gun_1), S_0)$ from the literal $Holds(Alive, Result(Shoot(Gun_1), S_0))$, for instance. This reflects the commonsense intuition that facts in the future cannot "cause" facts in the past.[4]

The remaining rules in the default theory are standard elements of the translation we are considering. The fifth and sixth rules reflect the obvious fact that each fluent is either true or false in the initial situation, by forcing each consistent extension of the default theory to include, for each fluent F, either the literal $Holds(F, S_0)$ or the literal $\neg Holds(F, S_0)$. Furthermore, these two rules guarantee that the default theory takes into account every possible initial situation. The seventh and eighth rules express the "commonsense law of inertia" — the principle that things don't change unless they are made to. For instance, since we have the literal $Holds(\, Trotting, S_0)$, one of the inertia rules allows us to derive the literal $Holds(\, Trotting, Result(Shoot(Gun_1), S_0))$, so long as it is consistent to do so (which, roughly speaking, it will be if and only if the first gun is initially unloaded). Notice that these inertia rules again reflect the commonsense belief that facts in the future do not "cause" facts in the past.

This action domain can also be correctly formalized in logic programming, under the answer set semantics of Gelfond and Lifschitz [GL91]. Because of the well-known equivalence of logic programming under the answer set semantics and the appropriate subset of default logic [GL90], the logic program in Figure 2.2 can be understood as a direct translation of the previous default theory, except for the first and second rules, which are handled in a slightly more complex fashion (to be explained in Section 6 of this paper).

Recall that one consequence of the action domain in this example is that the turkey would not be trotting if the pilgrim were to shoot his two guns, one after

[3]Here, as earlier, x appears as a meta-variable ranging over $\{Gun_1, Gun_2\}$.

[4]This point is discussed further in Section 5.2, where we briefly compare our translation to previous published proposals for representing actions in default logic [HM87, Mor88].

$$\frac{Holds(Open, S_0)}{False} \qquad \frac{True}{\neg Holds(Fastened(x), Result(Unfasten(x), s))}$$

$$\frac{\neg Holds(Fastened(Clasp_1), s) \wedge \neg Holds(Fastened(Clasp_2), s)}{Holds(Open, s)}$$

$$\frac{: Holds(f, S_0)}{Holds(f, S_0)} \qquad \frac{: \neg Holds(f, S_0)}{\neg Holds(f, S_0)}$$

$$\frac{Holds(f, s) : Holds(f, Result(a, s))}{Holds(f, Result(a, s))} \qquad \frac{\neg Holds(f, s) : \neg Holds(f, Result(a, s))}{\neg Holds(f, Result(a, s))}$$

FIGURE 2.3. Default theory for Example 2.

the other, in the initial situation. Accordingly, the literal

$$\neg Holds(Trotting, Result(Shoot(Gun_2), Result(Shoot(Gun_1), S_0)))$$

is a consequence of both the default theory and the logic program.

Example 2

Let us consider a second action domain, adapted from [Lin95], in which there is a spring-loaded briefcase with two clasps. We have actions that unfasten the clasps, one at a time. If both clasps are unfastened, the briefcase pops open. Initially the briefcase is not open. We can conclude in this case that the briefcase would be open after we unfastened the first clasp in the initial situation if and only if the second clasp is initially not fastened.

As in the previous example, this is an incomplete temporal projection domain in which there is a static causal law. Once again, we are interested in possible indirect effects, or ramifications, of actions.

This action domain can be formalized in $\mathcal{AC}$ by the following domain description.

initially $\neg Open$

$Unfasten(x)$ **causes** $\neg Fastened(x)$

$\neg Fastened(Clasp_1) \wedge \neg Fastened(Clasp_2)$ **suffices for** $Open$

The corresponding default theory appears in Figure 2.3.

The first three rules of the default theory correspond to the three propositions in the domain description. The last four rules again encode the completeness of the initial situation and the commonsense law of inertia. (As in the previous example, this domain description can also be translated into a logic program.)

The domain description entails the $\mathcal{AC}$ proposition

$$(\ Open \textbf{ after } Unfasten(Clasp_1)\ ) \equiv (\ \textbf{initially } \neg Fastened(Clasp_2)\ )$$

and, accordingly, the formula

$$Holds(Open, Result(Unfasten(Clasp_1), S_0)) \equiv \neg Holds(Fastened(Clasp_2), S_0)$$

is a consequence of the default theory. In contrast with the previous example, we are concerned in this case with a consequence of a more complex kind, relating fluent values at two different time points.

Traditionally, background knowledge in action domains has been represented in the form of state constraints, which are, intuitively speaking, formulas of classical logic that are said to hold in every possible state of the world. Thus, for example, one might write a state constraint

$$\textbf{always } \neg Fastened(Clasp_1) \wedge \neg Fastened(Clasp_2) \supset Open$$

in place of the proposition

$$\neg Fastened(Clasp_1) \wedge \neg Fastened(Clasp_2) \textbf{ suffices for } Open$$

which represents a static causal law. In general, static causal laws are more expressive than state constraints, as shown in [MT95b]. In fact, state constraints, as they have been traditionally understood, constitute a simple special case of static causal laws. In Section 7 we will discuss these issues at some length in relation to this and other examples.

Example 3

A third action domain, loosely adapted from [KL94, GKL95], involves flipping a coin and betting on the outcome.[5] After each toss the coin either lies heads or it doesn't. (Roughly speaking, the outcome of the toss action is nondeterministic.) If you bet heads when the coin lies heads, you become a winner. If you bet heads when the coin doesn't lie heads, you cease being a winner. Now, suppose that you toss and bet heads, after which you are a winner. In this case we can conclude that the coin was heads after the toss.

This is an action domain in which there is a "nondeterministic" action. Notice also that this is not a temporal projection domain, since we are told about the value of a fluent in a non-initial situation. In this case, we are interested in reasoning from a later to an earlier time.

This action domain can be formalized in $\mathcal{AC}$ as follows.

$Winner$ **after** $Toss; BetHeads$

$Toss$ **possibly changes** $Heads$

$BetHeads$ **causes** $Winner$ **if** $Heads$

$BetHeads$ **causes** $\neg Winner$ **if** $\neg Heads$

This domain description entails the $\mathcal{AC}$ proposition

$$Heads \textbf{ after } Toss$$

and, accordingly, the literal

$$Holds(Heads, Result(Toss, S_0))$$

is entailed by the corresponding logic program, listed in Figure 2.4.

The first rule of this program corresponds to the first proposition in the domain description. The next two rules correspond to the second proposition in the domain description. The fourth and fifth rules correspond to the third and fourth propositions in the domain description. Again, the last four rules encode the completeness of the initial situation and the commonsense law of inertia.

[5] This action domain also resembles Sandewall's "Russian Turkey Shoot" domain [San94].

1. $False \leftarrow not\, Holds(\, Winner, Result(BetHeads, Result(\, Toss, S_0)))$
2. $Holds(Heads, Result(\, Toss, s)) \leftarrow not \neg Holds(Heads, Result(\, Toss, s))$
3. $\neg Holds(Heads, Result(\, Toss, s)) \leftarrow not\, Holds(Heads, Result(\, Toss, s))$
4. $Holds(\, Winner, Result(BetHeads, s)) \leftarrow Holds(Heads, s)$
5. $\neg Holds(\, Winner, Result(BetHeads, s)) \leftarrow \neg Holds(Heads, s)$
6. $Holds(f, S_0) \leftarrow not \neg Holds(f, S_0)$
7. $\neg Holds(f, S_0) \leftarrow not\, Holds(f, S_0)$
8. $Holds(f, Result(a, s)) \leftarrow Holds(f, s), not \neg Holds(f, Result(a, s))$
9. $\neg Holds(f, Result(a, s)) \leftarrow \neg Holds(f, s), not\, Holds(f, Result(a, s))$

FIGURE 2.4. Logic program for Example 3.

Example 4

Finally, consider a fourth action domain, adapted from [KL94]. The door to your hotel room is closed. It can be opened by inserting the keycard, but that is not possible when you do not have the keycard.

In $\mathcal{AC}$ we write the following.

initially $\neg DoorOpen$

$InsertCard$ **causes** $DoorOpen$

impossible $InsertCard$ **if** $\neg HasCard$

Since it is not known whether or not you initially have your keycard, this domain description does not entail the $\mathcal{AC}$ proposition

$$DoorOpen \textbf{ after } InsertCard$$

but it does entail the following weaker $\mathcal{AC}$ proposition.

$$(\, DoorOpen \textbf{ after } InsertCard\,) \equiv (\, \textbf{initially } HasCard\,)$$

Accordingly, the corresponding logic program (Figure 2.5) does not entail the literal

$$Holds(DoorOpen, Result(InsertCard, S_0))$$

but each answer set for the program includes exactly one of the following two literals:

$$Holds(DoorOpen, Result(InsertCard, S_0))\ ,\ \neg Holds(HasCard, S_0)\ .$$

This domain description, unlike those considered in the previous examples, describes an "action precondition" for one of its actions: the action *InsertCard* can be performed only in situations where *HasCard* holds. Thus, for instance, the domain description fails to entail the proposition

$$True \textbf{ after } InsertCard$$

1. $False \leftarrow not\, \neg Holds(DoorOpen, S_0)$
2. $Holds(DoorOpen, Result(InsertCard, s)) \leftarrow Reachable(Result(InsertCard, s))$
3. $\neg Reachable(Result(InsertCard, s)) \leftarrow \neg Holds(HasCard, s)$
4. $Reachable(s) \leftarrow not\, \neg Reachable(s)$
5. $\neg Reachable(Result(a, s)) \leftarrow \neg Reachable(s)$
6. $Holds(f, S_0) \leftarrow not\, \neg Holds(f, S_0)$
7. $\neg Holds(f, S_0) \leftarrow not\, Holds(f, S_0)$
8. $Holds(f, Result(a, s)) \leftarrow Holds(f, s), Reachable(Result(a, s)),$
   $not\, \neg Holds(f, Result(a, s))$
9. $\neg Holds(f, Result(a, s)) \leftarrow \neg Holds(f, s), Reachable(Result(a, s)),$
   $not\, Holds(f, Result(a, s))$

FIGURE 2.5. Logic program for Example 4.

which says, roughly speaking, that the action of inserting the keycard can be performed in the initial situation.

The action language $\mathcal{AC}$ handles action preconditions in a flexible and robust manner. By contrast we note that the sole restriction placed in this paper on the $\mathcal{AC}$ domain descriptions translated into default logic will be a requirement that action preconditions be expressed in a particular "explicit" form. The domain description above satisfies this requirement, and therefore we are able to formalize it in logic programming, as shown in Figure 2.5, using a translation defined in Section 6 of this paper.

The first three rules of this program correspond to the three propositions in the domain description. The translation in this case is complicated by the fact that in this action domain, unlike the domains considered previously, there is an action that is sometimes impossible to perform. This additional difficulty is accommodated in the translation through the use of an additional predicate *Reachable*, which says of a sequence of actions that it can be performed in the initial situation. For instance, the third rule says that if you are in a situation in which you do not have your keycard, there is no "reachable" situation that can result from inserting your keycard — since you in fact cannot insert it. Rule 2 says that if it is indeed possible to insert your keycard in the current situation, then the door will be open after you have done so. Rule 4 expresses the assumption that situations are reachable unless we say otherwise. This assumption is based on the more fundamental assumption in $\mathcal{AC}$ that actions are performable unless we say otherwise (either explicitly or implicitly). Rule 5 says that if a given situation is not reachable, then it does not have any reachable successors. Notice that in this translation the assumption of inertia (in the last two rules) is also predicated on reachability.

3. PRELIMINARY DEFINITIONS

Given a set U of propositional atoms (not including the special constants *True* and *False*), we denote by $\mathcal{L}(U)$ the language of propositional logic with exactly the

atoms $U \cup \{ True, False \}$.[6] For any literal L, let $\overline{L}$ denote the literal complementary to L. For any set X of literals, let $\overline{X} = \{\overline{L} : L \in X\}$. By $Lit(U)$ we denote the set $U \cup \overline{U}$ of literals.[7] In this description we say nothing about the form of the atoms in U, but of course an important special case is when U is the set of all ground atoms of a many-sorted first-order language.

We represent an interpretation of $\mathcal{L}(U)$ as a maximal consistent set of literals from $Lit(U)$. We say that a set of formulas from $\mathcal{L}(U)$ is *logically closed* if it is closed under propositional logic (with respect to $\mathcal{L}(U)$). Inference rules over $\mathcal{L}(U)$ will be written as expressions of the form

$$\frac{\phi}{\psi}$$

where ϕ and ψ are formulas from $\mathcal{L}(U)$.

Let R be a set of inference rules, and let Γ be a set of formulas. We say that Γ is *closed under* R if for every rule $\frac{\phi}{\psi}$ in R, if ϕ belongs to Γ then ψ does too. By $Cn_U(R)$ we denote the least logically closed set of formulas from $\mathcal{L}(U)$ that is closed under R. We will often find it convenient to identify a formula ϕ with the inference rule

$$\frac{True}{\phi} .$$

Under this convention, $Cn_U(\Gamma)$ denotes the least logically closed set of formulas from $\mathcal{L}(U)$ that contains Γ. Similarly, $Cn_U(\Gamma \cup R)$ is the least logically closed set of formulas from $\mathcal{L}(U)$ that contains Γ and is also closed under R. We sometimes omit the subscript to Cn when there is no danger of confusion.

A *default rule* over $\mathcal{L}(U)$ is an expression of the form

$$\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \tag{3.1}$$

where all of $\alpha, \beta_1, \ldots, \beta_n, \gamma$ are formulas from $\mathcal{L}(U)$ ($n \geq 0$). Let r be a default rule of the form (3.1). We call α the *prerequisite* of r, and denote it by $pre(r)$. We call the formulas $\beta_1, \ldots, \beta_n$ the *justifications* of r, and write $just(r)$ to denote the set $\{ \beta_1, \ldots, \beta_n \}$. We call γ the *consequent* of r, and denote it by $cons(r)$. If $just(r)$ is empty, we say r is *justification-free*. If r is justification-free, we often identify r with the corresponding inference rule $\frac{\alpha}{\gamma}$. If $pre(r) = True$ we often omit it and write $\frac{: \beta_1, \ldots, \beta_n}{\gamma}$ instead. If $just(r) = \{cons(r)\}$, we say that r is *normal*.

A *default theory* over $\mathcal{L}(U)$ is a set of default rules over $\mathcal{L}(U)$. Let D be a default theory over $\mathcal{L}(U)$ and let E be a set of formulas from $\mathcal{L}(U)$. We define the *reduct of D by E*, denoted by D^E, as follows.

$$D^E = \left\{ \frac{pre(r)}{cons(r)} \ : \ r \in D \ and \ for \ all \ \ \beta \in just(r) \ , \ \neg\beta \notin E \right\}$$

We say that E is an *extension* of D if

$$E = Cn_U(D^E) .$$

[6] Thus, $\mathcal{L}(\emptyset)$ consists of all formulas in which only the atoms *True* and *False* occur.

[7] Notice that $Lit(U)$ is a proper subset of the set of literals from $\mathcal{L}(U)$, since $Lit(U)$ does not include the literals in which the special atoms *True* and *False* occur.

We say D is *consistent* if it has at least one consistent extension. We say that a formula is a *consequence* of D if it belongs to every extension of D. Default logic is due to Reiter [Rei80]. The definition of an extension given above follows [GLPT91], and is equivalent to Reiter's definition.

For the purposes of this paper, a *logic program rule* over $\mathcal{L}(U)$ is an expression of the form

$$L_0 \leftarrow L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n \tag{3.2}$$

with $0 \leq m \leq n$, where $L_0 \in Lit(U) \cup \{False\}$ and for all i $(1 \leq i \leq n)$, $L_i \in Lit(U)$. A *logic program* over $\mathcal{L}(U)$ is a set of logic program rules over $\mathcal{L}(U)$.

Under the answer set semantics of Gelfond and Lifschitz [GL90], logic programming corresponds to a subset of default logic. Because it is convenient for the purposes of this paper, we will define the notions of "answer sets" and "entailment" for logic programs indirectly, in terms of the related notions for default logic.

For each logic program P there is a corresponding default theory $dt(P)$, defined as follows. For each rule $r \in P$ of form (3.2), $dt(P)$ includes the rule

$$\frac{L_1 \wedge \cdots \wedge L_m : \overline{L_{m+1}}, \ldots, \overline{L_n}}{L_0}.$$

A subset X of $Lit(U)$ is an *answer set* for P if there is an extension E of $dt(P)$ such that $X = E \cap Lit(U)$. It follows that X is a consistent answer set for P if and only if $Cn(X)$ is a consistent extension of $dt(P)$. For any $L \in Lit(U)$, we say that P *entails* L if L belongs to all answer sets for P. It follows that P entails L if and only if $dt(P)$ does.

Although the definitions in this section are stated for the propositional case, they are taken, in the standard way, to apply naturally in the (quantifier-free) non-propositional case as well, by taking each non-ground expression to stand for all of its ground instances.

4. THE ACTION LANGUAGE $\mathcal{AC}$

In the high-level action language $\mathcal{AC}$, a description of an action domain is a set of propositions of the following five kinds:

1. value propositions, which restrict the values of fluents in situations that would result from the performance of sequences of actions;

2. sufficiency propositions, which say that whenever the truth of one fluent formula is caused, the truth of a second fluent formula is also caused;

3. effect propositions, which say that under certain conditions a fluent formula would be caused to hold as a result of the performance of an action;

4. influence propositions, which say that under certain conditions the performance of an action would "nondeterministically" change the value of a fluent;

5. executability propositions, which say that under certain conditions an action would be impossible to perform.

After specifying the syntax and semantics of $\mathcal{AC}$, we will illustrate the definitions with an example.

4.1. Syntax of $\mathcal{AC}$

We begin with two disjoint nonempty sets of symbols, a set $\mathbf{F}$ of *fluent names* and a set $\mathbf{A}$ of *action names*. We designate a subset $\mathbf{F}_f$ of $\mathbf{F}$ as the *frame fluents* and we call the members of $\mathbf{F} \setminus \mathbf{F}_f$ the *nonframe fluents*. A *fluent formula* is a formula from $\mathcal{L}(\mathbf{F})$. A *frame fluent formula* is a formula from $\mathcal{L}(\mathbf{F}_f)$.

An *atomic value proposition* is an expression of the form

$$\phi \textbf{ after } \overline{A}$$

where ϕ is a fluent formula, and $\overline{A}$ is a string of action names. If $\overline{A}$ is the empty string, we may write instead

$$\textbf{initially } \phi\,.$$

A *value proposition* is a propositional combination of atomic value propositions.

A *sufficiency proposition* is an expression of the form

$$\phi \textbf{ suffices for } \psi$$

where ϕ and ψ are fluent formulas. Sufficiency propositions represent static causal laws. Thus, such a proposition says that, in the action domain being described, whenever ϕ is caused ψ is also. We write

$$\textbf{always } \phi$$

as an abbreviation for the proposition *True* **suffices for** ϕ and we write

$$\textbf{never } \phi$$

as an abbreviation for the proposition ϕ **suffices for** *False*. Given a nonframe fluent F, an expression of the form

$$\textbf{always } F \equiv \phi$$

where ϕ is a frame fluent formula, is called an *explicit definition of* F. We require that $\mathcal{AC}$ domain descriptions include an explicit definition of every nonframe fluent.

An *effect proposition* is an expression of the form

$$A \textbf{ causes } \phi \textbf{ if } \psi$$

where A is an action name, and ϕ and ψ are fluent formulas. If ψ is the formula *True*, we may drop the **if** and simply write A **causes** ϕ.

An *influence proposition* is an expression of the form

$$A \textbf{ possibly changes } F \textbf{ if } \psi$$

where A is an action name, F is a fluent name, and ψ is a fluent formula. If ψ is the formula *True*, we may drop the **if** and simply write A **possibly changes** F.

An *executability proposition* is an expression of the form

$$\textbf{impossible } A \textbf{ if } \psi$$

where A is an action name and ψ is a fluent formula. In the semantics of $\mathcal{AC}$ such a proposition has the same meaning as the effect proposition A **causes** *False* **if** ψ, but the syntactic distinction becomes convenient in Sections 5 and 6 when we specify translations of $\mathcal{AC}$ domain descriptions into default logic and logic programming.

An $\mathcal{AC}$ *domain description* is a set of $\mathcal{AC}$ propositions that includes an explicit definition for each nonframe fluent.

4.2. Semantics of $\mathcal{AC}$

Let D be an $\mathcal{AC}$ domain description, with fluents $\mathbf{F}$ and frame fluents $\mathbf{F}_f$. A *structure for D* is a partial function from action strings to interpretations of $\mathcal{L}(\mathbf{F})$, whose domain is nonempty and prefix-closed.[8] By $Dom(\Psi)$ we denote the domain of a structure Ψ. Notice that for every structure Ψ, $Dom(\Psi)$ includes the empty string (denoted by ϵ).

Let R be the set of inference rules $\frac{\phi}{\psi}$ such that the sufficiency proposition

$$\phi \textbf{ suffices for } \psi$$

is in D. An interpretation S of $\mathcal{L}(\mathbf{F})$ is called a *state* if $Cn(S)$ is closed under R.

Let A be action name and S a state. We say that A is *prohibited in S* if there is an executability proposition

$$\textbf{impossible } A \textbf{ if } \psi$$

in D such that S satisfies ψ. Let $E(A,S)$ be the set of all fluent formulas ϕ for which there is an effect proposition

$$A \textbf{ causes } \phi \textbf{ if } \psi$$

in D such that S satisfies ψ. Similarly, let $F(A,S)$ be the set of all fluent names F for which there is an influence proposition

$$A \textbf{ possibly changes } F \textbf{ if } \psi$$

in D such that S satisfies ψ. A set E of fluent formulas is called an *explicit effect of A in S* if:

1. A is not prohibited in S, and
2. there is an interpretation I of $\mathcal{L}(F(A,S))$ such that $E = I \cup E(A,S)$.

We say that a state S' *may result from doing A in S* if there is an explicit effect E of A in S such that

$$Cn(S') = Cn[\,(\,S \cap S' \cap Lit(\mathbf{F}_f)\,) \cup E \cup R\,]\,.$$

This fixpoint definition guarantees that S' may result from doing A in S if and only if the value of every frame fluent in S' is caused — either it held the same value in S and was not made to change, or its value was changed (directly or indirectly) by the action. Thus, this definition essentially captures the commmonsense law of inertia, which is the assumption that things don't change unless they're made to.[9] Let $Res(A,S)$ denote the set of states that may result from doing A in S.

Given a structure Ψ, we say an atomic value proposition ϕ **after** $\overline{A}$ is *true in* Ψ if $\overline{A} \in Dom(\Psi)$ and $\Psi(\overline{A})$ satisfies ϕ. The general truth definition for value propositions is then given by the standard recursion over the propositional connectives.

[8] A set Σ of strings is *prefix-closed* if, for every string $\sigma \in \Sigma$, every prefix of σ is also in Σ.

[9] See [MT95b] and [PT95] for further discussion related to this fixpoint definition.

A structure Ψ for D is a *model of* D if it satisfies the following four conditions.

1. $\Psi(\epsilon)$ is a state.
2. For all $\overline{A} \in Dom(\Psi)$ and all action names A, if $Res(A, \Psi(\overline{A}))$ is nonempty then $\overline{A}; A \in Dom(\Psi)$.
3. For all $\overline{A}; A \in Dom(\Psi)$, $\Psi(\overline{A}; A) \in Res(A, \Psi(\overline{A}))$.
4. Every value proposition in D is true in Ψ.

A value proposition is *entailed* by D if it is true in every model of D.

4.3. An Example $\mathcal{AC}$ Domain Description

As an example illustrating the use of the preceding definitions, consider the following $\mathcal{AC}$ domain description D_1, in which *Dead* is the only nonframe fluent.

always $Dead \equiv \neg Alive$

initially $Walking$

$\neg Walking$ **after** $Shoot$

$\neg Alive$ **suffices for** $\neg Walking$

$Shoot$ **causes** $Dead \wedge \neg Loaded$

impossible $Shoot$ **if** $\neg Loaded$

Notice that we are describing here a different shoot action than in Example 1 (Section 2), where shooting was always possible. There, the direct effect $\neg Alive$ of the shoot action had a "fluent precondition" *Loaded*. Here, *Loaded* becomes instead an action precondition of *Shoot*.

Domain description D_1 has a unique model Ψ_1, as follows.

$$\begin{aligned} Dom(\Psi_1) &= \{\epsilon, Shoot\} \\ \Psi_1(\epsilon) &= \{Loaded, Alive, \neg Dead, Walking\} \\ \Psi_1(Shoot) &= \{\neg Loaded, \neg Alive, Dead, \neg Walking\} \end{aligned}$$

It is easy to check, for instance, that the following value proposition is true in Ψ_1.

$$(\textbf{initially } Loaded) \wedge (Dead \wedge \neg Loaded \textbf{ after } Shoot)$$

To exercise the definitions, we will verify that Ψ_1 is the unique model of D_1. It is clear that Ψ_1 is a structure for D_1, so we begin by showing that Ψ_1 is a model.

First, we must check that $\Psi_1(\epsilon)$ is a state. We see that domain description D_1 includes the sufficiency propositions

$$\textbf{always } Dead \equiv \neg Alive$$

and

$$\neg Alive \textbf{ suffices for } \neg Walking$$

from which we obtain the associated set of inference rules

$$R = \left\{ \frac{True}{Dead \equiv \neg Alive}, \frac{\neg Alive}{\neg Walking} \right\} .$$

It follows that there are exactly six states in this action domain; they are the six interpretations of $\mathcal{L}(\mathbf{F})$ that satisfy the fluent formulas $Dead \equiv \neg Alive$ and $\neg Alive \supset \neg Walking$. We see that $\Psi_1(\epsilon)$ is indeed one of these six states.

Second, we must check that $Res(Shoot, \Psi_1(Shoot))$ is empty. Since D_1 includes the executability proposition

$$\textbf{impossible } Shoot \textbf{ if } \neg Loaded$$

we see that $Shoot$ is prohibited in $\Psi_1(Shoot)$. Therefore there can be no explicit effect E of $Shoot$ in $\Psi_1(Shoot)$, which shows that $Res(Shoot, \Psi_1(Shoot)) = \emptyset$.

Third, we must verify that $\Psi_1(Shoot)$ belongs to $Res(Shoot, \Psi_1(\epsilon))$. That is, we must show that $\Psi_1(Shoot)$ may result from doing $Shoot$ in $\Psi_1(\epsilon)$. This requires that we check that

$$Cn(\Psi_1(Shoot)) = Cn((\Psi_1(\epsilon) \cap \Psi_1(Shoot) \cap \mathcal{L}(\mathbf{F}_f)) \cup E \cup R)$$

where E is an explicit effect of $Shoot$ in $\Psi_1(\epsilon)$. We first observe that $Shoot$ is not prohibited in $\Psi_1(\epsilon)$. Since D_1 includes no influence propositions, we have $F(Shoot, \Psi_1(\epsilon)) = \emptyset$. Thus the only interpretation of $\mathcal{L}(F(Shoot, \Psi_1(\epsilon)))$ is also $\emptyset$. Since D_1 includes the effect proposition

$$Shoot \textbf{ causes } Dead \wedge \neg Loaded$$

we have

$$E(Shoot, \Psi_1(\epsilon)) = \{Dead \wedge \neg Loaded\}\ .$$

Given these observations, we can conclude that the unique explicit effect E of $Shoot$ in $\Psi_1(\epsilon)$ is $\{Dead \wedge \neg Loaded\}$. It remains to observe that $\Psi_1(\epsilon) \cap \Psi_1(Shoot)$ is empty, so $\Psi_1(\epsilon) \cap \Psi_1(Shoot) \cap \mathcal{L}(\mathbf{F}_f)$ is also. Thus what we are to verify is that

$$Cn(\Psi_1(Shoot)) = Cn(\{Dead \wedge \neg Loaded\} \cup R)$$

which is clearly true. In fact, what we have shown is that

$$Res(Shoot, \Psi_1(\epsilon)) = \{\Psi_1(Shoot)\}$$

since $\Psi_1(Shoot)$ is the only state that satisfies $Dead \wedge \neg Loaded$.

Fourth, we must check that Ψ_1 satisfies the two value propositions in D_1, which it clearly does.

So we've shown that Ψ_1 is indeed a model of domain description D_1. Now let us verify that it is the only model.

Assume that Ψ is a model of D_1. By model condition 1 we know that $\Psi(\epsilon)$ is a state, and by model condition 4, we know that the value proposition

$$\textbf{initially } Walking$$

is true in Ψ. That is, $\Psi(\epsilon)$ must satisfy the fluent formula $Walking$. It follows that $\Psi(\epsilon)$ also satisfies $Alive$ and $\neg Dead$. Thus at this point we know everything about $\Psi(\epsilon)$ except whether or not it satisfies $Loaded$, so there are two states to consider.

Consider the state $S = \{\neg Loaded, Alive, \neg Dead, Walking\}$. We will show that $\Psi(\epsilon)$ cannot be S, which will be sufficient to show that $\Psi(\epsilon) = \Psi_1(\epsilon)$. Since D_1 includes the executability proposition **impossible** $Shoot$ **if** $\neg Loaded$ we know that

Shoot is prohibited in S. It follows that there can be no explicit effect E of *Shoot* in S, which allows us to conclude that $Res(Shoot, S)$ is empty. Now, by model condition 4 we know that D_1 must satisfy the value proposition

$$\neg \mathit{Walking} \textbf{ after } \mathit{Shoot}$$

so we can conclude that $Shoot \in Dom(\Psi)$. It follows by model condition 3 that $\Psi(Shoot) \in Res(Shoot, \Psi(\epsilon))$. Since $Res(Shoot, S) = \emptyset$, we have $\Psi(\epsilon) \neq S$. So $\Psi(\epsilon) = \Psi_1(\epsilon)$. And since we've already seen that $Res(Shoot, \Psi_1(\epsilon)) = \{\Psi_1(Shoot)\}$, we can conclude by model conditions 2 and 3 that $\Psi(Shoot) = \Psi_1(Shoot)$, which is sufficient to establish the fact that $\Psi = \Psi_1$. So Ψ_1 is the unique model of D_1.

5. REPRESENTING ACTIONS IN DEFAULT LOGIC

We begin by defining a class of $\mathcal{AC}$ domain descriptions called "qualification-free." Roughly speaking, in qualification-free domain descriptions, all action preconditions are stated "explicitly," in the form of executability propositions. We will specify a sound and complete translation of qualification-free $\mathcal{AC}$ domain descriptions into default theories. We will also specify a second, simpler translation for those $\mathcal{AC}$ domain descriptions in which there are no action preconditions whatsoever. These embeddings build on previous work in [Tur94, LT95, PT95].

We then compare the formalization of the Yale Shooting domain [HM87] obtained by our translation with the default theories discussed by Hanks and McDermott [HM87] and by Morris [Mor88].

5.1. Embedding $\mathcal{AC}$ in Default Logic

We say that an $\mathcal{AC}$ domain description is *qualification-free* if for all action names A and states S, A is prohibited in S whenever $Res(A, S)$ is empty.

Our default theories for reasoning about action use the situation calculus. For any fluent formula ϕ, we write $Holds(\phi, S)$ to stand for the formula obtained by replacing each fluent atom F in ϕ by $Holds(F, S)$. (Notice that the special atoms *True* and *False* are not replaced.) Given an action string $A_1; \cdots; A_m$, we write

$$[A_1; \cdots; A_m]$$

to stand for the term

$$Result(A_m, Result(A_{m-1}, \ldots Result(A_1, S_0) \cdots)) \, .$$

Given an atomic value proposition ϕ **after** $\overline{A}$, we write

$$[\phi \textbf{ after } \overline{A}]$$

to stand for the formula

$$(\, Holds(\phi, [\overline{A}]) \wedge Reachable([\overline{A}]) \,) \, .$$

Given a (non-atomic) value proposition V, we write $[V]$ to stand for the formula obtained by simultaneously replacing each atomic value proposition V' that occurs in V by the formula $[V']$.

Reachability axioms. Default theory $\delta(D)$ includes the rules

$$\frac{:\mathit{Reachable}(s)}{\mathit{Reachable}(s)} \quad \text{and} \quad \frac{\neg\mathit{Reachable}(s)}{\neg\mathit{Reachable}(\mathit{Result}(a,s))} .$$

Initial situation axioms. For each fluent literal L, $\delta(D)$ includes the rule

$$\frac{:\mathit{Holds}(L,S_0)}{\mathit{Holds}(L,S_0)} .$$

Inertia axioms. For each frame fluent literal L, $\delta(D)$ includes the rule

$$\frac{\mathit{Holds}(L,s) \wedge \mathit{Reachable}(\mathit{Result}(a,s)) : \mathit{Holds}(L,\mathit{Result}(a,s))}{\mathit{Holds}(L,\mathit{Result}(a,s))} .$$

FIGURE 5.1. Standard elements of the translation δ.

The translation δ takes an $\mathcal{AC}$ domain description D to a default theory $\delta(D)$ over the language $\mathcal{L}(U)$, where U is the least set of atoms such that, for every action string $\overline{A}$: (i) $\mathit{Reachable}([\overline{A}]) \in U$; (ii) for every fluent name F, $\mathit{Holds}(F,[\overline{A}]) \in U$.

For each value proposition V in D, $\delta(D)$ includes the rule

$$\frac{\neg[V]}{\mathit{False}} .$$

For each sufficiency proposition ϕ **suffices for** ψ in D, $\delta(D)$ includes the rule

$$\frac{\mathit{Holds}(\phi,s) \wedge \mathit{Reachable}(s)}{\mathit{Holds}(\psi,s)} .$$

For each effect proposition A **causes** ϕ **if** ψ in D, $\delta(D)$ includes the rule

$$\frac{\mathit{Holds}(\psi,s) \wedge \mathit{Reachable}(\mathit{Result}(A,s))}{\mathit{Holds}(\phi,\mathit{Result}(A,s))} .$$

For each influence proposition A **possibly changes** F **if** ψ in D, $\delta(D)$ includes the pair of rules

$$\frac{\mathit{Holds}(\psi,s) \wedge \mathit{Reachable}(\mathit{Result}(A,s)) : \mathit{Holds}(F,\mathit{Result}(A,s))}{\mathit{Holds}(F,\mathit{Result}(A,s))}$$

and

$$\frac{\mathit{Holds}(\psi,s) \wedge \mathit{Reachable}(\mathit{Result}(A,s)) : \neg\mathit{Holds}(F,\mathit{Result}(A,s))}{\neg\mathit{Holds}(F,\mathit{Result}(A,s))} .$$

For each executability proposition **impossible** A **if** ψ in D, $\delta(D)$ includes the rule

$$\frac{\mathit{Holds}(\psi,s)}{\neg\mathit{Reachable}(\mathit{Result}(A,s))} .$$

Default theory $\delta(D)$ also includes the additional, standard rules shown in Figure 5.1.

The following theorem shows that the translation δ is indeed sound and complete for qualification-free $\mathcal{AC}$ domain descriptions.

Theorem 5.1. **(Embedding Theorem)** *Let D be a qualification-free $\mathcal{AC}$ domain description. A value proposition V is entailed by D if and only if the formula $[V]$ is entailed by the default theory $\delta(D)$.*

The Embedding Theorem is an immediate consequence of the following stronger theorem, which is proved in Appendix B.

Theorem 5.2. **(Correspondence Theorem)** *Let D be a qualification-free $\mathcal{AC}$ domain description. There is a one-to-one correspondence between models of D and consistent extensions of $\delta(D)$ such that, for every model Ψ of D and its corresponding extension E, a value proposition V is true in Ψ iff $[V] \in E$.*

For example, recall domain description D_1, in which *Dead* is a nonframe fluent.

always $Dead \equiv \neg Alive$

initially $Walking$

$\neg\, Walking$ **after** $Shoot$

$\neg\, Alive$ **suffices for** $\neg\, Walking$

$Shoot$ **causes** $Dead \wedge \neg Loaded$

impossible $Shoot$ **if** $\neg Loaded$

Domain description D_1 entails, for instance, the value proposition

initially $Loaded$

and the Embedding Theorem guarantees that the corresponding formula

$$Holds(Loaded, S_0) \wedge Reachable(S_0)$$

is entailed by the corresponding default theory $\delta(D_1)$, listed in Figure 5.2.

If a domain description includes no executability propositions, we can eliminate the *Reachable* atoms in the corresponding default theory, thus obtaining a simpler translation, as follows. Let D be an $\mathcal{AC}$ domain description. By $\delta'(D)$ we denote the default theory obtained from $\delta(D)$ by first eliminating the reachability axioms and then replacing each *Reachable* atom in the remaining rules by the special atom *True*. Of course it is then straightforward to eliminate the resulting occurrences of *True* in the resulting default theory. Notice that the default theories in Examples 1 and 2 in Section 2 can be obtained by translation δ'.

For each atomic value proposition ϕ **after** $\overline{A}$, let

$$[\![\phi \textbf{ after } \overline{A}]\!]$$

denote the formula

$$Holds(\phi, [\overline{A}])$$

and for each (non-atomic) value proposition V, let $[\![V]\!]$ be the formula obtained from V by simultaneously replacing each atomic value proposition V' that occurs in V by the formula $[\![V']\!]$.

Corollary 5.1. **(Reachability Corollary)** *Let D be a qualification-free $\mathcal{AC}$ domain description with no executability propositions. There is a one-to-one correspondence between models of D and consistent extensions of $\delta'(D)$ such that, for every model Ψ of D and its corresponding extension E, a value proposition V is true in Ψ if and only if $[\![V]\!] \in E$.*

$$\frac{True \wedge Reachable(s)}{Holds(Dead,s) \equiv \neg Holds(Alive,s)} \quad \frac{\neg(Holds(Walking,S_0) \wedge Reachable(S_0))}{False}$$

$$\frac{\neg(\neg Holds(Walking,Result(Shoot,S_0)) \wedge Reachable(Result(Shoot,S_0)))}{False}$$

$$\frac{\neg Holds(Alive,s) \wedge Reachable(s)}{\neg Holds(Walking,s)} \quad \frac{\neg Holds(Loaded,s)}{\neg Reachable(Result(Shoot,s))}$$

$$\frac{True \wedge Reachable(Result(Shoot,s))}{Holds(Dead,Result(Shoot,s)) \wedge \neg Holds(Loaded,Result(Shoot,s))}$$

$$\frac{:Holds(f,S_0)}{Holds(f,S_0)} \quad \frac{:\neg Holds(f,S_0)}{\neg Holds(f,S_0)} \quad \frac{:Reachable(s)}{Reachable(s)} \quad \frac{\neg Reachable(s)}{\neg Reachable(Result(a,s))}$$

$$\frac{Holds(Alive,s) \wedge Reachable(Result(a,s)) : Holds(Alive,Result(a,s))}{Holds(Alive,Result(a,s))}$$

$$\frac{\neg Holds(Alive,s) \wedge Reachable(Result(a,s)) : \neg Holds(Alive,Result(a,s))}{\neg Holds(Alive,Result(a,s))}$$

$$\frac{Holds(Loaded,s) \wedge Reachable(Result(a,s)) : Holds(Loaded,Result(a,s))}{Holds(Loaded,Result(a,s))}$$

$$\frac{\neg Holds(Loaded,s) \wedge Reachable(Result(a,s)) : \neg Holds(Loaded,Result(a,s))}{\neg Holds(Loaded,Result(a,s))}$$

$$\frac{Holds(Walking,s) \wedge Reachable(Result(a,s)) : Holds(Walking,Result(a,s))}{Holds(Walking,Result(a,s))}$$

$$\frac{\neg Holds(Walking,s) \wedge Reachable(Result(a,s)) : \neg Holds(Walking,Result(a,s))}{\neg Holds(Walking,Result(a,s))}$$

FIGURE 5.2. The translation $\delta(D_1)$ of $\mathcal{AC}$ domain description D_1.

5.2. Comparison with Previous Approaches

At this point it may be interesting to briefly consider some of the ways in which our default theory for the Yale Shooting domain [HM87] differs from the one proposed and found inadequate by Hanks and McDermott [HM87], and from the more adequate solution later proposed by Morris [Mor88].

We represent the Yale Shooting domain by the following $\mathcal{AC}$ domain description.

initially *Alive*

Load **causes** *Loaded*

Shoot **causes** $\neg$*Alive* **if** *Loaded*

Of course this domain description entails the $\mathcal{AC}$ value proposition

$$\neg Alive \textbf{ after } Load; Wait; Shoot$$

and accordingly, we know by the Reachability Corollary that the corresponding literal

$$\neg Holds(Alive, Result(Shoot, Result(Wait, Result(Load, S_0))))$$

is a consequence of the corresponding default theory Y_1, which appears in Figure 5.3.

By comparison, the default theory that was introduced and rejected by Hanks and McDermott [HM87] is (essentially) the default theory Y_2 that appears in Figure 5.4. In default theory Y_2, the well-known technical difficulty results from the

$$\frac{\neg Holds(Alive, S_0)}{False}$$

$$\frac{True}{Holds(Loaded, Result(Load, s))} \qquad \frac{Holds(Loaded, s)}{\neg Holds(Alive, Result(Shoot, s))}$$

$$\frac{: Holds(f, S_0)}{Holds(f, S_0)} \qquad \frac{: \neg Holds(f, S_0)}{\neg Holds(f, S_0)}$$

$$\frac{Holds(f, s) : Holds(f, Result(a, s))}{Holds(f, Result(a, s))} \qquad \frac{\neg Holds(f, s) : \neg Holds(f, Result(a, s))}{\neg Holds(f, Result(a, s))}$$

FIGURE 5.3. Default theory Y_1

$$\frac{True}{Holds(Alive, S_0)} \qquad \frac{True}{Ab(Loaded, Load, s) \wedge Holds(Loaded, Result(Load, s))}$$

$$\frac{True}{Holds(Loaded, s) \supset (Ab(Alive, Shoot, s) \wedge \neg Holds(Alive, Result(Shoot, s)))}$$

$$\frac{True}{(Holds(f, s) \wedge \neg Ab(f, a, s)) \supset Holds(f, Result(a, s))}$$

$$\frac{True}{(\neg Holds(f, s) \wedge \neg Ab(f, a, s)) \supset \neg Holds(f, Result(a, s))}$$

$$\frac{: \neg Ab(f, a, s)}{\neg Ab(f, a, s)}$$

FIGURE 5.4. Default theory Y_2

fact that we can reason "backwards in time," using the rule describing the effect of *Shoot* and the inertia rules. In particular, by such "backwards" reasoning, we can derive

$$Ab(Loaded, Wait, Result(Load, S_0))$$

and

$$\neg Holds(Loaded, Result(Wait, Result(Load, S_0)))$$

from the default "supposition"

$$\neg Ab(Alive, Shoot, Result(Wait, Result(Load, S_0)))$$

thus obtaining the famous anomaly: according to default theory Y_2 the gun may become mysteriously unloaded during the wait action.

There is another peculiarity to be noted here, related to the fact that in the Yale Shooting domain (as originally described by Hanks and McDermott) we are not told whether or not the gun is initially loaded. Accordingly, the $\mathcal{AC}$ domain description entails neither

initially *Loaded*

nor

initially $\neg$*Loaded* .

From the Reachability Corollary it follows, for instance, that our default theory Y_1 does not entail the literal $\neg Holds(Loaded, S_0)$.

$$\frac{True}{Holds(Alive, S_0)} \qquad \frac{True}{Ab(Loaded, Load, s) \wedge Holds(Loaded, Result(Load, s))}$$

$$\frac{True}{Holds(Loaded, s) \supset (Ab(Alive, Shoot, s) \wedge \neg Holds(Alive, Result(Shoot, s)))}$$

$$\frac{Holds(f, s) : \neg Ab(f, a, s)}{Holds(f, Result(a, s))} \qquad \frac{\neg Holds(f, s) : \neg Ab(f, a, s)}{\neg Holds(f, Result(a, s))}$$

FIGURE 5.5. Default theory Y_3

By comparison, in the default theory Y_2 of Hanks and McDermott, we can "suppose"

$$\neg Ab(Alive, Shoot, S_0)$$

and from this default supposition we can derive

$$\neg Holds(Loaded, S_0)$$

by reasoning "backwards in time," using the rule describing the effect of the shoot action. This "backwards" reasoning seems a little suspicious, but so far we have only made an informal argument suggesting that $\neg Holds(Loaded, S_0)$ belongs to some extension of the default theory, which is of course not unexpected, since the $\mathcal{AC}$ domain description itself has a model in which *Loaded* is initially false. But as it turns out, $\neg Holds(Loaded, S_0)$ belongs to every extension of the default theory of Hanks and McDermott, which is therefore not only incomplete for the Yale Shooting domain, but also unsound.

Next consider the default theory for the Yale Shooting domain proposed by Morris [Mor88], which is (essentially) the default theory Y_3 shown in Figure 5.5. Even in Morris' default theory it appears that we can reason "backwards in time," using the rule describing the effect of the action *Shoot*. But notice that there is now no default rule allowing us to "suppose" literals of the form $\neg Ab(f, a, s)$. Moreover, there is no opportunity for "inappropriately" deriving atoms of the form $Ab(f, a, s)$ by reasoning backwards in time. Thus the famous anomaly is eliminated. On the other hand, it turns out that the formula $\neg Holds(Loaded, S_0)$ is once again inappropriately entailed. To see this, notice first that we cannot derive the literal

$$Ab(Alive, Shoot, S_0)$$

in default theory Y_3. Because of this, we are able to derive

$$Holds(Alive, Result(Shoot, S_0))$$

using one of the default rules expressing the commonsense law of inertia. And, from this, we can derive

$$\neg Holds(Loaded, S_0)$$

by reasoning backwards in time, using the rule describing the effect of the shoot action. Thus, Morris' default theory for the Yale Shooting domain is, apparently, complete but unsound.

It may be helpful to point out that the observed unsoundness of default theories Y_2 and Y_3 can be overcome simply by adding to them the following rules enforcing completeness of the initial situation.

$$\frac{:Holds(f,S_0)}{Holds(f,S_0)} \qquad \frac{:\neg Holds(f,S_0)}{\neg Holds(f,S_0)}$$

Recall that these rules are standard elements of our translations from $\mathcal{AC}$ into default logic. In a manner of speaking, they force a default theory to take into account every possible initial situation.

In our translations from $\mathcal{AC}$ into default logic, every default rule is either normal or justification-free. In this way our approach is simpler than that of Morris, which uses non-normal defaults, as well as an auxiliary predicate Ab. Moreover, our method has the considerable advantage of guaranteeing correct representations of commonsense knowledge for a wide variety of action domains.

6. LOGIC PROGRAMS FOR REPRESENTING ACTIONS

We begin by identifying a syntactically restricted class of $\mathcal{AC}$ domain descriptions for which the translation into logic programming is particularly convenient. We call such domain descriptions "simple." After specifying a sound and complete translation of simple, qualification-free domain descriptions into logic programming, we go on to consider a somewhat broader class of $\mathcal{AC}$ domain descriptions, which we call "vivid." We show how every vivid, qualification-free domain description can be transformed into an equivalent simple, qualification-free domain description. Thus we obtain a correct embedding in logic programming for every vivid, qualification-free $\mathcal{AC}$ domain description. Finally we briefly compare our work with previous similar work.

In the case of value propositions, the translation into logic program rules is rather more complicated than the translation into default rules specified in the previous section. For all other $\mathcal{AC}$ propositions, the translation is essentially the same.

6.1. Simple Domain Descriptions

We say that an atomic value proposition is *simple* if it has the form

$$L \textbf{ after } \overline{A}$$

where L is a fluent literal, and we say that a (non-atomic) value proposition is *simple* if it has the form

$$V_1 \vee \cdots \vee V_m \vee \neg V_{m+1} \vee \cdots \vee \neg V_n \qquad (0 \leq m \leq n\,, n > 0)$$

where each V_i $(1 \leq i \leq n)$ is a simple atomic value proposition.

We say that a sufficiency proposition is *simple* if it has either the form

$$L_1 \wedge \cdots \wedge L_n \textbf{ suffices for } L_0 \qquad (n > 0)$$

where each L_i $(0 \leq i \leq n)$ is a fluent literal, or the form

$$\textbf{always } L$$

where L is a fluent literal, or the form

$$\textbf{never } L_1 \wedge \cdots \wedge L_n \qquad (n > 0)$$

where each L_i $(1 \leq i \leq n)$ is again a fluent literal.

We say that an effect proposition is *simple* if it has either the form

$$A \textbf{ causes } L_0 \textbf{ if } L_1 \wedge \cdots \wedge L_n \qquad (n > 0)$$

where each L_i $(0 \leq i \leq n)$ is a fluent literal, or the form

$$A \textbf{ causes } L$$

where L is a fluent literal.

We say that an influence proposition is *simple* if it has either the form

$$A \textbf{ possibly changes } F \textbf{ if } L_1 \wedge \cdots \wedge L_n \qquad (n > 0)$$

where each L_i $(1 \leq i \leq n)$ is a fluent literal, or the form

$$A \textbf{ possibly changes } F\,.$$

Finally, we say that an executability proposition is *simple* if it has the form

$$\textbf{impossible } A \textbf{ if } L_1 \wedge \cdots \wedge L_n \qquad (n > 0)$$

where each L_i $(1 \leq i \leq n)$ is a fluent literal.

We say that an $\mathcal{AC}$ domain description is *simple* if all of its propositions are. Perhaps the most severe restriction on simple domain descriptions is that they cannot include explicit definitions, due to the restricted form of sufficiency propositions. Notice that three of the four example domain descriptions considered in Section 2 are in fact simple domain descriptions.

6.2. Embedding Simple Domain Descriptions in Logic Programming

Let D be a simple $\mathcal{AC}$ domain description. We define its translation into a logic program $\pi(D)$ as follows.

For each value proposition $V_1 \vee \cdots \vee V_m \vee \neg V_{m+1} \vee \cdots \vee \neg V_n$ in D, include the rule

$$\mathit{False} \leftarrow [\![V_{m+1}]\!], \ldots, [\![V_n]\!], \mathit{not}\,[\![V_1]\!], \ldots, \mathit{not}\,[\![V_m]\!]\,.$$

For each sufficiency proposition in D of the form $L_1 \wedge \cdots \wedge L_n$ **suffices for** L_0 include the rule

$$\mathit{Holds}(L_0, s) \leftarrow \mathit{Holds}(L_1, s), \ldots, \mathit{Holds}(L_n, s), \mathit{Reachable}(s)\,.$$

For each sufficiency proposition in D of the form **always** L include the rule

$$\mathit{Holds}(L, s) \leftarrow \mathit{Reachable}(s)\,.$$

For each sufficiency proposition in D of the form **never** $L_1 \wedge \cdots \wedge L_n$ include the rule

$$\mathit{False} \leftarrow \mathit{Holds}(L_1, s), \ldots, \mathit{Holds}(L_n, s), \mathit{Reachable}(s)\,.$$

For each effect proposition in D of the form A **causes** L_0 **if** $L_1 \wedge \cdots \wedge L_n$ include the rule

$$Holds(L_0, Result(A,s)) \leftarrow Holds(L_1,s), \ldots, Holds(L_n,s), \\ Reachable(Result(A,s)) .$$

For each effect proposition in D of the form A **causes** L include the rule

$$Holds(L, Result(A,s)) \leftarrow Reachable(Result(A,s)) .$$

For each influence proposition in D of the form

$$A \textbf{ possibly changes } F \textbf{ if } L_1 \wedge \cdots \wedge L_n$$

include the following two rules.

$$\begin{aligned} Holds(F, Result(A,s)) &\leftarrow Holds(L_1,s), \ldots, Holds(L_n,s), \\ &\quad Reachable(Result(A,s)), not \neg Holds(F, Result(A,s)) \\ \neg Holds(F, Result(A,s)) &\leftarrow Holds(L_1,s), \ldots, Holds(L_n,s), \\ &\quad Reachable(Result(A,s)), not\, Holds(F, Result(A,s)) \end{aligned}$$

For each influence proposition in D of the form A **possibly changes** F include the following two rules.

$$\begin{aligned} Holds(F, Result(A,s)) &\leftarrow Reachable(Result(A,s)), not \neg Holds(F, Result(A,s)) \\ \neg Holds(F, Result(A,s)) &\leftarrow Reachable(Result(A,s)), not\, Holds(F, Result(A,s)) \end{aligned}$$

Finally, for each executability proposition **impossible** A **if** $L_1 \wedge \cdots \wedge L_n$ in D, include the rule

$$\neg Reachable(s) \leftarrow Holds(L_1,s), \ldots, Holds(L_n,s) .$$

Also include the following six standard rules for reachability, completeness of the initial situation, and the commonsense law of inertia.

$$\begin{aligned} Reachable(s) &\leftarrow not \neg Reachable(s) \\ \neg Reachable(Result(a,s)) &\leftarrow \neg Reachable(s) \\ Holds(f, S_0) &\leftarrow not \neg Holds(f, S_0) \\ \neg Holds(f, S_0) &\leftarrow not\, Holds(f, S_0) \\ Holds(f, Result(a,s)) &\leftarrow Holds(f,s), Reachable(Result(a,s)), \\ &\quad not \neg Holds(f, Result(a,s)) \\ \neg Holds(f, Result(a,s)) &\leftarrow \neg Holds(f,s), Reachable(Result(a,s)), \\ &\quad not\, Holds(f, Result(a,s)) \end{aligned}$$

Notice that the logic program in the fourth example in Section 2 can be obtained by the translation π.

Theorem 6.1. **(LP Embedding Theorem)** *Let D be a simple, qualification-free $\mathcal{AC}$ domain description. A simple atomic value proposition L* **after** $\overline{A}$ *is entailed by D if and only if the literal $Holds(L, [\overline{A}])$ is entailed by the logic program $\pi(D)$.*

The LP Embedding Theorem is an immediate consequence of the following stronger theorem, which is proved in Appendix B using the Correspondence Theorem for default logic.

Theorem 6.2. **(LP Correspondence Theorem)** *Let D be a simple, qualification-free $\mathcal{AC}$ domain description. There is a one-to-one correspondence between models of D and consistent answer sets of $\pi(D)$ such that, for every model Ψ of D and corresponding answer set X, a simple value proposition*

$$V_1 \vee \cdots \vee V_m \vee \neg V_{m+1} \vee \cdots \vee \neg V_n$$

is true in Ψ if and only if at least one of the sets $\{ [\![V_1]\!], \ldots, [\![V_m]\!] \} \cap X$ and $\{ [\![V_{m+1}]\!], \ldots, [\![V_n]\!] \} \setminus X$ is nonempty.

If a simple domain description includes no executability propositions, we can eliminate the *Reachable* atoms in the corresponding logic program, thus obtaining a simpler translation. So let D be a simple $\mathcal{AC}$ domain description without executability propositions. By $\pi'(D)$ we denote the logic program obtained from $\pi(D)$ by first eliminating the reachability axioms and then deleting all *Reachable* atoms from the remaining rules. Notice that the logic program in the third example in Section 2 can be obtained by the translation π'.

Corollary 6.1. **(LP Reachability Corollary)** *Let D be a simple, qualification-free $\mathcal{AC}$ domain description without executability propositions. There is a one-to-one correspondence between models of D and consistent answer sets of $\pi'(D)$ such that, for every model Ψ of D and corresponding answer set X, a simple value proposition $V_1 \vee \cdots \vee V_m \vee \neg V_{m+1} \vee \cdots \vee \neg V_n$ is true in Ψ if and only if the set $\{ [\![V_1]\!], \ldots, [\![V_m]\!], \overline{[\![V_{m+1}]\!]}, \ldots, \overline{[\![V_n]\!]} \} \cap X$ is nonempty.*

6.3. Making Vivid Domain Descriptions Simple

The syntactic restrictions which define the class of simple domain descriptions are, fortunately, more strict than necessary. In this subsection we show that a much broader class of $\mathcal{AC}$ domain descriptions can be embedded into logic programming.

We say that a sufficiency proposition is *vivid* if it has either the form

$$\phi \textbf{ suffices for } \psi$$

where ψ is a nonempty conjunction of fluent literals, or the form

$$\phi \textbf{ suffices for } \mathit{False}\ .$$

Similarly, we say that an effect proposition is *vivid* if it has the form

$$A \textbf{ causes } \phi \textbf{ if } \psi$$

where ϕ is a nonempty conjunction of fluent literals.

We say that a domain description is *vivid* if all of its sufficiency propositions and effect propositions are. Any vivid domain description can be transformed into an equivalent simple domain description, in the manner described below.

We begin by assuming a function *CNF* that takes every fluent formula ϕ to an equivalent fluent formula $\mathit{CNF}(\phi)$ in conjunctive normal form, in which the special atoms *True* and *False* do not occur. We also assume a function *DNF* that takes every fluent formula ϕ to an equivalent fluent formula $\mathit{DNF}(\phi)$ in disjunctive normal form, in which the special atoms *True* and *False* do not occur.

For any atomic value proposition ϕ **after** $\overline{A}$, let $CNF(\phi$ **after** $\overline{A})$ be the result of simultaneously replacing each disjunct L of each conjunct of $CNF(\phi)$ with the simple atomic value proposition L **after** $\overline{A}$. Notice that ϕ **after** $\overline{A}$ is true in a structure Ψ if and only if $CNF(\phi$ **after** $\overline{A})$ is.

Next we describe a three-step transformation that takes any value proposition V to a corresponding family of simple value propositions.

1. Let V_s be the result of simultaneously replacing each atomic value proposition V' that occurs in V with the value proposition $CNF(V')$. Notice that V_s is a propositional combination of simple atomic value propositions. Notice also that V is true in a structure Ψ if and only if V_s is.

2. Let C be the set of conjuncts of the conjunctive normal form of V_s. Notice that each member of C is a disjunction of simple atomic value propositions or their negations. Notice also that V_s is true in a structure Ψ if and only if every member of C is.

3. Take the set of value propositions obtained by reordering the literals of each member of C so that each of the resulting expressions is a simple value proposition.

Observe that V is true in a structure Ψ if and only if all of the corresponding simple value propositions are.

For any vivid sufficiency proposition ϕ **suffices for** ψ, take the family of simple sufficiency propositions ϕ' **suffices for** L such that ϕ' is a disjunct of $DNF(\phi)$ and L is a conjunct of ψ.

For any vivid effect proposition A **causes** ϕ **if** ψ, take the family of simple effect propositions A **causes** L **if** ψ' such that L is a conjunct of ϕ and ψ' is a disjunct of $DNF(\psi)$.

For any influence proposition A **possibly changes** F **if** ψ, take the family of simple influence propositions A **possibly changes** F **if** ψ' such that ψ' is a disjunct of $DNF(\psi)$.

Finally, for each executability proposition **impossible** A **if** ψ, take the family of simple executability propositions **impossible** A **if** ψ' such that ψ' is a disjunct of $DNF(\psi)$.

Let *Simple* be a function that takes every vivid domain description to a simple domain description that can be obtained by transforming each of its propositions in the manner described above.

Theorem 6.3. **(Vivid Domains Theorem)** *Let D be a vivid $\mathcal{AC}$ domain description. The domain descriptions D and $Simple(D)$ have the same models. Moreover, D is qualification-free if and only if $Simple(D)$ is.*

Since we have already specified a correct embedding of simple, qualification-free domain descriptions into logic programming, the Vivid Domains Theorem establishes the more general fact that every vivid, qualification-free domain description can be correctly embedded in logic programming, by first transforming it into an equivalent simple, qualification-free domain description. For instance, the logic program in the first example in Section 2 can be obtained in this manner.

Finally, it is clear from the previous discussion that any value proposition V can be transformed into a family Q of simple value propositions such that V is true in

a structure Ψ if and only if every member of Q is. Thus we have shown that the LP Correspondence Theorem can be applied to any value proposition, for any vivid qualification-free $\mathcal{AC}$ domain description.

6.4. Comparison with Previous Similar Work

Previous work on translations from high-level action languages into logic programming has for the most part focused on the action language $\mathcal{A}$ [GL93].[10] All such translations have been restricted to "consistent" domain descriptions. For now we will refer to the set of all such $\mathcal{A}$ domain descriptions as "consistent $\mathcal{A}$."

The portion of the language $\mathcal{AC}$ that we correctly embed in logic programming subsumes consistent $\mathcal{A}$ [GL93]. In fact, consistent $\mathcal{A}$ corresponds to a strict subset of the simple, qualification-free $\mathcal{AC}$ domain descriptions. By way of comparison, the original translation in [GL93] of consistent $\mathcal{A}$ into logic programming was partial — it was not sound for some consistent $\mathcal{A}$ domain descriptions — and incomplete. Later sound and complete translations of consistent $\mathcal{A}$ utilized extensions or variants of logic programming: abductive logic programming [DD93, Dun93], equational logic programming [Thi94], or disjunctive logic programming [Tur94]. For the portion of $\mathcal{AC}$ that corresponds to consistent $\mathcal{A}$, our sound and complete translation π' is arguably simpler than any of those previously published, and does not require abduction or disjunction. Furthermore, the portion of $\mathcal{AC}$ that we embed in logic programming — that is, the set of all vivid, qualification-free $\mathcal{AC}$ domain descriptions — is considerably more expressive than consistent $\mathcal{A}$, since it allows non-atomic value propositions, nondeterministic actions, causal background information and action preconditions.

7. REMARKS ON THE ACTION LANGUAGE $\mathcal{AC}$

The action language $\mathcal{AC}$ closely resembles the language $\mathcal{AR}$ of Giunchiglia, Kartha and Lifschitz [GKL95] and its predecessor $\mathcal{AR}_0$ [KL94]. Unlike the language $\mathcal{AC}$, $\mathcal{AR}$ allows non-boolean fluents; but if we consider only the propositional portion of $\mathcal{AR}$, we find that the model structures for the languages are identical.

Syntactically, $\mathcal{AC}$ and $\mathcal{AR}$ are very similar. One difference is that $\mathcal{AR}$ does not include sufficiency propositions for representing background knowledge, which is instead represented by means of state constraints of the form **always** ϕ where ϕ is a fluent formula. In $\mathcal{AC}$ we understand such an expression as an abbreviation of the corresponding sufficiency proposition *True* **suffices for** ϕ. Thus $\mathcal{AR}$ state constraints are well-formed $\mathcal{AC}$ propositions. Another significant syntactic difference between $\mathcal{AC}$ and $\mathcal{AR}$ is that $\mathcal{AR}$ includes only atomic value propositions, whereas $\mathcal{AC}$ allows propositional combinations of atomic value propositions. A third difference is that in $\mathcal{AR}$ the expression **impossible** A **if** ψ is an abbreviation for the effect proposition A **causes** *False* **if** ψ whereas in $\mathcal{AC}$ these are distinct propositions.[11]

[10] An exception is [BG93], which translates the action language $\mathcal{A}_C$ — a language very similar to $\mathcal{A}$ but allowing concurrent and non-executable actions. But the translation there is similar to the translation of $\mathcal{A}$ in [GL93], and exhibits similar shortcomings.

[11] We have seen that this distinction becomes convenient when we specify the translations from $\mathcal{AC}$ into default logic and logic programming.

As the preceding observations suggest, the set of well-formed propositional $\mathcal{AR}$ expressions is a proper subset of the set of well-formed $\mathcal{AC}$ expressions. Given this, the relationship between $\mathcal{AR}$ and $\mathcal{AC}$ is captured in the following theorem.

Theorem 7.1. **($\mathcal{AR}$ Theorem)** *Let D be a propositional $\mathcal{AR}$ domain description such that every nonframe fluent in D has an explicit definition in terms of frame fluents. D is an $\mathcal{AC}$ domain description, and the $\mathcal{AC}$ models of D are exactly the $\mathcal{AR}$ models of D.*

The statement of the $\mathcal{AR}$ Theorem reflects the fact that some propositional $\mathcal{AR}$ domain descriptions are not $\mathcal{AC}$ domain descriptions. These are the propositional $\mathcal{AR}$ domain descriptions in which there is a nonframe fluent that is not explicitly defined in terms of frame fluents. On the other hand, we have observed that some $\mathcal{AC}$ domain descriptions are not $\mathcal{AR}$ domain descriptions. For example, consider the following $\mathcal{AC}$ formalization of the Two-Switches domain, adapted from [KL94] (and originally introduced in [Lif90]).

$$Up(Switch_1) \equiv Up(Switch_2) \textbf{ suffices for } On$$
$$Up(Switch_1) \not\equiv Up(Switch_2) \textbf{ suffices for } \neg On$$
$$Toggle(x) \textbf{ causes } Up(x) \textbf{ if } \neg Up(x)$$
$$Toggle(x) \textbf{ causes } \neg Up(x) \textbf{ if } Up(x)$$

The Two-Switches domain can be formalized in $\mathcal{AR}$ by declaring the fluent On to be nonframe and replacing the two sufficiency propositions by a single state constraint

$$\textbf{always } On \equiv (Up(Switch_1) \equiv Up(Switch_2)) .$$

In modifying the domain description in this manner, we seem to be replacing causal information — the fact that the state of the switches "causally" determines the state of the light — with a "non-causal" explicit definition. But in doing so, we do not change the set of models.[12]

Let us consider a slight variant of the domain description from Example 2 (Section 2), adapted from [Lin95], which demonstrates that it is not always possible to obtain intuitively correct results using state constraints augmented by the frame/nonframe distinction. In this action domain, there is a spring-loaded briefcase with two clasps. We have actions that unfasten the clasps, one at a time. If both clasps are unfastened, the briefcase pops open. Initially the briefcase is not open and the second clasp is not fastened. We can conclude in this case that the briefcase is open after we unfasten the first clasp. We formalize this action domain in $\mathcal{AC}$ as follows.

$$\textbf{initially } \neg Open \wedge \neg Fastened(Clasp_2)$$
$$Unfasten(x) \textbf{ causes } \neg Fastened(x)$$
$$\neg Fastened(Clasp_1) \wedge \neg Fastened(Clasp_2) \textbf{ suffices for } Open$$

This domain description entails the value proposition

$$Open \textbf{ after } Unfasten(Clasp_1) .$$

[12] Notice that in this case, the domain description we obtain is in fact a "legal" $\mathcal{AC}$ domain description, since the nonframe fluent On is explicitly defined in terms of the frame fluent formula $Up(Switch_1) \equiv Up(Switch_2)$.

Traditionally, background knowledge in action domains has been represented in the form of state constraints, which are, intuitively speaking, formulas of classical logic that are said to hold in every possible state of the world. Moreover, it has been common to use state constraints to derive indirect effects, or "ramifications" of actions, as is in fact done in the language $\mathcal{AR}$. Thus, for example, one might think of writing the state constraint

$$\textbf{always } (\neg \mathit{Fastened}(\mathit{Clasp}_1) \wedge \neg \mathit{Fastened}(\mathit{Clasp}_2)) \supset \mathit{Open}$$

in place of

$$\neg \mathit{Fastened}(\mathit{Clasp}_1) \wedge \neg \mathit{Fastened}(\mathit{Clasp}_2) \textbf{ suffices for } \mathit{Open}\,.$$

But it seems that there is no way of designating frame and nonframe fluents that will allow the resulting $\mathcal{AR}$ domain description to capture the intended models of the domain. For instance, if we declare *Open* nonframe, then the briefcase can open spontaneously, as it were, at any time. On the other hand, if we leave all fluents "in the frame," we find that unfastening the first clasp can sometimes have the unintended indirect effect of fastening the second clasp.

Lin and Reiter [LR94] have suggested the name "ramification constraints" for state constraints that are used to derive indirect effects. For example, state constraints in the languages $\mathcal{AR}_0$ and $\mathcal{AR}$ function as ramification constraints, as do state constraints in the action formalizations of Winslett [Win88] and Baker [Bak91]. One thing the $\mathcal{AR}$ Theorem shows is that $\mathcal{AC}$ expressions of the form

$$\textbf{always } \phi$$

correspond precisely to state constraints in $\mathcal{AR}$. Recall that in $\mathcal{AC}$ such an expression stands for the sufficiency proposition

$$\mathit{True} \textbf{ suffices for } \phi\,.$$

It is natural to call such $\mathcal{AC}$ propositions *ramification constraints.*

Lin and Reiter [LR94] describe another use of state constraints: as so-called "qualification constraints." Qualification constraints are state constraints that simply restrict the state space; they do not themselves lead to any indirect effects. Qualification constraints are so-named because they can lead to "derived action preconditions," or "qualifications."[13]It is straightforward to verify that $\mathcal{AC}$ sufficiency propositions of the form

$$\phi \textbf{ suffices for } \mathit{False}$$

in fact function as qualification constraints, since such propositions simply rule out any state in which ϕ holds, without leading to any indirect effects. Recall that we abbreviate such sufficiency propositions as

$$\textbf{never } \phi\,.$$

It is natural to call such $\mathcal{AC}$ propositions *qualification constraints.*[14]

[13]This idea was anticipated by Ginsberg and Smith [GS88].

[14]Much of this discussion is adapted from [MT95b], where the relationship of static causal laws to ramification and qualification constraints is addressed in a more abstract setting.

As an example of an $\mathcal{AC}$ domain description involving a qualification constraint, consider the following formalization of the Emperor Domain of Lin and Reiter [LR94].

$$\textbf{never } \mathit{Yellow}(\mathit{Block}_1) \wedge \mathit{Yellow}(\mathit{Block}_2)$$
$$\mathit{Paint}(x) \textbf{ causes } \mathit{Yellow}(x)$$

This domain description does not entail the $\mathcal{AC}$ value proposition

$$\mathit{Yellow}(\mathit{Block}_2) \textbf{ after } \mathit{Paint}(\mathit{Block}_2)$$

but it does entail the following weaker proposition.

$$(\textbf{ initially } \neg \mathit{Yellow}(\mathit{Block}_1)) \equiv (\mathit{Yellow}(\mathit{Block}_2) \textbf{ after } \mathit{Paint}(\mathit{Block}_2))$$

This reflects the fact that it is possible to paint the second block yellow if and only if the first block is not already yellow. Observe that in this case, in order to obtain an equivalent $\mathcal{AR}$ domain description we replace the sufficiency proposition with the state constraint

$$\textbf{always } \neg(\mathit{Yellow}(\mathit{Block}_1) \wedge \mathit{Yellow}(\mathit{Block}_2))$$

and also explictly describe the action preconditions, as follows.

$$\textbf{impossible } \mathit{Paint}(\mathit{Block}_1) \textbf{ if } \mathit{Yellow}(\mathit{Block}_2)$$
$$\textbf{impossible } \mathit{Paint}(\mathit{Block}_2) \textbf{ if } \mathit{Yellow}(\mathit{Block}_1)$$

Up to now we have not presented an example in which it is natural to use a ramification constraint (except to introduce an explicit definition). So consider a blocks world in which there are two blocks (A,B) and four locations (1,2,3,4). Each block is always in exactly one location. There are never two blocks in the same location. For each block, there is a move action that changes its location. We can describe this action domain in $\mathcal{AC}$ as follows.

$$\textbf{always } \mathit{Loc}(x,1) \vee \mathit{Loc}(x,2) \vee \mathit{Loc}(x,3) \vee \mathit{Loc}(x,4)$$
$$\textbf{always } \neg\mathit{Loc}(x,m) \vee \neg\mathit{Loc}(x,n) \qquad (m \neq n)$$
$$\textbf{never } \mathit{Loc}(A,n) \wedge \mathit{Loc}(B,n)$$
$$\mathit{Move}(x) \textbf{ causes } \neg\mathit{Loc}(x,n) \textbf{ if } \mathit{Loc}(x,n)$$

This domain description entails, for instance, the value propositions

$$(\textbf{ initially } \mathit{Loc}(A,1) \wedge \mathit{Loc}(B,2)) \supset (\mathit{Loc}(A,3) \not\equiv \mathit{Loc}(A,4) \textbf{ after } \mathit{Move}(A))$$

and

$$(\mathit{Loc}(A,1) \textbf{ after } \mathit{Move}(A)) \supset \textbf{ initially } \neg\mathit{Loc}(B,1).$$

Sufficiency propositions are closely related to inference rules, as is apparent from the definition of *Res* in the semantics of $\mathcal{AC}$. Brewka and Hertzberg [BH93] use inference rules to encode causal background knowledge for reasoning about action. Their definition differs markedly from ours though. For instance, as we point out in [MT95b], their approach cannot capture the notion of qualification constraints. In fact, it sometimes yields different results even when derived action preconditions are not involved.

The idea for the fixpoint definition of *Res* adopted in the semantics of $\mathcal{AC}$ was introduced in a pair of related papers [MT95b, PT95]. In [MT95b] we show how the fixpoint definition can be derived as a natural extension of the corresponding definition from Winslett's [Win88] possible models approach to reasoning about the effects of actions. In [PT95] the idea is explored in the more abstract setting of theory update. There we introduce a fixpoint definition of "rule update" — update by means of arbitrary sets of inference rules — and show that rule update not only extends Winslett's update definition but also extends the notion of revision programming, introduced by Marek and Truszczyński [MT93, MT94, MT95a].

In other recent related work, Baral [Bar95] proposes an action description language based closely upon revision programming. The semantics of his action language is given directly in terms of a translation into disjunctive logic programs. This would seem to be a weakness of his proposal, since disjunctive logic programs are relatively difficult to reason about. Nonetheless, it seems that where Baral's proposal overlaps with ours, it agrees. Lin [Lin95] introduces a circumscriptive approach to causal theories of action that is closely related to his previous work with Reiter [LR94]. Lin shows that for a special class of action descriptions, the meaning of a description can be obtained by a reasonably straightforward completion process; but in the general case, the semantics of Lin's action descriptions is given in terms of a complex minimization process. In the case of what Lin calls a "stratified" theory, it appears that Lin's proposal will agree with ours. Thielscher [Thi95] extends previous work, by himself and his colleagues, on reasoning about action in the formalism of equational logic programming. His proposal involves the use of state constraints accompanied by auxiliary information about directional, causal relationships between pairs of fluent atoms. The semantics of his action description language is given by a definition that is essentially procedural, and in fact seems motivated by computational (rather than declarative) concerns. It is unclear to what extent his proposal overlaps with ours.

One advantage of the action description language $\mathcal{AC}$ over those of [Bar95, Lin95, Thi95] is that it naturally accommodates the use of arbitrary propositional formulas in the description of static causal laws and effects of actions. This makes it possible to express traditional ramification constraints, for instance. Also, recall that such formulas are used when explicit definitions are introduced. Another advantage is that $\mathcal{AC}$ has a relatively transparent semantics, specially tailored for action domains, in which there is a simple definition of possible next states that is used in a straightforward manner to constrain a situation calculus model structure.

We conclude this section with three results concerning formal properties of $\mathcal{AC}$, which are modeled on similar results for the language $\mathcal{AR}$ [GKL95]. We omit the proofs, which are reasonably straightforward.

Theorem 7.2. **(Replacement Theorem)** *Let D be an $\mathcal{AC}$ domain description. Let T be a subset of the sufficiency propositions in D. Take*

$$R_T = \left\{ \frac{\phi}{\psi} : \phi \textbf{ suffices for } \psi \in T \right\}.$$

Let ϕ, ϕ' be fluent formulas such that $(\phi \equiv \phi') \in Cn(R_T)$. Let D' be an $\mathcal{AC}$ domain description obtained from D by replacing some or all occurrences of ϕ with ϕ' in some or all propositions that do not belong to T. Domain descriptions D and D' have the same models.

Corollary 7.1. **(Explicit Definitions Corollary)** *Let D be an $\mathcal{AC}$ domain description in which there is an explicit definition* **always** $F \equiv \phi$ *and furthermore there is no influence proposition A* **possibly changes** F **if** ψ. *Let D' be the domain description with fluents* $\mathbf{F} \setminus \{F\}$ *that can obtained from D by deleting the explicit definition* **always** $F \equiv \phi$ *and replacing all remaining occurrences of F with ϕ. For every value proposition V in which F does not occur, V is true in D if and only if V is true in D'.*

Theorem 7.3. **(Restricted Monotonicity Theorem)**[15] *Let D be an $\mathcal{AC}$ domain description. If D' can be obtained by adding value propositions to D, then every value proposition entailed by D is also entailed by D'.*

ACKNOWLEDGMENTS

Many thanks to Vladimir Lifschitz and Norman McCain. Thanks also to Chitta Baral, Michael Gelfond, Enrico Giunchiglia, G. Neelakantan Kartha, Teodor Przymusinski and Michael Thielscher for helpful discussions. This work is partially supported by National Science Foundation grant #IRI-9306751.

REFERENCES

[Bak91.] Andrew Baker. Nonmonotonic reasoning in the framework of situation calculus. *Artificial Intelligence*, 49, pages 5–23, 1991.

[Bar95.] Chitta Baral. Reasoning about actions: non-deterministic effects, constraints, and qualifications. In *Proc. of IJCAI-95*, pages 2017–2023, 1995.

[BG93.] Chitta Baral and Michael Gelfond. Representing concurrent actions in extended logic programming. In *Proc. of IJCAI-93*, pages 866–871, 1993.

[BH93.] Gerhard Brewka and Joachim Hertzberg. How to do things with worlds: On formalizing actions and plans. *Journal of Logic and Computation*, 3(5), 1993.

[DD93.] Marc Denecker and Danny DeSchreye. Representing incomplete knowledge in abductive logic programming. In *Logic Programming: Proc. of the 1993 Int'l Symposium*, pages 147–163, 1993.

[Dun93.] Phan Minh Dung. Representing actions in logic programming and its applications in database updates. In *Logic Programming: Proc. of the 10th Int'l Conference*, pages 7–25, 1993.

[Elk92.] Charles Elkan. Reasoning about action in first-order logic. In *Proc. of the 1992 Canadian Conf. on Artificial Intelligence*, 1992.

[Gef90.] Hector Geffner. Causal theories of nonmonotonic reasoning. In *Proc. of AAAI-90*, pages 524–530, 1990.

[GKL95.] Enrico Giunchiglia, G. Neelakantan Kartha, and Vladimir Lifschitz. Actions with indirect effects (extended abstract). In *Working Notes of the AAAI Spring Symposium on Extending Theories of Action*, pages 80–85, 1995.

[15] See [Lif93] for a general account of restricted monotonicity.

[GL88.] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, pages 1070–1080, 1988.

[GL90.] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In David Warren and Peter Szeredi, editors, *Logic Programming: Proc. of the 7th Int'l Conference*, pages 579–597, 1990.

[GL91.] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[GL93.] Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.

[GLPT91.] Michael Gelfond, Vladimir Lifschitz, Halina Przymusińska, and Miroslaw Truszczyński. Disjunctive defaults. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proc. of the 2nd Int'l Conference*, pages 230–237, 1991.

[GS88.] Matthew Ginsberg and D.E. Smith. Reasoning about actions II: The qualification problem. *Artificial Intelligence*, 35:311–342, 1988.

[HM87.] Steve Hanks and Drew McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

[KL94.] G. Neelakantan Kartha and Vladimir Lifschitz. Actions with indirect effects (preliminary report). In *Proc. of the Fourth Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 341–350, 1994.

[Lif90.] Vladimir Lifschitz. Frames in the space of situations. *Artificial Intelligence*, 46:365–376, 1990.

[Lif93.] Vladimir Lifschitz. Restricted monotonicity. In *Proc. AAAI-93*, pages 432–437, 1993.

[Lin95.] Fangzhen Lin. Embracing causality in specifying the indirect effects of actions. In *Proc. of IJCAI-95*, pages 1985–1991, 1995.

[LR94.] Fangzhen Lin and Raymond Reiter. State constraints revisited. *Journal of Logic and Computation*, 4(5):655–678, 1994. Special Issue on Actions and Processes.

[LT94.] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In Pascal Van Hentenryck, editor, *Proc. Eleventh Int'l Conf. on Logic Programming*, pages 23–37, 1994.

[LT95.] Vladimir Lifschitz and Hudson Turner. From disjunctive programs to abduction. In Jürgen Dix, Luis Pereira, and Teodor Przymusinski, editors, *Non-Monotonic Extensions of Logic Programming (Lecture Notes in Artificial Intelligence 927)*, pages 23–42. Springer-Verlag, 1995.

[Mor88.] Paul Morris. The anomalous extension problem in default reasoning. *Artificial Intelligence*, 35(3):383–399, 1988.

[MT93.] Wiktor Marek and Miroslaw Truszczyński. Revision programming. Manuscript, 1993.

[MT94.] Wiktor Marek and Miroslaw Truszczyński. Revision specifications by means of programs. In *Logics in AI. Proceedings of JELIA '94*, 1994.

[MT95a.] Wiktor Marek and Miroslaw Truszczyński. Revision programming, database updates and integrity constraints. In *Proc. of the 5th Int'l Conf. on Database Theory*, pages 368–382, 1995.

[MT95b.] Norman McCain and Hudson Turner. A causal theory of ramifications and qualifications. In *Proc. of IJCAI-95*, pages 1978–1984, 1995.

[PT95.] Teodor Przymusinski and Hudson Turner. Update by means of inference rules. In *Proc. of the 3rd Int'l Conf. on Logic Programming and Nonmonotonic Reasoning*, pages 156–174, 1995.

[Rei80.] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1,2):81–132, 1980.

[San94.] Erik Sandewall. *Features and Fluents*. Oxford University Press, 1994.

[Thi94.] Michael Thielscher. Representing actions in equational logic programming. In Pascal Van Hentenryck, editor, *Logic Programming: Proc. of the 11th Int'l Conference*, pages 207–224. MIT Press, 1994.

[Thi95.] Michael Thielscher. Computing ramifications by postprocessing. In *Proc. of IJCAI-95*, pages 1994–2000, 1995.

[Tur94.] Hudson Turner. Signed logic programs. In Maurice Bruynooghe, editor, *Logic Programming: Proc. of the 1994 Int'l Symposium*, pages 61–75, 1994.

[Tur96a.] Hudson Turner. Representing actions in default logic: A situation calculus approach. In *Working Papers of the Third Symposium on Logical Formalizations of Commonsense Reasoning*, 1996.

[Tur96b.] Hudson Turner. Splitting a default theory. In *Proc. of AAAI-96*, 1996. To appear.

[Win88.] Marianne Winslett. Reasoning about action using a possible models approach. In *Proc. of AAAI–88*, pages 89–93, 1988.

A. APPENDIX: SPLITTING A DEFAULT THEORY

In this appendix we briefly turn our attention from the specific problem of representing actions in default logic and logic programming, in order to present technical results concerning default theories in general. These results — the Splitting Set Theorem and Splitting Sequence Theorem for default logic from [Tur96b] — are needed for the proof of the Correspondence Theorem (in Appendix B).

Splitting Sets

Let D be a default theory over $\mathcal{L}(U)$ such that, for every rule $r \in D$, $pre(r)$ is in conjunctive normal form. (Of course any default theory can be easily transformed into an equivalent default theory, over the same language, satisfying this condition.) For any rule $r \in D$, a formula ϕ is a *constituent of* r if at least one of the following conditions holds: (i) ϕ is a conjunct of $pre(r)$; (ii) $\phi \in just(r)$; (iii) $\phi = cons(r)$.

A *splitting set* for D is a subset A of U such that for every rule $r \in D$ the following two conditions hold.

1. Every constituent of r belongs to $\mathcal{L}(A) \cup \mathcal{L}(U \setminus A)$.
2. If $cons(r)$ does not belong to $\mathcal{L}(U \setminus A)$, then r is a default rule over $\mathcal{L}(A)$.

If A is a splitting set for D, we say that A *splits* D. The *base of* D *relative to* A, denoted by $b_A(D)$, is the default theory over $\mathcal{L}(A)$ that consists of all members of D that are default rules over $\mathcal{L}(A)$.

Let $U_2 = \{a, b, c, d\}$. Consider the following default theory D_2 over $\mathcal{L}(U_2)$.

$$\frac{:\neg b}{a} \quad \frac{:\neg a}{b} \quad \frac{a \vee b : a, b}{c \vee d} \quad \frac{a \wedge (c \vee d) : \neg d}{\neg d} \quad \frac{b \wedge (c \vee d) : \neg c}{\neg c}$$

Take $A_2 = \{a, b\}$. It's easy to verify that A_2 splits D_2, with

$$b_{A_2}(D_2) \;=\; \left\{ \frac{:\neg b}{a}\,,\; \frac{:\neg a}{b} \right\} .$$

Notice that the default theory $b_{A_2}(D_2)$ over $\mathcal{L}(A_2)$ has two consistent extensions:

$$Cn_{A_2}(\{a\}) \quad \text{and} \quad Cn_{A_2}(\{b\})\,.$$

Given a splitting set A for D, and a set X of formulas from $\mathcal{L}(A)$, the *partial evaluation of D by X with respect to A*, denoted by $e_A\,(D, X)$, is the default theory over $\mathcal{L}(U \setminus A)$ obtained from D in the following manner. For each rule $r \in D \setminus b_A(D)$ such that

1. every conjunct of $pre(r)$ that belongs to $\mathcal{L}(A)$ also belongs to $Cn_A(X)$, and
2. no member of $just(r)$ has its complement in $Cn_A(X)$

there is a rule $r' \in e_A\,(D, X)$ such that

1. $pre(r')$ is obtained from $pre(r)$ by replacing each conjunct of $pre(r)$ that belongs to $\mathcal{L}(A)$ by $True$, and
2. $just(r') = just(r) \cap \mathcal{L}(U \setminus A)$, and
3. $cons(r') = cons(r)\,.$

For example, it is easy to verify that

$$e_{A_2}(D_2\,,\, Cn_{A_2}(\{a\})) \;=\; \left\{ \frac{True}{c \vee d}\,,\; \frac{True \wedge (c \vee d) : \neg d}{\neg d} \right\}$$

and that

$$e_{A_2}(D_2\,,\, Cn_{A_2}(\{b\})) \;=\; \left\{ \frac{True}{c \vee d}\,,\; \frac{True \wedge (c \vee d) : \neg c}{\neg c} \right\} .$$

Let A be a splitting set for D. A *solution to D with respect to A* is a pair $\langle X, Y \rangle$ of sets of formulas satisfying the following two properties.

1. X is a consistent extension of the default theory $b_A(D)$ over $\mathcal{L}(A)$.
2. Y is a consistent extension of the default theory $e_A\,(D, X)$ over $\mathcal{L}(U \setminus A)$.

For example, given our previous observations, it is easy to verify that D_2 has two solutions with respect to A_2:

$$\langle\, Cn_{A_2}(\{a\})\,,\; Cn_{U_2 \setminus A_2}(\{c, \neg d\}) \,\rangle \quad \text{and} \quad \langle\, Cn_{A_2}(\{b\})\,,\; Cn_{U_2 \setminus A_2}(\{\neg c, d\}) \,\rangle\,.$$

Theorem A.1. **(Splitting Set Theorem)** *Let A be a splitting set for a default theory D over $\mathcal{L}(U)$. A set E of formulas is a consistent extension of D if and only if $E = Cn_U(X \cup Y)$ for some solution $\langle X, Y \rangle$ to D with respect to A.*

Thus, for example, it follows from the Splitting Set Theorem that the default theory D_2 has exactly two consistent extensions:

$$Cn_{U_2}(\{a, c, \neg d\}) \quad \text{and} \quad Cn_{U_2}(\{b, \neg c, d\}) .$$

Corollary A.1. **(Splitting Set Corollary)** *Let A be a splitting set for a default theory D over $\mathcal{L}(U)$. If E is a consistent extension of D, then the pair*

$$\langle\, E \cap \mathcal{L}(A)\, ,\, E \cap \mathcal{L}(U \setminus A)\, \rangle$$

is a solution to D with respect to A.

Splitting Sequences

A (transfinite) *sequence* is a family whose index set is an initial segment of ordinals $\{\alpha : \alpha < \mu\}$. We say that a sequence $\langle A_\alpha \rangle_{\alpha<\mu}$ of sets is *monotone* if $A_\alpha \subseteq A_\beta$ whenever $\alpha < \beta$, and *continuous* if, for each limit ordinal $\alpha < \mu$, $A_\alpha = \bigcup_{\gamma<\alpha} A_\gamma$.

A *splitting sequence* for a default theory D over $\mathcal{L}(U)$ is a nonempty, monotone, continuous sequence $\langle A_\alpha \rangle_{\alpha<\mu}$ of splitting sets for D such that $\bigcup_{\alpha<\mu} A_\alpha = U$.

The definition of a solution with respect to a splitting set is extended to splitting sequences as follows. Let $A = \langle A_\alpha \rangle_{\alpha<\mu}$ be a splitting sequence for D. A *solution to D with respect to A* is a sequence $\langle E_\alpha \rangle_{\alpha<\mu}$ of sets of formulas that satisfies the following three conditions.

1. E_0 is a consistent extension of the default theory $b_{A_0}(D)$ over $\mathcal{L}(A_0)$.

2. For any α such that $\alpha + 1 < \mu$, $E_{\alpha+1}$ is a consistent extension of the default theory

$$e_{A_\alpha}\left(b_{A_{\alpha+1}}(D), \bigcup_{\gamma\leq\alpha} E_\gamma\right)$$

over $\mathcal{L}(A_{\alpha+1} \setminus A_\alpha)$.

3. For any limit ordinal $\alpha < \mu$, $E_\alpha = Cn_\emptyset(\emptyset)$.

We generalize the Splitting Set Theorem as follows.

Theorem A.2. **(Splitting Sequence Theorem)** *Let $A = \langle A_\alpha \rangle_{\alpha<\mu}$ be a splitting sequence for a default theory D over $\mathcal{L}(U)$. A set E of formulas is a consistent extension of D if and only if*

$$E = Cn_U\left(\bigcup_{\alpha<\mu} E_\alpha\right)$$

for some solution $\langle E_\alpha \rangle_{\alpha<\mu}$ to D with respect to A.

The proof of this theorem relies on the Splitting Set Theorem. We also have the following counterpart to the Splitting Set Corollary.

Corollary A.2. **(Splitting Sequence Corollary)** *Let $A = \langle A_\alpha \rangle_{\alpha<\mu}$ be a splitting sequence for a default theory D over $\mathcal{L}(U)$. Let $\langle U_\alpha \rangle_{\alpha<\mu}$ be the sequence of pairwise disjoint subsets of U such that for all $\alpha < \mu$*

$$U_\alpha = A_\alpha \setminus \bigcup_{\gamma<\alpha} A_\gamma \, .$$

If E is a consistent extension of D, then the sequence $\langle E \cap \mathcal{L}(U_\alpha) \rangle_{\alpha<\mu}$ is a solution to D with respect to A.

B. APPENDIX: PROOFS

We first prove the Correspondence Theorem and Reachability Corollary, showing that the translations from $\mathcal{AC}$ into default logic are sound and complete.

On the basis of these results, we go on to prove the LP Correspondence Theorem and LP Reachability Corollary, showing the correctness of our translations of simple, qualification-free $\mathcal{AC}$ domain descriptions into logic programming. Finally we prove the Vivid Domains Theorem, which shows that every vivid $\mathcal{AC}$ domain description can be transformed into an equivalent simple domain description.

B.1. Proof of Correspondence Theorem and Reachability Corollary

Our primary task is to prove the special case of the Correspondence Theorem in which the domain description has no value propositions. We'll call this intermediate result the Correspondence Lemma. Most of the work in this subsection is devoted to its proof.

Let D be a qualification-free domain description without value propositions, with fluents $\mathbf{F}$, and frame fluents $\mathbf{F}_f$. We will show that there is a one-to-one correspondence between models of D and consistent extensions of $\delta(D)$ such that a value proposition V is true in a model of D if and only if the formula $[V]$ belongs to the corresponding extension of $\delta(D)$.

We begin with two fundamental lemmas. The first of these will be used to show that our default theory $\delta(D)$ correctly characterizes the possible initial situations.

Let Δ_0 be the default theory

$$R \cup \left\{ \frac{: L}{L} : L \text{ is a fluent literal} \right\} .$$

Notice that Δ_0 is default theory over $\mathcal{L}(\mathbf{F})$.

Lemma B.1. A consistent set X of fluent formulas is an extension of Δ_0 if and only if there is a state S such that $X = Cn_{\mathbf{F}}(S)$.

Proof. Recall that an interpretation S is a state if and only if $Cn_{\mathbf{F}}(S)$ is closed under R. (Recall also that interpretations are maximal consistent sets of literals.)

(Left-to-right) Assume that X is a consistent extension of Δ_0. It is easy to verify that, for every fluent F, either F or $\neg F$ belongs to X. So there is an interpretation S such that $X = Cn_{\mathbf{F}}(S)$. Moreover, since $R \subseteq \Delta_0^X$, we know that X is closed under R; so S is a state.

(Right-to-left) Assume that S is a state, and take $X = Cn_{\mathbf{F}}(S)$. It is easy to verify that

$$\Delta_0^X = S \cup R$$

from which it follows that $X \subseteq Cn_{\mathbf{F}}(\Delta_0^X)$. Of course X is closed under S, and since S is a state, we know that X is also closed under R. And since X is logically closed, we can conclude that $X = Cn_{\mathbf{F}}(\Delta_0^X)$. □

The second fundamental lemma will be used to show that in the consistent extensions of $\delta(D)$, non-initial situations respect the transition function Res. This result is very similar to an embedding theorem from [PT95].

For any state S and action name A, let $\Delta(A, S)$ be the default theory obtained by taking the union of the following four sets of rules.

1. All rules of the form $\frac{:L}{L}$ where L is a frame fluent literal in S.
2. $E(A, S)$
3. All rules of the forms $\frac{:F}{F}$ and $\frac{:\neg F}{\neg F}$ where $F \in F(A, S)$.
4. R

Notice that $\Delta(A, S)$ is a default theory over $\mathcal{L}(\mathbf{F})$.

Lemma B.2. Let S be a state and A an action that is not prohibited in S. The following hold.

1. *A state S' belongs to $Res(A, S)$ if and only if $Cn_{\mathbf{F}}(S')$ is a consistent extension of $\Delta(A, S)$.*
2. *If X is a consistent extension of $\Delta(A, S)$, then there is a state S' such that $X = Cn_{\mathbf{F}}(S')$.*

Proof. For the first part, let S' be a state, and let

$$E = E(A, S) \cup (S' \cap \mathcal{L}(F(A, S))) .$$

Observe that E is an explicit effect of A in S. It is not difficult to verify that

$$\Delta(A, S)^{Cn(S')} = (S \cap S' \cap Lit(\mathbf{F}_f)) \cup E \cup R .$$

Thus we see that $Cn(S')$ is a consistent extension of $\Delta(A, S)$ if and only if $Cn(S') = Cn[(S \cap S' \cap Lit(\mathbf{F}_f)) \cup E \cup R]$.

Since E is an explicit effect of A in S, we have shown that if $Cn(S')$ is a consistent extension of $\Delta(A, S)$ then $S' \in Res(A, S)$. To see the other direction, assume that $S' \in Res(A, S)$. Thus there is an explicit effect E' of A in S such that $Cn(S') = Cn[(S \cap S' \cap Lit(\mathbf{F}_f)) \cup E' \cup R]$. It is clear that $E' = E(A, S) \cup (S' \cap \mathcal{L}(F(A, S)))$, which is to say that $E' = E$. Thus we can conclude that $Cn(S')$ is a consistent extension of $\Delta(A, S)$.

For the second part, assume that X is a consistent extension of $\Delta(A, S)$. Suppose there is a fluent name F such that $F \notin X$ and $\neg F \notin X$. Since every nonframe fluent in an $\mathcal{AC}$ domain description must have a definition in terms of frame fluents,

we can assume without loss of generality that F is a frame fluent. But in this case, since S is a state, $\Delta(A,S)$ includes one of the following two rules.

$$\frac{:F}{F} \quad \frac{:\neg F}{\neg F}$$

From this we can conclude that $Cn(\Delta(A,S)^X)$ includes either F or $\neg F$. So $Cn(\Delta(A,S)^X) \neq X$, which is a contradiction. So we have shown that for every fluent name F, either $F \in X$ or $\neg F \in X$. And since X is consistent, it follows that there is an interpretation S' of $\mathcal{L}(\mathbf{F})$ such that $X = Cn(S')$. Now, since $\Delta(A,S)$ contains the inference rules R, we know that $Cn(S')$ is closed under R. So S' is a state. □

Next we prepare to move these results into the language of default theory $\delta(D)$. This will require three preliminary lemmas.

Let U be the set of atoms such that $\mathcal{L}(U)$ is the language of the default theory $\delta(D)$. We can view $\mathcal{L}(U)$ as including a tree of copies of the language $\mathcal{L}(\mathbf{F})$: one copy for each action string $\overline{A}$. For any set Γ of rules (that is, any combination of default rules, inference rules and formulas) over $\mathcal{L}(\mathbf{F})$ and any action string $\overline{A}$, let $\Sigma(\Gamma,\overline{A})$ denote the set of rules over $\mathcal{L}(U)$ obtained by replacing each occurrence of each fluent atom F in Γ by the atom $Holds(F,[\overline{A}])$. (Notice that occurrences of the special atoms *True* and *False* are left unchanged.) Observe that for each action string $\overline{A}$, the language $\mathcal{L}(\Sigma(\mathbf{F},\overline{A}))$ is a subset of $\mathcal{L}(U)$ such that the rules over $\mathcal{L}(\Sigma(\mathbf{F},\overline{A}))$ and the rules over $\mathcal{L}(\mathbf{F})$ are in one-to-one correspondence.

Lemma B.3. For every set Γ of inference rules over $\mathcal{L}(\mathbf{F})$ and every action string $\overline{A}$, a set X of formulas from $\mathcal{L}(\mathbf{F})$ is closed under Γ if and only if $\Sigma(X,\overline{A})$ is closed under $\Sigma(\Gamma,\overline{A})$.

Proof. (Left-to-right) Assume X is closed under Γ. Let r' be a rule from $\Sigma(\Gamma,\overline{A})$ such that $pre(r') \in \Sigma(X,\overline{A})$. We must show that $cons(r') \in \Sigma(X,\overline{A})$. We know there is a rule $r \in \Gamma$ such that r' can be obtained from r by replacing each occurrence of each fluent atom F in r by the atom $Holds(F,[\overline{A}])$. Since $pre(r') \in \Sigma(X,\overline{A})$, we know that $pre(r) \in X$. Since X is closed under Γ, $cons(r) \in X$, from which it follows that $cons(r') \in \Sigma(X,\overline{A})$. Proof in the other direction is similar. □

Notice that the previous lemma is sufficient to establish also that X is logically closed if and only if $\Sigma(X,\overline{A})$ is.

Lemma B.4. For every set Γ of inference rules over $\mathcal{L}(\mathbf{F})$ and every action string $\overline{A}$, we have $\Sigma(Cn_{\mathbf{F}}(\Gamma),\overline{A}) = Cn_{\Sigma(\mathbf{F},\overline{A})}(\Sigma(\Gamma,\overline{A}))$.

Proof. Follows easily from the previous lemma. □

Lemma B.5. For every default theory D over $\mathcal{L}(\mathbf{F})$ and every action string $\overline{A}$, a set X of formulas from $\mathcal{L}(\mathbf{F})$ is an extension of D if and only if $\Sigma(X,\overline{A})$ is an extension of $\Sigma(D,\overline{A})$.

Proof. By Lemma B.3 we know that $X = Cn_{\mathbf{F}}(D^X)$ if and only if $\Sigma(X,\overline{A}) = \Sigma(Cn_{\mathbf{F}}(D^X),\overline{A})$. By Lemma B.4 we have $\Sigma(Cn_{\mathbf{F}}(D^X),\overline{A}) = Cn_{\Sigma(\mathbf{F},\overline{A})}(\Sigma(D^X,\overline{A}))$. Finally, it is not difficult to verify that $\Sigma(D^X,\overline{A}) = \Sigma(D,\overline{A})^{\Sigma(X,\overline{A})}$ which suffices to establish the lemma. □

In order to apply the two fundamental lemmas (B.1 & B.2) to the default theory

$\delta(D)$, we will split $\delta(D)$ into simpler parts, using the Splitting Sequence Theorem. To this end, we introduce a partial mapping σ from an initial segment of ordinals $\{\alpha : \alpha < \mu\}$ to action strings, which satisfies the following three conditions.

1. For each action string $\overline{A}$ there is a non-limit ordinal $\alpha < \mu$ such that $\sigma(\alpha) = \overline{A}$.
2. For each non-limit ordinal $\alpha < \mu$ there is an action string $\overline{A}$ such that $\sigma(\alpha) = \overline{A}$.
3. For all non-limit ordinals α and β such that $\alpha < \beta < \mu$, $\sigma(\alpha) \neq \sigma(\beta)$ and the length of $\sigma(\alpha)$ is no greater than the length of $\sigma(\beta)$.

Notice that $\sigma(0) = \epsilon$.

Let $\langle U_\alpha \rangle_{\alpha<\mu}$ be the sequence of pairwise disjoint subsets of U with the following two properties.

1. For each limit ordinal $\alpha < \mu$, $U_\alpha = \emptyset$.
2. For each non-limit ordinal $\alpha < \mu$, U_α consists of all atoms from U with the situation argument $[\sigma(\alpha)]$.

Let $\langle A_\alpha \rangle_{\alpha<\mu}$ be the sequence of subsets of U such that for all $\alpha < \mu$

$$A_\alpha = \bigcup_{\gamma \leq \alpha} U_\gamma \, .$$

It is not difficult to verify that $\langle A_\alpha \rangle_{\alpha<\mu}$ is a splitting sequence for $\delta(D)$.

Now we can prove that default theory $\delta(D)$ is correct with respect to the initial situation S_0, which we can also write as $[\epsilon]$ and as $[\sigma(0)]$. We do this by showing that the default theory $b_{A_0}(\delta(D))$ behaves correctly, as follows.

Lemma B.6. A set X of formulas from $\mathcal{L}(U_0)$ is a consistent extension of $b_{A_0}(\delta(D))$ if and only if there is a state S such that

$$X = Cn_{U_0}[\{Reachable(S_0)\} \cup \Sigma(S, \epsilon)] \, .$$

PROOF. It is easy to verify that

$$b_{A_0}(\delta(D)) = \Sigma(\Delta_0, \epsilon) \cup \left\{ \frac{: Reachable(S_0)}{Reachable(S_0)} \right\} \, .$$

The lemma follows in a straightforward fashion from this observation, by Lemmas B.1 and B.5. □

We next show that default theory $\delta(D)$ uses the structure of the situation calculus to build what is essentially a tree of embeddings of the definition of *Res*.

For each $\alpha + 1 < \mu$, let

$$D_{\alpha+1} = b_{A_{\alpha+1}}(\delta(D)) \setminus b_{A_\alpha}(\delta(D)) \, .$$

So $D_{\alpha+1}$ is the default theory over $\mathcal{L}(A_{\alpha+1})$ which can be described as follows. Let $\overline{A}$ be the action string and A the action name such that $\sigma(\alpha + 1) = \overline{A}; A$. For each sufficiency proposition ϕ **suffices for** ψ in D, we have the rule

$$\frac{Holds(\phi, [\overline{A}; A]) \wedge Reachable([\overline{A}; A])}{Holds(\psi, [\overline{A}; A])} \, .$$

For each effect proposition A **causes** ϕ **if** ψ in D, we have the rule

$$\frac{Holds(\psi, [\overline{A}]) \wedge Reachable([\overline{A}; A])}{Holds(\phi, [\overline{A}; A])} .$$

For each influence proposition A **possibly changes** F **if** ψ in D, we have the rules

$$\frac{Holds(\psi, [\overline{A}]) \wedge Reachable([\overline{A}; A]) : Holds(F, [\overline{A}; A])}{Holds(F, [\overline{A}; A])}$$

and

$$\frac{Holds(\psi, [\overline{A}]) \wedge Reachable([\overline{A}; A]) : \neg Holds(F, [\overline{A}; A])}{\neg Holds(F, [\overline{A}; A])} .$$

For each executability proposition **impossible** A **if** ψ in D, we have the rule

$$\frac{Holds(\phi, [\overline{A}])}{\neg Reachable([\overline{A}; A])} .$$

We also have a number of additional rules, as specified below.
Reachability axioms.

$$\frac{: Reachable([\overline{A}; A])}{Reachable([\overline{A}; A])} \text{ and } \frac{\neg Reachable([\overline{A}])}{\neg Reachable([\overline{A}; A])} .$$

Inertia axioms. For each frame fluent literal L,

$$\frac{Holds(L, [\overline{A}]) \wedge Reachable([\overline{A}; A]) : Holds(L, [\overline{A}; A])}{Holds(L, [\overline{A}; A])} .$$

For any set Y of formulas from $\mathcal{L}(A_\alpha)$, let

$$E_{\alpha+1}(Y) = e_{A_\alpha}(D_{\alpha+1}, Y) .$$

Notice that $E_{\alpha+1}(Y)$ is a default theory over $\mathcal{L}(U_{\alpha+1})$.

Lemma B.7. Let α be such that $\alpha + 1 < \mu$. Let $\overline{A}$ be the action string and A the action name such that $\sigma(\alpha + 1) = \overline{A}; A$. Let γ be such that $\sigma(\gamma) = \overline{A}$. For any logically closed set Y of formulas from $\mathcal{L}(A_\alpha)$, we have

$$E_{\alpha+1}(Y) = E_{\alpha+1}(Y \cap \mathcal{L}(U_\gamma)) .$$

Proof. Follows easily from the fact that every constituent of every rule in $D_{\alpha+1}$ belongs to $\mathcal{L}(U_\gamma) \cup \mathcal{L}(U_{\alpha+1})$. □

Lemma B.8. Let α be such that $\alpha + 1 < \mu$. Let $\overline{A}$ be the action string and A the action name such that $\sigma(\alpha + 1) = \overline{A}; A$. Let γ be such that $\sigma(\gamma) = \overline{A}$. Let $Y = Cn_{U_\gamma}(\{\neg Reachable([\overline{A}])\})$. Let S be a state. Let $Z = Cn_{U_\gamma}[\{Reachable([\overline{A}])\} \cup \Sigma(S, \overline{A})]$. The following hold.

1. *The unique extension of $E_{\alpha+1}(Y)$ is $Cn_{U_{\alpha+1}}(\{\neg Reachable([\overline{A}; A])\})$.*
2. *If A is prohibited in S, then the unique extension of $E_{\alpha+1}(Z)$ is*

$$Cn_{U_{\alpha+1}}(\{\neg Reachable([\overline{A}; A])\}) .$$

3. *If A is not prohibited in S, then X is an extension of $E_{\alpha+1}(Z)$ if and only if there is a state $S' \in Res(A,S)$ such that*

$$X = Cn_{U_{\alpha+1}}[\{Reachable([\overline{A};A])\} \cup \Sigma(S', \overline{A}; A)] .$$

PROOF. For this lemma we will use the Splitting Set Theorem, with splitting set $B = \{Reachable([\overline{A};A])\}$. Notice that B splits both $E_{\alpha+1}(Y)$ and $E_{\alpha+1}(Z)$.

For the first part, it's not hard to verify that the unique extension of $b_B(E_{\alpha+1}(Y))$ is $Cn_B(\{\neg Reachable([\overline{A};A])\})$. Moreover, it is easy to verify that

$$e_B(E_{\alpha+1}(Y), Cn_B(\{\neg Reachable([\overline{A};A])\})) = \emptyset .$$

Thus

$$\langle\ Cn_B(\{\neg Reachable([\overline{A};A])\})\ ,\ Cn_{\Sigma(\mathbf{F},\overline{A};A)}(\emptyset)\ \rangle$$

is a solution to $E_{\alpha+1}(Y)$ with respect to B, and by the Splitting Set Theorem it follows that $Cn_{U_{\alpha+1}}(\{\neg Reachable([\overline{A};A])\})$ is the unique extension of $E_{\alpha+1}(Y)$.

For part two, assume that A is prohibited in S. Thus there is fluent formula ϕ such that the rule

$$\frac{Holds(\phi, [\overline{A}])}{\neg Reachable(\overline{A}; A)}$$

belongs to $D_{\alpha+1}$ and ϕ is satisfied in S. Since ϕ is satisfied in S, $\phi \in Cn_{\mathbf{F}}(S)$. Thus we have $Holds(\phi, [\overline{A}]) \in \Sigma(Cn_{\mathbf{F}}(S), \overline{A})$, and it follows by Lemma B.4 that $Holds(\phi, [\overline{A}]) \in Cn_{\Sigma(\mathbf{F},\overline{A})}(\Sigma(S, \overline{A}))$. And since $Cn_{\Sigma(\mathbf{F},\overline{A})}(\Sigma(S, \overline{A})) \subseteq Z$, we have $Holds(\phi, [\overline{A}]) \in Z$. It follows easily that the unique extension of $b_B(E_{\alpha+1}(Z))$ is $Cn_B(\{\neg Reachable([\overline{A};A])\})$. Again, it is easy to verify that

$$e_B(E_{\alpha+1}(Z), Cn_B(\{\neg Reachable([\overline{A};A])\})) = \emptyset .$$

Thus, by essentially the same reasoning as in the previous case, we can conclude that the unique extension of $E_{\alpha+1}(Z)$ is $Cn_{U_{\alpha+1}}(\{\neg Reachable([\overline{A};A])\})$.

For the third part, assume that A is not prohibited in S. Reasoning much as before, we see that unique extension of $b_B(E_{\alpha+1}(Z))$ is $Cn_B(B)$. Now take

$$D' = e_B(E_{\alpha+1}(Z), Cn_B(B)) .$$

The key step is to recognize that

$$D' = \Sigma(\Delta(A,S), \overline{A}; A) .$$

Thus, we can apply Lemma B.2 relating $Res(A,S)$ and $\Delta(A,S)$, as follows.

(Left-to-right) Assume that X is an extension of $E_{\alpha+1}(Z)$. By the Splitting Set Corollary, $\langle X \cap \mathcal{L}(B), X \cap \mathcal{L}(\Sigma(\mathbf{F}, \overline{A}; A))\rangle$ is a solution to $E_{\alpha+1}(Z)$ with respect to B, and it follows immediately that $X \cap \mathcal{L}(B)$ is a consistent extension of $b_B(E_{\alpha+1}(Z))$ and that $X \cap \mathcal{L}(\Sigma(\mathbf{F}, \overline{A}; A))$ is a consistent extension of D'. Let $X' = X \cap \mathcal{L}(\Sigma(\mathbf{F}, \overline{A}; A))$. Notice that $X = Cn_{U_{\alpha+1}}(B \cup X')$. Since $D' = \Sigma(\Delta(A,S), \overline{A}; A)$, we know by Lemmas B.2 and B.5 that there is a state $S' \in Res(A,S)$ such that $X' = Cn_{\Sigma(\mathbf{F},\overline{A};A)}[\Sigma(S', \overline{A}; A)]$. It follows that $X = Cn_{U_{\alpha+1}}(B \cup \Sigma(S', \overline{A}; A))$.

(Right-to-left) Assume there is a state $S' \in Res(A,S)$ such that $X = Cn_{U_{\alpha+1}}(B \cup \Sigma(S', \overline{A}; A))$. We know that $Cn_B(B)$ is a consistent extension of $b_B(E_{\alpha+1}(Z))$.

Let $X' = Cn_{\Sigma(\mathbf{F},\overline{A},A)}[\Sigma(S',\overline{A};A)]$. Since $D' = \Sigma(\Delta(A,S),\overline{A};A)$. we know by Lemmas B.2 and B.5 that X' is a consistent extension of D'. It follows by the Splitting Set Theorem that X is an extension of $E_{\alpha+1}(Z)$. □

At this point we have applied the fundamental lemmas (B.1 & B.2) to the parts of $\delta(D)$ that we obtain using the splitting sequence $\langle A_\alpha\rangle_{\alpha<\mu}$. The resulting lemmas (B.6 & B.8) capture essential properties of $\delta(D)$ in a form that will be convenient for our proof of the Correspondence Lemma.

In what follows, we will first show that for any model Ψ of D we can construct a corresponding extension of $\delta(D)$ (Lemma B.14). It will then remain to show that each consistent extension of $\delta(D)$ corresponds to a unique model of D (Lemma B.21). These results together will establish that there is indeed a one-to-one correspondence between models of D and consistent extensions of $\delta(D)$. Moreover, we'll show that each model and its corresponding extension agree on the truth of all value propositions, which will suffice to establish the Correspondence Lemma.

At this point we associate with each structure for D a unique set of literals from $\mathcal{L}(U)$. It will be our goal to show that, for any model of D, the associated set of literals is a consistent extension of $\delta(D)$.

For any structure Ψ for D, let $\delta(\Psi)$ be the least set of literals from $\mathcal{L}(U)$ such that for every action string $\overline{A}$:

1. if $\overline{A} \notin Dom(\Psi)$, then $\neg Reachable([\overline{A}]) \in \delta(\Psi)$; and
2. if $\overline{A} \in Dom(\Psi)$, then $Reachable([\overline{A}]) \in \delta(\Psi)$ and $\Sigma(\Psi(\overline{A}),\overline{A}) \subseteq \delta(\Psi)$.

In Lemma B.10 below, we will show that $\delta(\Psi)$ has the following crucial property: a value proposition V is true in Ψ if and only if $[V] \in Cn_U(\delta(\Psi))$.

Lemma B.9. Let Ψ be a structure for D. For all action strings $\overline{A}$, we have

$$Cn_U(\delta(\Psi)) \cap \mathcal{L}(\Sigma(\mathbf{F},\overline{A})) = Cn_{\Sigma(\mathbf{F},\overline{A})}(\delta(\Psi) \cap \mathcal{L}(\Sigma(\mathbf{F},\overline{A}))) .$$

Proof. Straightforward. □

Lemma B.10. A value proposition V is true in a structure Ψ for D if and only if $[V] \in Cn_U(\delta(\Psi))$.

Proof. Notice that it is sufficient to prove that the lemma holds for all atomic value propositions. So consider any value proposition ϕ **after** $\overline{A}$.

(Left-to-right) Assume that ϕ **after** $\overline{A}$ is true in Ψ. Thus, $\overline{A} \in Dom(\Psi)$ and ϕ is satisfied in $\Psi(\overline{A})$. Since ϕ is satisfied in $\Psi(\overline{A})$, we know that $\phi \in Cn_{\mathbf{F}}(\Psi(\overline{A}))$. Thus, $Holds(\phi,[\overline{A}]) \in \Sigma[Cn_{\mathbf{F}}(\Psi(\overline{A})),\overline{A}]$. We can conclude by Lemma B.4 that $Holds(\phi,[\overline{A}]) \in Cn_U(\Sigma(\Psi(\overline{A}),\overline{A}))$. By the definition of $\delta(\Psi)$, $Reachable([\overline{A}]) \in \delta(\Psi)$ and $\Sigma(\Psi(\overline{A}),\overline{A}) \subseteq \delta(\Psi)$. So we've shown that $Reachable([\overline{A}]) \wedge Holds(\phi,[\overline{A}]) \in Cn_U(\delta(\Psi))$. That is, $[\phi$ **after** $\overline{A}] \in Cn_U(\delta(\Psi))$.

(Right-to-left) Assume that $[\phi$ **after** $\overline{A}] \in Cn_U(\delta(\Psi))$. That is just to say that $Reachable([\overline{A}]) \wedge Holds(\phi,[\overline{A}]) \in Cn_U(\delta(\Psi))$. Thus $Holds(\phi,[\overline{A}]) \in Cn_U(\delta(\Psi))$ and $Reachable([\overline{A}]) \in Cn_U(\delta(\Psi))$. By the definition of $\delta(\Psi)$, it follows that $\overline{A} \in Dom(\Psi)$. Again by the definition of $\delta(\Psi)$, we can conclude that $\Sigma(\Psi(\overline{A}),\overline{A}) = \delta(\Psi) \cap \mathcal{L}(\Sigma(\mathbf{F},\overline{A}))$. Thus, $Cn_{\Sigma(\mathbf{F},\overline{A})}(\Sigma(\Psi(\overline{A}),\overline{A})) = Cn_{\Sigma(\mathbf{F},\overline{A})}(\delta(\Psi) \cap \mathcal{L}(\Sigma(\mathbf{F},\overline{A})))$. By Lemmas B.4 and B.9 it follows that

$$\Sigma[Cn_{\mathbf{F}}(\Psi(\overline{A})),\overline{A}] = Cn_U(\delta(\Psi)) \cap \mathcal{L}(\Sigma(\mathbf{F},\overline{A})) .$$

And since we know that $Holds(\phi, [\overline{A}]) \in Cn_U(\delta(\Psi)) \cap \mathcal{L}(\Sigma(\mathbf{F}, \overline{A}))$, it follows that $Holds(\phi, [\overline{A}]) \in \Sigma[Cn_{\mathbf{F}}(\Psi(\overline{A})), \overline{A}]$. So $\phi \in Cn_{\mathbf{F}}(\Psi(\overline{A}))$. That is, $\Psi(\overline{A})$ satisfies ϕ and thus ϕ **after** $\overline{A}$ is true in Ψ. □

Now we begin the main part of the proof of the left-to-right direction of the Correspondence Lemma.

Let Ψ be a model of D. We will show that $Cn_U(\delta(\Psi))$ is an extension of $\delta(D)$. We begin by putting $Cn_U(\delta(\Psi))$ in a form more suitable for application of the Splitting Sequence Theorem.

Let $\langle X_\alpha \rangle_{\alpha < \mu}$ be defined as follows.

1. $X_0 = Cn_{U_0}(\delta(\Psi) \cap \mathcal{L}(U_0))$.
2. For all α such that $\alpha + 1 < \mu$, $X_{\alpha+1} = Cn_{U_{\alpha+1}}(\delta(\Psi) \cap \mathcal{L}(U_{\alpha+1}))$.
3. For all limit ordinals $\alpha < \mu$, $X_\alpha = Cn_\emptyset(\emptyset)$.

Lemma B.11. We have

$$Cn_U(\delta(\Psi)) = Cn_U\left(\bigcup_{\alpha<\mu} X_\alpha\right).$$

Proof. It is straightforward to verify that

$$\delta(\Psi) = \bigcup_{\alpha<\mu} (\delta(\Psi) \cap \mathcal{L}(U_\alpha)).$$

It is clear from the definitions that for all $\alpha < \mu$, $X_\alpha = Cn_{U_\alpha}(\delta(\Psi) \cap \mathcal{L}(U_\alpha))$. The lemma follows easily from these observations. □

We will show that $\langle X_\alpha \rangle_{\alpha<\mu}$ is a solution to $\delta(D)$ with respect to $\langle A_\alpha \rangle_{\alpha<\mu}$.

Lemma B.12. Let α be a non-limit ordinal such that $\alpha < \mu$. Let $\overline{A}$ be the action string such that $\overline{A} = \sigma(\alpha)$. The following hold.

1. *If $\overline{A} \in Dom(\Psi)$ then $X_\alpha = Cn_{U_\alpha}[\{Reachable([\overline{A}])\} \cup \Sigma(\Psi(\overline{A}), \overline{A})]$.*
2. *If $\overline{A} \notin Dom(\Psi)$ then $X_\alpha = Cn_{U_\alpha}(\{\neg Reachable([\overline{A}])\})$.*

Proof. By the definition of $\langle X_\alpha \rangle_{\alpha<\mu}$ we know that $X_\alpha = Cn_{U_\alpha}(\delta(\Psi) \cap \mathcal{L}(U_\alpha))$. For part one, assume that $\overline{A} \in Dom(\Psi)$. From the definition of $\delta(\Psi)$ we can conclude that $\delta(\Psi) \cap \mathcal{L}(U_\alpha) = \{Reachable([\overline{A}])\} \cup \Sigma(\Psi(\overline{A}), \overline{A})$. For part two, assume that $\overline{A} \notin Dom(\Psi)$. From the definition of $\delta(\Psi)$ we can conclude that $\delta(\Psi) \cap \mathcal{L}(U_\alpha) = \{\neg Reachable([\overline{A}])\}$. □

Lemma B.13. For each α such that $\alpha + 1 < \mu$, $X_{\alpha+1}$ is a consistent extension of the default theory

$$E_{\alpha+1}\left(\bigcup_{\beta\leq\alpha} X_\beta\right)$$

over $\mathcal{L}(U_{\alpha+1})$.

PROOF. Let $\overline{A}$ be the action string and A the action name such that $\sigma(\alpha{+}1) = \overline{A}; A$. Let γ be such that $\sigma(\gamma) = \overline{A}$. By the definition of $E_{\alpha+1}$ we know that

$$E_{\alpha+1}\left(\bigcup_{\beta\leq\alpha} X_\beta\right) = E_{\alpha+1}\left(Cn_{\boldsymbol{A}_\alpha}\left(\bigcup_{\beta\leq\alpha} X_\beta\right)\right) .$$

It is easy to verify that

$$Cn_{\boldsymbol{A}_\alpha}\left(\bigcup_{\beta\leq\alpha} X_\beta\right) \cap \mathcal{L}(U_\gamma) \;=\; X_\gamma .$$

Thus, by Lemma B.7 we can conclude that

$$E_{\alpha+1}\left(\bigcup_{\beta\leq\alpha} X_\beta\right) = E_{\alpha+1}(X_\gamma) .$$

So we will show that $X_{\alpha+1}$ is an extension of $E_{\alpha+1}(X_\gamma)$. Consider three cases.

Case 1: $\overline{A}; A \in Dom(\Psi)$. Since the domain of Ψ is prefix-closed, $\overline{A} \in Dom(\Psi)$. Let $S = \Psi(\overline{A})$ and $S' = \Psi(\overline{A}; A)$. By the previous lemma we have the following.

$$X_\gamma = Cn_{U_\gamma}[\{Reachable([\overline{A}])\} \cup \Sigma(S, \overline{A})]$$
$$X_{\alpha+1} = Cn_{U_{\alpha+1}}[\{Reachable([\overline{A}; A])\} \cup \Sigma(S', \overline{A}; A)]$$

Since $\overline{A}; A \in Dom(\Psi)$, we know that A is not prohibited in S. Furthermore, since Ψ is a model of D, we have $S' \in Res(A, S)$. Thus, by part three of Lemma B.8, $X_{\alpha+1}$ is a consistent extension of $E_{\alpha+1}(X_\gamma)$.

Case 2: $\overline{A}; A \notin Dom(\Psi)$ and $\overline{A} \in Dom(\Psi)$. Let $S = \Psi(\overline{A})$. In this case, X_γ is the same as in the previous case. By the previous lemma

$$X_{\alpha+1} = Cn_{U_{\alpha+1}}(\{\neg Reachable([\overline{A}; A])\}) .$$

Since D is a qualification-free domain, and $\overline{A}; A \notin Dom(\Psi)$ while $\overline{A} \in Dom(\Psi)$, we can conclude that A is prohibited in S. Thus, by part two of Lemma B.8, $X_{\alpha+1}$ is a consistent extension of $E_{\alpha+1}(X_\gamma)$.

Case 3: $\overline{A}; A \notin Dom(\Psi)$ and $\overline{A} \notin Dom(\Psi)$. In this case, $X_{\alpha+1}$ is the same as in the previous case. By the previous lemma

$$X_\gamma = Cn_{U_\gamma}(\{\neg Reachable([\overline{A}])\}) .$$

By part one of Lemma B.8, $X_{\alpha+1}$ is a consistent extension of $E_{\alpha+1}(X_\gamma)$. □

Lemma B.14. Let D be a qualification-free domain theory without value propositions. If Ψ is a model of D, then $Cn_U(\delta(\Psi))$ is a consistent extension of $\delta(D)$.

PROOF. First, the fact that X_0 is a consistent extension of $b_{\boldsymbol{A}_0}(\delta(D))$ follows easily from Lemma B.6. Second, the previous lemma shows that for each α such that $\alpha + 1 < \mu$, $X_{\alpha+1}$ is a consistent extension of

$$E_{\alpha+1}\left(\bigcup_{\gamma\leq\alpha} X_\gamma\right) .$$

Finally, by the definition of $\langle X_\alpha \rangle_{\alpha<\mu}$, we know that for all limit ordinals $\alpha < \mu$, $X_\alpha = Cn_\emptyset(\emptyset)$. These observations are sufficient to establish that $\langle X_\alpha \rangle_{\alpha<\mu}$ is a solution to $\delta(D)$ with respect to $\langle A_\alpha \rangle_{\alpha<\mu}$. By Lemma B.11

$$Cn_U(\delta(\Psi)) = Cn_U\left(\bigcup_{\alpha<\mu} X_\alpha\right)$$

so we can conclude by the Splitting Sequence Theorem that $Cn_U(\delta(\Psi))$ is a consistent extension of $\delta(D)$. □

We have essentially established the left-to-right direction of the Correspondence Lemma. Now we turn our attention to the other direction.

Let X be a consistent extension of $\delta(D)$. We will show that there is a (unique) model Ψ of D such that $X = Cn_U(\delta(\Psi))$ (Lemma B.21). To this end, we specify a construction that, as we will show, yields a unique model of D for each consistent extension of $\delta(D)$.

Let Ψ_X be the partial function from actions strings to sets of fluent literals that satisfies the following two conditions.

1. $Dom(\Psi_X) = \{\overline{A} : Reachable([\overline{A}]) \in X\}$.
2. For all non-limit ordinals $\alpha < \mu$, if $\sigma(\alpha) \in Dom(\Psi_X)$, then $\Psi_X(\sigma(\alpha))$ is the greatest set of fluent literals such that $\Sigma(\Psi_X(\sigma(\alpha)), \sigma(\alpha)) \subseteq X \cap \mathcal{L}(U_\alpha)$.

Thus, for every action string $\overline{A} \in Dom(\Psi_X)$, $\Psi_X(\overline{A})$ is the greatest set S of fluent literals such that $\Sigma(S, \overline{A})$ is a subset of X. One thing we will show is that such sets S are in fact states (Lemma B.17). More generally, we will establish the fact that Ψ_X is a structure for D such that $X = Cn_U(\delta(\Psi_X))$ (Lemma B.18).

Lemma B.15. The domain of Ψ_X is nonempty and prefix-closed.

Proof. By Lemma B.6 (and the Splitting Sequence Corollary), we can conclude that $Reachable(S_0) \in X$. Thus the domain of Ψ_X is nonempty.

For every action string $\overline{A}$, $\delta(D)$ includes the rule

$$\frac{: Reachable([\overline{A}])}{Reachable([\overline{A}])}$$

from which it follows that for every action string $\overline{A}$, either $Reachable([\overline{A}]) \in X$ or $\neg Reachable([\overline{A}]) \in X$. Furthermore, for every action string $\overline{A}$ and action name A, $\delta(D)$ includes the rule

$$\frac{\neg Reachable([\overline{A}])}{\neg Reachable([\overline{A}; A])}$$

which guarantees that if $\neg Reachable([\overline{A}]) \in X$ then $\neg Reachable([\overline{A}; A]) \in X$. Thus we can conclude that if $\overline{A} \notin Dom(\Psi_X)$ then $\overline{A}; A \notin Dom(\Psi_X)$, which is just to say that the domain of Ψ_X is prefix-closed. □

Once again we will be looking to apply Lemmas B.6 and B.8. In order to do this, we need an appropriate sequence, which is defined next.

Let $\langle X_\alpha \rangle_{\alpha<\mu}$ be the sequence such that for every $\alpha < \mu$, $X_\alpha = X \cap \mathcal{L}(U_\alpha)$. We know by the Splitting Sequence Corollary that $\langle X_\alpha \rangle_{\alpha<\mu}$ is a solution to $\delta(D)$ with

respect to $\langle A_\alpha \rangle_{\alpha<\mu}$. Moreover, it is not hard to verify that

$$X = Cn_U\left(\bigcup_{\alpha<\mu} X_\alpha\right) .$$

Lemma B.16. Let α be such that $\alpha+1<\mu$. Let $\overline{A}$ be the action string and A the action name such that $\sigma(\alpha+1) = \overline{A}; A$.

1. *If $Reachable([\overline{A}; A]) \in X_{\alpha+1}$, then there is a state S such that*

$$X_{\alpha+1} = Cn_{U_{\alpha+1}}[\{Reachable([\overline{A}; A])\} \cup \Sigma(S, \overline{A}; A)] .$$

2. *If $Reachable(\overline{A}; A]) \notin X_{\alpha+1}$, then*

$$X_{\alpha+1} = Cn_{U_{\alpha+1}}(\{\neg Reachable([\overline{A}; A])\}) .$$

Proof. Proof is by induction on α. Let γ be such that $\sigma(\gamma) = \overline{A}$. By the definition of $E_{\alpha+1}$ we know that

$$E_{\alpha+1}\left(\bigcup_{\beta\leq\alpha} X_\beta\right) = E_{\alpha+1}\left(Cn_{\boldsymbol{A}_\alpha}\left(\bigcup_{\beta\leq\alpha} X_\beta\right)\right) .$$

It is not difficult to verify that

$$Cn_{\boldsymbol{A}_\alpha}\left(\bigcup_{\beta\leq\alpha} X_\beta\right) \cap \mathcal{L}(U_\gamma) = X_\gamma .$$

Thus, by Lemma B.7 we can conclude that

$$E_{\alpha+1}\left(\bigcup_{\beta\leq\alpha} X_\beta\right) = E_{\alpha+1}(X_\gamma) .$$

So $X_{\alpha+1}$ is an extension of $E_{\alpha+1}(X_\gamma)$. Now consider three cases.

Case 1: $\gamma = 0$. Thus $\overline{A} = \epsilon$. By Lemma B.6 there is a state S such that

$$X_\gamma = Cn_{U_\gamma}[\{Reachable([\overline{A}])\} \cup \Sigma(S, \overline{A})] .$$

To show part one for this case, assume that $Reachable([\overline{A}; A]) \in X_{\alpha+1}$. By part two of Lemma B.8 we can conclude that A is not prohibited in S and the desired conclusion then follows from part three of Lemma B.8. Part two for this case can be proved similarly.

Case 2: $\gamma \neq 0$ and $Reachable([\overline{A}]) \in X_\gamma$. In this case we use the inductive hypothesis, which guarantees that there is a state S such that

$$X_\gamma = Cn_{U_\gamma}[\{Reachable([\overline{A}])\} \cup \Sigma(S, \overline{A})] .$$

From this point the proof proceeds as in the previous case.

Case 3: $\gamma \neq 0$ and $Reachable([\overline{A}]) \notin X_\gamma$. In this case we again use the inductive hypothesis, which guarantees that $X_\gamma = Cn_{U_\gamma}(\{\neg Reachable([\overline{A}])\})$. We reach the desired conclusion by applying part one of Lemma B.8. □

Lemma B.17. Ψ_X *is a partial function from action strings to states.*

PROOF. We show that for all non-limit ordinals $\alpha < \mu$, if $\sigma(\alpha) \in Dom(\Psi_X)$, then $\Psi_X(\sigma(\alpha))$ is a state. First, if $\alpha = 0$, we can conclude by Lemma B.6 (and the Splitting Sequence Corollary), along with the definition of Ψ_X, that $\sigma(\alpha) \in Dom(\Psi_X)$ and that $\Psi_X(\sigma(\alpha))$ is a state. Let α be such that $\alpha + 1 < \mu$. If $\sigma(\alpha+1) \in Dom(\Psi_X)$, then $Reachable([\sigma(\alpha+1)]) \in X_{\alpha+1}$, and we can conclude by the previous lemma that $\Psi_X(\sigma(\alpha+1))$ is a state. □

Lemma B.18. Ψ_X *is a structure for* D *such that* $X = Cn_U(\delta(\Psi_X))$.

PROOF. By Lemma B.15 and the previous lemma, Ψ_X is partial function from action strings to states whose domain is nonempty and prefix-closed, which shows that Ψ_X is a structure for D. Thus, $\delta(\Psi_X)$ is defined. We must show that $X = Cn_U(\delta(\Psi_X))$. To begin, it is easy to verify that

$$Cn_U(\delta(\Psi_X)) = Cn_U\left(\bigcup_{\alpha<\mu} Cn_{U_\alpha}(\delta(\Psi_X) \cap \mathcal{L}(U_\alpha))\right) .$$

Recall that

$$X = Cn_U\left(\bigcup_{\alpha<\mu} X_\alpha\right) .$$

Given these observations, we see that it will be sufficient to show that for every $\alpha < \mu$, $X_\alpha = Cn_{U_\alpha}(\delta(\Psi_X) \cap \mathcal{L}(U_\alpha))$.

If α is a limit ordinal, this is trivially true; so assume that α is a non-limit ordinal and let $\overline{A}$ be the action string such that $\overline{A} = \sigma(\alpha)$. Now consider two cases.

Case 1: $\overline{A} \notin Dom(\Psi_X)$. In this case, by the definition of δ, we have $\delta(\Psi_X) \cap \mathcal{L}(U_\alpha) = \{\neg Reachable([\overline{A}])\}$. Similarly, by the definition of Ψ_X, we know that $Reachable([\overline{A}]) \notin X_\alpha$. It follows by Lemma B.6 that $\alpha \neq 0$. Thus, by part two of Lemma B.16 we can conclude that $X_\alpha = Cn_{U_\alpha}(\delta(\Psi_X) \cap \mathcal{L}(U_\alpha))$.

Case 2: $\overline{A} \in Dom(\Psi_X)$. By the definition of Ψ_X, we have $Reachable([\overline{A}]) \in X_\alpha$. Now, if $\overline{A} = \epsilon$ we know by Lemma B.6 that there is a state S such that

$$X_\alpha = Cn_{U_\alpha}(\{Reachable([\overline{A}])\} \cup \Sigma(S, \overline{A})) .$$

On the other hand, if $\overline{A} \neq \epsilon$, the same thing follows from part one of Lemma B.16. By the definition of Ψ_X we know that $\Sigma(\Psi_X(\overline{A}), \overline{A}) \subseteq X_\alpha$. Since X_α is consistent, we can conclude that $\Psi_X(\overline{A}) = S$. It follows by the definition of δ that

$$\delta(\Psi_X) \cap \mathcal{L}(U_\alpha) = \{Reachable([\overline{A}])\} \cup \Sigma(S, \overline{A}) .$$

Thus $X_\alpha = Cn_{U\alpha}(\delta(\Psi_X) \cap \mathcal{L}(U_\alpha))$. □

Now that we know Ψ_X is a structure for D, we'll need just two more lemmas in order to establish that Ψ_X is in fact a model for D (Lemma B.21).

Lemma B.19. For all $\overline{A} \in Dom(\Psi_X)$ *and all action names* A, *if* $Res(A, \Psi_X(\overline{A}))$ *is nonempty, then* $\overline{A}; A \in Dom(\Psi_X)$.

PROOF. Let $S = \Psi_X(\overline{A})$. By Lemma B.17, S is a state. Let α and γ be such that $\sigma(\alpha+1) = \overline{A}; A$ and $\sigma(\gamma) = \overline{A}$. By the construction of Ψ_X from X, we know

that $\Sigma(S, \overline{A}) \subseteq X_\gamma$, and also that $Reachable([\overline{A}]) \in X_\gamma$. If $\gamma = 0$ it follows by Lemma B.6 that

$$X_\gamma = Cn_{U_\gamma}[\{Reachable([\overline{A}])\} \cup \Sigma(S, \overline{A})]$$

since X_γ is consistent. On the other hand, if $\gamma \neq 0$, the same thing follows from part one of Lemma B.16. Since $Res(A, S)$ is nonempty, we know that A is not prohibited in S. It follows by part three of Lemma B.8 that $Reachable([\overline{A}; A]) \in X_{\alpha+1}$. Thus, by the construction of Ψ_X from X, $\overline{A}; A \in Dom(\Psi_X)$. □

Lemma B.20. For all $\overline{A}; A \in Dom(\Psi_X)$, $\Psi_X(\overline{A}; A) \in Res(A, \Psi_X(\overline{A}))$.

Proof. By Lemma B.15, $\overline{A} \in Dom(\Psi_X)$, since $\overline{A}; A$ is. Let $S = \Psi_X(\overline{A})$ and $S' = \Psi_X(\overline{A}; A)$. Let α and γ be such that $\sigma(\alpha + 1) = \overline{A}; A$ and $\sigma(\gamma) = \overline{A}$. By Lemma B.17, S and S' are states. By essentially the same reasoning used in the proof of the previous lemma, we can show each of the following.

$$X_\gamma = Cn_{U_\gamma}[\{Reachable([\overline{A}])\} \cup \Sigma(S, \overline{A})]$$
$$X_{\alpha+1} = Cn_{U_{\alpha+1}}[\{Reachable([\overline{A}; A])\} \cup \Sigma(S', \overline{A}; A)]$$

It follows from part two of Lemma B.8 that A is not prohibited in S. We can conclude by part three of Lemma B.8 that $S' \in Res(A, S)$. That is, $\Psi_X(\overline{A}; A) \in Res(A, \Psi_X(\overline{A}))$. □

Lemma B.21. Let D be a qualification-free domain theory without value propositions. If X is a consistent extension of $\delta(D)$, then Ψ_X is a (unique) model of D such that $X = Cn_U(\delta(\Psi_X))$.

Proof. By Lemma B.18, Ψ_X is a structure for D such that $X = Cn_U(\delta(\Psi_X))$. Moreover, it is clear that Ψ_X is the only such structure. Lemma B.17 shows that $\Psi_X(\epsilon)$ is a state. Given this, Lemmas B.19 and B.20 establish the fact that Ψ_X is a model of D. □

Lemma B.22. **(Correspondence Lemma)** *Let D be a qualification-free $\mathcal{AC}$ domain description without value propositions. There is a one-to-one correspondence between models of D and consistent extensions of $\delta(D)$ such that a value proposition V is true in a model of D if and only if the formula $[V]$ belongs to the corresponding extension of $\delta(D)$.*

Proof. We have defined a total function from models Ψ of D to consistent extensions $Cn_U(\delta(\Psi))$ of $\delta(D)$ (Lemma B.14). Notice that this function is injective. To see that it is also surjective, notice that we have also defined a total function from consistent extensions X of $\delta(D)$ to models Ψ_X of D such that $X = Cn_U(\delta(\Psi_X))$ (Lemma B.21). Thus δ can be used to define a one-to-one correspondence between models of D and consistent extensions of $\delta(D)$. Finally, we've shown that a value proposition V is true in a model Ψ of D if and only if the formula $[V]$ belongs to the corresponding extension $Cn_U(\delta(\Psi))$ of $\delta(D)$ (Lemma B.10). □

We require one more lemma for the proof of the Correspondence Theorem.

Lemma B.23. Let D be a qualification-free domain description. Let D' be the domain description obtained by deleting all value propositions from D. Let E be a consistent set of formulas from $\mathcal{L}(U)$. $E = Cn_U(\delta(D)^E)$ if and only if $E = Cn_U(\delta(D')^E)$ and for every value proposition $V \in D$, $[V] \in E$.

PROOF. (Left-to-right) Assume that $E = Cn_U(\,\delta(D)^E\,)$. It is clear that $E = Cn_U(\,\delta(D')^E\,)$, since every rule in $\delta(D) \setminus \delta(D')$ has the form $\frac{\phi}{False}$. By the Correspondence Lemma, there is a model Ψ of D' such that for every value proposition V, V is true in Ψ if and only if $[V] \in E$. Consider any value proposition $V \in D$. Since $\frac{\neg[V]}{False} \in \delta(D)$, we know that $\neg[V] \notin E$. It follows that $\neg V$ is not true in Ψ, and thus that V is true in Ψ. So we can conclude that $[V] \in E$.

Proof in the other direction is similar, but slightly simpler. □

PROOF OF CORRESPONDENCE THEOREM. Let D be a qualification-free domain description. Let D' be the domain description obtained by deleting all value propositions from D. By the Correspondence Lemma, we know that there is a one-to-one correspondence between models of D' and consistent extensions of $\delta(D')$ such that a value proposition V is true in a model of D' if and only if the formula $[V]$ belongs to the corresponding extension of $\delta(D')$.

Let C be the set of pairs $\langle \Psi, E \rangle$ such that Ψ is a model of D' and E is the corresponding extension of $\delta(D')$. Since every model of D is a model of D' and similarly every consistent extension of $\delta(D)$ is a consistent extension of $\delta(D')$, we can complete our proof by showing that, for each pair $\langle \Psi, E \rangle$ that belongs to C, Ψ is a model of D if and only if E is a consistent extension of $\delta(D)$. So consider any $\langle \Psi, E \rangle \in C$.

(Left-to-right) Assume that Ψ is a model of D. We can conclude by the Correspondence Lemma that for every value proposition $V \in D$, $[V] \in E$. Since E is a consistent extension of $\delta(D')$, it follows by the previous lemma that E is a consistent extension of $\delta(D)$.

(Right-to-left) Assume that E is a consistent extension of $\delta(D)$. It follows by the previous lemma that E is a consistent extension of $\delta(D')$ such that for every value proposition $V \in D$, $[V] \in E$. We can conclude by the Correspondence Lemma that Ψ is a model of D' that satisfies every value proposition in D. That is, Ψ is a model of D. □

PROOF OF REACHABILITY COROLLARY. Let A be the set of all *Reachable* atoms in U. Since D includes no executability propositions, A is a splitting set for $\delta(D)$. Furthermore, it is clear that $Cn_A(A)$ is the unique extension of $b_A(\delta(D))$. Notice that

$$\delta'(D) = e_A(\delta(D), Cn_A(A))\,.$$

It follows by the Splitting Set Theorem that there is a one-to-one correspondence between the consistent extensions of $\delta(D)$ and the consistent extensions of $\delta'(D)$ such that for every value proposition V, $[V]$ belongs to a consistent extension of $\delta(D)$ if and only if $[\![V]\!]$ belongs to the corresponding extension of $\delta'(D)$. Given this fact, the corollary follows immediately from the Correspondence Theorem. □

B.2. Proof of LP Correspondence Theorem, LP Reachability Corollary, and Vivid Domains Theorem

Proof of the LP Correspondence Theorem is based on the following lemma, which is shown to follow from the Correspondence Theorem for default logic.

Lemma B.24. **(LP Correspondence Lemma)** *Let D be a simple, qualification-free $\mathcal{AC}$ domain description without value propositions. There is a one-to-one*

correspondence between models of D and consistent answer sets of $\pi(D)$ such that, for every model Ψ of D and corresponding answer set X, a simple value proposition $V_1 \vee \cdots \vee V_m \vee \neg V_{m+1} \vee \cdots \vee \neg V_n$ is true in Ψ if and only if at least one of the sets $\{ [\![V_1]\!], \ldots, [\![V_m]\!] \} \cap X$ and $\{ [\![V_{m+1}]\!], \ldots, [\![V_n]\!] \} \setminus X$ is nonempty.

PROOF. To begin, because D includes no value propositions, it is straightforward to determine that the default theories $dt(\pi(D))$ and $\delta(D)$ have precisely the same extensions. By the Correspondence Theorem, we know there is a one-to-one correspondence between models of D and consistent extensions of $\delta(D)$ such that, for every model Ψ of D and corresponding extension E, a value proposition V is true in Ψ if and only if $[V] \in E$. Let C be the set of pairs $\langle \Psi, E \rangle$ such that Ψ is a model of D and E is the corresponding consistent extension of $\delta(D)$. Since $\delta(D)$ and $dt(\pi(D))$ have the same extensions, we know that the set $\{ E \cap Lit(U) : E \textit{ is a consistent extension of } \delta(D) \}$ is precisely the set of consistent answer sets for $\pi(D)$. Take $A = \{ \langle \Psi, E \cap Lit(U) \rangle : \langle \Psi, E \rangle \in C \}$.

Consider any $\langle \Psi, E \rangle \in C$, along with the corresponding pair $\langle \Psi, X \rangle \in A$. It is clear from the construction of $\pi(D)$ that for any simple atomic value proposition L **after** $\overline{A}$, $[\![L \textbf{ after } \overline{A}]\!] \in X$ if and only if both $[\![L \textbf{ after } \overline{A}]\!] \in X$ and $\mathit{Reachable}([\overline{A}]) \in X$. We can conclude that $[\![L \textbf{ after } \overline{A}]\!] \in X$ if and only if $[L \textbf{ after } \overline{A}] \in E$. It follows by the Correspondence Theorem that for any simple atomic value proposition V, V is true in Ψ if and only if $[\![V]\!] \in X$.

Assume that $\langle \Psi, X \rangle$ and $\langle \Psi', X' \rangle$ are distinct members of A. It follows that Ψ and Ψ' are distinct, and so must differ on the truth of some simple atomic value proposition V. We can conclude that exactly one of the two sets X, X' includes $[\![V]\!]$. Thus $X \neq X'$. This shows that A captures a one-to-one correspondence between models of D and consistent answer sets of $\pi(D)$.

Now consider any simple value proposition $V = V_1 \vee \cdots \vee V_m \vee \neg V_{m+1} \vee \cdots \vee \neg V_n$ and any $\langle \Psi, X \rangle \in A$. We know that V is true in Ψ if and only if at least one of $V_1, \ldots, V_m$ is true in Ψ or at least one of $V_{m+1}, \ldots, V_n$ is not true in Ψ. Since each of $V_1, \ldots, V_m$ is a simple atomic value proposition, we can conclude that at least one of $V_1, \ldots, V_m$ is true in Ψ if and only if $\{ [\![V_1]\!], \ldots, [\![V_m]\!] \} \cap X$ is nonempty. Similarly, since each of $V_{m+1}, \ldots, V_n$ is a simple atomic value proposition, we can conclude that at least one of $V_{m+1}, \ldots, V_n$ is not true in Ψ iff $\{ [\![V_{m+1}]\!], \ldots, [\![V_n]\!] \} \setminus X$ is nonempty. Summing up, we have shown that $V_1 \vee \cdots \vee V_m \vee \neg V_{m+1} \vee \cdots \vee \neg V_n$ is true in Ψ iff at least one of the sets $\{ [\![V_1]\!], \ldots, [\![V_m]\!] \} \cap X$ and $\{ [\![V_{m+1}]\!], \ldots, [\![V_n]\!] \} \setminus X$ is nonempty. □

PROOF OF LP CORRESPONDENCE THEOREM. Let D be a simple, qualification-free domain description. Let D' be the domain description obtained from D by deleting all value propositions. By the LP Correspondence Lemma, we know that there is a one-to-one correspondence between models of D' and consistent answer sets of $\pi(D')$ such that, for every model Ψ of D' and corresponding answer set X, a simple value proposition $V_1 \vee \cdots \vee V_m \vee \neg V_{m+1} \vee \cdots \vee \neg V_n$ is true in Ψ if and only if either $\{ [\![V_1]\!], \ldots, [\![V_m]\!] \} \cap X \neq \emptyset$ or $\{ [\![V_{m+1}]\!], \ldots, [\![V_n]\!] \} \setminus X \neq \emptyset$.

Notice that $\pi(D) \setminus \pi(D')$ consists of all rules of the form

$$\mathit{False} \leftarrow [\![V_{m+1}]\!], \ldots, [\![V_n]\!], \mathit{not}\, [\![V_1]\!], \ldots, \mathit{not}\, [\![V_m]\!]$$

where $V_1 \vee \cdots \vee V_m \vee \neg V_{m+1} \vee \cdots \vee \neg V_n$ is a (simple) value proposition in D. It is a consequence of this observation that, for any set X, X is an answer set for $\pi(D)$ if and only if X is an answer set for $\pi(D')$ such that, for every value

proposition $V_1 \vee \cdots \vee V_m \vee \neg V_{m+1} \vee \cdots \vee \neg V_n$ in D, either $\{ [\![V_1]\!], \ldots, [\![V_m]\!] \} \cap X \neq \emptyset$ or $\{ [\![V_{m+1}]\!], \ldots, [\![V_n]\!] \} \setminus X \neq \emptyset$. We will rely on this result below.

Let A be the set of pairs $\langle \Psi, X \rangle$ such that Ψ is a model of D' and X is the corresponding answer set for $\pi(D')$. Since every model of D is a model of D' and similarly every consistent answer set for $\pi(D)$ is a consistent answer set for $\pi(D')$, we can complete our proof by showing that, for each pair $\langle \Psi, X \rangle$ that belongs to A, Ψ is a model of D if and only if X is a consistent answer set for $\pi(D)$. So consider any $\langle \Psi, X \rangle \in A$.

(Left-to-right) Assume that Ψ is a model of D. Thus, every value proposition in D is true in Ψ. It follows by the LP Correspondence Lemma that for every value proposition $V_1 \vee \cdots \vee V_m \vee \neg V_{m+1} \vee \cdots \vee \neg V_n$ in D, either $\{ [\![V_1]\!], \ldots, [\![V_m]\!] \} \cap X \neq \emptyset$ or $\{ [\![V_{m+1}]\!], \ldots, [\![V_n]\!] \} \setminus X \neq \emptyset$. We can conclude that X is an answer set for $\pi(D)$.

Proof in the other direction is similar. □

Next we sketch a proof of the LP Reachability Corollary. A complete proof would use the Splitting Set Theorem for logic programs [LT94].

Proof of LP Reachability Corollary (Sketch).
Let $B = \{Reachable([\overline{A}]) : \overline{A}$ *is an action string*$\}$. Since D is qualification-free, we can show that for every consistent answer set X for $\pi(D)$, $X \cap Lit(B) = B$. Given this, show that for any subset X of $Lit(U \setminus B)$, $X \cup B$ is a consistent answer set for $\pi(D)$ if and only if X is a consistent answer set for $\pi'(D)$. Finally, since every action string $\overline{A}$ belongs to $Dom(\Psi)$, we know that for every simple atomic value proposition V, V is true in Ψ if and only if $\neg V$ is not true in Ψ. We can conclude that $\{ [\![V_1]\!], \ldots, [\![V_m]\!], \overline{[\![V_{m+1}]\!]}, \ldots, \overline{[\![V_n]\!]} \} \cap X \neq \emptyset$ if and only if either $\{ [\![V_1]\!], \ldots, [\![V_m]\!] \} \cap X \neq \emptyset$ or $\{ [\![V_{m+1}]\!], \ldots, [\![V_n]\!] \} \setminus X \neq \emptyset$. □

Finally we begin the proof of the Vivid Domains Theorem.

For any domain description D, let

$$Rules(D) = \left\{ \frac{\phi}{\psi} : \phi \textbf{ suffices for } \psi \in D \right\} .$$

We will need the following lemma.

Lemma B.25. **(Vivid Domains Lemma)** *Let D be a vivid $\mathcal{AC}$ domain description. For any set X of fluent literals,*

$$Cn(X \cup Rules(D)) = Cn(X \cup Rules(Simple(D))) .$$

Proof. It is easy to verify that for any set X of fluent formulas we have

$$Cn(X \cup Rules(Simple(D))) \subseteq Cn(X \cup Rules(D)) .$$

For the other direction, let X be a set of fluent literals, and take

$$Y = Cn(X \cup Rules(Simple(D))) .$$

We will show that Y is closed under $X \cup Rules(D)$. We begin by observing that because X is a set of fluent literals and every rule in $Rules(Simple(D))$ has either the form $\frac{\phi}{L}$ where L is a fluent literal or the form $\frac{\phi}{False}$, we can conclude that there is a set Y' of fluent literals such that $Y = Cn(Y')$. It's clear that Y is closed under

X, so consider any rule $\frac{\phi}{\psi} \in Rules(D)$ such that $\phi \in Y$. We must show that ψ also belongs to Y. Since $\phi \in Y$ and $Y = Cn(Y')$ for some set Y' of fluent literals, we can conclude that some disjunct ϕ' of $DNF(\phi)$ belongs to Y. Since ϕ **suffices for** ψ is vivid, we know that ψ is either a nonempty conjunction of fluent literals or the formula *False*. If ψ is *False*, then Y is inconsistent and we're done; so assume that ψ is a nonempty conjunction of fluent literals. Consider any conjunct L of ψ. The rule ϕ' **suffices for** L belongs to $Simple(D)$, and since $\phi' \in Y$, we have $L \in Y$. We can conclude that each conjunct of ψ belongs to Y; and since Y is logically closed, we have $\psi \in Y$. □

Notice that the preceding lemma establishes the fact that domain descriptions D and $Simple(D)$ have the same set of states.

Proof of Vivid Domains Theorem. We have already observed that, for any structure Ψ, all of the value propositions in D are true in Ψ if and only all of the value propositions in $Simple(D)$ are true in Ψ. By the Vivid Domains Lemma, we know that domains D and $Simple(D)$ have the same set of states. Consider any action A and state S. We complete the first part of the proof by showing that a state S' may result from doing A in S in domain D if and only if S' may result from doing A in S in domain $Simple(D)$.

It is easy to verify that A is prohibited in S in domain D if and only if A is prohibited in S in domain $Simple(D)$. If A is prohibited in S in the two domains, we're done. So assume otherwise. It is also easy to see that the two domains agree on the set $F(A,S)$. Furthermore, although the two domains may not agree precisely on the set $E(A,S)$, it is clear that they do agree on $Cn(E(A,S))$. It follows that E' is an explicit effect of A in S in domain $Simple(D)$ if and only if there is an explicit effect E of A in S in domain D such that $E' = Cn(E) \cap Lit(\mathbf{F})$. Moreover, for any such E and E', it is clear that, for any set Γ of inference rules, $Cn(\Gamma \cup E) = Cn(\Gamma \cup E')$.

(Left-to-right) Assume that S' may result from doing A in S in domain D. So there is an explicit effect E of A in S in domain D such that

$$Cn(S') = Cn[\,(\,S \cap S' \cap Lit(\mathbf{F}_f)\,) \cup E \cup Rules(D)\,]\,.$$

We have already observed that there must be an explicit effect E' of A in S in domain $Simple(D)$ such that $E' = Cn(E) \cap Lit(\mathbf{F})$ and, for any set Γ of inference rules, $Cn(\Gamma \cup E) = Cn(\Gamma \cup E')$. Thus

$$Cn(S') = Cn[\,(\,S \cap S' \cap Lit(\mathbf{F}_f)\,) \cup E' \cup Rules(D)\,]\,.$$

Furthermore, because $(\,S \cap S' \cap Lit(\mathbf{F}_f)\,) \cup E'$ is a set of fluent literals, we can conclude by the Vivid Domains Lemma that

$$Cn(S') = Cn[\,(\,S \cap S' \cap Lit(\mathbf{F}_f)\,) \cup E' \cup Rules(Simple(D))\,]\,.$$

That is, S' may result from doing A in S in domain $Simple(D)$.

Proof in the other direction is similar. Thus we have shown that the two domains agree on Res, which is sufficient to establish the fact that they have the same models, given the earlier observation that they have equivalent sets of value propositions (Section 6). Moreover, since they also agree, for each action A and state S, on the question of whether or not A is prohibited in S, we can conclude that either both domain descriptions are qualification-free or neither is. □