Updates, actions, and planning

Andreas Herzig	Jérôme Lang	Pierre Marquis	Thomas Polacsek
IRIT/UPS	IRIT/UPS	CRIL/Université d'Artois	IRIT/UPS
F-31062 Toulouse Cedex	F-31062 Toulouse Cedex	F-62307 Lens Cedex	F-31062 Toulouse Cedex
France	France	France	France
herzig@irit.fr	lang@irit.fr	marquis@cril.univ-artois.fr	polacsek@irit.fr

Abstract

A general framework for update-based planning is presented. We first give a new family of dependence-based update operators that are well-suited to the representation of simple actions and we identify the complexity of query entailment from an updated belief base. Then we introduce conditional, nondeterministic and concurrent updates so as to encode the corresponding types of action effects. Plan verification and existence are expressed in this update-based framework.

1 Introduction

Three subareas of AI seem significantly connected and still, in the literature, work in each of these areas has been so far disconnected, up to a few exceptions. These areas are (a) belief update, (b) reasoning about action and change, and (c) planning in nondeterministic environments.

Belief update (a) mainly focuses on determining how a belief state should evolve after adding a new piece of information reflecting an explicit evolution of the world; how beliefs persist is studied in depth (this often – yet not always – relies on an assumption of minimal change), as well as causality and dependence between pieces of information; in particular, disjunctive information (reflecting uncertainty about the world after update) is taken into account, and many theoretical results exist (characterization of operators thanks to representation theorems, and computational complexity results).

Reasoning about action (b) focuses on the nature of pieces of information to take account for (preconditions for executability, direct changes, indirect changes or ramifications, static laws of the domain) and on the nature of the actions themselves (determinism, conditional effects, normal and/or exceptional effects, concurrent execution...) using sophisticated languages, generally more expressive than those of belief update but slightly less worked from a theoretical point of view (especially to what concerns representation theorems and complexity results – yet some results exist).

Finally, while (a) and (b) care on determining the consequences on the agent's current belief state of, respectively, a new piece of information and the execution of an action, most of the previous works about planning (c) are centered on plan generation; algorithmic developments have been pushed forward especially when actions are very simple – like in STRIPS – but less so when they are more complex[1].

Our purpose is to show how existing works on belief update can be extended so as to represent complex actions representations and planning in nondeterministic environments.

After some formal preliminaries (Section 2), we introduce in Section 3 a new family of update operators based on literal dependence. Such operators generalize existing dependence-based operators but grasp in a better way the incorporation of the effects of an action in a belief base. Interestingly, this generalization has no influence on computational properties since the complexity of query entailment from an updated belief base is just as hard as classical entailment in the general case (unlike other operators like Winslett's PMA – unless the polynomial hierarchy collapses). Then we show in Section 4 how the effects of more sophisticated (ontic) actions can be represented in our framework by generalizing our family of update operators to deal with conditionals, nondeterminism (distinct from disjunction!) and concurrency; this establishes a closer parallel between updates and effects of complex actions such as considered in action languages. In Section 5, we show how to use our (extended) update operators so as to compute the effect of a (possibly conditional) plan in a nondeterministic domain. We successively consider the case where the environment is fully observable and unobservable. Complexity results for plan verification and plan existence are given for both cases. Finally, connections to related work are discussed in Section 6, before the concluding section.

2 Formal preliminaries

We consider a propositional language $PROP_{PS}$ built up from a finite set PS of propositional variables and the boolean constants $\top$ and $\bot$. Propositional formulas are denoted by A, B etc. and interpretations over PS (or *worlds*) are denoted by ω etc. We represent them as tuples or conjunctions of literals over PS. In the following, we will identify every propositional formula A with its set of models $Mod(A)$.

Let LIT be the set of literals of the language, i.e., $LIT = \bigcup_{x \in PS}\{x, \neg x\}$. We recall that a formula A is in Nega-

[1] Only a few recent works such as [McCain and Turner, 1998], [Rintanen, 1999] and [Ferraris and Giunchiglia, 2000] considered planners with nondeterministic actions represented in sophisticated action languages.

tive Normal Form iff it makes use of the connectives $\wedge$, $\vee$, $\neg$, only and the scope of negation connectives appearing in A includes propositional symbols only. For instance, $A = \neg((\neg a \wedge b) \vee (a \wedge c))$ is equivalent to the NNF formula $(a \vee \neg b) \wedge (\neg a \vee \neg c)$ [2]. $A[x \leftarrow C]$ denotes the formula obtained by uniformly replacing every occurrence of propositional variable x by the formula C. $A[l \leftarrow C]$ denotes the formula obtained by replacing in a uniform way every positive (resp. negative) occurrence of literal l in the NNF of A by C (resp. $\neg C$). We denote by $Lit(A) \subseteq LIT$ the set of literals appearing in the NNF of A. In the previous example, $Lit(A) = \{a, \neg a, \neg b, \neg c\}$. For any $L \subseteq LIT$, we note $NEG(L) = \{\neg l \mid l \in L\}$, supposing $\neg\neg x$ identified with x. Lastly, if ω is a world and L a consistent set of literals from LIT, then $Force(\omega, L)$ is the world that gives the same truth value as ω to all variables except the variables of literals of L and $Force(\omega, L) \models \bigwedge_{l \in L} l$. For instance, if $\omega = (a, \neg b, c, d)$ and $L = \{b, \neg d\}$ then $Force(\omega, L) = (a, b, c, \neg d)$.

A *belief update operator* $\diamond$ maps the propositional belief base (a formula) B representing the initial beliefs of a given agent and an input formula A reflecting some explicit evolution of the world [Katsuno and Mendelzon, 1991], to a new set of beliefs $B \diamond A$ held by the agent after this evolution has taken place. Katsuno and Mendelzon [Katsuno and Mendelzon, 1991] proposed a general semantics for update. The most prominent feature of KM updates (distinguishing updates from revision) is that update must be performed modelwise, i.e., $Mod(B \diamond A) = \bigcup_{\omega \models B} \omega \diamond A$.

Finally, we assume the reader familiar with some basic notions of computational complexity (see [Papadimitriou, 1994] otherwise).

3 A new family of dependence-based updates

Many proposals for update operators have been made. Recently, several authors showed that there are good reasons for building a belief operator from a dependence relation [Doherty *et al.*, 1998] [Herzig and Rifi, 1999]. The dependence-based update of a belief base B by an input formula A consists in first forgetting in B all information "relevant" to A (leaving unchanged the truth value of variables not relevant to the update), and then expanding the result with A. What remains to be defined is the notion of "being relevant to".

3.1 Formula-variable dependence

A *formula-variable dependence function* is modelled a mapping Dep from $PROP_{PS}$ to 2^{PS}. Many choices for Dep are possible (see [Herzig and Rifi, 1999] for details). Whatever the choice, the *dependence-based update* of a world ω by a formula A w.r.t. Dep, denoted by $\omega \diamond_{Dep} A$, is the set of all worlds ω' such that $\omega' \models A$, and for every propositional variable x from PS such that $x \notin Dep(A)$, ω and ω' assign the same truth value to x.

Interestingly, the complexity of query entailment from an updated belief base is "only" coNP-complete when dependence-based updates are considered [Liberatore, 2000] while entailment when most usual other operators - like Winslett's PMA - are used is at the second level of the polynomial hierarchy.

3.2 Formula-literal dependence

Forgetting everything about the *variables* involved in the update often leads to *forgetting too much*. Consider a robot being told to go to a room with two doors 1 and 2, and to ensure that at least one of them is red (possibly by painting one of the doors), which can be represented as an update by $\texttt{red1} \vee \texttt{red2}$. Suppose now that both doors are initially red: $B = \texttt{red1} \wedge \texttt{red2}$. Then, using any dependence function Dep s.t. $\{\texttt{red1}, \texttt{red2}\} \subseteq Dep(\texttt{red1} \vee \texttt{red2})$, we get $B \diamond_{Dep} (\texttt{red1} \vee \texttt{red2}) = \texttt{red1} \vee \texttt{red2}$. Thus, we forgot that both doors were already red. Clearly, this is not what is expected: intuitively, we should not forget that doors 1 and 2 are red, because updating by $\texttt{red1} \vee \texttt{red2}$ has no *negative* influence on red1 nor on red2. Hence, only *negative* occurrences of red1 and red2 should be forgotten before expanding by the input formula, not positive ones.

This may not be a problem in some contexts, but clearly, when reasoning about actions, these update operators make us forget too much and therefore do not handle the frame problem correctly. In order to cope with this limitation, we introduce a new family of dependence-based update operators. Such operators generalize existing ones because they are based on formula-literal dependence [Lang *et al.*, 2000], a more general notion than formula-variable dependence.

From now on, a formula-literal dependence function is a mapping $Dep : PROP_{PS} \rightarrow 2^{LIT}$. Many full-sense Dep can be considered. Like formula-variable dependence functions, $Dep(A)$ can be basically defined as $Lit(A)$ or $DepLit(A)$, where the latter is defined as follows:

Definition 1 (FL independence) *[Lang* et al.*, 2000] Let A be a formula from $PROP_{PS}$ and $l(= x$ or $\neg x)$ be a literal from LIT. A is* literal-independent *of x iff $A[x \leftarrow \top] \models A$, and A is* literal-independent *of $\neg x$ iff $A[x \leftarrow \bot] \models A$. Finally, A depends on l iff it is not literal-independent of l. We denote by $DepLit(A)$ the set of literals A depends on.*

Thus, $A = (a \vee b) \wedge (\neg a \vee c) \wedge (a \vee b \vee \neg c)$ is literal-dependent on a, $\neg a$, b and c; but not on $\neg c$, because it is equivalent to $(a \vee b) \wedge (\neg a \vee c)$, and $\neg c$ does not appear in the NNF of the latter.

More sophisticated dependence functions can also be designed by considering explicit dependence (binary) relations δ on LIT. One of them is $Dep(A) = \bigcup_{l \in Lit(A)} \delta(l)$ (and similarly with $DepLit(A)$). We can also take into account persistent literals, i.e., literals remaining true whatever happens, such as $\neg$ alive (contrarily to alive). For example, $Dep(A) = DepLit(A) \cap NEG(Pers)$ where $Pers$ is a given set of persistent literals is a possible dependence function. Clearly, persistent literals cannot be taken into account by formula-variable dependence. The other way round, every formula-variable dependence function can be simulated by a formula-literal dependence function.

3.3 Update based on formula-literal dependence

Definition 2 *The update $\omega \diamond_{Dep} A$ of the world ω by the formula A w.r.t Dep is the set of all worlds ω' s.t. $\omega' \models A$ and there exists $L \subseteq Dep(A)$ s.t. $\omega = Force(\omega', NEG(L))$.*

[2] Note that when A is built up from the connectives $\wedge$, $\vee$, $\neg$, $\Rightarrow$, it can be turned into an equivalent NNF formula in linear time by "pushing down" every occurrence of $\neg$ and removing double negations. Abusing words, we call the resulting formula *the NNF* of A.

The next result gives us a way to compute $B \diamond_{Dep} A = \bigcup_{\omega \in Mod(B)} \omega \diamond_{Dep} A$ in practice. We first need to define *literal forgetting* [Lang *et al.*, 2000].

Definition 3 (literal forgetting) *For a formula B and $L \subseteq LIT$, $ForgetLit(B, L)$ is the formula inductively defined by*
(1) $ForgetLit(B, \emptyset) = B$;
(2) $ForgetLit(B, \{l\}) = B[l \leftarrow \top] \vee (\neg l \wedge B)$;
(3) $ForgetLit(B, \{l\} \cup L) = ForgetLit(ForgetLit(B, L), \{l\})$

Proposition 1
$B \diamond_{Dep} A \equiv A \wedge ForgetLit(B, NEG(Dep(A)))$.

Example 1 $B = (\neg a \vee \neg d) \wedge (a \vee b) \wedge (a \vee c \vee d)$, $A = d$ *and let us take $Dep = DepLit$. We have $ForgetLit(B, Dep(A)) = ForgetLit(B, \{d\}) = (\neg a \vee \neg d) \wedge (a \vee b)$; hence $B \diamond_{Dep} A \equiv d \wedge \neg a \wedge b$.*

Interestingly, for all the update operators based on literal dependence listed above, we have the following complexity result (this slightly extends Theorem 15 from [Liberatore, 2000]):

Proposition 2 *Given three formulas A, B and C, $B \diamond_{Dep} A \models^{?} C$ is* coNP*-complete.*

If $DepLit(A) \subseteq Dep(A)$, then A and $ForgetLit(B, NEG(Dep(A)))$ are strongly separable, in the sense that, if $l \in DepLit(A)$, then $\neg l \notin DepLit(ForgetLit(B, NEG(Dep(A))))$ holds. This is an important property from a computational point of view when CNF queries are considered since:

Proposition 3 *If C and D are two strongly separable formulas, then they are separable in the sense of Levesque [Levesque, 1998], i.e., for every clause E, we have $C \wedge D \models E$ iff $C \models E$ or $D \models E$.*

Accordingly, one global entailment test can be replaced by two local (simpler) entailment tests, and exponential savings can be achieved this way in practice. Now, clausal entailment from a $ForgetLit$ formula reduces to clausal entailment from a classical formula:

Proposition 4 *Let E be a non-tautological clause. Then $Forget(B, L) \models E$ iff $B \models \bigvee_{l \in Lit(E) \setminus L} l$.*

Alltogether, the two previous propositions give us some tractable restrictions for clausal query entailment from an updated belief base. Let us say that a formula is tractable when it belongs to a class of formulas for which clausal entailment is known as tractable (e.g., the Horn CNF class). When $\diamond_{Dep}$ satisfies $DepLit(A) \subseteq Dep(A)$, A is a tractable formula for which literal-(in)dependence can be tested in polynomial time and B is a tractable formula, then determining whether $B \diamond_{Dep} A \models C$ can be achieved in polynomial time for every CNF formula C. This is the case for instance when $Dep = DepLit$, B is a Horn CNF formula and A is a set of binary clauses[3].

[3] Here, clausal entailment from the updated base $B \diamond_{Dep} A$ is even easier than clausal entailment from the corresponding expanded base $B \wedge A$ since determining whether $A \wedge B \models C$ is coNP-complete in this situation – tractable classes are known not to mix well.

4 Updates and actions

The aim of this section is to show that belief update is not far from being able to determine the effects of an arbitrary action. In order to do so, we introduce the notion of *extended input*.

Definition 4 (extended inputs) Extended inputs *are defined inductively as follows:*

- *every formula A of $PROP_{PS}$ is an extended input;*
- *if Φ and Ψ are extended inputs then $\Phi \cup \Psi$ is an extended input (*nondeterminism*);*
- *if Φ and Ψ are extended inputs then $\Phi || \Psi$ is an extended input (*concurrency*);*
- *if Φ and Ψ are extended inputs and C is a formula of $PROP_{PS}$ then* `if` C `then` Φ `else` Ψ *is an extended input (*conditional*).*

Definition 5 (updates by extended inputs) *Let $\diamond$ be an arbitrary update operator, ω a world and Φ an extended input. We define the update of ω by Φ inductively as follows:*

- *if $A \in PROP_{PS}$, then $\omega \diamond A$ is given by the definition of $\diamond$;*
- $\omega \diamond (\Phi \cup \Psi) = (\omega \diamond \Phi) \cup (\omega \diamond \Psi)$*;*
- $\omega \diamond \texttt{if } C \texttt{ then } \Phi \texttt{ else } \Psi = \begin{cases} \omega \diamond \Phi & \text{if } \omega \models C \\ \omega \diamond \Psi & \text{if } \omega \models \neg C \end{cases}$
- $\omega \diamond (\Phi || \Psi) = \omega \diamond (\Gamma(\Phi || \Psi))$
where $\Gamma(\Phi || \Psi))$ is defined inductively by:
$$\Gamma(\Phi||\Psi) = \begin{cases} \Phi \wedge \Psi \text{ if } \Phi, \Psi \in PROP_{PS} \\ \Gamma(\Phi||\Psi_1) \cup \Gamma(\Phi||\Psi_2) \text{ if } \Psi = \Psi_1 \cup \Psi_2 \\ \texttt{if } C \texttt{ then } \Gamma(\Phi \diamond \Psi_1) \texttt{ else } \Gamma(\Phi \diamond \Psi_2) \\ \qquad \text{if } \Psi = (\texttt{if } C \texttt{ then } \Psi_1 \texttt{ else } \Psi_2) \\ \Gamma(\Psi||\Phi) \textit{ otherwise} \end{cases}$$

As before, $B \diamond \Phi = \cup_{\omega \in Mod(B)} (\omega \diamond \Phi)$.

$\omega \diamond (\Phi \cup \Psi)$ is a *nondeterministic update*, namely, ω is nondeterministically updated by Φ or by Ψ. This has been first introduced by Brewka and Hertzberg [Brewka and Hertzberg, 1993], who have observed that $\omega \diamond (A \cup B)$ is in general different from $\omega \diamond (A \vee B)$. In the case where $\Phi = A$ and $\Psi = B$ are simple propositional formulas, updating ω by $A \cup B$ intuitively means that a nondeterministic action has been performed, whose possible effect is either A or B, and that (quoting from [Brewka and Hertzberg, 1993]) the planner has no way to know whether one of the postcondition is already satisfied and nondeterministically choose to update by A or by B. A first example is the `toss` action, representable by the extended input `head` $\cup \neg$ `head`. Another example (adapted from [Brewka and Hertzberg, 1993]): recall the red doors example (Section 2). If the robot is told to paint door 1 or door 2 in red but has no way to know whether one of the doors is already red (because it has no sensors), then the action should be modelled by `red1` $\cup$ `red2`. On the other hand, the action consisting of an update by `red1` $\vee$`red2` corresponds to a context where the robot can check whether one of the doors is initially red (and leaves the world unchanged if this is the case). The introduction of these two kinds of nondeterministic actions needs both standard disjunction and $\cup$. Note that we have $B \diamond (A \cup C) \equiv (B \diamond A) \vee (B \diamond C)$.

$\omega \diamond$ `if` C `then` Φ `else` Ψ is a *conditional update*: the initial model is updated by Φ or by Ψ depending on whether ω

satisfies the branching condition C or not. In the case where $\Phi = A$ and $\Psi = D$ are simple propositional formulas, we have $B \diamond \texttt{if}\ C\ \texttt{then}\ A\ \texttt{else}\ D \equiv (B \wedge C) \diamond A \vee (B \wedge \neg C) \diamond D$. Conditional updates are needed for action with conditional effects, such as the well-used turkey shooting action, which corresponds to an update by `if` loaded `then` (¬alive ∧¬loaded) `else` ⊤. Conditional actions are not well handled with pure unconditional updates.

$\omega \diamond (\Phi || \Psi)$ is a *concurrent update*. It basically works by gathering conjunctively all possible combinations of alternative results, and then performing the corresponding updates separately; in particular, $B \diamond ((A_1 \cup A_2) || (B_1 \cup B_2)) \equiv B \diamond (A_1 \wedge B_1) \vee B \diamond (A_1 \wedge B_2) \vee B \diamond (A_2 \wedge B_1) \vee B \diamond (A_2 \wedge B_2)$. This can be easily generalized to the case where more than two extended inputs must be considered as concurrent update formulas. Concurrent updates aim at representing compactly independent effects of actions. Consider the action of loading the gun which has two independent effects, namely: (1) the gun is loaded and (2) the turkey goes hiding if it is not deaf. It corresponds to the concurrent update by $\alpha =$ (loaded || `if` ¬deaf `then` hidden `else` ⊤). Concurrency can as well be used for concurrent (or parallel) actions[4].

Consider parallel actions having some inconsistent effects. As the effect of the update by their combination is an empty set of models, this has no impact on the final result. This means that our notion of action compatibility is *very loose*. For instance, if $\omega = (a, b, c)$ and if we have two actions α and β corresponding to the extended inputs $\alpha = (a \wedge b) \cup (a \wedge c)$ and $\beta = \neg a \cup \neg b$ then $\omega \diamond (\alpha || \beta) = (\omega \diamond \bot) \cup (\omega \diamond \bot) \cup (\omega \diamond \bot) \cup (\omega \diamond (a \wedge \neg b \wedge c)) = \{(a, \neg b, c)\}$. More precisely, two nondeterministic actions $\alpha = A_1 \cup ... \cup A_p$ and $\beta = B_1 \cup ... \cup B_q$ are compatible (i.e., concurrently executable) iff there is an effect A_i of α and an effect B_j of β such that $A_i \wedge B_j$ is consistent[5].

The previous definition is independent of the choice of a particular update operator. In the case of a dependence-based operator, the complexity results established in Section 3 enable us to say that the prediction problem (i.e., the problem of determining if the execution of an action in a given belief state leads to satisfying a given formula) is coNP-complete[6].

Note that usual ramification methods, such as the use of integrity constraints or causal rules (see [Doherty *et al.*, 1998] for their handling in an update framework) can be added on top of our framework. We do not discuss them since it is not a contribution of our paper.

5 Update-based planning

5.1 Definitions

In the following, we consider standard *ontic* actions, which are meant to act on the world, and do not have any other effects on the knowledge state of the agent than the realization of its ontic effects. In our framework, a standard ontic action α is described by an extended input, denoted by α as well (thus, we identify the action and its effects, which will not lead to ambiguities), and determining its effects on a world ω simply consists in updating ω by α.

What is missing now in order to define a plan is the notion of *update sequences*.

Definition 6 (plans) *Given a finite set of standard ontic actions ACT_O,* (branching) plans *are defined inductively by:*

- *λ (empty plan) is a plan;*
- *for any extended input $\alpha \in ACT_O$, α is a plan;*
- *if π_1 and π_2 are plans, then $\pi_1; \pi_2$ is a plan;*
- *if π_1 and π_2 are plans and C is a propositional formula, then* `if` C `then` π_1 `else` π_2 *is a plan.*

Thus, plans are sequential and conditional extended updates (these extended updates being themselves conditional, nondeterministic and concurrent)[7]. At the plan level, nondeterministic choice and parallel composition are not allowed.

Definition 7 (planning problems) *A* planning problem *is a triple $\mathcal{P} = \langle I, ACT_O, G \rangle$ where:*

- *I is a propositional formula representing the knowledge on the initial state;*
- *$ACT_O = \{\alpha_1, ..., \alpha_q\}$ is a set of standard ontic actions;*
- *G is a propositional formula representing the goal.*

Note that these definitions do not mention anything regarding observability. They can therefore be used as generic definitions. What changes when various assumptions about observability are made is the type of plans allowed. When full observability is assumed, any branching is allowed, while with unobservability, branching is never allowed. With partial observability we may branch only on conditions whose truth value can be determined by the agent.

Definition 8 (trajectories) *A* trajectory *τ is a nonempty sequence $\tau = \langle \omega_0, ..., \omega_T \rangle$ of models ($T \geq 0$). If $\tau = \langle \omega_0, ..., \omega_T \rangle$, then $length(\tau) = T$. If π is a plan and I an initial knowledge base, τ is a* possible trajectory *for (I, π) w.r.t. ACT_O iff (1) $\omega_0 \models I$ and (2) one of the following conditions holds:*

1. *$\pi = \lambda$ and $T = 0$;*
2. *$\pi \in ACT_O$ and $T = 1$ and $\omega_1 \in Mod(\omega_0 \diamond \pi)$;*

[4] Note that our notion of concurrency differs from concurrency in dynamic logic which is defined by $(\omega \diamond \Phi) \cap (\omega \diamond \Psi)$. Like in language $\mathcal{C}$ [Giunchiglia and Lifschitz, 1998], concurrency means that the effects of the concurrent actions must be gathered before being used as inputs for belief update. Consider for instance the two actions "open the door" and "open the window", corresponding to the updates `door` and `window`, applied concurrently to the initial state $\omega =$ (¬ `door` , ¬ `window`). For any dependence-based update operator $\diamond$ for which `door` and `window` are independent we get $\omega \diamond$ `door` = (`door`, ¬ `window`), $\omega \diamond$ `window` = (¬ `door`, `window`), and thus ($\omega \diamond$ `door`) ∩ ($\omega \diamond$ `window`) = ∅, while with our definition we get the intended result $\omega \diamond$ (`door` || `window`) = $\omega \diamond$ (`door` ∧ `window`) = (`door`, `window`).

[5] Other notions of compatibility can be represented in our framework. They are not developed further due to space limitations.

[6] If the update operator was instead Π_2^p-complete, as many operators are, then extended update would be Π_2^p-complete too.

[7] Note that `if .. then .. else` is used both at the action and at the plan levels. Formally, both structures are identical but both kinds of conditionals do not share the same purpose; especially, observability considerations may restrict the range of allowed branching conditions at the plan level, but not at the action level.

3. *$\pi = \pi_1; \pi_2$ and τ can be decomposed in two subtrajectories $\tau_1 = \langle \omega_0, ..., \omega_i \rangle$, $\tau_2 = \langle \omega_i, ..., \omega_T \rangle$, s.t. τ_1 is a possible trajectory for (I, π_1), and τ_2 is a possible trajectory for (ω_i, π_2));*
4. *$\pi =$ `if` C `then` π_1 `else` π_2 and*
   - *either $\omega_0 \models C$ and τ is a possible trajectory for (I, π_1)*
   - *or $\omega_0 \models \neg C$ and τ is a possible trajectory for (I, π_2).*

Definition 9 (succeeding plans) *A plan π is a* succeeding plan *for a planning problem $\mathcal{P} = \langle I, ACT_O, G \rangle$ iff every possible trajectory τ for (I, π) is s.t. $\omega_{length(\tau)} \models G$.*

5.2 Fully observable environments

Under full observability, any plan is allowed. Let us consider the complexity of plan verification and existence.

Proposition 5 (complexity of plan verification/existence) *Suppose that the update operator used $\diamond$ is one of the dependence-based operator given in the previous sections.*

1. *determining whether π is a succeeding plan for $\mathcal{P} = \langle I, ACT_O, G \rangle$ is* coNP*-complete;*
2. *determining whether there exists a succeeding plan π for $\mathcal{P} = \langle I, ACT_O, G \rangle$ is* EXPTIME*-complete, and determining whether there exists a succeeding plan π of polynomially bounded length for $\mathcal{P}$ is* PSPACE*-complete.*

Point 1. is a key result. It is a strong generalization of Theorem 20 from [Liberatore, 2000]. Point 2. is easier to establish: the upper bounds are a byproduct of Point 1., and the lower bounds are derived from Littman's results on probabilistic planning [Littman, 1997]. These results (especially Point 1.) show that update-based planning in fully observable environments is not harder than STRIPS-based nondeterministic planning, which (somewhat) means that the facilities brought with the use of update are "for free".

5.3 Unobservable environments

Definition 10 (unbranching plans) *An* unbranching plan *is a (possibly empty) sequence $\alpha_0; ...; \alpha_T$ of standard ontic actions, or equivalently a plan in which no* `if` ... `then` ... `else` *appears at the outer level*[8].

Trajectories and succeeding plans are defined the same way as above. The following property, reformulating succeeding plans in terms of iterated updates, is almost straightforward:

Proposition 6 *Let $\pi = \alpha_1; ...; \alpha_T$ be an unbranching plan, and $\mathcal{P} = \langle I, ACT_O, G \rangle$ a planning problem. Then π is a succeeding plan for $\mathcal{P}$ iff $((..(I \diamond \alpha_1) \diamond \alpha_2) \diamond ...) \diamond \alpha_T \models G$.*

Proposition 7 (complexity of plan verification/existence) *Suppose the update operator used $\diamond$ is one of the dependence-based operator given in the previous sections.*

1. *determining whether an unbranching plan π is a succeeding plan for $\mathcal{P} = \langle I, ACT_O, G \rangle$ is* coNP*-complete.*
2. *determining whether there exists a succeeding unbranching plan π for $\mathcal{P} = \langle I, ACT_O, G \rangle$ is* EXPSPACE*-complete, and determining whether there exists a succeeding unbranching plan π for $\mathcal{P}$ of polynomially bounded length is Σ_2^p-complete.*

The EXPSPACE-hardness result is a direct consequence of a result from [Haslum and Jonsson, 1999]. The latter point (Σ_2^p-completeness) is similar to results in [Baral *et al.*, 1999] and in [da Costa *et al.*, 1997], which again suggests that dependence-based updates can be added "for free".

6 Related work

[Brewka and Hertzberg, 1993] were the first to use belief update operators for representing actions with conditional and nondeterministic effects. In Sections 3 and 4, we truly follow in their steps. They use Winslett's PMA update operator, which has been shown in many places to possess severe drawbacks (both from the points of view of expressivity and complexity). By making use of further advances in belief update, especially in what concerns dependence-based update, we were able to go much further than they did towards the practical computation of plans (and as well by allowing for concurrent actions, conditional plans and epistemic actions). See also the transition-based update-like operators of Cordier and Siegel [Cordier and Siegel, 1995].

[Fargier *et al.*, 2000] use dependence-based update for one-stage decision making processes (that can be viewed as degenerated planning problems). Their use of a formula-variable dependence is, as we show in Section 3, not the best choice for reasoning about action, and they do not use nondeterministic updates, nor sequences of updates.

[del Val and Shoham, 1994] point out some correspondances between the Katsuno-Mendelzon postulates and generic properties of action (by rewriting KM postulates in the situation calculus and showing that each KM postulate can be expressed as a specific property of actions), which enable them to give some intuitive reasons to accept or reject some of the KM postulates. Rather than practically proposing action languages based on belief update operators as we did, their work remains at the meta-level and made us think that such an update-based framework for planning was feasible.

[Shapiro *et al.*, 2000] represent iterated updates as well as epistemic actions in the situation calculus. They distinguish between ontic actions (leading to updates) and epistemic (leading to revisions). They represent the set of possible worlds (or situations) explicitly, while, using STRIPS-like representation and a propositional formalism, we represent it in a compact way, which makes our framework very different from theirs.

A line of work which is parallel to ours is the planning as satisfiability frameworks with actions languages, especially [Giunchiglia, 2000] based on the language $\mathcal{C}$ and [McCain and Turner, 1998] based on the causal theory developed by the authors. The main differences between these languages and update is in the handling of disjunction: the literal completions used in [Giunchiglia, 2000] and [McCain and Turner, 1998] require that dynamic laws expressing the effects of actions have a single literal in their head. The exact links

[8] However, `if` ... `then` ... `else` may appear inside the extended inputs, for representing conditional effects.

between these expressive languages and belief update is an interesting topic for further research. [Geffner, 1996] also makes use of causal theories for actions and planning, and uses plausibility functions to distiguish various levels of exceptionality in the effects of actions.

7 Conclusion

In this paper, we have presented a new update-based framework for planning. Our contribution is manyfold. We have introduced a new family of dependence-based update operators, based on formula-literal dependence, which includes existing dependence-based operators as specific cases, suits better the needs of reasoning about action, and whose computational complexity is not being above that of classical entailment. We have also shown how belief update has to be augmented with conditionals, nondeterminism and concurrency in order to be fully adapted to the representation of actions. Third, we have shown how to define (conditional or unconditional) plans in our framework. Finally, we have shown that the complexity of plan verification and existence in our framework is not higher than with a STRIPS-like representation.

This work calls for several perspectives. One of them concerns the introduction of partial observability in our framework. The simplest way to do it consists in considering that a fixed set of formulas is directly observable; allowed branching conditions are in this case formulas whose truth value can be determined from the truth values of these formulas. A more expressive solution consists in allowing for epistemic actions in the framework. Unlike ontic actions, epistemic actions leave the world unchanged and therefore do not imply an update, but instead, a knowledge expansion. In order to distinguish between facts and knowledge, an epistemic modality $\mathcal{K}$ can be introduced. Plans can be defined in a similar way as before except for branching conditions, for which we allow only for epistemically interpretable branching conditions as in [Herzig *et al.*, 2000].

Acknowledgements

The third author has been partly supported by the IUT de Lens and the Région Nord/Pas-de-Calais.

References

[Baral *et al.*, 1999] C. R. Baral, V. Kreinovich, and R. Trejo. Computational complexity of planning and approximate planning in presence of incompleteness. In *Proc. of IJCAI'99*, pages 948–955, 1999.

[Brewka and Hertzberg, 1993] G. Brewka and J. Hertzberg. How to do things with worlds: on formalizing actions and plans. *J. of Logic and Computation*, 3(5):517–532, 1993.

[Cordier and Siegel, 1995] M.O. Cordier and P. Siegel. Prioritized transitions for updates. In *Proc. of ECSQARU'95*, pages 142–150. LNAI 946, Springer-Verlag, 1995.

[da Costa *et al.*, 1997] C. da Costa, F. Garcia, J. Lang, and R. Martin-Clouaire. Possibilistic planning: Representation and complexity. In *Proc. of ECP'97*, pages 143–155, 1997.

[del Val and Shoham, 1994] A. del Val and Y. Shoham. Deriving properties of belief update from theories of action. *J. of Logic, Language, and Information*, 3:81–119, 1994.

[Doherty *et al.*, 1998] P. Doherty, W. Łukasziewicz, and E. Madalińska-Bugaj. The PMA and relativizing change for action update. In *Proc. of KR'98*, pages 258–269, 1998.

[Fargier *et al.*, 2000] H. Fargier, J. Lang, and P. Marquis. Propositional logic and one-stage decision making. In *Proc. of KR'2000*, pages 445–456, 2000.

[Ferraris and Giunchiglia, 2000] P. Ferraris and E. Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proc. of AAAI'00*, pages 748–753, 2000.

[Geffner, 1996] H. Geffner. A qualitative model for temporal reasoning with incomplete information. In *Proc. of AAAI'96*, pages 1176–1181, 1996.

[Giunchiglia and Lifschitz, 1998] E. Giunchiglia and V. Lifschitz. An action language based on causal explanation: Preliminary report. In *Proc. of AAAI'98,* pp. 623-630, 1998.

[Giunchiglia, 2000] E. Giunchiglia. Planning as satisfiability with expressive action languages: Concurrency, constraints, and nondeterminism. In *Proc. of KR'2000*, pages 657–666, 2000.

[Haslum and Jonsson, 1999] P. Haslum and P. Jonsson. Some results on the complexity of planning with incomplete information. In *Proc. of ECP'99,* 308-318, 1999.

[Herzig and Rifi, 1999] A. Herzig and O. Rifi. Propositional belief update and minimal change. *Artificial Intelligence*, 115:107–138, 1999.

[Herzig *et al.*, 2000] A. Herzig, J. Lang, D. Longin, and T. Polacsek. A logic for planning under partial observability. In *Proc. of AAAI'2000*, pages 768–773, 2000.

[Katsuno and Mendelzon, 1991] H. Katsuno and A. Mendelzon. Propositional knowledge base revision and minimal change. *Artificial Intelligence*, 52:263–294, 1991.

[Lang *et al.*, 2000] J. Lang, P. Liberatore, and P. Marquis. Propositional independence, part I: formula-variable dependence and forgetting. Technical report, IRIT, 2000.

[Levesque, 1998] H. Levesque. A completeness result for reasoning with incomplete first-order knowledge bases. In *Proc. of KR'98*, pages 14–23, 1998.

[Liberatore, 2000] P. Liberatore. The complexity of belief update. *Artificial Intelligence*, 119:141–190, 2000.

[Littman, 1997] M. L. Littman. Probabilistic propositional planning: Representations and complexity. In *Proc. of AAAI'97*, pages 748–754, 1997.

[McCain and Turner, 1998] N. McCain and H. Turner. Satisfiability planning with causal theories. In *Proc. of KR'98*, pages 212–223, 1998.

[Papadimitriou, 1994] Ch. Papadimitriou. *Computational complexity*. Addison–Wesley, 1994.

[Rintanen, 1999] J. Rintanen. Constructing conditional plans by a theorem prover. *Journal of Artificial Intelligence Research*, 10:323–353, 1999.

[Shapiro *et al.*, 2000] S. Shapiro, M. Pagnucco, Y. Lespérance, and H. J. Levesque. Iterated belief change in the situation calculus. In *Proc. of KR'00*, pages 527–538, 2000.