Abstract. A macro operator is a planning operator that is generated from a sequence of actions. Macros have mostly been used for macro planning, where the planner considers the macro as a single action and expands it into the original sequence during execution, but they can also be applied to other problems, such as maintaining a plan library. There are several approaches to macro operator generation, which differ in restrictions on the original actions and in the way they represent macros. However, all existing approaches are either restricted to STRIPS domains, only work on grounded actions, or they do not synthesize macros but consider the original sequence instead.

We study the synthesis of macro operators for ADL domains. We describe our synthesis method based on an ADL semantics for ADL. We use the synthesis method for ADL macro planning and evaluate it on a number of domains from the IPC. As a second application, we describe how macro operator synthesis can be useful for maintaining a plan library by computing the precondition and effects of the parameterized library plans.

1 Introduction

Automated planning can be thought of as the task to determine a sequence of actions from a given initial state to a desired goal state. Macro planning is a technique to improve planner performance by treating sequences of actions, which frequently occur together, as atomic. Macro planning is a well-studied field and a number of macro planners exist, e.g., Marvin [11], MUM [5], MACROFF [2], or DBMP/S [20]. However, all existing macro planners are either restricted to STRIPS, or they apply macros by applying each action of the sequence one-by-one. In comparison to STRIPS, ADL adds quantified preconditions and affects, disjunctive preconditions, and conditional effects. This makes ADL macro operator synthesis a challenging problem, as “the precondition and effect formulas of a macro are hard to infer from the formulas of contained operators” [2]. In this paper, we investigate the synthesis of ADL macro operators. Given a parameterized sequence of operators, the task is to determine a precondition formula such that the macro operator can be applied if and only if the respective action sequence can be applied, and an effect formula such that the resulting states after applying the macro operator and the respective action sequence are the same. Consider a robot that can carry a bag of potentially fragile objects (Listing 1). If we combine the actions drop and fix into the macro drop-fix, the precondition formula of the macro should state that (1) the robot must be carrying the bag, and (2) the object to fix must be either broken or it must be fragile and in the bag. The effect formula of the macro should state that (1) the robot is not carrying the bag anymore, (2) the object that the robot fixed is not broken, and (3) all fragile objects in the bag other than the object to be fixed are still broken. The resulting macro is shown in Listing 2.

Apart from macro planning, macro operator synthesis can be used to generate and maintain a plan library. A plan library is a collection of pre-computed plans and can be used as replacement for planning at run-time, e.g., in robotics domains [1, 18]. Such a plan library is either hand-crafted or computed from previous planning results. In both cases, macro operator synthesis alleviates the problem of creating and maintaining a consistent library: In the former case, macro operator synthesis can be used to compute the overall preconditions and effects, which can then be used at run-time to check whether the plan is suitable in the given situation. In the latter case, macro operator synthesis can be used to generalize the sample plans.

We describe our synthesis method based on an ADL semantics, which we summarize in Section 3. In Section 4, we describe how to synthesize the macro operator by regressing the actions’ preconditions into a single precondition, and by chaining the actions’ effects into a single effect. We show that the synthesized macro operators are indeed correct. As the main application of ADL operator synthesis, we extend the macro planner DBMP/S to ADL by using the synthesized macro operators for planning in Section 5. We evaluate the approach by comparing it to MACROFF and planning without macros. In Section 6, we de-
scribe how macro operator synthesis can also be used to generate and maintain a plan library, before we conclude in Section 7.

2 Related Work

Already the original STRIPS planner was extended with macros in the form of generalized plans [14], (partial) solutions to previous problems which are generalized by substituting constants with parameters. Marvin [11] based on FF uses macros to escape search plateaus. Marvin has been extended to store macros in a library so they can be re-used later [10]. To maintain a small library, macros are filtered based on usage count, number of problems since last use, instantiation count, and length of the macro. Wizard [27] learns STRIPS-based macros using a genetic algorithm. It generates initial macros from solutions of simple seeding problems. It evaluates macros based on the percentage of problems solved, the mean time gain/loss, and the percentage of problems solved faster [26]. M macro operators by analyzing a plan database for frequent action sequences. In contrast to M macro operators, B [5] is a STRIPS macro planner that learns macros from the solutions of simple seeding problems. It evaluates macros based on the percentage of problems solved, the mean time gain/loss, and the percentage of problems solved faster [26]. M macro operators by analyzing a plan database for frequent action sequences. In contrast to M macro operators, B [5] is a STRIPS macro planner that learns macros from the solutions of simple seeding problems. It evaluates macros based on the percentage of problems solved, the mean time gain/loss, and the percentage of problems solved faster [26].

24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

3 Foundations: $\mathcal{ES}$ and ADL Semantics

For the definition of ADL macro operators, we use an ADL semantics based on $\mathcal{ES}$ [9]. Based on this semantics, we will define ADL macro operators and proof that such an operator is equivalent to the original action sequence with respect to its preconditions and effects.

3.1 The Logic $\mathcal{ES}$

The logic $\mathcal{ES}$ [22] is an epistemic variant of the Situation Calculus [25, 24] where situations do not appear as terms in the language but are part of the semantics. We only present the non-epistemic subset of $\mathcal{ES}$ and refer to [22, 21] for details.

Syntax $\mathcal{ES}$ is a first-order modal logic with equality, infinitely many fluent predicate symbols $F^k$ and rigid function symbols $G^k$ of every arity $k$, and connectives $\land, \lor, \neg, \forall, [\cdot]$. The terms of the language are the least set of expressions such that (1) Every first-order variable is a term; (2) If $t_1, \ldots, t_k$ are terms and $g \in G^k$, then $g(t_1, \ldots, t_k)$ is a term. A term is called ground if it does not contain any variables.

We let $R$ denote the set of all ground terms. As in [22], we do not distinguish between sorts action and object but allow any term to be used as an action or as an object.

The well-formed formulas of the language are the least set of expressions such that (1) If $t_1, \ldots, t_k$ are terms and $F \in F^k$, then $F(t_1, \ldots, t_k)$ is an atomic formula; (2) If $t_1$ and $t_2$ are terms, then $(t_1 = t_2)$ is a formula; (3) If $t$ is a term and $\alpha$ is a formula, then $[\alpha ]t$ is a formula; (4) If $\alpha$ and $\beta$ are formulas, then so are $(\alpha \land \beta), \neg \alpha, \forall x, \alpha, C\alpha$.

We read $[\alpha ]t$ as "$\alpha$ holds after action $t$" and $\Box \alpha$ as "$\alpha$ holds after any sequence of actions". As usual, we treat $\exists, \alpha (\lor \beta, \alpha \lor \beta)$ as abbreviations. A formula without free variables is called a sentence. A formula with no $\Box$ operators is called bounded, a sentence with no $\Box$ and $[\cdot]$ operators and not mentioning $\text{Poss}$ is called fluent.

Semantics Let $P$ denote the set of all pairs $\sigma, \alpha$ where $\sigma \in R^*$ is considered a sentence of actions and $\rho = F(r_1, \ldots, r_k)$ is a ground fluent atom from $F^k$. A world is then a mapping from $P$ to truth values $\{0, 1\}$. Variables are interpreted substitutionally over the rigid terms $R$, i.e., $R$ is treated as being isomorphic to a fixed universe of discourse. This is similar to the logic $L$ [23].

Given a world $w$, for any formula $\alpha$ with no free variables, we write $w, \sigma \models \alpha$ instead of $w, \sigma \models \alpha$ where $\sigma$ denotes the empty action sequence, and

$$w, \sigma \models F(r_1, \ldots, r_n) \iff w [\sigma : F(r_1, \ldots, r_n)] = 1$$

$$w, \sigma \models (r_1 = r_2) \iff r_1 \text{ and } r_2 \text{ are identical}$$

$$w, \sigma \models (\alpha \land \beta) \iff w, \sigma \models \alpha \text{ and } w, \sigma \models \beta$$

$$w, \sigma \models \neg \alpha \iff w, \sigma \models \neq \alpha$$

$$w, \sigma \models \forall x, \alpha \iff w, \sigma \models \alpha^x \text{ for every } r \in R$$

$$w, \sigma \models [r] \alpha \iff w, \sigma \cdot r \models \alpha$$

$$w, \sigma \models \Box \alpha \iff w, \sigma \cdot d' \models \alpha \text{ for every } d' \in R$$

The notation $\alpha^x$ means the result of simultaneously replacing all free occurrences of the variable $x$ by the term $t$; $\alpha^t \cdot \alpha^s$ denotes the concatenation of the two action sequences.

Basic Action Theories Given a set $F$ of fluent predicates, a set of sentences $\Sigma$ is called a basic action theory over $F$ iff it only mentions the fluents in $F$ and is of the form $\Sigma = \Sigma_0 \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}}$, where (1) $\Sigma_0$ is a finite set of fluent sentences; (2) $\Sigma_{\text{pre}}$ is a singleton of the form $\Box \text{Poss} (\alpha) \equiv \pi$, where $\pi$ is fluent with $\alpha$ being the only free variable; (3) $\Sigma_{\text{post}}$ is a finite set of successor state axioms of the form $\Box [\alpha] F(\vec{x}) \equiv \gamma_F$, one for each fluent $F \in F \setminus \{\text{Poss}\}$, where $\gamma_F$ is a fluent sentence whose free variables are among $\vec{x}$ and $\alpha^t_0$ represents the initial database, $\Sigma_{\text{pre}}$ is one large precondition axiom, and $\Sigma_{\text{post}}$ the set of successor state axioms for all fluents in $F$.

Regression Given a basic action theory $\Sigma$, a common task is projection, i.e., determining what holds after a sequence of actions has occurred. For a given basic action theory $\Sigma$, ground terms $r_1, \ldots, r_k$, and an arbitrary sentence $\alpha$, the projection task is to determine whether $\Sigma \models [r_1] \ldots [r_k] \alpha$. One method to do projection is to use regression. The idea of regression is to successively replace fluents in $\alpha$ by the right-hand side of their successor state axioms until the resulting sentence does not contain any actions. In $\mathcal{ES}$, any bounded, objective sentence $\alpha$ is considered regressable. The regression of $\alpha$ over a sequence of (necessarily ground) terms $\sigma$ is denoted as $\mathcal{R}[\sigma, \alpha]$. As $\sigma$ does not need to be ground, we can also regress a
As shown in \cite{22}, for any ADL precondition formula $A_n$, where for any sequence of terms $\tau$, $R[\tau, \alpha]$ is defined inductively on $\alpha$:

1. $R[\tau, (t_1, t_2)] = (t_1, t_2)$;
2. $R[\tau, \forall \sigma] = \forall \tau R[\tau, \alpha]$;
3. $R[\tau, (A \land \beta)] = (R[\tau, \alpha] \land R[\tau, \beta])$;
4. $R[\tau, \neg \alpha] = \neg R[\tau, \alpha]$;
5. $R[\tau, [\alpha]] = R[\tau, t, \alpha]$;
6. $R[\tau, \text{Pos}(t)] = R[\tau, \pi]$;
7. $R[\tau, F(t_1, \ldots, t_k)]$ is defined inductively on $\sigma$ by
   
   (a) $R[\tau, F(t_1, \ldots, t_k)] = F(t_1, \ldots, t_k)$;
   
   (b) $R[\tau, \exists A(t_1, \ldots, t_k)] = R[\tau, (\gamma_F)_{t_1}^a \cdots t_k^a]$.

For any world $w$ and basic action theory $\Sigma$, we define a world $w_{\Sigma}$ which is like $w$ except that it satisfies the $\Sigma_{pre}$ and $\Sigma_{post}$ sentences of $\Sigma$. Formally, $w_{\Sigma}$ is a world satisfying the following conditions:

1. If $F \notin F$, $w_{\Sigma}[F(\vec{r})] = w[\sigma,F(\vec{r})]$;
2. If $F \in F$, $w_{\Sigma}[\sigma,F(\vec{r})]$ is defined inductively:

   (a) $w_{\Sigma}([\gamma_F]) = w([\gamma_F])$;

   (b) $w_{\Sigma}[\tau : \forall \sigma,F(\vec{r})] = 1$ if $w_{\Sigma}[\sigma] = \sigma_R$;

   (c) $w_{\Sigma}[\sigma : \text{Pos}(\vec{r})] = 1$ iff $w_{\Sigma}[\sigma] = \pi_R$.

As shown in \cite{22}, for any $w$, $w_{\Sigma}$ exists and is unique.

**Theorem 1** (Regression Theorem \cite{22}). Let $\Sigma = \Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post}$ be a basic action theory, $w$ a world, $\alpha$ a bounded sentence, and $\tau$ and $\sigma$ a sequence of ground terms. Then:

1. $w \models R[\tau, \alpha]$ iff $w_{\Sigma}[\sigma] = \alpha$;
2. $\Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post} \models \alpha$ iff $\Sigma_0 \models R[\tau, \alpha]$.

**3.2 ADL Semantics**

We summarize a declarative semantics of ADL based on ES \cite{9}, which translates ADL operators into precondition and successor state axioms of ES.

**ADL Operators** First, we describe how we can define ADL operators consisting of a precondition formula and an effect formula in ES.

An ADL precondition formula is an ES formula of the following form:

- (1) An atomic formula $F(t)$ is a precondition formula if each of the $t_i$ is either a variable or a constant.
- (2) An equality atom $\forall \phi$ is a precondition formula if each of the $t_i$ is either a variable or a constant.
- (3) If $\psi_1$ and $\psi_2$ are effect formulas, then $\psi_1 \land \psi_2$ and $\forall \exists \tau : \psi_1$ are effect formulas.
- (4) If $\gamma$ is a precondition formula and $\psi$ is an effect formula not containing "\( \Rightarrow \)" and "\( \neg \)\( \gamma \)", then $\gamma \Rightarrow \psi$ is an effect formula.

An ADL operator $O$ is given by a quadruple $(A, \vec{y}, \vec{t}, \pi_A, \epsilon_A)$, where (1) $A$ is a symbol from $G^0$, with $p = |\vec{y}|$; (2) $\vec{y}, \vec{t}$ is a list of variable symbols with associated types; (3) $\pi_A$ is a precondition formula with free variables from $\vec{y}$, and $\epsilon_A$ is an effect formula with free variables from $\vec{y}$. We call $\lambda$ the operator name and $\vec{y}, \vec{t}$ the parameters of $O$. An ADL operator $(A, \vec{y}, \vec{t}, \pi_A, \epsilon_A)$ is in normal form, if its effect $\epsilon_A$ is of the following form:

\[
\begin{align*}
&\forall \vec{x} : \exists ! F_j : (\gamma_{F_j}^+_{\vec{A}}(\vec{x})) \Rightarrow F_j(\vec{x}) \land \\
&\forall \vec{x} : \exists ! F_j : (\neg \gamma_{F_j}^-_{\vec{A}}(\vec{x})) \Rightarrow \neg F_j(\vec{x})
\end{align*}
\]

If an ADL operator is in normal form, then for each $F_j$, there is at most one positive effect formula of the form $F_j(\vec{x})$ and at most one negative effect formula of the form $\neg F_j(\vec{x})$.

The ADL operator for $\text{drop}$ is $O_{\text{drop}} = (\vec{y}, \vec{t}, \vec{A}, \vec{B}, \vec{A}_{\text{drop}}, \vec{B}_{\text{drop}})$ with $\vec{A}_{\text{drop}} : \text{carrying}(\vec{y})$ and $\vec{B}_{\text{drop}} : \forall \vec{x} : \text{obj}(\vec{y}) \Rightarrow \neg \text{broken}(\vec{y})$.

**ADL Problem Description** Using the definitions above, we can now describe how we can formulate an ADL problem description in ES. A problem description for ADL is given by (1) a finite list of types $t_1, \ldots, t_n$, $\text{Object}$, where $\text{Object}$ is a special type that must always be included, (2) a finite list of statements of the form $t_i(\text{either } t_1, \ldots, t_k)$ defining some of the types as compound types, where $\tau_i$ is the union of all $\tau_i$, and $k_i$ is the number of sub-types of $\tau_i$; a primitive type is a type other than $\text{Object}$ that does not occur on the left-hand side of such a definition, (3) a finite list of fluent predicates $F_1, \ldots, F_n$ with a list of types $t_{j_1}, \ldots, t_{j_k}$ for each $F_j$, which defines the types of the arguments of $F_j$, (4) a finite list of objects with associated primitive types $o_1 : t_{j_1}, \ldots, o_k : t_{j_k}$, with each $o_k$ is a symbol from $G^0$, (5) a finite list of ADL operators $O_1, \ldots, O_m$ with $O_i = (A_i, \vec{y}_i, \vec{t}_i, \pi_{A_i}, \epsilon_{A_i})$, in normal form, where each operator only contains symbols from the operator’s parameters, and from (1), (3), and (4), (6) an initial state $I$ in form of an effect formula that only contains symbols from (1), (3), (4), and (7) a goal description $G$ in form of a precondition formula, which only contains symbols from (1), (3), and (4).

**ADL Basic Action Theories** Given an ADL problem description, a corresponding ES basic action theory can be constructed as follows:

**Successor State Axioms** $\Sigma_{post}$ A set of operator descriptions $\{O_1, \ldots, O_m\}$ can be transformed into a set of successor state axioms $\Sigma_{post}$. Let

\[
\begin{align*}
\gamma_{F_j}^+ &\overset{\text{def}}{=} \bigvee_{\gamma_{F_j}^+_{\vec{A}} \in \Sigma_{\text{obj}}} \exists ! \vec{y}_i : a = A_i(\vec{y}_i) \land \gamma_{F_j}^+_{\vec{A}} \\
\gamma_{F_j}^- &\overset{\text{def}}{=} \bigvee_{\gamma_{F_j}^-_{\vec{A}} \in \Sigma_{\text{obj}}} \exists ! \vec{y}_i : a = A_i(\vec{y}_i) \land \gamma_{F_j}^-_{\vec{A}}
\end{align*}
\]

Using the definitions for $\gamma_{F_j}^+_{\vec{A}}$, we can define the successor state axiom for $F_j$ (cf. \cite{28}).

\[
\square [a] F_j(\vec{x}) \equiv \gamma_{F_j}^+ \land F_j(\vec{x}) \lor \neg F_j(\vec{x}) \land \neg \gamma_{F_j}^-
\]
In our example, the successor state axiom for broken is:

\[ \square [a] \text{broken}(o) \equiv \exists b. a = \text{drop}(b) \land (\text{in}(o, b) \land \text{fragile}(o)) \land \text{obj}(a) \land \text{broken}(o) \lor \text{fix}(a) \]

The Precondition Axiom \( \Sigma_{pre} \). The precondition axiom is a disjunction over all \( m \) operators of the problem domain:

\[ \pi \equiv \bigvee_{1 \leq i \leq m} \exists \vec{y}_i. \pi_i = A_i(\vec{y}_i) \land \pi_{A_i} \]

In our example, the precondition axiom is:

\[ \pi = \exists b. \text{bag}. a = \text{drop}(b) \land \text{carrying}(b) \lor \exists o. \text{obj}. a = \text{fix}(a) \land \text{broken}(a) \]

Initial Description \( \Sigma_0 \). The initial description \( \Sigma_0 \) is a conjunction of fluent formulas describing the initial state of the world and all information about the types of objects, e.g.:

\[ \Sigma_0 = \text{bag}(b_1) \land \text{obj}(a_1) \land \text{carrying}(b_1) \land \text{in}(a_1, b_1) \]

4 ADL Macro Operator Synthesis

Using ES and the ADL semantics described above, we can now define ADL macro operators. For a given sequence \( \sigma \) of ADL operator names, we construct a new macro operator \( O_\sigma \) whose precondition formula is satisfied iff all preconditions of \( \sigma \) are satisfied when applied subsequently, and whose effect is the same as the accumulated effect after applying \( \sigma \). In the following, let \( \sigma = (A_1, \ldots, A_n) \) be a non-empty sequence of ADL operator names with corresponding operators \( O_i = (A_i, \vec{y}_A, \vec{w}_A, \vec{A}_e, \epsilon_A) \).

Macro Preconditions

For the sequence \( \sigma \), we need to compute a precondition formula \( \pi_{A_\sigma} \) for the corresponding macro operator \( O_{A_\sigma} \). Intuitively, for any grounding \( \rho = \sigma(t) \) of \( \sigma \), \( w \models \pi_{A_\sigma}(t) \) must hold iff \( \rho \) is executable, i.e., iff it is possible to execute all actions of \( \rho \) subsequently. First, we define under what condition an action sequence is executable:

Definition 1 (Executable action sequence). Given an ADL problem description with ADL operators \( O_1, \ldots, O_n \) and corresponding basic action theory \( \Sigma \). Let \( \rho = (a_1, \ldots, a_n) = (A_1(\vec{t}_1), \ldots, A_n(\vec{t}_n)) \) be a ground sequence of actions with preconditions \( \pi_{A_n} = A_i(\vec{t}_i) \). We say \( \rho \) is executable in world \( w \) iff the following holds:

\[ w \models \pi_{A_n} \land [a_1](\pi_{a_2} \land [a_2](\pi_{a_3} \land \ldots \land [a_n](\pi_{A_n}))) \]

Next, we define the precondition formula of the macro operator \( O_{A_\sigma} \).

Definition 2 (Macro Precondition). The macro precondition \( \pi_{A_\sigma} \) is defined inductively:

\[ \pi_{A_n} = \pi_{A_n} \]

\[ \pi_{(A_1, A_2, \ldots, A_n)} = \pi_{A_1} \land R((A_1), \pi_{A_2}, \ldots, \pi_{A_n})) \]

Note that we define the precondition by regressing over the non-ground operator \( A_i \), i.e., \( \pi_{A_1, \ldots, A_n} \) may have free variables \( \vec{x}_1, \ldots, \vec{x}_n \). Given ground terms \( \vec{t}_1, \ldots, \vec{t}_n \), we denote the grounded precondition \( \pi_{A_1, \ldots, A_n}(\vec{t}_1, \ldots, \vec{t}_n) \) as \( \pi_{a_1, \ldots, a_n} \). Coming back to our example, the macro precondition of the sequence \( \sigma = (\text{drop}(b), \text{fix}(o)) \) is:

\[ \pi_{\sigma} = \pi_{\text{drop}} \land R((\text{drop}(b)), \pi_{\text{fix}}) = \pi_{\text{carrying}(b)} \land R((\text{drop}(b)), \text{broken}(o)) = \pi_{\text{carrying}(b)} \land (\text{broken}(o) \lor \text{in}(o, b) \land \text{fragile}(o)) \]

The macro precondition \( \pi_{\sigma} \) satisfies the desired property:

Theorem 2. Let \( \Sigma = \Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post} \) be a BAT. Then:

\[ \Sigma_0 \models \pi_{a_1} \land [a_1](\pi_{a_2} \land \ldots \land a_{n-1}) \pi_{a_n} \]

Proof. By Regression Theorem:

\[ \Sigma \models \pi_{a_1} \land [a_1](\pi_{a_2} \land \ldots \land a_{n-1}) \pi_{a_n} \iff \Sigma_0 \models R([a_1] \land [a_1] \pi_{a_2} \land \ldots \land a_{n-1}) \pi_{a_n} \]

where

\[ R([a_1 \land [a_1] \pi_{a_2} \land \ldots \land a_{n-1}) \pi_{a_n} \]

Corollary 1. The action sequence \( \sigma = (A_1(\vec{t}_1), \ldots, A_n(\vec{t}_n)) \) is executable in world \( w \) iff \( w \models \pi_{a_1, \ldots, a_n}(\vec{t}_1, \ldots, \vec{t}_n) \).

Macro Effects

For the sequence \( \sigma \) of ADL operator names, we need to compute the accumulated effect of \( \sigma \). To do so, we first define the chaining of two ADL operators:

Definition 3. Let \( O_1, O_2 \) be ADL operators in normal form with \( O_1 = (A_1, \vec{y}_A, \vec{w}_A, \vec{A}_e, \epsilon_A) \). We define the chaining \( C(\epsilon_1, \epsilon_2) \) of \( O_1 \) with \( O_2 \) as:

\[ \begin{align*}
C(\epsilon_1, \epsilon_2) &= \bigvee_{F} \vec{x}_F ; F_j \left( \gamma_{F_j, (A_1, A_2)}(\vec{x}_j) \Rightarrow F_j(\vec{x}_j) \right) \land \bigvee_{F} \vec{x}_F ; F_j \left( \gamma_{F_j, (A_1, A_2)}(\vec{x}_j) \Rightarrow \neg F_j(\vec{x}_j) \right) \\
\end{align*} \]

where

\[ \gamma_{F_j, (A_1, A_2)}(\vec{x}_j) = \gamma_{F_j, A_1}(\vec{x}_j) \land
\neg R((A_1), \gamma_{F_j, A_2}(\vec{x}_j) \lor R((A_1), \gamma_{F_j, A_2}(\vec{x}_j))\]

The subformula \( \gamma_{F_j, (A_1, A_2)}(\vec{x}_j) \) defines when the chained actions \( (A_1, A_2) \) cause the fluent \( F_j \) to be true. This is the case if \( A_1 \) causes the fluent to be true \( (\gamma_{F_j, A_1}(\vec{x}_j)) \) and after doing \( A_1, A_2 \) does not cause it to be false again \( (\neg R((A_1), \gamma_{F_j, A_2}(\vec{x}_j))) \), or if after doing
$A_1, A_2$ causes the fluent to be true ($R((A_1), \gamma^+_{F, A_2}(\tilde{t}_j))$). As an example, for the chaining of the effects of $\text{drop}(b)$ and $\text{fix}(o)$ and the successor state axiom of the fluent $\text{broken}$, $\gamma^+$ looks as follows:

$$
\begin{align*}
\begin{array}{l}
\gamma^+_{\text{broken}, \text{drop}(b), \text{fix}(o)}(o') \\
= \gamma^+_{\text{broken}, \text{drop}(b)}(o') \land \\
\neg R((\text{drop}(b)), \gamma^+_{\text{broken}, \text{fix}(o)}(o')) \\
\lor R((\text{drop}(b)), \gamma^+_{\text{broken}, \text{fix}(o)}(o'))
\end{array}
\end{align*}
$$

We use the chaining $C$ to define the effect $\epsilon_{\sigma}$ of the macro:

**Definition 4.** We define the macro effect $\epsilon_{\sigma}$ inductively:

$$
\epsilon(A_1) = \epsilon_{A_1}
$$

$$
\epsilon(A_1, A_2, \ldots, A_n) = C(\epsilon(A_1, A_2, \ldots, A_{n-1}); \epsilon_{A_n})
$$

With precondition $\pi_{\sigma}$ and effect $\epsilon_{\sigma}$, we can define the ADL macro operator for $\sigma$:

**Definition 5 (Macro Operator).** We call the ADL operator $O_{\sigma} = \langle A_0, \{\tilde{t}_1, \tilde{t}_2, \ldots, \tilde{t}_{n-1}\}, \pi_{\sigma}, \epsilon_{\sigma} \rangle$ the macro operator for $\sigma$.

We do not require the $\tilde{y}_{A_i}$ to be distinct, i.e., we can assign a parameter of $A_i$ to the same name as another parameter of $A_j$, as long as they are of the same type. We denote the distinct joint parameters of $O_{\sigma}$ as $\tilde{y}_{A_i}$. The PDDL representation of the synthesized operator for the sequence $(\text{drop}, \text{fix})$ is shown in Listing 2. We show that $O_{\sigma}$ has the same effect as $\sigma$:

**Lemma 1.** Given an ADL problem description $A$ with corresponding basic action theory $\Sigma$. Let $\tilde{t}_1, \ldots, \tilde{t}_n$ be ground terms of type $\tilde{t}_1, \tilde{t}_2, \ldots, \tilde{t}_{n-1}$, and $O_\sigma$ the corresponding macro operator for $\sigma = \langle A_1, \ldots, A_n \rangle$ with operator name $A_0$. Let $\Sigma_\sigma$ be the same basic action theory as $\Sigma$ but with additional ADL (macro) operators $O_0(A_1, A_2), O_0(A_1, A_2, \ldots, A_n)$. For any bounded sentence $\phi$:

$$
\begin{align*}
\Sigma_{\sigma} \models [A_{\sigma}(\tilde{t}_1, \ldots, \tilde{t}_n)] \alpha & \Leftrightarrow \\
\Sigma \models [A_0(\tilde{t}_1)][A_2(\tilde{t}_2)] \ldots [A_n(\tilde{t}_n)] \alpha
\end{align*}
$$

**Proof.** Proof by induction over the length of $\phi$ for atomic formulas $\alpha$ and then by structural induction over $\alpha$. We denote $A_1(\tilde{t}_1)$ as $A_i$ and $A_0(\tilde{t}_0)$ as $A_0$.

**Base case** Let $F(\tilde{t})$ be a ground atomic formula.

$$
\Sigma \models [a_1] \ldots [a_n] F(\tilde{t}).
$$

**Induction step** Let $\Gamma(\tilde{t})$ be a ground atomic formula.

$$
\Sigma \models [a_1] \ldots [a_{n-1}] (F(\tilde{t}) \land \neg \gamma^-_F(\tilde{t}))
$$

We conclude $\Sigma \models [a_1] \ldots [a_{n-1}] \neg \gamma^-_F(\tilde{t})$. We thus have the same effects as the corresponding ground action sequence $(A_1(\tilde{f}_1), \ldots, A_n(\tilde{f}_n))$:
Theorem 3. For any bounded sentence $\alpha$:
\[ \Sigma_a \models [A_0(\vec{t}_1), \ldots, \vec{t}_n] \alpha \iff \Sigma \models [A_0(\vec{t}_1), \ldots, A_n(\vec{t}_n)] \alpha \]

Proof. \( \implies \)
Assume $\Sigma_a \models [A_0(\vec{t}_1), \ldots, \vec{t}_n] \alpha$. Let $w_x$ be a world with $w_x \models \Sigma_a$, therefore $w_x \models w_{x_n}$, and thus by Lemma 1:
\[ w_{x_n} \models [A_0(\vec{t}_1), \ldots, A_n(\vec{t}_n)] \alpha \]

Assume $\Sigma \models [A_0(\vec{t}_1), \ldots, A_n(\vec{t}_n)] \alpha$. Then there is a world $w_{x_n} \models \Sigma$ but $w_{x_n} \not\models [A_0(\vec{t}_1), \ldots, A_n(\vec{t}_n)] \alpha$. Contradiction to the uniqueness of $w_{x_n}$.

\[ \Longleftrightarrow \]

5 Macro Planning

We extended the implementation of DBMP/S [20] with ADL macros. DBMP/S generates macro-augmented domains as follows:
1. Identify frequent action sequences in a plan database,
2. Generate macro operators for those sequences and add them to the domain,
3. Select the macro-augmented domains that are most promising.

From extending the planner to ADL, we also modified the macro selection. We score a macro $O$ with two properties:

1. The normalized frequency $f(O)$ of the macro in the training solutions. Let $n$ be the number of occurrences of the sequence $\vec{t}$ and $f$ the total number of actions in the database. The frequency of $\sigma$ is defined as $f(\sigma) = \frac{n}{f}$. 
2. The number of joint parameters in the operators of the macro. The parameter reduction $r(O_a)$ is defined by
\[ r(O_a) = \frac{\sum_{A_i(\vec{t}_i) \in \sigma} |\vec{t}_i| - |\vec{t}'_i|}{\sum_{A_i(\vec{t}_i) \in \sigma} |\vec{t}'_i|} \]

As an example, the parameter reduction for $\sigma_1 = \langle \text{drop}(b), \text{fix}(o), \text{drop}(b) \rangle$ (dropping the same bag twice) is $r(O_{\sigma_1}) = \frac{1}{4}$, because the two parameters of $\text{drop}$ are replaced by the common parameter $b$. On the other hand, for $\sigma_2 = \langle \text{drop}(b_1), \text{fix}(o), \text{drop}(b_2) \rangle$ (i.e., dropping two possibly different bags), $r(O_{\sigma_2}) = 0$, as the number of parameters is the same as in the original sequence. The operator $O_{\sigma_2}$ is more general but also has more parameters, possibly resulting in a larger search space.

For macro-augmented domains, we use the complementarity of the domain macros as an additional property. Let $M = \{O_{\sigma_1}, \ldots, O_{\sigma_n}\}$ be the macros of the domain and $\sigma_i = \langle A_{i_1}, \ldots, A_{i_m} \rangle$. The complementarity of $M$ is defined as
\[ C(M) = \frac{\bigcup_{i=1}^{n} \bigcup_{A_i \in \sigma_i} \{A_i\}}{\sum_{i=1}^{n} |\bigcup_{A_i \in \sigma_i} \{A_i\}|} \]

Intuitively, the complementarity is a measure for the difference of the macros in the domain, corresponding to a higher difference of the macro operators.

Using those properties, we define an evaluator
\[ E(M) = |M|^{-w_x} C(M) \sum_{O \in M} w_f f(O) + (1 - w_f) r(O) \]

where $w_f, w_x, w_c$ are weights from $[0, 1]$. The higher $w_f$, the higher we prefer domains with a small number of macros. The higher $w_x$, the more we use frequently occurring macros at the cost of less parameter reduction. The higher $w_c$, the higher we penalize macros that have the same actions.

1 The code is open-source and available at https://github.com/morxa/dbmp.

5.1 Evaluation

As benchmark domains, we used the STRIPS and ADL domains from the IPC-14 and IPC-18, in addition to a modified version of the ADL robotics domain Cleanup [13, 19]. We excluded domains that require functions or action costs, and we excluded Maintenance, as it consists of only one action. We generated macros using the DBMP/S framework with the ADL operator synthesis and the modified selection procedure as described above. As comparison, we used FF [17] and LAMA [29] without macro actions, in addition to MACROFF. Since our focus is on ADL, we did not include any STRIPS macro planner. As our primary focus is an application in robotics, we followed the rules of the IPC-18 Agile Track for scoring, which specifies a time limit of 5 min and a memory limit of 8 GB. For each task, the planner scores 1 point if the run-time $t$ was less than $1 \sec$, $1 - \frac{\log t}{\log 300}$ points if $1 \leq t \leq 300$, and 0 points otherwise.

For each domain, we split the problem set into sets for training (25%), validation (25%), and testing (50%). The training set is used for generating macro actions, the validation set is used for evaluator parameter tuning, and the test set is used for the final evaluation. We used the smallest problems for the training set and randomly assigned the remaining problems to the other sets. We generated solutions for the training set with the optimal variant of LAMA with a time limit of 60 min and a memory limit of 8 GB. Based on these solutions, we created macros consisting of up to 4 actions and macro-augmented domains containing up to 4 macros. In the next step, we selected the best macro-augmented domain for each possible configuration of $(w_f, w_x, w_c) \in \{0.0, 0.1, \ldots, 1.0\}^3$ and ran the planners FF and LAMA (2011) — modified to terminate when the first solution is found — with the augmented domains on the validation set.

Table 1. The weights for frequency ($w_f$), complementarity ($w_x$), and number of macros ($w_c$) of the evaluators which performed best on problems of the validation set and which were used on the test set.

<table>
<thead>
<tr>
<th>Domain</th>
<th>$w_f$</th>
<th>$w_x$</th>
<th>$w_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocksworld</td>
<td>0.1</td>
<td>0.30</td>
<td>0.00</td>
</tr>
<tr>
<td>Caldera</td>
<td>0.02</td>
<td>0.09</td>
<td>0.00</td>
</tr>
<tr>
<td>Narkabe</td>
<td>0.5</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Terms</td>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Barman</td>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Childsnack</td>
<td>0.0</td>
<td>0.6</td>
<td>0.1</td>
</tr>
<tr>
<td>Hiking</td>
<td>0.0</td>
<td>0.9</td>
<td>0.6</td>
</tr>
<tr>
<td>Thoughtful</td>
<td>0.0</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>Cleanup</td>
<td>0.9</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 2. Planner scores on the validation set using the DBMP macro domains with the best configurations according to Table 1.

<table>
<thead>
<tr>
<th>Domain</th>
<th>FF</th>
<th>LAMA</th>
<th>MACROFF</th>
<th>DBMP</th>
<th>DBMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocksworld</td>
<td>8.87</td>
<td>15.00</td>
<td>11.80</td>
<td>10.99</td>
<td>14.82</td>
</tr>
<tr>
<td>Caldera</td>
<td>0.85</td>
<td>1.35</td>
<td>0.75</td>
<td>0.85</td>
<td>2.06</td>
</tr>
<tr>
<td>Narkabe</td>
<td>0.00</td>
<td>0.73</td>
<td>0.00</td>
<td>0.90</td>
<td>0.76</td>
</tr>
<tr>
<td>Terms</td>
<td>0.00</td>
<td>2.00</td>
<td>0.00</td>
<td>1.83</td>
<td>2.00</td>
</tr>
<tr>
<td>Barman</td>
<td>0.00</td>
<td>4.00</td>
<td>0.00</td>
<td>4.36</td>
<td>4.96</td>
</tr>
<tr>
<td>Childsnack</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>3.97</td>
<td>3.90</td>
</tr>
<tr>
<td>Hiking</td>
<td>0.65</td>
<td>1.87</td>
<td>1.29</td>
<td>3.01</td>
<td>3.54</td>
</tr>
<tr>
<td>Thoughtful</td>
<td>1.55</td>
<td>1.82</td>
<td>3.06</td>
<td>2.95</td>
<td>3.00</td>
</tr>
<tr>
<td>Cleanup</td>
<td>1.00</td>
<td>0.70</td>
<td>0.00</td>
<td>4.11</td>
<td>0.73</td>
</tr>
</tbody>
</table>

Based on the validation results, we chose the configuration with the highest total score for each domain-planner pair. The chosen configurations are shown in Table 1, the scores on the evaluation set...
A plan library is a collection of pre-computed plans, which are either hand-crafted. The preconditions of the plan actions are checked during execution to make sure the actions are actually executable, and the actions’ effects are applied on the agent’s world model. However, the decision criteria when to use which plan is manually engineered. Thus, a plan may be selected even though it is not executable or does not accomplish the desired goal. By using macro operator synthesis, we can compute the preconditions and effects of the overall plan and check them before selecting a plan. Listing 3 shows an example for such a plan from the library: In this plan, a robot moves to a machine (move), gets a workpiece from the shelf (wp-get-shelf), and feeds it into the machine with wp-put, whose action definition is shown in Listing 4. To ensure that no other robot uses the machine at the same time, it locks the location before moving there (location-lock) and unlocks it when it has finished the other actions (location-unlock). Listing 5 shows the macro operator generated from the plan.

### 7 Conclusion

We introduced a formal method to synthesize an ADL macro operator from an ADL operator sequence and showed that the resulting operator has the same preconditions and effects as the respective sequence, using an ADL semantics based on the Situation Calculus. The presented approach is the first synthesis method for ADL macro operators and is able to synthesize operators with quantified preconditions and effects, disjunctive preconditions, and conditional effects. By representing a macro operator as a regular PDDL operator, we are able to use off-the-shelf planners such as LAMA without any modifications to the planner. We applied the synthesis method to macro
planning using the macro planner DBMP and showed that it outperforms MACROFF, a state-of-the-art ADL macro planner, but it could not always improve the performance of LAMA. Finally, we described that ADL macro operators can also be useful for maintaining a plan library by computing the precondition and effects of the plans in the library, and for generating such a plan library by generalizing a set of observed plans.

Acknowledgments

T. Hofmann is supported by the DFG grant GL-747/23-1 on Constraint-based Transformations of Abstract Task Plans into Executable Actions for Autonomous Robots.2

REFERENCES


2 http://gepris.dfg.de/gepris/projekt/288705857