Macro Operator Synthesis for ADL Domains

Till Hofmann[1] and **Tim Niemueller**[1,2] and **Gerhard Lakemeyer**[1]

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 how to compute the parameterized preconditions and effects of a macro operator such that they are equivalent to the preconditions and effects of the respective action sequence and prove the correctness of the synthesized macro operators based on a Situation Calculus 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

```
(:action drop :parameters (?b - bag)
  :precondition (carrying ?b)
  :effect (and (not (carrying ?b))
   (forall (?o - obj)
     (when (and (in ?o ?b) (fragile ?o))
       (broken ?o)))))
(:action fix :parameters (?o - obj)
  :precondition (broken ?o)
  :effect (not (broken ?o)))
```

Listing 1. An extract from a simple PDDL domain of a robot carrying a bag with potentially fragile objects.

```
; MACRO [drop,fix] PARAMETERS [[1],[2]]
(:action drop-fix :parameters (?p1 - bag ?p2 - obj)
 :precondition (and (carrying ?p1)
    (or (and (in ?p2 ?p1) (fragile ?p2))
        (broken ?p2)))
 :effect (and
    (not (broken ?p2)) (not (carrying ?p1))
    (forall (?o - obj)
      (when (and (in ?o ?p1) (fragile ?o)
                 (not (= ?o ?p2)))
           (broken ?o)))))
```

Listing 2. The macro operator consisting of the actions `drop` and `fix`.

already, 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 in $\mathcal{ES}$ [9], a dialect of the Situation Calculus, 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-

[1] Knowledge-Based Systems Group, RWTH Aachen University, Germany, email: (hofmann,lakemeyer)@kbsg.rwth-aachen.de
[2] T. Niemueller is now with X, The Moonshot Factory, email: timdn@x.team

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]. MACROFF [2] uses two different approaches to macro planning: *Component Abstraction-Enhanced Domains* (CA-ED) and a *Solution-Enhanced Planner* (SOL-EP). CA-ED supports only STRIPS domains. SOL-EP represents a macro as a partially ordered sequence of operators [3], which can be applied in one step during search. SOL-EP supports ADL domains. *MUM* [8] is a STRIPS macro planner that learns macros from the solutions of less complex training problems and ranks macros based on the concept of *outer entanglements* [6] and *independent actions* [5]. BLOMA [7] decomposes plans into blocks, sub-sequences that must not be interleaved with other actions. From those blocks, BLOMA generates macro actions, which can again be represented as PDDL actions. BLOMA is restricted to STRIPS. DBMP/S [20] determines macro operators by analyzing a plan database for frequent action sequences. In contrast to MACROFF, it synthesizes an operator from the action sequence and represents the macro as regular PDDL operator. DBMP/S only supports the STRIPS fragment.

Similar to the proposed operator synthesis method, Rintanen [30] proposes to use regression to compute the preconditions and effects of an ADL action sequence. However, in contrast to the proposed method, they only allow grounded action sequences, which is not sufficient to generate macro operators or to maintain a plan library.

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 $\wedge, \neg, \forall, \Box, [\cdot]$. 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 α is a formula, then $[t]\alpha$ is a formula; (4) If α and β are formulas, then so are $(\alpha \wedge \beta)$, $\neg\alpha$, $\forall x.\, \alpha$, $\Box\alpha$.

We read $[t]\alpha$ as "α holds after action t" and $\Box\alpha$ as "α holds after any sequence of actions". As usual, we treat $\exists.\, \alpha$, $(\alpha \vee \beta)$, $\alpha \supset \beta$, and $(\alpha \equiv \beta)$ 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 $Poss$ is called *fluent*.

Semantics Let P denote the set of all pairs $\sigma{:}\rho$ where $\sigma \in R^*$ is considered a sequence 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 $\mathcal{L}$ [23].

Given a world w, for any formula α with no free variables, we write $w \models \alpha$ instead of $w, \langle\rangle \models \alpha$ where $\langle\rangle$ 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 and r_2 are identical
$w, \sigma \models (\alpha \wedge \beta)$	iff $w, \sigma \models \alpha$ and $w, \sigma \models \beta$
$w, \sigma \models \neg\alpha$	iff $w, \sigma \not\models \alpha$
$w, \sigma \models \forall x.\, \alpha$	iff $w, \sigma \models \alpha^x_r$ for every $r \in R$
$w, \sigma \models [r]\,\alpha$	iff $w, \sigma \cdot r \models \alpha$
$w, \sigma \models \Box\alpha$	iff $w, \sigma \cdot \sigma'$ for every $\sigma' \in R^*$

The notation α^x_t means the result of simultaneously replacing all free occurrences of the variable x by the term t; $\sigma_1 \cdot \sigma_2$ denotes the concatenation of the two action sequences.

Basic Action Theories Given a set $\mathcal{F}$ of fluent predicates, a set of sentences Σ is called a *basic action theory over* $\mathcal{F}$ iff it only mentions the fluents in $\mathcal{F}$ and is of the form $\Sigma = \Sigma_0 \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}}$, where (1) Σ_0 is a finite set of fluent sentences; (2) Σ_{pre} is a singleton of the form $\Box Poss(a) \equiv \pi$, where π is fluent with a being the only free variable; (3) Σ_{post} is a finite set of successor state axioms of the form $\Box[a]F(\vec{x}) \equiv \gamma_F$, one for each fluent $F \in \mathcal{F} \setminus \{Poss\}$, where γ_F is a fluent sentence whose free variables are among $\vec{x}$ and a. Σ_0 represents the initial database, Σ_{pre} is one large precondition axiom, and Σ_{post} the set of successor state axioms for all fluents in $\mathcal{F}$.

Regression Given a basic action theory Σ, a common task is *projection*, i.e., determining what holds after a sequence of actions has occurred. For a given basic action theory Σ, ground terms $r_1, \ldots, r_k$, and an arbitrary sentence α, the projection task is to determine whether $\Sigma \models [r_1] \ldots [r_k]\alpha$ holds. One method to do projection is to use *regression*. The idea of regression is to successively replace fluents in α 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 α is considered regressable. The regression of α over a sequence of (not necessarily ground) terms σ is denoted as $\mathcal{R}[\sigma, \alpha]$. As σ does not need to be ground, we can also regress a sentence over a sequence of operators $A_1 \dots A_n$, which we will use to define the preconditions and effects of a macro operator. The regression $\mathcal{R}[\alpha]$ of α wrt Σ is defined to be the fluent formula $\mathcal{R}[\langle\rangle, \alpha]$, where for any sequence of terms σ, $\mathcal{R}[\sigma, \alpha]$ is defined inductively on α:

1. $\mathcal{R}[\sigma, (t_1 = t_2)] = (t_1 = t_2)$;
2. $\mathcal{R}[\sigma, \forall x\, \alpha] = \forall x\, \mathcal{R}[\sigma, \alpha]$;
3. $\mathcal{R}[\sigma, (\alpha \wedge \beta)] = (\mathcal{R}[\sigma, \alpha] \wedge \mathcal{R}[\sigma, \beta])$;
4. $\mathcal{R}[\sigma, \neg\alpha] = \neg\mathcal{R}[\sigma, \alpha]$;
5. $\mathcal{R}[\sigma, [t]\alpha] = \mathcal{R}[\sigma \cdot t, \alpha]$;
6. $\mathcal{R}[\sigma, Poss(t)] = \mathcal{R}[\sigma, \pi^a_t]$;
7. $\mathcal{R}[\sigma, F(t_1, \dots, t_k)]$ is defined inductively on σ by
   (a) $\mathcal{R}[\langle\rangle, F(t_1, \dots, t_k)] = F(t_1, \dots, t_k)$;
   (b) $\mathcal{R}[\sigma \cdot t, F(t_1, \dots, t_k)] = \mathcal{R}[\sigma, (\gamma_F)^{a\, x_1}_{t\, t_1} \cdots {}^{x_k}_{t_k}]$

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

1. For $F \notin \mathcal{F}$, $w_\Sigma[\sigma{:}F(\vec{r})] = w[\sigma{:}F(\vec{r})]$;
2. For $F \in \mathcal{F}$, $w_\Sigma[\sigma{:}F(\vec{r})]$ is defined inductively:
   (a) $w_\Sigma[\langle\rangle{:}F(\vec{r})] = w[\langle\rangle{:}F(\vec{r})]$
   (b) $w_\Sigma[\sigma \cdot r{:}F(\vec{r})] = 1$ iff $w_\Sigma, \sigma \models (\gamma_F)^{a\, \vec{x}}_{r\, \vec{r}}$
3. $w_\Sigma[\sigma{:}Poss(r)] = 1$ iff $w_\Sigma, \sigma \models \pi^a_r$

As shown in [22], for any w, w_Σ exists and is unique.

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

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

3.2 ADL Semantics

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

ADL Operators First, we describe how we can define ADL operators consisting of a precondition formula and an effect formula in $\mathcal{ES}$. An *ADL precondition formula* is an $\mathcal{ES}$ formula of the following form: (1) An atomic formula $F(\vec{t})$ is a precondition formula if each of the t_i is either a variable or a constant. (2) An equality atom $(t_1 = t_2)$ is a precondition formula if each t_i is a variable or a constant. (3) If ϕ_1 and ϕ_2 are precondition formulas, then so are $\phi_1 \wedge \phi_2$, $\neg\phi_1$, and $\forall x{:}\tau\, \phi_1$.

The quantifier $\forall x{:}\tau$ stands for "all x of type τ", where a type τ is a unary predicate from F^1 and $\tau(x)$ means that object x has type τ. The quantifier $\forall x{:}\tau$ is defined as $\forall x{:}\tau\, \phi \overset{def}{=} \forall x\, \tau(x) \supset \phi$. Furthermore, we denote tuples of terms and types by vectors, e.g., $\vec{t}$ and $\vec{\tau}$. If $\vec{\tau}$ denotes $\tau_1, \dots, \tau_k$, $\vec{r}$ denotes $r_1, \dots, r_k$ and $\vec{t}$ denotes $t_1, \dots, t_k$, then we use the following short-hand notations: (1) $(\vec{r} = \vec{t}) \overset{def}{=} (r_1 = t_1) \wedge \dots \wedge (r_k = t_k)$, (2) $\vec{\tau}(\vec{t}) \overset{def}{=} \tau_1(t_1) \wedge \dots \wedge \tau_k(t_k)$.

An *ADL effect formula* is an $\mathcal{ES}$ formula of the following form: (1) An atomic formula $F(\vec{t})$ is an effect formula if each of the t_i is either a variable or a constant. (2) A negated atomic formula $\neg F(\vec{t})$ is an effect formula if each of the t_i is either a variable or a constant. (3) If ψ_1 and ψ_2 are effect formulas, then $\psi_1 \wedge \psi_2$ and $\forall x{:}\tau.\ \psi_1$ are effect formulas. (4) If γ is a precondition formula and ψ is an effect formula not containing "$\Rightarrow$" and "$\forall$", then $\gamma \Rightarrow \psi$ is an effect formula.

An *ADL operator* O is given by a quadruple $(A, \vec{y}{:}\vec{\tau}, \pi_A, \epsilon_A)$, where (1) A is a symbol from G^p, with $p = |\vec{y}|$, (2) $\vec{y}{:}\vec{\tau}$ is a list of variable symbols with associated types, (3) π_A is a precondition formula with free variables from $\vec{y}$, (4) and ϵ_A is an effect formula with free variables from $\vec{y}$. We call A the operator name and $\vec{y}{:}\vec{\tau}$ the parameters of O. An ADL operator $(A, \vec{y}{:}\vec{\tau}, \pi_A, \epsilon_A)$ is in *normal form*, if its effect ϵ_A is of the following form:

$$\bigwedge_{F_j} \forall \vec{x}_j{:}\vec{\tau}_{F_j}.\ \left(\gamma^+_{F_j,A}(\vec{x}_j) \Rightarrow F_j(\vec{x}_j)\right) \wedge$$
$$\bigwedge_{F_j} \forall \vec{x}_j{:}\vec{\tau}_{F_j}.\ \left(\gamma^-_{F_j,A}(\vec{x}_j) \Rightarrow \neg F_j(\vec{x}_j)\right)$$

If an ADL operator is in normal form, then for each F_j, there is at most one positive effect formula of the form $\cdots \Rightarrow F_j(\vec{x})$ and at most one negative effect formula of the form $\cdots \Rightarrow \neg F_j(\vec{x})$. The ADL operator for `drop` is $O_{drop} = (drop, b{:}bag, \pi_{drop}, \epsilon_{drop})$ with $\pi_{drop} = carrying(b)$ and $\epsilon_{drop} = \forall o{:}obj\ [(in(o, b) \wedge fragile(o)) \supset broken(o)] \wedge \neg carrying(b)$.

ADL Problem Description Using the definitions above, we can now describe how we can formulate an ADL problem description in $\mathcal{ES}$. A problem description for ADL is given by (1) a finite list of types $\tau_1, \dots, \tau_l$, *Object*, where *Object* is a special type that must always be included, (2) a finite list of statements of the form $\tau_i(\text{either}\ \tau_{i_1} \dots \tau_{i_{k_i}})$ defining some of the types as compound types, where τ_i is the union of all τ_{i_j} and k_i is the number of sub-types of τ_i; a *primitive type* is a type other than *Object* that does not occur on the left-hand side of such a definition, (3) a finite list of fluent predicates $F_1, \dots, F_n$ with a list of types $\tau_{j_1}, \dots \tau_{j_{k_j}}$ for each F_j, which defines the types of the arguments of F_j, (4) a finite list of objects with associated primitive types $o_1{:}\tau_{o_1}, \dots, o_k{:}\tau_{o_k}$, where each o_i is a symbol from G^0, (5) a finite list of ADL operators $O_1, \dots, O_m$ with $O_i = (A_i, \vec{y}_i{:}\vec{\tau}_i, \pi_{A_i}, \epsilon_{A_i})$, 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), and (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 $\mathcal{ES}$ basic action theory can be constructed as follows:

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

$$\gamma^+_{F_j} \overset{def}{=} \bigvee_{\gamma^+_{F_j,A_i} \in NF(O_i)} \exists \vec{y}_i.\ a = A_i(\vec{y}_i) \wedge \gamma^+_{F_j,A_i}$$
$$\gamma^-_{F_j} \overset{def}{=} \bigvee_{\gamma^-_{F_j,A_i} \in NF(O_i)} \exists \vec{y}_i.\ a = A_i(\vec{y}_i) \wedge \gamma^-_{F_j,A_i}$$

Using the definitions for $\gamma^\pm_{F_j,A_i}$, we can define the successor state axiom for F_j (cf., [28]).

$$\square\, [a]\, F_j(\vec{x}_j) \equiv \gamma^+_{F_j} \wedge \vec{\tau}_{F_j}(\vec{x}_j) \vee F_j(\vec{x}_j) \wedge \neg\gamma^-_{F_j}$$

In our example, the successor state axiom for $broken$ is:

$$\begin{aligned}&\square\,[a]\,broken(o) \equiv \\ &\quad \exists b.\, a = drop(b) \wedge (in(o,b) \wedge fragile(o)) \wedge obj(o) \\ &\quad \vee broken(o) \wedge \neg a = fix(o)\end{aligned}$$

The Precondition Axiom Σ_{pre} The precondition axiom π is a disjunction over all m operators of the problem domain:

$$\pi \overset{def}{=} \bigvee_{1\leq i\leq m} \exists \vec{y}_i{:}\vec{\tau}_i.\, a = A_i(\vec{y}_i) \wedge \pi_{A_i}$$

In our example, the precondition axiom is:

$$\begin{aligned}\pi =&\exists b{:}bag.\, a = drop(b) \wedge carrying(b) \\ &\vee \exists o{:}obj.\, a = fix(o) \wedge broken(o)\end{aligned}$$

Initial Description Σ_0 The initial description Σ_0 is a conjunction of fluent formulas describing the initial state of the world and all information about the types of objects, e.g.,:

$$\Sigma_0 = bag(b_1) \wedge obj(o_1) \wedge carrying(b_1) \wedge in(o_1, b_1)$$

4 ADL Macro Operator Synthesis

Using $\mathcal{ES}$ and the ADL semantics described above, we can now define ADL macro operators. For a given sequence σ of ADL operator names, we construct a new macro operator O_σ whose precondition formula is satisfied iff all preconditions of σ are satisfied when applied subsequently, and whose effect is the same as the observed accumulated effect after applying σ. In the following, let $\sigma = \langle A_1, \ldots, A_n\rangle$ be a non-empty sequence of ADL operator names with corresponding operators $O_i = (A_i, \vec{y}_{A_i}{:}\vec{\tau}_{A_i}, \pi_{A_i}, \epsilon_{A_i})$.

Macro Preconditions

For the sequence σ, we need to compute a precondition formula π_σ for the corresponding macro operator O_σ. Intuitively, for any grounding $\rho = \sigma(\vec{t})$ of σ, $w \models \pi_\sigma(\vec{t})$ must hold iff ρ is executable, i.e., iff it is possible to execute all actions of ρ 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 $O_1, \ldots, O_l$ and corresponding basic action theory Σ. Let $\rho = \langle a_1, \ldots, a_n\rangle = \langle A_1(\vec{t}_1), \ldots, A_n(\vec{t}_n)\rangle$ be a ground sequence of actions with preconditions $\pi_{a_i} = \pi_{A_i}(\vec{t}_i)$. We say ρ is* executable in world w *iff the following holds:*

$$w \models \pi_{a_1} \wedge [a_1]\big(\pi_{a_2} \wedge [a_2](\pi_{a_3} \wedge \ldots [a_{n-1}](\pi_{a_n})\big)$$

Next, we define the precondition formula of the macro operator O_σ.

Definition 2 (Macro Precondition)**.** *The macro precondition π_σ is defined inductively:*

$$\begin{aligned}\pi_{\langle A_n\rangle} &= \pi_{A_n} \\ \pi_{\langle A_1, A_2, \ldots, A_n\rangle} &= \pi_{A_1} \wedge \mathcal{R}(\langle A_1\rangle, \pi_{\langle A_2, \ldots, A_n\rangle})\end{aligned}$$

Note that we define the precondition by regressing over the non-ground operator A_i, i.e., $\pi_{\langle A_1, \ldots, A_n\rangle}$ 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 $\pi_{\langle A_1, \ldots, A_n\rangle}(\vec{t}_1, \ldots, \vec{t}_n)$ as $\pi_{\langle a_1, \ldots, a_n\rangle}$. Coming back to our example, the macro precondition of the sequence $\sigma = \langle drop(b), fix(o)\rangle$ is:

$$\begin{aligned}\pi_\sigma &= \pi_{drop} \wedge \mathcal{R}(\langle drop(b)\rangle, \pi_{fix}) \\ &= carrying(b) \wedge \mathcal{R}(\langle drop(b)\rangle, broken(o)) \\ &= carrying(b) \wedge (broken(o) \vee in(o,b) \wedge fragile(o))\end{aligned}$$

The macro precondition π_σ satisfies the desired property:

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

$$\Sigma_0 \models \pi_{\langle a_1, \ldots, a_n\rangle} \Leftrightarrow \Sigma \models \pi_{a_1} \wedge [a_1]\big(\pi_{a_2} \wedge \ldots [a_{n-1}]\pi_{a_n}\big)$$

Proof. By Regression Theorem:
$\Sigma \models \pi_{a_1} \wedge [a_1]\big(\pi_{a_2} \wedge \ldots [a_{n-1}]\pi_{a_n}\big)$ iff

$$\Sigma_0 \models \mathcal{R}\big[\pi_{a_1} \wedge [a_1]\pi_{a_2} \wedge \ldots [a_{n-1}]\pi_{a_n}\big]$$

where

$$\begin{aligned}&\mathcal{R}\big[\pi_{a_1} \wedge [a_1]\pi_{a_2} \wedge \ldots [a_{n-1}]\pi_{a_n}\big] \\ &= \pi_{a_1} \wedge \mathcal{R}\big[[a_1]\pi_{a_2} \wedge \ldots [a_{n-1}]\pi_{a_n}\big] \\ &= \pi_{a_1} \wedge \mathcal{R}\big[\langle a_1\rangle, \pi_{a_2} \wedge \ldots \wedge [a_{n-1}]\pi_{a_n}\big] \\ &= \pi_{a_1} \wedge \mathcal{R}[\langle a_1\rangle, \pi_{a_2}] \wedge \mathcal{R}[\langle a_1\rangle, [a_2](\pi_{a_3} \wedge \ldots \wedge [a_{n-1}]\pi_{a_n})] \\ &= \pi_{a_1} \wedge \mathcal{R}[\langle a_1\rangle, \pi_{a_2}] \wedge \mathcal{R}[\langle a_1, a_2\rangle, \pi_{a_3} \wedge \ldots \wedge [a_{n-1}]\pi_{a_n}] \\ &= \ldots \\ &= \pi_{a_1} \wedge \mathcal{R}[\langle a_1\rangle, \pi_{a_2}] \wedge \mathcal{R}[\langle a_1, a_2\rangle, \pi_{a_3}] \wedge \ldots \\ &\quad \wedge \mathcal{R}[\langle a_1, a_2, \ldots, a_{n-1}\rangle, \pi_{a_n}] \\ &= \pi_{\langle a_1, \ldots, a_n\rangle}\end{aligned}$$

□

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

Macro Effects

For the sequence σ of ADL operator names, we need to compute the accumulated effect of σ. 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 $O_i = (A_i, \vec{y}_{A_i}{:}\vec{\tau}_{A_i}, \pi_{A_i}, \epsilon_{A_i})$. We define the chaining $\mathcal{C}(\epsilon_{A_1}, \epsilon_{A_2})$ of O_1 with O_2 as:*

$$\begin{aligned}\mathcal{C}(\epsilon_{A_1}, \epsilon_{A_2}) = &\bigwedge_{F_j} \forall \vec{x}_j{:}\vec{\tau}_{F_j}\, \Big(\gamma^+_{F_j,\langle A_1, A_2\rangle}(\vec{x}_j) \Rightarrow F_j(\vec{x}_j)\Big) \wedge \\ &\bigwedge_{F_j} \forall \vec{x}_j{:}\vec{\tau}_{F_j}\, \Big(\gamma^-_{F_j,\langle A_1, A_2\rangle}(\vec{x}_j) \Rightarrow \neg F_j(\vec{x}_j)\Big)\end{aligned}$$

where

$$\begin{aligned}&\gamma^+_{F_j,\langle A_1, A_2\rangle}(\vec{x}_j) = \gamma^+_{F_j,A_1}(\vec{x}_j) \wedge \\ &\qquad \neg\mathcal{R}(\langle A_1\rangle, \gamma^-_{F_j,A_2}(\vec{x}_j)) \vee \mathcal{R}(\langle A_1\rangle, \gamma^+_{F_j,A_2}(\vec{x}_j)) \\ &\gamma^-_{F_j,\langle A_1, A_2\rangle}(\vec{x}_j) = \gamma^-_{F_j,A_1}(\vec{x}_j) \wedge \\ &\qquad \neg\mathcal{R}(\langle A_1\rangle, \gamma^+_{F_j,A_2}(\vec{x}_j)) \vee \mathcal{R}(\langle A_1\rangle, \gamma^-_{F_j,A_2}(\vec{x}_j))\end{aligned}$$

The subformula $\gamma^+_{F_j,\langle A_1, A_2\rangle}(\vec{x}_j)$ defines when the chained actions $\langle A_1, A_2\rangle$ 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 ($\neg\mathcal{R}(\langle A_1\rangle, \gamma^-_{F_j,A_2}(\vec{x}_j))$), or if after doing

A_1, A_2 causes the fluent to be true ($\mathcal{R}(\langle A_1\rangle, \gamma^+_{F_j,A_2}(\vec{x}_j))$). As an example, for the chaining of the effects of $drop(b)$ and $fix(o)$ and the successor state axiom of the fluent *broken*, γ^+ looks as follows:

$$\begin{aligned}
&\gamma^+_{broken,\langle drop(b),fix(o)\rangle}(o')\\
&= \gamma^+_{broken,drop(b)}(o') \wedge\\
&\quad \neg\mathcal{R}\big(\langle drop(b)\rangle, \gamma^-_{broken,fix(o)}(o')\big)\\
&\quad \vee \mathcal{R}\big(\langle drop(b)\rangle, \gamma^+_{broken,fix(o)}(o')\big)\\
&= in(o',b) \wedge fragile(o')\\
&\quad \wedge \neg\mathcal{R}(\langle drop(b)\rangle, o = o') \vee \mathcal{R}(\langle drop(b)\rangle, \bot)\\
&= in(o',b) \wedge fragile(o') \wedge o \neq o'
\end{aligned}$$

We use the chaining $\mathcal{C}$ to define the effect ϵ_σ of the macro:

Definition 4. *We define the* macro effect ϵ_σ *inductively:*

$$\begin{aligned}
\epsilon_{\langle A_1\rangle} &= \epsilon_{A_1}\\
\epsilon_{\langle A_1,A_2\ldots,A_n\rangle} &= \mathcal{C}(\epsilon_{\langle A_1,A_2,\ldots,A_{n-1}\rangle}, \epsilon_{A_n})
\end{aligned}$$

With precondition π_σ and effect ϵ_σ, we can define the ADL macro operator for σ:

Definition 5 (Macro Operator)**.** *We call the ADL operator* $O_\sigma = (A_\sigma, (\vec{y}_{A_1}{:}\vec{\tau}_{A_1}, \ldots, \vec{y}_{A_n}{:}\vec{\tau}_{A_n}), \pi_\sigma, \epsilon_\sigma)$ *the* macro operator for σ.

We do not require the $\vec{y}_{A_i,j}$ 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_σ as $\vec{y}_\sigma{:}\vec{\tau}_\sigma$. The PDDL representation of the synthesized operator for the sequence $\langle drop, fix\rangle$ is shown in Listing 2. We show that O_σ has the same effect as σ:

Lemma 1. *Given an ADL problem description $\mathcal{A}$ with corresponding basic action theory Σ. Let $\vec{t}_1, \ldots, \vec{t}_n$ be ground terms of type $\vec{\tau}_{A_1}, \ldots, \vec{\tau}_{A_n}$, and O_σ the corresponding macro operator for $\sigma = \langle A_1, \ldots, A_n\rangle$ with operator name A_σ. Let Σ_σ be the same basic action theory as Σ but with additional ADL (macro) operators $O_{\langle A_1,A_2\rangle}, O_{\langle A_1,A_2,A_3\rangle}, \ldots, O_\sigma$. For any bounded sentence α:*

$$\begin{aligned}
&w_{\Sigma_\sigma} \models [A_\sigma(\vec{t}_1, \ldots, \vec{t}_n)]\alpha \Leftrightarrow\\
&\quad w_\Sigma \models [A_1(\vec{t}_1)][A_2(\vec{t}_2)]\ldots[A_n(\vec{t}_n)]\alpha
\end{aligned}$$

Proof. Proof by induction over the length of σ for atomic formulas α and then by structural induction over α. We denote $A_i(\vec{t}_i)$ as a_i and $(\vec{t}_1, \ldots, \vec{t}_n)$ as $\vec{t}_\sigma$.

Base case $n = 1$: Follows directly from $\pi_\sigma = \pi_{A_1}, \epsilon_\sigma = \epsilon_{A_1}$ and thus $A_\sigma = A_1$.

Induction step Let $F(\vec{t})$ be a ground atomic formula.

$\Leftarrow$**:** Assume $w_\Sigma \models [a_1]\ldots[a_n]F(\vec{t})$. Following the successor state axiom, we have two cases:

1. $w_\Sigma \models [a_1][a_2]\ldots[a_{n-1}]\gamma^+_F \wedge \vec{\tau}_F(\vec{t})$ and thus $w_\Sigma \models \mathcal{R}(\langle a_1, \ldots, a_{n-1}\rangle, \gamma^+_F)$. By definition of γ^+_F, we can follow: $w_\Sigma \models \mathcal{R}(\langle a_1, \ldots, a_{n-1}\rangle, \gamma^+_{F,A_n}(\vec{t}))$, where $\gamma^+_{F,A_n}(\vec{t})$ is defined by the disjunct for A_n of γ^+_F in the successor state axiom for F. By definition of the chaining $\mathcal{C}(A_{\langle A_1,\ldots,A_{n-1}\rangle}, A_n)$ and by induction: $w_{\Sigma_\sigma} \models \gamma^+_{F,\langle A_1,\ldots A_n\rangle}(\vec{t})$. By definition of the macro effect $\epsilon_{\langle A_1,\ldots A_n\rangle}$, it follows: $w_{\Sigma_\sigma} \models [A_\sigma(\vec{t}_\sigma)]F(\vec{t})$.
2. $w_\Sigma \models [a_1][a_2]\ldots[a_{n-1}](F(\vec{t}) \wedge \neg\gamma^-_F)$. By induction: $w_{\Sigma_{\sigma'}} \models [A_{\sigma'}(\vec{t}_{\sigma'})](F(\vec{t}) \wedge \neg\gamma^-_F)$ for the sub-sequence $\sigma' = \langle A_1, \ldots, A_{n-1}\rangle$. In particular, $w_{\Sigma_{\sigma'}} \models [A_{\sigma'}(\vec{t}_{\sigma'})]\neg\gamma^-_{F,A_n}$ (again γ^-_{F,A_n} being defined by the disjunct for A_i of γ^-_F in the successor state axiom for F) and by Regression Theorem and Theorem 2: $w_{\Sigma_{\sigma'}} \models \mathcal{R}(\langle A_{\sigma'}(\vec{t}_{\sigma'})\rangle, \neg\gamma^-_{F,A_n})$, and therefore $w_{\Sigma_\sigma} \models \neg\gamma^-_{F,\langle A_{\sigma'},A_n\rangle}$. It follows: $w_{\Sigma_\sigma} \models F(\vec{t}) \wedge \neg\gamma^-_F$ and thus $w_{\Sigma_\sigma} \models [A_\sigma(\vec{t}_\sigma)]F(\vec{t})$.

$\Rightarrow$**:** Assume $w_{\Sigma_\sigma} \models [A_\sigma(\vec{t}_\sigma)]F(\vec{t})$. Again, we have two cases:

1. $w_{\Sigma_\sigma} \models (\gamma^+_F \wedge \vec{\tau}_F(\vec{t}))|^a_{A_\sigma(\vec{t})}$. By definition of γ^+_F, it follows that $w_{\Sigma_\sigma} \models \gamma^+_{F,A_\sigma}(\vec{t})$ and thus

$$w_{\Sigma_\sigma} \models \gamma^+_{F,A_{\sigma'}}(\vec{t}) \wedge \neg\mathcal{R}(\langle A_{\sigma'}\rangle, \gamma^-_{F,A_n}(\vec{t})) \vee \mathcal{R}(\langle A_{\sigma'}\rangle, \gamma^+_{F,A_n}(\vec{t}))$$

for the sub-sequence $\sigma' = \langle A_1, \ldots, A_{n-1}\rangle$.

(a)

$$w_{\Sigma_\sigma} \models \gamma^+_{F,A_{\sigma'}}(\vec{t}) \wedge \neg\mathcal{R}(\langle A_{\sigma'}\rangle, \gamma^-_{F,A_n}(\vec{t}))$$

With $w_{\Sigma_\sigma} \models \gamma^+_{F,A_{\sigma'}}(\vec{t})$ and $w_{\Sigma_\sigma} \models \vec{\tau}(\vec{t})$, it follows that $w_{\Sigma_\sigma} \models [A_{\sigma'}(\vec{t})]F(\vec{t})$, and thus by induction, $w_\Sigma \models [a_1]\ldots[a_{n-1}]F(\vec{t})$. Because of $w_{\Sigma_\sigma} \models \neg\mathcal{R}(\langle A_{\sigma'}\rangle, \gamma^-_{F,A_n}(\vec{t}))$, and by Regression Theorem, it follows that $w_{\Sigma_\sigma} \models [A_{\sigma'}(\vec{t})]\neg\gamma^-_{F,A_n}(\vec{t})$.
We conclude by induction $w_\Sigma \models [a_1]\ldots[a_{n-1}]\neg\gamma^-_{F,A_n}(\vec{t})$ and therefore $w_\Sigma \models [a_1]\ldots[a_n]F(\vec{t})$.

(b)

$$w_{\Sigma_\sigma} \models \mathcal{R}(\langle A_{\sigma'}\rangle, \gamma^+_{F,A_n}(\vec{t}))$$

By induction and Theorem 2, $w_\Sigma \models [a_1]\ldots[a_{n-1}]\gamma^+_{F,A_n}(\vec{t})$ and also $w_\Sigma \models [a_1]\ldots[a_{n-1}]\vec{\tau}(\vec{t})$. It follows: $w_\Sigma \models [a_1]\ldots[a_n]F(\vec{t})$.

2. $w_{\Sigma_\sigma} \models F(\vec{t}) \wedge \neg\gamma^-_F$. Thus, $w_{\Sigma_\sigma} \models F(\vec{t})$, and

$$w_{\Sigma_\sigma} \models \neg\left(\gamma^-_{F,A_{\sigma'}}(\vec{t}) \wedge \neg\mathcal{R}\left(\langle A_{\sigma'}\rangle, \gamma^+_{F,A_n}(\vec{t})\right) \vee \mathcal{R}\left(\langle A_{\sigma'}\rangle, \gamma^-_{F,A_n}(\vec{t})\right)\right)$$

By induction and Regression Theorem:

$$w_\Sigma \models [a_1, \ldots, a_{n-1}]\neg\gamma^-_{F,A_n}(\vec{t})$$

We conclude: $w_\Sigma \models [a_1, \ldots, a_n]F(\vec{t})$.

Structural induction over α:

1. $\alpha = (t_1 = t_2)$: Follows directly from the semantics of $\mathcal{ES}$.
2. $\alpha = \phi \wedge \psi$: By induction, $w_{\Sigma_\sigma} \models [A_\sigma(\vec{t}_1, \ldots, \vec{t}_n)]\phi$ iff $w_\Sigma \models [A_1(\vec{t}_1)][A_2(\vec{t}_2)]\ldots[A_n(\vec{t}_n)]\phi$, similarly for ψ. Thus, by $\mathcal{ES}$ semantics, $w_{\Sigma_\sigma} \models [A_\sigma(\vec{t}_1, \ldots, \vec{t}_n)](\phi \wedge \psi)$ iff $w_\Sigma \models [A_1(\vec{t}_1)][A_2(\vec{t}_2)]\ldots[A_n(\vec{t}_n)](\phi \wedge \psi)$.
3. $\alpha = \neg\varphi$: Follows directly by induction for φ.
4. $\alpha = \forall x.\,\alpha$: By induction, for every $r \in R$: $w_{\Sigma_\sigma} \models [A_\sigma(\vec{t}_1, \ldots, \vec{t}_n)]\varphi^x_r$ iff $w_\Sigma \models [A_1(\vec{t}_1)]\ldots[A_n(\vec{t}_n)]\varphi^x_r$.
By $\mathcal{ES}$ semantics, we can follow: $w_{\Sigma_\sigma} \models [A_\sigma(\vec{t}_1, \ldots, \vec{t}_n)]\alpha$ iff $w_\Sigma \models [A_1(\vec{t}_1)][A_2(\vec{t}_2)]\ldots[A_n(\vec{t}_n)]\alpha$. ☐

Finally, we can show that any grounded instance $A_\sigma(\vec{t}_1, \ldots, \vec{t}_n)$ of a macro operator O_σ has the same effects as the corresponding ground action sequence $\langle A_1(\vec{t}_1), \ldots, A_n(\vec{t}_n)\rangle$:

Theorem 3. *For any bounded sentence α:*

$$\Sigma_\sigma \models [A_\sigma(\vec{t}_1, \dots, \vec{t}_n)]\alpha \Leftrightarrow \Sigma \models [A_1(\vec{t}_1)] \dots [A_n(\vec{t}_n)]\alpha$$

Proof. $\Rightarrow$**:** Assume $\Sigma_\sigma \models [A_\sigma(\vec{t}_1, \dots, \vec{t}_n)]\alpha$. Let w_σ be a world with $w_\sigma \models \Sigma_\sigma$, therefore $w_\sigma = w_{\Sigma_\sigma}$, and thus by Lemma 1:

$$w_\Sigma \models [A_1(\vec{t}_1)][A_2(\vec{t}_2)] \dots [A_n(\vec{t}_n)]\alpha$$

Assume $\Sigma \not\models [A_1(\vec{t}_1)][A_2(\vec{t}_2)] \dots [A_n(\vec{t}_n)]\alpha$.
Then there is a world w'_Σ with $w'_\Sigma \models \Sigma$ but $w'_\Sigma \not\models [A_1(\vec{t}_1)][A_2(\vec{t}_2)] \dots [A_n(\vec{t}_n)]\alpha$. Contradiction to the uniqeness of w_Σ.
$\Leftarrow$**:** Analogously to "$\Rightarrow$". □

5 Macro Planning

We extended the implementation[1] 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. Apart from extending the planner to ADL, we also modified the *macro selection*. We score a macro O_σ with two properties:

1. The normalized *frequency* $f(O_\sigma)$ of the macro in the training solutions. Let n be the number of occurrences of the sequence σ and l the total number of actions in the database. The frequency of σ is defined as $f(\sigma) = \frac{n}{l}$.
2. The number of joint parameters in the operators of the macro. The *parameter reduction* $r(O_\sigma)$ is defined by

$$r(O_\sigma) = \frac{\sum_{A_i(\vec{y}_i) \in \sigma} |\vec{y}_i| - |\vec{y}_\sigma|}{\sum_{A_i(\vec{y}_i) \in \sigma} |\vec{y}_i|}$$

As an example, the parameter reduction for $\sigma_1 = \langle drop(b), fix(o), drop(b) \rangle$ (dropping the same bag twice) is $r(O_{\sigma_1}) = \frac{1}{3}$, because the two parameters of *drop* are replaced by the common parameter b. On the other hand, for $\sigma_2 = \langle drop(b_1), fix(o), drop(b_2) \rangle$ (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_{σ_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 $\mathcal{M} = \{O_{\sigma_1}, \dots, O_{\sigma_n}\}$ be the macros of the domain and $\sigma_i = \langle A_{i_1}, \dots, A_{i_{j_i}} \rangle$. The complementarity of $\mathcal{M}$ is defined as

$$C(\mathcal{M}) = \frac{\left| \bigcup_{i=1}^{n} \bigcup_{A_j \in \sigma_i} \{A_j\} \right|}{\sum_{i=1}^{n} \left| \bigcup_{A_j \in \sigma_i} \{A_j\} \right|}$$

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

Using those properties, we define an evaluator

$$E(\mathcal{M}) = |\mathcal{M}|^{-w_l} C(\mathcal{M})^{w_c} \sum_{O \in \mathcal{M}} w_f f(O) + (1 - w_f) r(O)$$

where w_f, w_l, w_c are weights from $[0, 1]$. The higher w_l, the higher we prefer domains with a small number of macros. The higher w_f, 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 $5\,\mathrm{min}$ and a memory limit of $8\,\mathrm{GB}$. For each task, the planner scores 1 point if the run-time t was less than $1\,\mathrm{sec}$, $1 - \frac{\log t}{\log 300}$ 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 $60\,\mathrm{min}$ and a memory limit of $8\,\mathrm{GB}$. 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 $(w_f, w_c, w_l) \in \{0.0, 0.1, \dots, 1.0\}^3$ 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_c), and number of macros (w_l) of the evaluators which performed best on problems of the validation set and which were used on the test set.

Domain	FF			LAMA		
	w_f	w_l	w_c	w_f	w_l	w_c
Blocksworld	0.1	1.0	0.0	0.0	1.0	0.0
Caldera	0.0	0.9	0.0	0.0	0.0	0.0
Nurikabe	0.5	0.0	0.1	0.1	0.0	0.0
Termes	0.6	0.0	0.9	0.0	0.4	0.3
Barman	0.7	0.0	0.0	0.5	1.0	0.0
Childsnack	0.0	0.6	1.0	0.0	0.6	1.0
Hiking	0.6	0.5	0.9	0.6	0.6	0.7
Thoughtful	0.5	0.8	0.4	0.7	0.0	0.0
Cleanup	0.9	0.0	0.0	0.0	0.0	0.2

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

Domain	FF	LAMA	MACROFF	DBMP FF	DBMP LAMA
Blocksworld	8.87	**15.00**	11.80	10.99	14.82
Caldera	0.85	1.35	0.75	0.85	**2.06**
Nurikabe	0.00	0.73	0.00	**0.90**	0.76
Termes	0.00	**2.00**	0.00	1.83	**2.00**
Barman	0.00	4.00	0.00	4.36	**4.96**
Childsnack	0.00	1.00	0.00	**3.97**	3.90
Hiking	0.65	1.87	1.29	3.01	**3.54**
Thoughtful	1.55	1.82	**3.06**	2.95	3.00
Cleanup	1.00	0.70	0.00	**4.11**	0.73

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

Table 3. Planner scores on the test set. The macro-augmented domain was selected based on the validation set. Scores for FF and LAMA on the original domains, MACROFF with its own macros, and FF and LAMA with DBMP domains augmented by up to 4 macros of maximum length 4.

Domain	FF	LAMA	MACROFF	DBMP FF	DBMP LAMA
Blocksworld	17.73	**30.00**	28.40	23.00	29.72
Caldera	3.57	**3.63**	2.47	3.58	3.05
Nurikabe	**2.79**	2.36	1.68	2.50	2.26
Termes	0.97	**6.00**	0.00	2.63	3.00
Barman	0.00	**10.00**	0.00	8.80	9.96
Childsnack	0.00	1.00	0.00	6.11	**7.57**
Hiking	3.28	**9.40**	2.89	5.95	6.55
Thoughtful	4.73	**8.11**	3.78	3.00	5.22
Cleanup	3.82	0.70	0.00	**12.67**	0.73

are shown in Table 2. For each pair, we selected the macro-augmented domain with the highest score and evaluated it on the test set. The results are shown in Table 3.

On the validation set, we can see that DBMP improves planner performance on most domains with both FF and LAMA as base planner. It also generally performs better than MACROFF. However, in the test set the macro-augmented domains did not improve and sometimes even impaired the performance of LAMA, with *Childsnack* being a notable exception. For FF, macros are more helpful, and FF with DBMP macros generally outperforms FF also in the test set. MACROFF was not able to generate any macros for *Barman*, *Childsnack*, and *Cleanup*, because it could not solve any of the training problems. This clearly shows the advantage of having a planner-independent macro representation, as we can use a different (and if desired optimal) planner to solve the training problems.

Summarizing the results, DBMP with ADL macros performs better than MACROFF, and thus, ADL operator synthesis is a viable approach to macro planning with ADL domains. On the other hand, DBMP with ADL macros does not perform better than LAMA without macros on the test set, which may partly be due to an over-fitting macro selection process. This may be remedied by modifying the training and selection process, e.g., by determining training problems based on the *structural similarity* [4] of the problems.

6 Creating a Plan Library with ADL Macros

A plan library is a collection of pre-computed plans, which are either collected from previous runs or manually created. Such a plan library can be used directly, e.g., for planning on robots [1, 18], for case-based planning [15, 12], or to define abstract tasks that are distributed to agents in a multi-agent environment [16]. Similar to macros, a plan in a library can be represented by the sequence of actions that it consists of, as a single operator with preconditions and effects, or as grounded plan instances. Using macro operator synthesis can simplify the creation and maintenance of such plan libraries. Instead of manually specifying the preconditions and effects of the (abstract) plan, those can be computed automatically from the observed or manually specified plans. Additionally, the parameter reduction technique described in Section 5 can be used to generalize a set of observed plans into a parameterized plan.

We present an example from a production logistics scenario [18]: In this domain, a team of robots has to operate a number of machines to manufacture products. The approach presented in [18] uses a predefined plan library of PDDL actions to accomplish simple tasks and combines those tasks to achieve the overall objective. Those plans are hand-crafted. The preconditions of the plan actions are checked

```
(location-lock ?mps INPUT)
(move ?robot ?from ?from-side ?mps INPUT)
(wp-get-shelf ?robot ?cap-carrier ?mps ?shelf-spot)
(wp-put ?robot ?cap-carrier ?mps)
(location-unlock ?mps INPUT)
```

Listing 3. A plan from the logistics robots plan library.

```
(:action wp-put
 :parameters (?r - robot ?wp - workpiece ?m - mps)
 :precondition (and (at ?r ?m INPUT)
  (wp-usable ?wp) (holding ?r ?wp)
  (mps-side-free ?m INPUT))
 :effect (and (wp-at ?wp ?m INPUT)
  (not (holding ?r ?wp)) (can-hold ?r)
  (not (mps-side-free ?m INPUT))))
```

Listing 4. The action `wp-put`, one action of the plan in Listing 3.

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

```
(:action prefill-cap-station
 :parameters (?mps - mps ?robot - robot
   ?from - location ?from-side - mps-side
   ?cap-carrier - cap-carrier ?spot - shelf-spot)
 :precondition (and
  (not (location-locked ?mps ?p2))
  (at ?robot ?from ?from-side)
  (or (not (= ?mps ?from))
      (not (= ?from-side INPUT)))
  (wp-on-shelf ?cap-carrier ?mps ?spot)
  (can-hold ?robot) (mps-side-free ?mps INPUT))
 :effect (and
  (wp-at ?cap-carrier ?mps INPUT)
  (not (mps-side-free ?mps INPUT))
  (not (wp-on-shelf ?cap-carrier ?mps ?spot))
  (wp-usable ?cap-carrier) (spot-free ?mps ?spot)
  (not (at ?robot ?from ?from-side))
  (at ?robot ?mps INPUT)))
```

Listing 5. The computed macro operator for the plan in Listing 3.

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

[1] Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, Lorenz Mösenlechner, Dejan Pangercic, Thomas Rühr, and Moritz Tenorth, 'Robotic roommates making pancakes', in *Proceedings of the 11th IEEE-RAS International Conference on Humanoid Robots*, (2011).
[2] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer, 'Macro-FF: Improving AI planning with automatically learned macro-operators', *Journal of Artificial Intelligence Research (JAIR)*, **24**, (2005).
[3] Adi Botea, M. Müller, and Jonathan Schaeffer, 'Learning partial-order macros from solutions', in *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, (2005).
[4] L. Chrpa and M. Vallati, 'Determining representativeness of training plans: A case of macro-operators', in *Proceedings of the 30th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, (2018).
[5] Lukáš Chrpa, 'Generation of macro-operators via investigation of action dependencies in plans', *The Knowledge Engineering Review*, **25**(3), (2010).
[6] Lukáš Chrpa and Thomas Leo McCluskey, 'On exploiting structures of classical planning problems: Generalizing entanglements', in *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*, (2012).
[7] Lukáš Chrpa and Fazlul Siddiqui, 'Exploiting block deordering for improving planners efficiency', in *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, (2015).
[8] Lukáš Chrpa, Mauro Vallati, and Thomas Leo McCluskey, 'MUM: A technique for maximising the utility of macro-operators by constrained generation and use', in *Proceedings of the 24th International Conference on Automated Planning and Scheduling*, (2014).
[9] Jens Claßen, Patrick Eyerich, Gerhard Lakemeyer, and Bernhard Nebel, 'Towards an integration of planning and Golog', in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, (2007).
[10] Andrew Coles, Maria Fox, and Amanda Smith, 'Online identification of useful macro-actions for planning', *Artificial Intelligence*, (2007).
[11] Andrew Coles and Amanda Smith, 'Marvin: A heuristic search planner with online macro-action learning', *Journal of Artificial Intelligence Research (JAIR)*, **28**, (2007).
[12] Michael T. Cox, Héctor Muñoz-Avila, and Ralph Bergmann, 'Case-based planning', *The Knowledge Engineering Review*, **20**(3), 283–287, (September 2005).
[13] Christian Dornhege and Andreas Hertle, 'Integrated symbolic planning in the tidyup-robot project', in *AAAI Spring Symposium - Designing Intelligent Robots: Reintegrating AI II*, (2013).
[14] Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson, 'Learning and executing generalized robot plans', *Artificial Intelligence*, **3**, (1972).
[15] Kristian J. Hammond, *Case-Based Planning: Viewing Planning As a Memory Task*, Academic Press, Inc., 2016.
[16] Andreas Hertle and Bernhard Nebel, 'Efficient auction based coordination for distributed multi-agent planning in temporal domains using resource abstraction', in *Proceedings of the 41st German Conference on Artificial Intelligence (KI)*, (2018).
[17] Jörg Hoffmann and Bernhard Nebel, 'The FF planning system: Fast plan generation through heuristic search', *Journal of Artificial Intelligence Research (JAIR)*, **14**, (2001).
[18] Till Hofmann, Nicolas Limpert, Victor Mataré, Alexander Ferrein, and Gerhard Lakemeyer, 'Winning the RoboCup Logistics League with fast navigation, precise manipulation, and robust goal reasoning', in *RoboCup 2019: Robot World Cup XXIII*, (2019).
[19] Till Hofmann, Tim Niemueller, Jens Claßen, and Gerhard Lakemeyer, 'Continual planning in Golog', in *Proceedings of the 30th Conference on Artificial Intelligence (AAAI)*, (2016).
[20] Till Hofmann, Tim Niemueller, and Gerhard Lakemeyer, 'Initial results on generating macro actions from a plan database for planning on autonomous mobile robots', in *27th International Conference on Automated Planning and Scheduling (ICAPS)*, (2017).
[21] Gerhard Lakemeyer and Hector Levesque, 'Semantics for a useful fragment of the Situation Calculus', in *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, (2005).
[22] Gerhard Lakemeyer and Hector J Levesque, 'Situations, si! Situation terms, no!', in *Proceedings of the 9th Conference on Principles of Knowledge Representation and Reasoning (KR)*, (2004).
[23] Hector Levesque and Gerhard Lakemeyer, *The Logic of Knowledge Bases*, MIT Press, 2001.
[24] Hector Levesque, Fiora Pirri, and Ray Reiter, 'Foundations for the Situation Calculus', *Linköping Electronic Articles in Computer and Information Science*, **3**(18), (1998).
[25] John McCarthy, 'Situations, actions, and causal laws', Technical report, Stanford University, (1963).
[26] M.A. Hakim Newton, John Levine, Maria Fox, and Derek Long, 'Learning macro-actions for arbitrary planners and domains', in *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, (2007).
[27] M.A. Hakim Newton, John Levine, Maria Fox, and Derek Long, 'Wizard: Compiled macro-actions for planner-domain pairs', in *Booklet for the 6th International Planning Competition Learning Track*, (2008).
[28] Raymond Reiter, *Knowledge in action: logical foundations for specifying and implementing dynamical systems*, MIT Press, 2001.
[29] S. Richter and M. Westphal, 'The LAMA planner: Guiding cost-based anytime planning with landmarks', *Journal of Artificial Intelligence Research (JAIR)*, **39**, (2010).
[30] Jussi Rintanen, 'Regression for classical and nondeterministic planning', *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI)*, (2008).

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