A Dynamic Epistemic Logic with Finite Iteration and Parallel Composition
Andreas Herzig, Frédéric Maris, Elise Perrotin

To cite this version:

HAL Id: hal-03450078
https://ut3-toulouseinp.hal.science/hal-03450078
Submitted on 25 Nov 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A Dynamic Epistemic Logic with Finite Iteration and Parallel Composition

Andreas Herzig\textsuperscript{1}, Frédéric Maris\textsuperscript{2}, Elise Perrotin\textsuperscript{2}

\textsuperscript{1}IRIT, CNRS, Toulouse
\textsuperscript{2}IRIT, Univ. Toulouse III

Abstract
Existing dynamic epistemic logics combine standard epistemic logic with a restricted version of dynamic logic. Instead, we here combine a restricted epistemic logic with a rich version of dynamic logic. The epistemic logic is based on ‘knowing-whether’ operators and basically disallows disjunctions and conjunctions in their scope; it moreover captures ‘knowing-what’. The dynamic logic has not only all the standard program operators of Propositional Dynamic Logic, but also parallel composition as well as an operator of inclusive nondeterministic composition: its atomic programs are assignments of propositional variables. We show that the resulting dynamic epistemic logic is powerful enough to capture several kinds of sequential and parallel planning, both in the unbounded and in the finite horizon versions.

1 Introduction
Dynamic epistemic logics (DELs) combine epistemic logic and dynamic logic. Most approaches in the literature combine full-fledged epistemic logic with restricted versions of dynamic logic. The latter typically lack the program operators of propositional dynamic logic (PDL), in particular the operator of finite iteration (‘Kleene star’). The reason is that the latter causes undecidability of satisfiability (Miller and Moss 2005) and even of model checking. The latter is directly related to the general undecidability of DEL-based planning (Bolander and Andersen 2011; Aucher and Bolander 2013; Bolander et al. 2020). In previous work we have proposed a different route whose starting point is a restricted version of standard epistemic logic: boolean combinations of ‘knowing-whether’ operators followed by a propositional variable (Cooper et al. 2016; Cooper et al. 2020a). In that lightweight epistemic logic, EL-O, the existence of sequential and parallel epistemic plans can be decided in PSPACE (Cooper et al. 2020b). A dynamic extension, DEL-PAO, was proposed in (Herzig, Lorini, and Maffre 2015), and a modelling of planning tasks with no common knowledge and no parallel plans was given in (Cooper et al. 2016). We here give a fuller and more succinct dynamic logic, adding in particular an operator of parallel composition as well as an operator of inclusive nondeterministic composition. Both are imported from Dynamic Logic of Parallel Propositional Assignments DL-PPA (Herzig, Maris, and Vianey 2019), which is conceptually and mathematically simpler than several other proposals for adding parallel composition to dynamic logic such as (Benevides and Schechter 2014; Balbiani and Boudou 2018; Boudou, Herzig, and Troquard 2021). In the resulting DEL the solvability of various planning tasks can be captured: the existence of sequential and parallel plans, both in the unbounded and in the finite horizon versions. The operator of inclusive nondeterministic composition turns out to be instrumental for the succinct modelling of parallel planning.

We generalize the epistemic ‘knowing-whether’ operator to a ‘knowing-what’ (or ‘knowing the value’) operator, introduced and argued for in (Plaza 2007; Wang and Fan 2013; Wang 2018) as a useful and natural addition to logics of knowledge, particularly in the field of AI: agents may not only need to know whether or not a given proposition is true, but also what another agent’s telephone number is or the code to open a door. We show that this notion of ‘knowing what’ can be added to EL-O without much difficulty, leading to an enlarged field of possibilities regarding the planning tasks that can be modelled and worked with. We illustrate our logic by means of the gossip problem where each agent $i \in \text{Agt}$ knows a secret $s_i$ that none of the other agents knows and where the goal is to find a sequence of phone calls after which every agent knows every secret (Landau 1954; Knödel 1975; van Ditmarsch, van der Hoek, and Kuijer 2020). Several interesting variants of that problem can be designed, in particular where the goal is to obtain higher-order knowledge (Cooper et al. 2019; Cooper et al. 2020a; van Ditmarsch, Gattinger, and Ramezanian 2020).

We present the static epistemic logic EL-OC in Section 2 and extend it to the dynamic logic DEL-PPAOC in Section 3. In Section 4 we apply DEL-PPAOC to planning. Proofs of the results can be found in (Perrotin forthcoming, Ch. 6).

2 The Static Logic: EL-OC
We introduce the Epistemic Logic of Observation with Constants, abbreviated to EL-OC.

2.1 Atoms, Introspective Atoms, and Atomic Consequence
Let $Prop$ be a countable set of propositional variables, let $Cst$ be a countable set of constants, and let $Agt$ be a finite set
of agents. The set of observability operators is

\[ \text{OBS} = \{ S_i : i \in \text{Agt} \} \cup \{ S \}, \]

where \( S_i \) stands for individual observability of agent \( i \) and \( S \) stands for joint observability of all agents. The set of all sequences of observability operators is noted \( \text{OBS}' \) and the set of all non-empty sequences is noted \( \text{OBS}'' \). We use \( \sigma \), \( \sigma' \), etc. to denote elements of \( \text{OBS}' \).

Observability atoms, or atoms for short, are finite sequences of observability operators followed by a propositional variable. The set of all atoms is

\[ \text{ATM} = \{ \sigma p : \sigma \in \text{OBS}' \text{ and } p \in \text{Prop} \} \cup \{ \sigma c : \sigma \in \text{OBS}'' \text{ and } c \in \text{Cst} \}. \]

We use \( \alpha, \alpha', \beta, \ldots \) to denote atoms, unless specified as members of \( \text{ATM} \cup \text{Cst} \). Here are some examples: \( S_1 p \) reads “I sees the truth value of \( p \)”, i.e., 1 knows whether \( p \) is true or false. \( S_1 c \) reads “I sees the value of \( c \)”. \( J S \) reads “all agents jointly see whether agent 2 sees the value of \( c \)”, i.e., there is joint attention in the group of all agents concerning 2’s observation of \( c \). Agent 2 may or may not see the value of \( c \), and in both cases this is jointly observed. An atom with an empty sequence of observability operators is nothing but a propositional variable; constants are always preceded by at least one observability operator as they have no truth value.

We follow the principles of EL-O and call an atom introspective if it contains two consecutive \( S_i \) or a \( J S \) that is preceded by a non-empty sequence of observability operators. The set of all introspective atoms is

\[ \text{I-ATM} = \{ \sigma S_i S_j \alpha : \sigma \in \text{OBS}' \text{ and } \alpha \in \text{ATM} \cup \text{Cst} \} \cup \{ \sigma J S \alpha : \sigma \in \text{OBS}'' \text{ and } \alpha \in \text{ATM} \cup \text{Cst} \}. \]

We finally define a relation of atomic consequence between observability atoms as follows:

\[ \alpha \models \beta \text{ iff } \alpha = \beta \text{ or there are } \alpha' \in \text{ATM} \cup \text{Cst}, \sigma \in \text{OBS}' \text{ such that } \alpha = J S \alpha' \text{ and } \beta = \sigma \alpha'. \]

When \( \alpha \models \beta \) we say that \( \alpha \) is a cause of \( \beta \) and that \( \beta \) is a consequence of \( \alpha \). We denote by \( \alpha^= \) the set of causes of \( \alpha \), and by \( \alpha^\supset \) the set of its consequences. We generalize this to sets of atoms \( \forall V \subseteq \text{ATM}: V^= = \bigcup_{\alpha \in V} \alpha^= \).

### 2.2 Language and Semantics

The language of EL-OC is defined by the grammar

\[ \varphi ::= \alpha \mid \neg \varphi \mid (\varphi \land \varphi), \]

where \( \alpha \) ranges over \( \text{ATM} \). The boolean operators \( \top, \bot, \lor, \rightarrow \) and \( \equiv \) are defined in the standard way.

A state is a subset of the set of sets of atoms \( \text{ATM} \). We denote states by \( V, U, W \), etc. The set of all states is \( 2^{\text{ATM}} \).

As in EL-O semantics, we interpret formulas such that introspection is simulated. The only non-standard case is:

\[ \parallel \alpha \parallel = \{ V : \alpha \in V^= \cup \text{I-ATM} \}. \]

Hence \( \alpha \) is true in state \( V \) if and only if \( \alpha \) is introspective or \( \beta \models \alpha \) for some \( \beta \in V \).

### 3 Adding Dynamic Logic

We now describe the full dynamic logic \( \text{DEL-PPAOC} \).

#### 3.1 Language of \( \text{DEL-PPAOC} \)

The language of \( \text{DEL-PPAOC} \) extends that of \( \text{EL-OC} \) with the dynamic operator \( (\pi) \), where \( \pi \) is a program. Programs \( \pi \) and formulas \( \varphi \) are defined by the grammar

\[ \varphi ::= \pi \varphi \mid (\pi \varphi), \]

where \( \alpha \) ranges over the set of atomic formulas \( \text{ATM} \). The formula \( (\pi)\varphi \) reads “there is a possible execution of \( \pi \) such that \( \varphi \) is true afterwards”. The program \( \pi \leftarrow \varphi \) assigns the truth value of \( \varphi \) to \( \alpha \). \( \varphi \) tests whether \( \varphi \) is true (and fails when \( \varphi \) is false). \( \pi_1; \pi_2 \) executes \( \pi_1 \) and \( \pi_2 \) in sequence. \( \pi_1 \cup \pi_2 \) nondeterministically chooses between executing either \( \pi_1 \) or \( \pi_2 \); and \( \pi_1 \parallel \pi_2 \) nondeterministically chooses between executing either \( \pi_1 \) or \( \pi_2 \), or both. \( \pi_1 \parallel \pi_2 \) is the parallel composition of \( \pi_1 \) and \( \pi_2 \). The set of all formulas is \( \text{Fml}_{\text{DEL-PPAOC}} \).

The formula \( [\pi]\varphi \) abbreviates \( \neg (\pi)\neg \varphi \) and therefore has to be read “\( \varphi \) is true after every possible execution of \( \pi \)”. We define \( n \)-times iteration of \( \pi \) by induction on \( n \): \( \pi^0 = \top \) and \( \pi^{n+1} = \pi \parallel \pi^n \). Finally, we define \( \pi^{=\infty} \) as \( \bigcup_{i \in \text{N}} \pi^i \). \( \pi^{=\infty} \) is the unbounded iteration of \( \pi \).

#### 3.2 Semantics of \( \text{DEL-PPAOC} \)

The interpretation of a formula \( \varphi \) is a set of valuations \( \parallel \varphi \parallel \subseteq 2^{\text{ATM}} \) as just as in \( \text{EL-OC} \). The interpretation of a program \( \pi \) is a ternary relation on the set of valuations: \( \parallel \pi \parallel \subseteq 2^{\text{ATM}} \times 2^{\text{ATM}} \times 2^{\text{ATM}} \). When \( (V, U, W) \in \parallel \pi \parallel \) there is an execution of \( \pi \) from state \( V \) to state \( U \) assigning the variables at \( W \). The interpretation function is defined by mutual recursion. The main clauses are given in Table 2; the others are standard.

The interpretation of the assignment \( \alpha \leftarrow \varphi \) is that, either

<table>
<thead>
<tr>
<th>( S_i, S_j, \alpha )</th>
<th>( \text{J S} \alpha \rightarrow S_i \alpha )</th>
<th>( \text{V S}_4 )</th>
<th>( \text{J S} \text{J S} \alpha )</th>
<th>( \text{V S}_5 )</th>
<th>( \text{J S} S_i, \alpha )</th>
<th>( \text{V S}_6 )</th>
</tr>
</thead>
</table>
| Example 1. In the initial state \( \forall V \subseteq \text{ATM} \) every agent only knows her own secret. Therefore \( \forall V \subseteq \text{ATM} \) where all \( S_i \) are constants. (This differs from modellings of the gossip problem without ‘knowing-what’ where secrets are assumed to be either true or false.) Then \( \forall V \subseteq \{ \parallel S_i, S_j \rightarrow \neg S_i, S_j \parallel \} \) for every \( i \in \text{Agt} \).

A formula \( \varphi \) is \( \text{EL-OC} \) valid if \( \forall V \subseteq \text{ATM} \); it is satisfiable if \( \forall V \subseteq \text{ATM} \). Clearly, atom \( \alpha \) is valid if and only if it is introspective. It is easily shown that all properties of \( \text{EL-O} \) given in (Cooper et al. 2020a), in particular NP-completeness of the satisfiability problem, still hold in \( \text{EL-OC} \). The \( \text{EL-OC} \) validities are axiomatised in Table 1; the completeness proof closely follows that of (Cooper et al. 2020a).
variables is the singleton \{\alpha\}, or (2) \varphi is false, all causes of \alpha get the value false and the set of assigned variables is the set \alpha^= of all causes of \alpha.

The interpretation of parallel composition \pi_1 \uparrow \pi_2 is that each subprogram \pi_i is executed locally; then it is checked that the modifications (in terms of assigned variables) are compatible, in the sense that variables that are modified by both subprograms (i.e., those at \pi_1 \cap \pi_2) take the same truth value.

The interpretation of inclusive nondeterministic composition \pi_1 \cap \pi_2 is the exclusive nondeterministic composition of the three programs \pi_1, \pi_2 and \pi_1 \cap \pi_2.

4 Epistemic Planning

In this section we show how DEL-PPAO captures simple epistemic planning tasks where actions change not only the world, but also the agents’ knowledge. We assume deterministic actions with conditional effects described by add- and delete-lists. Such conditional effects are crucial: when an agent performs an action then the effects on another agent’s epistemic state typically depend on whether that agent sees the variables that are modified by the action (Andersen, Bolander, and Jensen 2012; Cooper et al. 2020a).

4.1 Action Descriptions

An action description is a pair \(a = (\text{pre}(a), \text{eff}(a))\) where \text{pre}(a) \in \text{Fml}_{\text{DEL-PPAO}} (the precondition of a) and \text{eff}(a) \subseteq \text{Fml}_{\text{DEL-PPAO}} \times 2^{A^=} \times 2^{A^=}

are the conditional effects of \(a\). For a triple \(ce = \langle \text{cnd}(ce), \text{eff}^+(ce), \text{eff}^-(ce)\rangle\) in \text{eff}(a), \text{cnd}(ce) is the condition of \(ce\), \text{eff}^+(ce) are the added atoms, and \text{eff}^-(ce) are the deleted atoms. We require consistency of effects: if \(ce_1, ce_2 \in \text{eff}(a)\) and \(\text{pre}(a) \land \text{cnd}(ce_1) \land \text{cnd}(ce_2)\) then \(\text{eff}^-(ce_1) \cap \text{eff}^+(ce_2)^= = 0\).

Example 2. In the gossip problem a call has two conditional effects per secret: if \(k\) knows \(s_i\) then \(s_i\) becomes known to \(j\), and vice versa. Hence \(\text{pre}([\text{call}]^i) = \top\) and

\[\text{eff}([\text{call}]^i) = \{\langle S_i, s_k, \{S_j, s_k, \emptyset\} : 1 \leq k \leq n \}} \cup \{\langle S_j, s_k, \{S_i, s_k, \emptyset\} : 1 \leq k \leq n \}

To every action \(a\) we associate a partial function \(\tau_a\) on states as follows: \(\tau_a(V)\) is defined if \(V \in [\text{pre}(a)]\); and when \(\tau_a(V)\) is defined then for every conditional effect \(ce \in \text{eff}(a)\), if its triggering condition \(\text{cnd}(ce)\) is satisfied (‘ce fires’) then the negative effects of \(ce\) are removed, as well as their causes, and the positive effects of \(ce\) are added:

\[\tau_a(V) = V \setminus \bigcup_{ce \in \text{eff}(a)} (\text{eff}^- (ce)^+ \cap \text{cnd}(ce) \setminus \text{pre}(a)) \cup \bigcup_{ce \in \text{eff}(a)} (\text{eff}^+ (ce) \cap \text{pre}(a) \setminus \text{cnd}(ce))\]

We capture \(\tau_a\) by a program testing the precondition and processing in parallel the conditional effects:

\[\text{exeAct} (a) = \text{pre} (a) \land \bigcap_{ce \in \text{eff} (a)} \left( \left\{ \begin{array}{l} \neg \text{cnd} (ce) \land \text{cnd} (ce) \setminus \text{pre} (a) \\ \neg \text{cnd} (ce) \land \text{cnd} (ce) \setminus \text{pre} (a) \end{array} \right. \right) \land \left( \left\{ \begin{array}{l} \text{cnd} (ce) \supseteq \text{pre} (a) \\ \text{cnd} (ce) \supseteq \text{pre} (a) \end{array} \right. \right) \right)\]

Note that thanks to the constraint on consistency of action effects the big parallel composition is always executable.

4.2 Consistency of a Set of Actions at a State

We give two consistency constraints for parallel execution.

First, \(a_1\) and \(a_2\) have no contradictory effects at \(V\) if for every \(ce_1 \in \text{eff}(a_1)\) and \(ce_2 \in \text{eff}(a_2)\), if \(V \in [\text{cnd}(ce_1) \land \text{cnd}(ce_2)]\) then \(\text{eff}^+(ce_1) \cap \text{eff}^-(ce_2)^= = 0\). Hence no effect of one action is destroyed by the effect of another action executed in parallel.\(^1\) For example, two actions of moving the same object to two different spots have contradictory effects.

Second, two different actions \(a_1\) and \(a_2\) have no cross-interaction at \(V\) if

1. \(V\) and \(\tau_{a_1}(V)\) agree on \(\text{pre}(a_2)\) and on the condition \(\text{cnd}(ce_2)\) of every conditional effect \(ce_2 \in \text{eff}(a_2)\).

\(^1\)The case \(a_1 = a_2\) is already excluded by our requirement on action descriptions in Section 4.1.
2. \( V \) and \( \tau_A(V) \) agree on \( \text{pre}(a_1) \) and on the condition 
\( \text{cnd}(ce_1) \) of every conditional effect \( ce_1 \in \text{eff}(a_1) \),
where ‘\( V \) and \( V' \) agree on \( \varphi \)’ means that either both \( V \in \| \varphi \| \) and \( V' \in \| \varphi \| \), or both \( V \notin \| \varphi \| \) and \( V' \notin \| \varphi \| \). Hence neither action preconditions nor effect conditions are modified by the effect of another action executed in parallel, guaranteeing that they can be interleaved. For example, the actions of two different agents picking up the same object cross-interact.

A set of actions \( A \) is consistent at \( V \) if for every \( a, a' \in A \) such that \( a \neq a' \), (1) \( a \) and \( a' \) have no contradictory effects at \( V \); (2) \( a \) and \( a' \) have no cross interaction at \( V \).

**Example 3.** A gossipping conference call \( \text{call}^j_i, \text{call}^k_i \) is consistent. A way to exclude this is to replace \( \text{call}^j_i \) by \( T\text{call}^j_i \), with \( \text{pre}(T\text{call}^j_i) = T \) and
\[
eff(T\text{call}^j_i) = \text{eff}(\text{call}^j_i) \cup \{(tg_i, 0, \{tg_i\}) \cup \{\neg\text{tg}_i, \{tg_i\}, 0\} \cup \{(\text{tg}_i, 0, \{\text{tg}_i\}) \cup \{\neg\text{tg}_i, \{\text{tg}_i\}, 0\}.\]

Then two different calls involving \( i \) each toggle the value of \( tg_i \), ensuring that they cross-interact in any state.

### 4.3 Semantics of a Set of Actions

A set of actions \( A \) determines a partial function \( \tau_A \) on valuations. It is defined at \( V \) if \( V \in \| \\bigwedge_{a \in A} \text{pre}(a) \| \) and \( A \) is consistent at \( V \). When it is defined at \( V \) then:
\[
\tau_A(V) = \left(V \setminus \bigcup_{a \in A, \text{cnd}(ce) \in \text{eff}(a)} \left(\bigcup_{V \in \| \text{cnd}(ce) \|} \text{eff}(ce)\right)\right) \cup \left(\bigcup_{a \in A, \text{cnd}(ce) \in \text{eff}(a)} \left(\bigcup_{V \in \| \text{cnd}(ce) \|} \text{eff}(ce)\right)\right).
\]

Thanks to consistency of \( A \) the order in which negative and positive effects are processed does not matter.

Let us capture this in DEL-PPAOC. First, the formula

\[
\text{Insensitive}(a, a') = \text{pre}(a) \wedge \bigwedge_{ce \in \text{eff}(a)} \left(\text{cnd}(ce) \leftrightarrow \text{eff}(a')\right)
\]

expresses that neither executability nor effects of \( a \) are sensitive to the execution of \( a' \). Then to every \( A \) we associate the DEL-PPAOC program

\[
\text{exeAct}(A) = \bigwedge_{a, a' \in A, a \neq a'} \text{Insensitive}(a, a') \wedge \bigwedge_{a \in A} \text{exeAct}(a).
\]

**Proposition 1.** For every finite set of actions \( A \):

1. \( \tau_A \) is defined at \( V \) if and only if \( \langle V, U, W \rangle \in \| \text{exeAct}(A) \| \) for some \( U, W \).
2. If \( \tau_A \) is defined at \( V \) then
   \( \tau_A(V) = U \) iff \( \langle V, U, W \rangle \in \| \text{exeAct}(A) \| \) for some \( W \).

### 4.4 Simple Epistemic Planning Tasks

A *simple epistemic planning task* is a triple \( \langle \text{Act}, V_0, \text{Goal} \rangle \) where \( \text{Act} \) is a finite set of actions, \( V_0 \in \mathcal{A}^{2\text{ATM}} \) is a finite state (the initial state), and \( \text{Goal} \in \text{Fin}l_{\text{DEL-PPAOC}} \) is a DEL-PPAOC formula.

**Example 4.** The gossip problem can be viewed as the planning task \( G = \langle \text{Act}^G, V_0^G, \text{Goal}^G \rangle \) with
\[
\text{Act}^G = \{ \text{Tcall}^j_i : i, j \in \text{Agt} \text{ and } i \neq j \} \text{ (cf. Example 3),}
\]
\[
V_0^G = \{ S, s_i : i \in \text{Agt} \} \text{ (cf. Example 1),}
\]
\[
\text{Goal}^G = \bigwedge_{i \in \text{Agt}} \bigwedge_{j \not\in \text{Agt}} S_i S_j.
\]

State \( V \) is reachable by a parallel plan from \( V_0 \in \mathcal{A}^{2\text{ATM}} \) via a set of actions \( \text{Act} \) if there is an \( m \geq 0 \) and sequences \( \langle V_0, \ldots, V_m \rangle \) and \( \langle A_0, \ldots, A_m \rangle \) such that \( V_m = V \) and for \( 1 \leq k \leq m, V_k \in \mathcal{A}^{2\text{ATM}} \). \( A_k \subseteq \text{Act} \), and \( \tau_A(V_{k-1}) = V_k \).

A simple epistemic planning task \( \langle \text{Act}, V_0, \text{Goal} \rangle \) is *solvable by a parallel plan* if there is a state \( V \) that is reachable by a parallel plan from \( V_0 \) via \( \text{Act} \) such that \( V \in \| \text{Goal} \| \); otherwise it is unsolvable by a parallel plan.

**Theorem 1.** A planning task \( \langle \text{Act}, V_0, \text{Goal} \rangle \) is solvable by a parallel plan with no more than \( k \) steps if and only if:
\[
V_0 \in \left\| \left( \bigwedge_{a \in \text{Act}} x_a \leftrightarrow \top \cup \bigwedge_{a \in \text{Act}} x_a \leftrightarrow \bot \wedge \pi_k \right) \right\|_{\text{Goal}^k},
\]
where the \( x_a \) are fresh variables and where
\[
\pi_k = \bigwedge_{a, a' \in \text{Act}, a \neq a'} \left( x_a \wedge x_{a'} \rightarrow \text{Insensitive}(a, a') \right) \wedge \bigwedge_{a \in \text{Act}} \left( \neg x_a \wedge (x_a ? \text{exeAct}(a)) \right).
\]

**Example 5.** Task \( G \) can be solved in \( \lceil \log_2 n \rceil \) steps of parallel calls if the number of agents \( n \) is even, and in \( \lceil \log_2 n \rceil + 1 \) steps if \( n \) is odd (Bavelas 1950; Landau 1954; Knödel 1975; Cooper et al. 2019). For instance, for \( n = 4 \) the parallel plan \( \langle \text{Tcall}^1_i, \text{Tcall}^2_i, \text{Tcall}^3_i, \text{Tcall}^4_i \rangle \) solves \( G \) in 2 steps.

Solvability by a sequential plan is the special case where the parallel plan is a sequence of singletons.

**Theorem 2.** A planning task \( \langle \text{Act}, V_0, \text{Goal} \rangle \) is solvable by a sequential plan with no more than \( k \) actions if and only if:
\[
V_0 \in \left\| \left( \bigcup_{a \in \text{Act}} \text{exeAct}(a) \right)^{\leq k} \right\|_{\text{Goal}^k}.
\]

### 5 Conclusion

Our logic DEL-PPAOC extends the lightweight epistemic logic EL-O in two directions. First, we generalise the ‘knowing-whether’ operator to a ‘knowing-what’ (or ‘knowing the value’) operator. Second, we add dynamic operators: all standard program operators of PDL plus operators of parallel composition and of inclusive nondeterministic composition. We show how to model solvability of simple epistemic planning tasks by bounded parallel or sequential plans.

We conjecture that DEL-PPAOC satisfiability and model checking are both PSPACE-complete. One way of proving this is to combine the reduction of DL-PPA to DL-PA (of Herzig, Maris, and Vianey 2019) and that of DEL-PAO to DL-PA of (Herzig, Lorini, and Maffre 2015). From that, PSPACE membership of the problem of solvability of planning tasks can be obtained by polynomially eliminating finite iteration \( \pi^{\leq k} \) by means of counters as done in (Herzig, Maris, and Vianey 2019). We believe that thanks to its decidability and relatively low complexity, DEL-PPAOC provides an interesting alternative to DEL-based epistemic planning.
Acknowledgements

We would like to thank the KR 2021 reviewers for their comments. Our work is partially supported by the EU ICT-48 2020 project TAILOR (No. 952215) and the ANR project CoPains (No. ANR-18-CE33-0012).

References


