Unifying Planning and Replanning in the ARF

Roman van der Krogt*, (r.p.j.vanderkrogt@cs.tudelft.nl)
Mathijs de Weerdt† (m.m.deweerdt@cs.tudelft.nl) and
Cees Witteveen (c.witteveen@cs.tudelft.nl)
Parallel and Distributed Systems Group, Delft University of Technology, PO Box 5031, 2600 GA Delft, The Netherlands

Abstract. The aim of this paper is to combine standard planning, replanning and multi-agent planning into a rigorous unifying framework, extending an existing logic-based approach to resource-based planning. In this Action Resource Framework (ARF), actions and resources are the primitive concepts. Actions consume and produce resources. Plans are structured objects composed of actions and resource schemes and an explicit dependency function specifying their interrelationships. Plans can be changed by applying plan operators to them. We consider both planning and replanning as plan transformation processes and we introduce some operators for plan transformation. We prove some sets of plan operators to be complete. The explicit representation of the plans and their inputs and outputs (resources) turns out to be particular useful in modeling multi-agent (re)planning as processes of exchanging services and resources among agents. Finally, we show that existing (re)planning methods and heuristics fit into this framework.

Keywords: planning, replanning, multi-agent planning, resource/plan logic

AMS: Primary 68T27 (Logic in artificial intelligence), also 68W15 (Distributed algorithms)

1. Introduction

Planning is an important ability required for intelligent agents to accomplish goals they have set for themselves, or to reach goals they have accepted from others. Usually, a planning problem is specified using a description of (i) the *current* (or *initial*) *state* the agent is in, (ii) the set of *actions* (together with their prerequisites and consequences) the agent is capable to perform, and (iii) the *goals* the agent is aiming at, specified as a set of states. The planning *problem* then is to find the right sequence (or partial order) of actions leading the agent from the initial state to one of the desired states specified by the goals.

* Supported by the Freight Transport Automation and Multi-modality (FTAM) research program.

† Supported by the Seamless Multi-modal Mobility (SMM) research program. Both programs are carried out within the TRAIL research school for Transport, Infrastructure and Logistics.

1.1. Existing approaches

Currently, there exists many systems and methods (e.g. [2, 11, 19, 1, 5, 9, 30, 32] that try to tackle the planning problem. For example, planners like HSP [4] and FF [20] construct plans bottom-up: they add the best action to the end of the partial plan. To determine which action is the best, they use *heuristic search* methods, e.g., a relaxation of the original planning problem such as ignoring the negative effects of actions. Other, so-called least-commitment planners, like UCPOP [23], do not use a particular order (top-down or bottom-up) to add actions, but keep track of all dependencies and constraints among actions within their partial plan representation.

These systems, however, have some serious drawbacks.

First of all, almost all of these planning systems rely on the tacit assumption that planning problems always can be solved *off-line*: the goal and the initial state are assumed to remain unchanged. For a large number of domains, however, this assumption does not hold due to the dynamical nature of planning problems. For example, in typical transportation problems, planners cannot rely on plans that are constructed off-line: due to e.g., traffic accidents, broken equipment and last-minute changes in orders, the feasibility of the plan can be seriously affected by a change in the initial state or in the conditions that need to be fulfilled during the execution of the plan. Hence, in these cases, a practical planning approach should also pay attention to *replanning* to repair a plan that has become infeasible, possibly already during the planning phase.

Secondly, most planning systems assume that a planning agent has to start from scratch. Often, however, this assumption is not realistic: agents are able to use results of their previous planning experiences, or knowledge given to them by a (human) domain expert and therefore use one or more existing –maybe incomplete or not completely suitable– plans P' as a starting point. Such initial plans may be *transformed* and *combined* into a suitable plan. Therefore, the standard approach to planning, i.e., to start from an empty plan, seems to be just a *limiting case* of standard practice and usually needs to be generalized to include the adaptation of existing plans.

Finally, it is quite common for planning systems to assume that their planner is the single entity that is capable to use resources and to execute actions. In reality, however, planning is not performed by a single autocratic entity: resources are often controlled by other parties, not all actions required can be performed by a single agent and in order to achieve its goal an agent has to coordinate its actions with other planners.

Fortunately, there are planning systems that at least partially deal with these issues. For example, the Systematic Plan Adaptor (SPA) [19] and CHEF [18] systems meet the second objection by addressing plan adaptation. These *case-based planners* maintain a database of past problems and their solution plans, and choose an appropriate starting plan whenever they face a planning problem. The selection of the starting plan is based on similarity with respect to the planning problem. This plan then is modified to match the current goal and initial state requirements.

Other systems focus on the replanning aspect. A system as GPG [17] is a *static* replanner, i.e., whenever their system is used to solve a discrepancy between the original planning problem and the current state of the world, it assumes that the world remains static. Only after a solution is found, the state of the world is examined again for changes. The term *continual planning* [10] is used to refer to systems in which planning, replanning and execution are all continuously interleaved.

Also, some planning systems support a (limited) form of *multi-agent planning*. For example , in *plan merging* approaches [14, 16, 7] each agent constructs its own plan and afterwards conflicts and mutual benefits are analyzed. In multi-agent *hierarchical (task reduction) planning* approaches [26] conflicts are being determined first and then are resolved by reducing the tasks. In *partial global planning* approaches [13] information about (potential) conflicts is maintained in a global structure. Finally, *implicit distributed planning* methods [27, 31, 8] share resources, for example via some sort of price mechanism, such as an electronic market or an auction. None of these multi-agent planning methods, however, describes a framework to use existing planning algorithms, uses domain knowledge, and contains replanning algorithms as well.

1.2. Research contributions

The goal of this paper is, first of all, to bring together ideas from both standard planning and replanning approaches into a unifying (re)planning approach. We base this framework upon an existing logic-based framework for resource based planning. The outcome is a unified framework to represent planning and replanning from a single agent perspective and we show that refinement strategies can be built on top of this framework to supply computational support for (re)planning. To show that this framework also can be used to cover the multi agent perspective on planning, we show that multi-agent aspects of planning as plan and resource sharing and coordination can be easily incorporated in the current framework.

We consider such a unifying framework for planning, replanning, and multi-agent planning to have (at least) the following benefits. First of all, it should offer a common platform to develop new heuristics and algorithms. Secondly, it should offer possibilities to compare the quality of competing (planning) algorithms. Finally, it should help to transfer existing single agent planning methods in a multi-agent context where planning comes down to designing methods for exchanging resources between agents. Moreover, this perspective also enables us to deal with privacy issues where agents might be reluctant to reveal details of their plans to other agents.

1.3. Outline

This paper is organized as follows. First, we give a concise introduction to an existing resource-based planning approach [24]. We use this formalism to deal with replanning and we introduce plan transformation operators that are able to modify resource-based plans. We show that all necessary plan adaptations can be performed by a single plan adaptation operator. By assuming that an agent is able to use a *plan library*, these operators can be used to transform an initial (inadequate) plan into an adequate plan. Viewed from a single-agent perspective, this framework unifies both planning and replanning approaches. We would like to find a suitable sequence of these plan modification steps to obtain an adequate plan form the initial plan. Therefore, we present a method to use existing planning techniques in this framework, and illustrate this by translating the FF-approach [20] and SPA [19] algorithm for plan adaptation. Finally, we put the framework in a multi-agent perspective and we show that such an extension can be obtained by generalizing existing *plan-merging* approaches.

2. The action resource formalism

In this section we introduce a framework for planning and replanning. This *Action Resource Framework*, abbreviated ARF, is based upon work by Tonino et al. [24, 7]. This framework has the following properties. Firstly, the planning problem is specified by two sets of *resources* (one set specifying the set of resources available, the other the set of resources to be achieved). Secondly, plans are specified as *structured objects* composed of unified general action schemata. Thirdly, a clear distinction is made between a plan as *embedded* in a specification of the initial situation and the goal and a plan as a (free) structured set of actions. Finally, both available and goal resources can be exchanged, and this is very useful to do multi-agent planning.

We start with a concise overview of the main elements of this framework. Subsequently, we extend the framework by providing a more sophisticated notion of plans, and we formalize the notion of a gap in a plan.

2.1. The ARF framework: basic notions

The ARF distinguishes two basic notions in planning: *resource (facts)* and *actions*. Goals and plans are derived notions that are defined using resources and actions.[1]

A *resource* is an object that is relevant to an agent with respect to the planning problem at hand. Such a resource is either a physical object such as a truck or a block, or an abstract conceptual notion such as the right to do something. To describe the connection between a resource and the (current) state of the world it is in, we use the notion of a *resource fact*. A resource fact specifies the state of a given resource. Syntactically, a resource fact is denoted by a *predicate name* together with a complete specification of the *sorts* of all its *attributes* together with their *values*. The predicate name serves to indicate the *type* of resource mentioned in the fact (e.g., a carrier cycle or a taxi). If *cycle* is a resource type, having attributes like its location *loc*, its driver *driver* and its capacity *capacity*, then $\mathit{cycle}(\mathrm{A} : \mathit{loc}, \mathrm{Tom} : \mathit{driver}, 500 : \mathit{capacity})$ is a resource fact describing a carrier cycle in A whose driver is Tom with capacity 500. To uniquely identify resource facts, a special attribute *identity* is used to distinguish it from other resources having the same type and possibly the same values of their attributes. Because of the special nature of this identifier, we denote a resource of type t with identifier i as $t_i(\ldots)$

When the values of all attributes of a resource fact are ground, i.e., they are constant, we call this a *ground resource fact*. However, attributes may also be variables or functions. In this case, a resource fact describes a *set* of ground resource facts (instances) of the same resource type.[2] For example, the following resource refers to all cycles in location A: $\mathit{cycle}_{i:id}(\mathrm{A} : \mathit{loc}, d : \mathit{driver}, c : \mathit{capacity})$.

To specify one specific ground resource fact (ground resource) denoted by such a *general resource fact*, we introduce the notion of a *substitution*. A substitution θ replaces variables occurring in a resource r by terms of the appropriate sort. We write $r\theta$ to denote the re-

[1] In the original formalism [7], actions are called ‘skills’ and plans are called ‘services’. We have chosen to use the more generally accepted terms ‘actions’ and ‘plans’.

[2] Abusing language, in the sequel we use the notions of a resource and a resource fact interchangeably.

source r' that results from replacing the variables occurring in r according to θ. A substitution is *ground* if it replaces variables by ground terms, i.e., terms that do not contain variables. If R is a set of general resources, $R\theta$ is a shorthand for $\{r\theta \mid r \in R\}$. The set of ground resources denoted by a general resource g is the set $R_g = \{g\theta : \theta \text{ is a ground substitution for } g\}$.

Goals can be efficiently specified by such general resources. Usually, a set of goals G is specified by a set of general resources $G = \{g_1, \ldots, g_n\}$. We say that a set of goals G is *satisfied* by a given set of resources R, abbreviated by $R \models G$, if there exists a ground substitution θ such that $G\theta \subseteq R$, i.e., there is a set of ground instances of the goals that is provided by the resources in R. Two resources r_1 and r_2 are called *compatible*, denoted by $r_1 \equiv r_2$, when they are equal except for the value of their identity attribute.

Example. Within some city in the Netherlands, a company named FOO transports goods from one place to the other. A map of this city is shown in Figure 1. There are 4 locations: Home (the location of FOO's offices), A, B and C, and a road network linking them. The planning problems that FOO has to solve typically consist of bringing crates containing goods from one location to another. FOO uses two carrier cycles for this. For simplicity, we assume that each carrier cycle is able to carry only one crate at the same time.

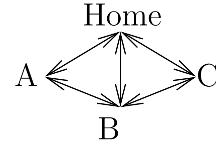

Figure 1. Map of the city used in the running example

FOO uses a planning system for planning its daily work, using ARF as the domain describing language. Its model has the following resources. A $road(f : loc, t : loc)$ resource is used to denote that there is a road leading from some location f to another t. A resource $cycle(i : id, l : loc)$ defines a cargo bike at a certain location l with a certain identifier i. Cargo items with a certain identifier i at a certain location l are denoted by a $crate(i : id, l : loc)$ resource. Finally, the ability to transport cargo between two locations f and t is expressed by a $cargospace(f : loc, t : loc)$ resource.

Resource facts are used to model the state of the world (as far as it is relevant to a planner) by enumerating the set of resource facts that are true at a certain time point. Possible *transitions* from one state to another are described by actions. An *action* is a basic process that consumes and produces resources. An action o has a set of input resources $in(o)$ that it consumes, and a set of output resources $out(o)$ that it produces. Furthermore, an action may contain a specification of some variables occurring in the set of output resources as *parameters*

$param(o)$ of the action. To ensure that output resources are uniquely defined, these resources may only contain variables that already occur in the input resources or in the set of the parameters.

An action o can be applied to a set of (ground) resources R if a ground substitution θ exists such that $in(o)\theta \subseteq R$. Application of this action to R results in consuming the set $in(o)\theta$ of input resources while producing the set $out(o)\theta$. The result of o applied to R (under θ) therefore is a resource transformation: starting with R, the set $R - in(o)\theta + out(o)\theta$ is produced.[3]

Example. Two actions are present in the domain of FOO. A *pedal* action denotes cycling from one location to another. This action creates a *cargospace* resource: the ability to transport cargo along. The other action, *carry*, uses this ability to actually transport a cargo item from one location to another. Both actions have a parameter *dst* specifying the destination.

$$\begin{aligned}
pedal(dst : loc): \quad & cycle(i : id, src : loc), road(src : loc, dst : loc) \Rightarrow \\
& cycle(i : id, dst : loc), cargospace(src : loc, dst : loc), \\
& road(src : loc, dst : loc) \\
carry(dst : loc): \quad & crate(i : id, src : loc), cargospace(src : loc, dst : loc) \Rightarrow \\
& crate(i : id, dst : loc)
\end{aligned}$$

Note that the domain is structured differently from conventional transportation domains. This is done to be able to explicitly reason about opportunities to transport cargo. Whereas in conventional models the actual transportation of objects is hidden inside the movement of the truck (or other vehicle) that it was loaded in, this description hides the loading and unloading of goods, but explicitly describes that goods are being transported.

In general, a single action applied on an initial set of resources may not be sufficient to achieve a desired state. Often, actions have to be applied in a partial order to produce the desired effect. A specification of the ordering of actions, however, is not sufficient. We also need to specify for each consumed resource, which produced resource it is *dependent* upon. Such a partially ordered set of actions together with a specification of the resources dependency relation is called a *plan*.

[3] We use + and − to denote (left associative) set-union and set-subtraction.

2.2. Dependency functions and plans in ARF

In the remainder of the paper, we assume that when O is a finite set of actions, for each o, o' in O we have $\mathrm{var}(o) \neq \mathrm{var}(o')$ when $o \neq o'$. First we define the set of resources produced and consumed in a set of actions O.

DEFINITION 1. *Let O be a finite set of actions. The set $res(O)$ is the set of all resources mentioned in the input $in(o)$ or output $out(o)$ of actions $o \in O$. The set $cons(O)$ specifies the set of all resources consumed using actions in O: $cons(O) = \bigcup_{o \in O} in(o)$ while the set of resources produced equals $prod(O) = \bigcup_{o \in O} out(o)$.*

We define plans over O as structured objects composed of actions in O. First of all, we have to specify how actions are interrelated. To this end, we use the notion of a *dependency function*.

DEFINITION 2. *Let O be a set of actions. A dependency function is an injective function $d : cons(O) \rightarrow prod(O) + \{\bot\}$ specifying (in a unique way) for each resource r to be consumed which resource r' produced by another action is used to provide r (or $\bot$ if r is not produced by an action in O, i.e., r is an input resource).*

As we mentioned before, plans are composed of partially ordered actions. Since a dependency function d specifies an immediate dependency of input resources of an action on output resources of another action, d can only specify a valid dependency if (i) the resources involved are compatible and (ii) d generates a partial order between the actions.

The first requirement is met if there exists a substitution θ such that for two resources r and r', $d(r) = r'$ implies $r\theta \equiv r'\theta$, that is θ is a *unifier* for every pair of resources $(r, d(r))$. In particular, we are looking at a *most general unifier* (mgu) θ with this property.[4]

The second condition requires that there are no loops in the dependency relation between actions generated by d: we say that o directly depends on o', abbreviated as $o' \ll_d o$, if resources $r \in in(o)$ and $r' \in out(o')$ exist such that $d(r) = r'$. Let $<_d = \ll^+$ be the transitive closure of $\ll_d$. Then the second condition simply requires $<_d$ to be a (strict) partial order on O. Now a plan can be defined as follows.

DEFINITION 3 (Free Plan). *A (free) plan P over a set of actions O is a triple $P = (O, d, \theta)$ where d is a dependency relation specifying dependencies between compatible relations and generates a partial order $<_d$ on O, while θ is the mgu of all dependency pairs $(r, d(r))$ where $r, d(r) \in res(O)$.*

[4] The mgu is unique upon renaming of variables.

Note that dependencies on resources can be easily defined using dependencies on actions. A resource r is said to be dependent on a resource r' in a plan $P = (O, d, \theta)$ if (i) $r \in out(o)$ and $r' \in in(o)$ for some $o \in O$ or actions $o, o' \in O$ exist such that $o' <_d o$ and $r \in res(o)$ and $r' \in res(o')$. Furthermore, a subset $S \subseteq prod(P)$ is said to be an *independent* set of resources, if for any pair of resources $r, r' \in S$, neither r is dependent on r', nor r' is dependent on r in P.

Sometimes we need the inverse d^{-1} of d, which is defined as follows: for every $r' \in prod(O)$ such that $d(r) = r'$, $d^{-1}(r') = r$, and for every $r' \in prod(O)$ not occurring in $\mathrm{ran}(d)$, $d^{-1}(r') = \bot$.

Given a plan $P = (O, d, \theta)$, the set of input resources of a plan, denoted by $In(P)$, is the set of resources $\{r\theta \mid d(r) = \bot\}$ not depending on other resources in the plan. The set of output resources denoted by $Out(P)$ is the set $\{r\theta \mid d^{-1}(r) = \bot\}$ of resources that are not consumed by actions in the plan. Furthermore, we define $prod(P) = prod(O)\theta$ and $cons(P) = cons(O)\theta$. Resources from $prod(P)$ may be used by other plans to produce other resources. Finally, we use $||P||$ to denote the size of P which equals the sum of the total number of all resources mentioned in P and the number of actions occurring in P.

Free plans are used to transform a set of resources into another set of resources. Given a set of goals G and an initial set of resources I, a plan $P = (O, d, \theta)$ is *applicable* to I if there exists a substitution σ such that $In(P)\sigma \subseteq I$. P *realizes* a set of goals G if there exists a substitution σ such that $G\sigma \subseteq (I - In(P)\sigma) + Out(P)\sigma$. The tuple (I, P, σ, G) where σ is a substitution of variables occurring in the goals G and input resources $In(P)$ is called an *embedded plan*. An embedded plan (I, P, σ, G) is called *adequate* when $In(P)\sigma \subseteq I$ and $G\sigma \subseteq (I - In(P)\sigma) + Out(P)\sigma$. A tuple (I, P, G) is called adequate, denoted by $I \models_P G$ when there exists a substitution σ such that (I, P, σ, G) is adequate.

Example. Figure 2.a shows an example plan. We have used the following abbreviations: r denotes a *road* resource, c denotes a *cycle*, cr abbreviates a *crate* resource and *cargospace* resources are denoted by cs. Also, we have not included the attribute sorts, only the values of the attributes are given. From a $road(f : loc, t : loc)$, a $cycle(i_1 : id, f : loc)$ and a $crate(i_2 : id, f : loc)$ resource, it is able to produce a $crate(i_2 : id, f : loc)$ by using a *pedal* action (P_1) and a *carry* action (C_2). That is, this plan can be used to transport a piece of cargo to another location. Throughout the remainder of this paper, we use a simplified representation when depicting plans in figures. The simplified representation of this plan is given in Figure 2.b. Two things have changed. Firstly, the road resource is no longer produced by the *pedal* action, since it is only a precondition for the action.

We assume that an unlimited amount of *road* resources is available for each of the *road* resources that is contained in the initial state. Secondly, the two *cargospace* resources and their dependency are condensed into a single resource that is simultaneously produced by the producing action and consumed by the consumer. For an initial set of resources $I = \{road(\mathrm{A} : loc, \mathrm{B} : loc), cycle(1 : id, \mathrm{A} : loc), crate(3 : id, \mathrm{A} : loc)\}$ and goal resources $G = \{crate(3 : id, \mathrm{B} : loc)\}$, the tuple (I, P, G) is adequate since (I, P, σ, G) is adequate for a substitution $\sigma = \{f \to \mathrm{A}, t \to \mathrm{B}, i_1 \to 1, i_2 \to 3\}$.

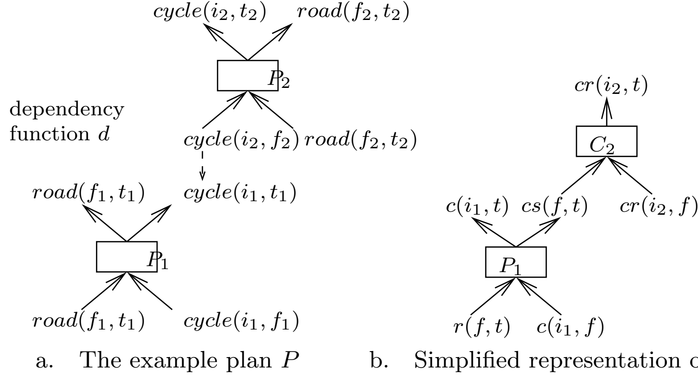

Figure 2. An example plan. The dashed arrow denotes the dependency relation d.

An *initial sub plan* P' of P is a plan $P' = (O', d', \theta')$ where (i) $O' \subseteq O$ and O' is downward closed with respect to $<_d$, (ii) d' is the restriction of d to $cons(O')$ and (iii) θ' is a most general unifier (mgu) for all dependent pairs $(r, d'(r)) \in d'$.[5]

The following proposition summarizes some properties of the planning relation $\models_P$ induced by P:

PROPOSITION 1. *Let $P = (O, d, \theta)$ be a plan, I and R be disjoint sets of resources and G a set of goals. Then we have*

1. *If $I \models_P G$ then $I \models_P G'$ for every $G' \subseteq G$ and $I' \models_P G$ for every $I' \supseteq I$, that is $\models_P$ is anti-monotonic in its right argument and monotonic in its first argument.*
2. *For every ground substitution σ with* $\mathrm{dom}(\sigma) \supseteq res(P)$, *$In(P)\sigma \models_P Out(P)\sigma$.*
3. *Let $S_1 + S_2$ denote the disjoint union of S_1 and S_2. Then, if $In(P)\sigma + R$ is defined $In(P)\sigma + R \models_P Out(P)\sigma + R$.*

[5] Downward closed means that for every $o <_d o'$, if $o' \in O'$ then also $o \in O'$.

4. If P is applicable to I then for every independent subset $S \subseteq prod(P)$ of general resources produced by P, P contains an initial sub plan P' such that $I \models_{P'} S$.

Proof. See Appendix.

The following observation is an easy consequence of the preceding proposition (part 3):

OBSERVATION 1. *If $I \models_P G$ and R is a set of ground resources such that $I \cap R = \emptyset$, then for every $R' \subseteq R$, $(I + R') \models_P (G + R')$.*

Example. FOO's initial state I has three components: (i) the cycles that are available: $cycle_1(1 : id, \mathrm{Home} : loc), cycle_2(2 : id, \mathrm{Home} : loc) \subset I$, (ii) the road network that is given in Figure 1: $road_3(\mathrm{A} : loc, \mathrm{B} : loc), road_4(\mathrm{A} : loc, \mathrm{C} : loc), \ldots \subset I$, and (iii) the current location of the goods that have to be transported. Today, FOO has only two orders: $crate_{100}(1 : id, \mathrm{B} : loc)$, and $crate_{101}(2 : id, \mathrm{D} : loc) \subset I$. Its goal state G has two components: (i) the carrier cycles have to return home at the end of the day: $cycle_{102}(1 : id, \mathrm{Home} : loc), cycle_{103}(2 : id, \mathrm{Home} : loc) \subset G$ and (ii) the goods have to be transported to their final destinations: $crate_{100}(1 : id, \mathrm{B} : loc), crate_{101}(2 : id, \mathrm{C} : loc) \subset G$.

2.3. Plans with gaps

The plans discussed above are perfect plans: whenever an instance of $In(P)$ has been determined, an instance of $Out(P)$ is guaranteed to be produced if P is executed. Sometimes, however, we have to deal with less perfect plans. In these cases plans contain *undefined actions*. An undefined action u specifies, like a real action, a relation between a set of its inputs $in(u)$ and its output resources $out(u)$. It is called an undefined action with respect to a set of actions $\mathcal{O}$ if there is no single action $o \in \mathcal{O}$ and a substitution θ such that $in(u) \supseteq in(o)\theta$ and $out(u) \subseteq out(o)\theta$. We call such an undefined action a *gap*.

DEFINITION 4 (Plan with gaps). *A plan $P = (O + U, d, \theta)$ is a plan with gaps over $\mathcal{O}$ if every action $u \in U$ is undefined with respect to $\mathcal{O}$.*

The idea of a plan with gaps is that it can be extended to a plan without gaps by substituting (other) plans for undefined actions. To this end we need the notion of *fitting* into a plan.

DEFINITION 5. *Let $P = (O + U, d, \theta)$ be a plan with gaps over $\mathcal{O}$ and $P' = (O', d', \theta')$ be a plan. If, for some $u \in U$ and a substitution σ,*

P' realizes $out(u)\sigma$ using (a subset of) $in(u)\sigma$, then P' fits into P and $P'' = (O + O' + (U - u), d'', \theta\theta'\sigma)$ is a plan with gaps over $\mathcal{O}$ using P' as a sub plan, where d'' is defined as follows:

$$d''(r) = \begin{cases} d(r) & \text{if } r \in res(P) - res(u) \\ d'(r) & \text{if } r \in cons(P') \text{ and } d'(r) \neq \bot \\ r' & \text{if } r' \in Out(P'),\ d(r) \in out(u) \text{ and } r'\sigma \equiv d(r)\sigma \\ r'' & \text{if } r \in In(P'),\ r' \in in(u) \text{ and } r'\sigma \equiv r\sigma \text{ and } d(r') = r'' \\ \bot & \text{otherwise} \end{cases}$$

Remark. In Section 3.3 we define a general operator that is capable of replacing some part of a plan by another plan. Substituting plans for gaps then is a special case of the application of this replacement operator.

This concludes the introduction to ARF and its extension to plans with dependency functions and plans with gaps. In the next section we use the (extended) ARF framework to discuss planning and replanning problems, showing that both can be defined as subproblems of a more general plan transformation problem, and we show how such transformations can be done.

3. Planning and replanning

Using the ARF terminology, traditional planning problems can be easily defined. A planning problem is a tuple $\Pi = (\mathcal{O}, I, G)$ where

1. I is a finite set of ground resources specifying the initial situation,
2. $\mathcal{O}$ is the set of possible actions an agent is capable to execute, and
3. G is a finite set of general resources specifying the goals.

A *solution* to Π is a plan $P = (O, d, \theta)$ such that

1. $O \subset \mathcal{O}$, i.e., it contains actions the agent can execute, and
2. (I, P, G) is adequate, i.e., $I \models_P G$.

Typically, traditional planning approaches implement various ways of searching for a (partially ordered) sequence of actions that leads the agent from a state I to a state that fulfills G. As remarked in the introduction, the search for a plan from scratch should be considered a limiting case of planning. For in most cases, agents start from some, although not completely adequate plan P' instead of starting from the empty plan $\diamondsuit$, and then try to *transform* it into an adequate plan.

Therefore, we consider planning to be an activity where existing plans are transformed and combined into adequate plans. This perspective also enables us to consider planning and replanning as almost identical problems.

A *replanning* problem occurs if an agent is able to achieve a set of goals G using a plan P with initial resources I, i.e., the triple (I, P, G) is adequate, but due to changes the agent discovers that its actual set of available resources is I', the realizable part of its plan is P', and the actual set of goals is G' and the triple (I', P', G') is no longer adequate. Hence, its current plan P' has to be adapted.

Note that now instances of both the planning and the replanning problem can be defined by

(i) the availability of an adequate triple (I, P, G),

(ii) the existence of an inadequate triple (I', P', G'), and

(iii) the goal of obtaining an adequate triple (I', P'', G').

In a *planning* problem, (i) denotes the availability of a plan P that has been used previously in a resource context (I, G). This plan ($P' = P$) is proposed in the current resource context (I', G') as a starting point (ii). However, by (iii) P has to be transformed to an adequate plan P''.

In a *replanning* problem, (i) means that there exists a valid plan for the initial resource context (I, G), but due to changes in the initial state, the action set and/or the goal specification, we end up with an invalid plan P' that has to be transformed into a valid plan P'' in the (changed) context (I', G').

Obviously, in most cases, such a transformation consists of several smaller transformation steps. Therefore, both planning and replanning can be described as constructing *sequences* of plan transformation steps. Of course, such plan transformation steps are guided by the available knowledge of the agent. Here, we propose to represent this knowledge by a set of available *free* plans in the form of a *plan library* that can be used by the agent to transform existing plans. A transformation step then involves the application of some plan transformation operators on the existing plan and some free plan selected from the plan library to compose a new plan.

In the next subsections we first discuss a number of plan transformation operators and we show that this set of operators is complete with respect to (re)planning using a plan library. Thereafter, we discuss some details of the plan library.

3.1. Plan operators

We introduce two simple plan operators (addition and deletion) that are used to transform a plan P using some set $\{P_j\}_{j=1}^n$ of given plans (the plan library). We show that assuming some simple properties of the plan library they form a *complete* set, that is every (source) plan P can be transformed into a target plan P' applying only the plan operators. Then we introduce a new plan transformation operator (replacement) and we show that this operator is able to simulate the two operators already introduced. Moreover, we provide a lower bound L for the number of plan transformations required given a source plan P, a target plan P' and a plan library $\{P_j\}_{j=1}^n$ and we prove that there always exists a sequence of transformations requiring no more than $2L$ applications of the replacement operator.

3.1.1. *Addition*

Analogously to the action concatenation operator used in traditional planning to construct a plan by composing actions, the addition operator $\oplus$ is an operator that glues two plans together by connecting input resources to output resources. Clearly, $\oplus$ needs the specification of a *glue function* g, like the dependency function d we used in a plan, to specify how exactly the new dependencies between in- and output resources in both plans are created. This gluing function g is a partial function overriding the specifications of the existing dependency functions d_1 and d_2 in both plans.[6] More precisely, if P_1 and P_2 are two plans to be added, whenever $r \in \mathit{In}(P_i)$ occurs in $\mathrm{dom}(g)$, $g(r) \in \mathit{Out}(P_j)$ for $i, j \in \{1, 2\}$.

DEFINITION 6 (Addition). *Let* $P_1 = (O_1, d_1, \theta_1)$ *and* $P_2 = (O_2, d_2, \theta_2)$ *be plans and* $g : \mathit{Out}(P_1) + \mathit{Out}(P_2) \rightarrow \mathit{In}(P_1) + \mathit{In}(P_2)$ *a partial dependency function mapping input resources to output resources. Then* $P_1 \oplus_g P_2$ *is defined as the plan* $P = (O_1 + O_2, d, \theta)$ *where*

1. $O = O_1 + O_2$,
2. $d = (d_1 + d_2) \dagger g$ *is a valid dependency function, and*[7]
3. $\theta = \theta_1\theta_2\theta_g$ *where* θ_g *is the mgu of the set of pairs* $\{(r\theta_1\theta_2, g(r)\theta_1\theta_2) \mid r \in \mathrm{dom}(g)\}$.

We discuss two extreme cases:

[6] Formally, the result of overriding f by g, denoted as $f \dagger g$ is the function h where $h(x) = g(x)$ if $x \in dom(g)$ and $h(x) = f(x)$ else.

[7] That is, $<_d$ is a strict partial order.

1. If $\text{dom}(g) = \emptyset$, P is the simple *concurrent combination* of P_1 and P_2, that is $P_1 \oplus_g P_2 = (O_1 + O_2, d_1 + d_2, \theta_1 + \theta_2)$;

2. If $\text{dom}(g) \neq \emptyset$ and $P_2 = \diamondsuit$, the resulting plan P is a *sequential refinement* of P_1, that is $<_d$ extends the partial order $<_{d_1}$.

Note that we always assume that whenever two plans P_1 and P_2 are combined, $\text{var}(P_1) \cap \text{var}(P_2) = \emptyset$.

Example. We continue our running example. FOO needs to find a plan for the problem discussed in the previous example. Since there is no current plan to work on yet, the first step is to select a proper starting plan. Suppose that the planner selects the plan P depicted in Figure 2.[8] The starting plan is shown in Figure 3.a. To embed the plan P in the context (I, P) we create a substitution σ binding goals from G to output resources $Out(P)$ of the plan and input resources $In(P)$ to the initial resources I. The result of this substitution could be the plan of Figure 3.b. Here, we have chosen to use the resources $cycle(2 : id, \text{Home} : loc)$ and $crate(1 : id, \text{Home} : loc)$ to satisfy the input resources of the plan. Also, we have satisfied the goal $crate(1 : id, \text{B} : loc)$.

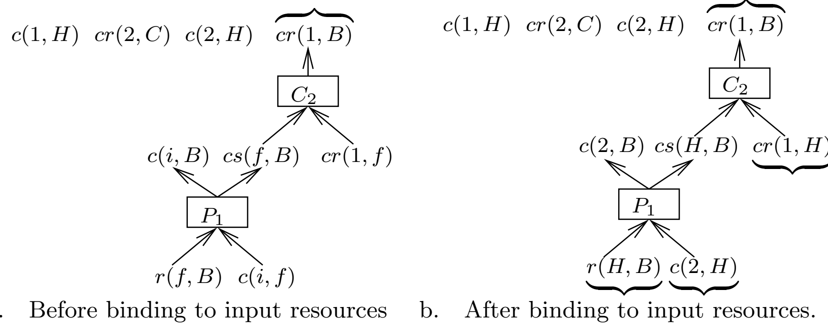

Figure 3. Partial plan after one planning step. Resources linked to an initial resource are denoted by $\smile$, whereas $\frown$ denotes a resource that is used to satisfy a goal.

To continue our example, suppose that the next resource that we try to satisfy is the $cycle(2 : id, \text{Home} : loc)$ resource. Obviously, we can satisfy this goal by riding the bike from B to Home. So, let's suppose that our planner has a plan P_1 available that contains just the *cycle* action. The connection relation g then contains one link from the $cycle(2 : id, \text{B} : loc)$ resource of the current partial plan to the *cycle* resource that is an input to plan P_1.

We state a simple property of the addition operator:

[8] Deciding which plan to add is of course the hard part, this is discussed later on in the paper.

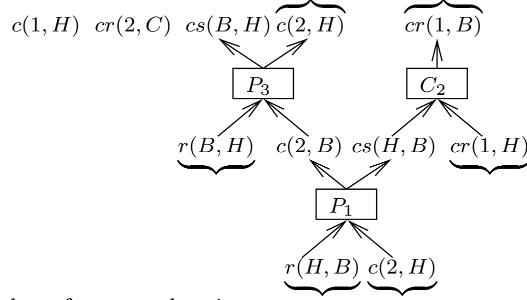

Figure 4. Partial plan after two planning steps.

PROPOSITION 2. *If $P_1 = (O_1, d_1, \theta_1)$ is a subplan of $P = (O, d, \theta)$ then there always exists a dependency function g such that $P = P_1 \oplus_g P_2$ for some subplan $P_2 = (O - O_1, d_2, \theta_2$ of P.*

Proof. Note that d_2 can be specified as the restriction of the dependency function d to the set $O - O_1$. Then g is the partial dependency function consisting of those pairs (r, r') that do occur in d but not in d_1 or d_2. □

3.1.2. *Deletion*

Addition extends a plan, e.g., in order to obtain new (goal) resources available in an other plan from the plan library. Sometimes, however, the resources we need are already available in a plan P, but are not available as output resources because they are consumed by some action. In that case we like to free those resources and even are prepared to delete some actions depending on these resources.[9] We therefore define an operator $\ominus$ that is able (i) to free resources and (ii) to delete actions by updating the dependency function using a partial dependency function g in a plan P and by specifying the set of actions to be removed from P.

More specifically, $\ominus$ takes two plans P_1 and P_2 where P_2 is a subplan of P_1 specifying the actions to be removed from P_1, and a partial dependency function g. The purpose of g is to specify which dependencies occurring in d have to be removed additionally, in order to free resources in P_1. Hence, the domain of g consists of resources occurring in the input set $in(o)$ of some actions in the given plan P_1, mapping them to $\perp$ and thereby overriding the definition of d.

DEFINITION 7 (Deletion). *Let $P_1 = (O_1, d_1, \theta_1)$ be a plan and $P_2 = (O_2, d_2, \theta_2)$ be a subplan of P_1. Let g be a partial function such that*

[9] The deletion operator can be applied to a plan in the plan library, or directly to the current plan to be transformed.

$(r, \perp) \in g$ and $\mathrm{dom}(g) \subseteq \mathrm{dom}(d_1)$. The plan $P = P_1 \ominus_g P_2$ then is defined as the plan $P = (O, d, \theta)$ where

1. *$O = O_1 - O_2$; that is all actions in O_2 are removed.*
2. *$d = ((d_1 - d_2) \dagger h) \dagger g$ where h is the dependency function defined as $h(r) = \perp$ whenever $d(r) \in res(O_2)$*
3. *θ is the mgu of all pairs $(r, d(r'))$ occurring in d.*

We distinguish two extreme cases:

1. $P_2 = \diamondsuit$. Then $\ominus_g$ acts as an operator only freeing up resources by removing dependencies between resources in plans, without removing actions from P.
2. When g is the empty function and $P_2 \neq \diamondsuit$, the resulting plan is the plan P_1 after removing the actions and dependencies from P_2.

As expected, there exists a partial duality between the operators $\oplus$ and $\ominus$. Formally, the relation between $\ominus_\emptyset$ and the operator $\oplus$ can be described as follows.

PROPOSITION 3. *Suppose that a plan P consists of two sub plans P_1 and P_2 such that P can be written as $P = P_1 \oplus_g P_2$ for some gluing function g. Then it holds that $P \ominus_\emptyset P_2 = P_1$.*

Proof. Immediately from the definitions of $\oplus$ and $\ominus$. □

Example. Let P be the (partial) plan that we constructed in the previous example. Suppose that we remove the $pedal_3$ action again that was added during the last step. The subplan P' of P that we then have to remove from P is $P' = (\{pedal_3\}, \emptyset, \emptyset)$. To free the resources that $pedal_3$ consumes, the connection function g maps those resources that are produced by other actions and which the $pedal_3$ action depends on to $\perp$: $g = \{r \rightarrow \perp | d^{-1}(r) \in in(pedal_3)\} = \{cycle(2 : id, \mathrm{B} : loc) \rightarrow \perp\}$. Then, $P \ominus_g P'$ results in the plan with the $pedal_3$ action removed (i.e., the plan of Figure 3.b).

3.2. Completeness of the addition and deletion operators

An important question is whether the two plan operators we just discussed are complete in the sense that every existing (inadequate) plan can be transformed into an adequate plan by applying the plan addition and the plan deletion operators.

Clearly, this completeness property is relative to the properties of the plan library that can be used to perform addition and deletion. We assume the following properties to hold for the set of plans $\{P_j\}$ occurring in the plan library:

Assumption 1
: Every plan is completely *accessible*, that is, for every plan in the plan library, its complete structure is available to the planner and the planner is always capable to retrieve any subplan from it.

Assumption 2
: Every action o that belongs to the capability of a planning agent A occurs in some plan occurring in its plan library.

We show that if these two assumptions are met, every existing plan P can be transformed into an adequate plan P' using only the addition and deletion operators.

First we show a simple result stating that by using a series of deletion operators we always can extract from any plan P containing an action o a subplan P_o containing only the action o.

PROPOSITION 4. *Let P be a plan containing an action o. Then there exists a sequence of plan operators resulting in the plan P_o containing exactly the action o.*

Proof. Since P contains o, P can be written as the composition of three plans:

1. The subplan $P_1 = (O_1, d_1, \theta_1)$ of P generated by all the actions the input resources occurring in $in(o)$ are dependent upon, i.e., the subplan below o;

2. The subplan P_2 of P generated by all the actions $in(o)$ is *not* dependent upon i.e., the plan generated by $O - O_1 - \{o\}$.

3. The plan P_o exactly consisting of the action o.

By Assumption 1, these plans can be specified if P is an existing plan, since P_1, P_2 and P_o are sub plans of P. By Proposition 2, there exists some $g_1, g_2 \subseteq d$ such that P can be written as $P = (P_1 \oplus_{g_1} P_o) \oplus_{g_2} P_2$. But then, by Proposition 3, $P_o = (P \ominus_{\emptyset} P_2) \ominus_{\emptyset} P_1$. □

The following corollary is an immediate generalization of the preceding proposition:

COROLLARY 1. *Let $P = (O, d, \theta)$ be a plan and If $O' \subseteq O$ an arbitrary subset of actions. Then there exists a sequence of plan deletion operators that applied to P results in a plan P' just containing O' as a set of independently executable actions, that is $P' = (O', \emptyset, id)$.*

Using this proposition and corollary it is not difficult to prove the result that the set $\{\oplus, \ominus\}$ is a complete set of plan operators under the assumptions stated:

PROPOSITION 5. *Suppose (I, P, G) is inadequate. If there exists an adequate plan (I, P', G) and there exists a plan library $\{P_j\}_{j=1}^{n}$ such that every action o occurring in P' also occurs in some P_j, then there exists at least one series of plan transformations which, when applied to P and the set $\{P_j\}$, results in the plan P'.*

Proof. [Sketch] Let $P = (O, d, \theta)$ and $P' = (O', d', \theta)$. Let $O_s = O \cap O'$. By the previous corollary, there exists a series of deletion operators which, applied to P, result in a plan $P_s = (O_s, \emptyset, id)$. By the previous proposition, there also exists a series of applications of deletion operators to plans resulting in a set of plans P_o for every $o \in O' - O$. Finally, using the addition operator, assemble the plan P' from the plans P_s and $\{P_o\}_{o \in O' - O}$. □

3.3. Replacement

Instead of using two separate plan operators, we would like to combine both operators into one single plan operator that is able to simulate them both. To this end we introduce the replacement operator $\otimes$ that serves to replace a subplan of a plan P by some other plan P'. Like the deletion operator, the replacement operator needs an additional parameter specifying the subplan to be replaced. In addition, the replacement operator also needs a parameter specifying how the new plan is attached to the existing plan. We therefore use an operator $\otimes$ with three parameters: the subplan to be deleted from P, a dependency function g to free up additional resources and a dependency function h specifying how the new plan P' should be glued to free resources in P.

DEFINITION 8. *Let $P_1 = (O_1, d_1, \theta_1)$ and $P_2 = (O_2, d_2, \theta_2)$ be plans, P_1' a subplan of P_1 and g, h be partial dependency functions. The plan $P = P_1 \otimes_{P_1', g, h} P_2$ is defined as the plan $P = (O, d, \theta)$ where:*

1. $O = (O_1 - O_1') + O_2$;
2. $d = ((((r, r') \in d_1 \mid r, r' \notin res(P_1')\}) \dagger g) + d_2) \dagger h$

3. θ is the unifier of the dependency pairs in d.

We show that $\otimes$ is able to simulate both the addition and the deletion operator.

PROPOSITION 6. *The replacement operator is able to simulate both the plan addition as well as the plan deletion operator.*

Proof. Let $P = P_1 \oplus_g P_2$. Then by definition of the replacement operator, $P = P_1 \otimes_{\not\Diamond,\emptyset,g} P_2$. Note that here g is used to glue the replacement for the empty plan to P_1.

Let $P = P_1 \ominus_g P_2$. Then $P = P_1 \otimes_{P_2,g,\emptyset} \not\Diamond$. That is, plan deletion is the same as replacing a subplan by an empty plan. □

Example. Suppose that the carrier cycle that FOO was going to use to transport the first crate brakes down, i.e., the resource $cycle(2 : id, \text{Home} : loc)$ is no longer available. Quickly, they hire a van and modify the domain to include a *van* resource and a *drive* action. The *van* resource is defined analogously to a *cycle* resource, whereas the *drive* is identical to a *pedal* action, except that the former consumes and produces a *van* resource.

In order to repair the partial plan P of Figure 4, the *pedal* actions have to be replaced by *drive* actions. Let P_{van} be the plan that consists of executing two *drive* actions in sequence, moving a van two steps. Furthermore, let P_{cycle} be the plan P restricted to the set of cycle actions. Then, we can replace the *pedal* actions by *drive* actions by executing $P \otimes_{P_{cycle},g_{del},g_{add}} P_{van}$, where $g_{del} = \{cargospace_{P_1}(\text{Home} : loc, \text{B} : loc) \rightarrow \bot\}$ and $g_{add} = \{cargospace_D(\text{Home} : loc, \text{B} : loc) \rightarrow cargospace_{C_2}(\text{Home} : loc, \text{B} : loc)\}$.[10]

3.4. A stronger completeness result for replacement

As an immediate corollary of Proposition 6 we have that the replacement operator is a complete operator. In fact, we can show a somewhat stronger result stating that the minimum number of applications of the replacement operator needed to transform a plan P into another plan P' using a plan library $\mathcal{P} = \{P_j\}_{j=1}^n$ is about twice the minimum number of plan transformations that are needed in any case. [11]

To determine this lower bound, let us first define some additional notions. Let $P' = (O', d', \theta')$ be a plan and $\{P_j = (O_j, d_j, \theta_j)\}_{j=1}^n$ a set

[10] Here, we denote a resource r that is input or output of the action o by r_o.

[11] Here, we assume that whenever a plan P_j from the plan library is involved in composing P' at least one transformation step is needed.

of plans. A *labeling* for P' using $\{P_j\}$ is a function $\lambda : O' \to \bigcup_j O_j$ such that

1. $\lambda(o') = o_j$ implies that $o'\theta = o_j\theta$ for some substitution θ, i.e., the actions with their input and output resources are compatible;

2. $\lambda(o') = \lambda(o'')$ implies $o' = o''$, i.e. an action o_j cannot be used twice as a label for different actions in O' (hence λ is injective).

The size $size(\lambda)$ of a labeling λ is the cardinality of the set $\{P_j \mid \exists o' \in O'[\lambda(o') \in O_j]\}$. Intuitively, $size(\lambda)$ is a lower bound for the number of operations needed to obtain P' from the plans used in the plan library if $size(\lambda)$ different plans are used from this library to compose P'.

The labeling λ for P' is said to be *minimal* with respect to a subset $T \subseteq \{P_j\}_{j=1}^n$ if a *minimum number* of plans in T is used to label the actions in P'. That is, as few plans from T as possible are involved in composing P' using $\{P_j\}$. Now we can define a lower bound on the number of transformations needed to transform a plan P to a plan P' as follows.

PROPOSITION 7. *Let P be a source plan, $\{P_j\}$ be a plan library and P' a target plan. Then the minimum number of transformation steps needed to transform P to P' using plans in $\{P\} + \{P_j\}$ equals at least $size(\lambda) - 1$ where λ is a minimal labeling of P' with respect to $\{P_j\}$ using plans in $\{P\} + \{P_j\}$.*

Proof. [Sketch] Since λ is minimal with respect to $\{P_j\}$, if an action in O' is labeled with an action in some P_j, the plan P_j used at least once in a transformation step. Therefore, there are at least $size(\lambda) - 1$ (the number of plans required besides P) different transformation steps needed. □

Now our completeness result can be stated as follows;

PROPOSITION 8. *Let P be a source plan, $\{P_j\}_{j=1}^n$ be a plan library and P' a target plan. Let λ be a minimal labeling with respect to $\{P_j\}$, using $\{P\} + \{P_j\}$. Then there exists a transformation from P to P' using at most $2size(\lambda) - 1$ applications of the replacement operator.*

Proof. [Sketch] Consider the actions o' in P' such that $\lambda(o') \in O$. Apply a replacement operator on P replacing the set of actions o not occurring in $ran(\lambda)$ by $\diamondsuit$ and returning a plan consisting of a set of independent actions. Then, for each plan P_j used in λ apply the replacement operator once to remove all actions not used in the labeling and once to insert the remaining actions into the current plan. This requires $1 + 2(size(\lambda) - 1) = 2size(\lambda) - 1$ transformation steps. □

Note that the plan operators we introduced require a current plan P to be transformed and another plan P' used to transform P. Hence, for an agent to be able to use these plan operators it must have access to a set of such plans P'. For this purpose, we assumed that the agent has a knowledge base containing plans to choose from. We called this knowledge base the *plan library*. We now discuss some aspects of such a plan library into more detail.

In its simplest form, a plan library can be conceived as a simple collection of plans. Often, however, it occurs that a number of plans share the same subplan or two plans are exactly equal except for some subplan in which they differ. In both cases, we could easily reduce the size of the plan library by using *plans with gaps* instead of complete plans. For example, suppose that $P_1, \dots P_k$ share a common subplan P', we might replace every occurrence of P' in each P_i by an action $u \in U$ and add the subplan P' to the library, effectively reducing its size with $(k-1)(||P'||-1)$.

If the plan library may contain plans with gaps, we need additional constraints to hold for the library in order to guarantee that valid plans can be created out of plans with gaps. An intuitive constraint is that for each gap u that is present in a plan P from the library, another plan P' in the library must fit in the gap, i.e., a plan P' such that $in(u) \models_{P'} out(u)$. The resulting plan is the result of replacing u by P' in P and is written as $P \otimes_u P'$.

This requirement, however, does not solve our problem: for example, take a library with a plan A with a gap u_a and a plan B with a gap u_b where B fits u_a and A fits u_b. If there are no other plans that fit in either u_a or u_b, then it is not possible to create a valid ground plan out of these plans. The following requirement prevents such infinite regress to occur:

DEFINITION 9. *A plan library L is* groundable *if for each plan P with a nonempty set U_P of gaps it holds that there there exists a finite subset of plans $P_1, P_2, \dots P_k$ from L such that*

1. *for every $i = 1, 2, \dots, k-1$ the plans $P'_0 = P$, $P'_{i+1} = P'_i \otimes_{u_i} P_i$ are well-defined, where each u_i is a gap occurring in $U_{P'_i}$;*

2. $|U_{P'_k}| < |U_{P'_0}| = U_P$

Example. Figure 5 shows the plan library of FOO. It includes plans to reach 3 goals: P_1 can be used to satisfy a *cycle* goal (i.e. having a cycle at a certain location) by using a *pedal* action. The second plan, P_2, can satisfy a *van* goal using a *drive* action. Finally, the last goal,

a *crate* resource at a specific location, can be satisfied by two plans P_3 and P_4. The former uses a *pedal* action to deliver the goods by cycle, the latter uses a *drive* action to deliver using a van. Note that this library does not contain plans with gaps. Therefore, this is a groundable plan library. A more interesting example can be found if we introduce the

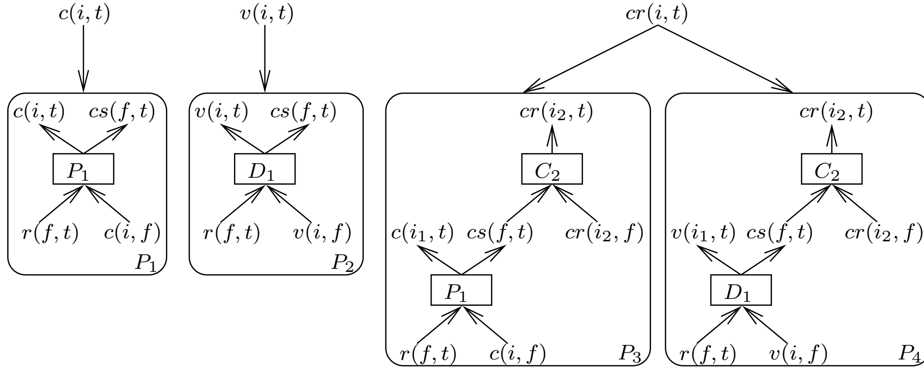

Figure 5. FOO's plan library.

following action:

$$\begin{aligned} \textit{combine}: \quad & \textit{cargospace}(a:\textit{loc}, b:\textit{loc}), \textit{cargospace}(b:\textit{loc}, c:\textit{loc}) \Rightarrow \\ & \textit{cargospace}(a:\textit{loc}, c:\textit{loc}) \end{aligned}$$

This action simply states that if it is possible to transport cargo from a to b and from b to c, then it is also possible to transport cargo from a to c. Now, we can add the plans of Figure 6 to our plan library. Assuming the availability of $road(l, l)$ resources, this library allows us to combine an arbitrary number of *cargospace* resources created by *pedal* actions into a single *cargospace* resource, since both P_5 and P_6 fit into the gap of P_5. Note that this is a groundable plan library, since P_6 has no gaps.

4. A refinement (re)planning framework

To create and/or improve plans using the addition and deletion operators (or the replacement operator) and the plan library presented in the previous section, we have two options: Either we can design new algorithms to implement these operators, or we can show how existing algorithms and heuristics can be used to implement them. Choosing for the last option, in this section we reformulate and generalize the classical refinement planning template algorithm [22] in order to use it

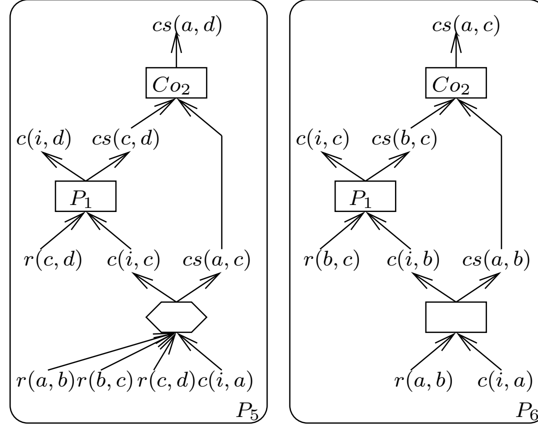

Figure 6. A more interesting plan library. We use ⬭ to denote a gap.

in the ARF-replanning framework including the use of plan libraries. Hereafter, we present two examples of existing planning techniques that can be seen as instances of this new refinement template algorithm.

An important distinction between the ARF refinement framework and the classical planning framework is that in the ARF framework replanning is included. Consequently, if a replanning step is performed (during a refinement step), the set of possible candidates may be enlarged, while classically it is required that the size of the set of possible candidates monotonically decreases.

4.1. A TEMPLATE ALGORITHM

As Kambhampati [22] argued, planning approaches do not only have in common the data structures they use, but they also share a common algorithmic structure. This common part, reformulated in ARF framework, is presented in the REFINE algorithm (Algorithm 1). The part that is different for each existing planning algorithm and depends on the specific planning method used is represented by the REFINESTEP (Algorithm 2).

The REFINE algorithm specifies how a solution *result* can be obtained from a partial plan P. Each time the function REFINE is executed and P is not a solution, one or more refined versions of this plan are produced in a refinement step. Then repeatedly a component of the resulting set $\mathcal{P}$ is selected, and the function REFINE is called recursively using this component. Once a solution has been found, this process stops, and the result is returned.

Algorithm 1 (REFINE $((I, P, \sigma, G), L)$)

Input: *An embedded plan* (I, P, σ, G) *and a plan library* L

Output: *A plan to attain G with I or 'fail'.*

begin

1. **if** $I \models In(P)\sigma$ *and* $Out(P)\sigma \models G\sigma$ *and* P *does not have any gaps* **then**
   1.1. ***return*** P*;*

2. **else**
   2.1. $result :=$ *fail;*
   2.2. $\mathcal{P} :=$ RefineStep $((I, P, \sigma, G), L)$*;*
   2.3. **while** $\mathcal{P} \neq \emptyset$ *and* $result =$ *fail* **do**
      2.3.1. *select a plan* P_i *of* $\mathcal{P}$*;*
      2.3.2. $\mathcal{P} := \mathcal{P} - \{P_i\}$*;*
      2.3.3. *determine a suitable* σ_i *for* P_i*;*[12]
      2.3.4. $result :=$ Refine$((I, P_i, \sigma_i, G), L)$*;*
   2.4. ***return*** *result;*

end

The RefineStep that is called in line 2.2 is described in Algorithm 2. Because this part of the planning algorithm is different for each existing planning method, we present a common template for these methods. Some instances (i.e., planning methods) may skip some of these steps, and others pay much more attention to one or more other steps. Different choices for each of the steps lead to different algorithms.

In the template algorithm we treat gaps in a plan identically to missing resources. The subgoals R are selected from all resources that need to be obtained: goals, missing input resources and gaps. For these selected subgoals one or more possible plans are selected from the plan library to attain these subgoals. The selected plans are combined with the original plan P. The result of one plan step is a set of one or more combinations of a selected plan with P.

Algorithm 2 (RefineStep $((I, P, \sigma, G), L)$)

Input: *An embedded plan* (I, P, σ, G) *and a plan library* L

Output: *A set of possible refinements* $\mathcal{P}$

begin

1. *select subgoals* $R \subseteq (G\sigma - Out(P)\sigma) \cup (In(P)\sigma - I) \cup \bigcup_{u \in P} out(u)$*;*

2. *select one or more subplans* $P_i \in L$ *to attain* R*;*

3. **for** ***each*** P_i **do**
   determine a subplan P'_i *of* P *and functions* g_i *and* h_i *to combine* P *and* P_i*;*

4. $\mathcal{P} := \left\{ P \otimes_{P'_i, g_i, h_i} P_i \middle| P_i \right\}$*;*

5. **return** $\mathcal{P}$*;*

end

[12] For the sake of clarity, we omitted further details here.

4.2. Examples of instances of the refine step template algorithm

To illustrate that existing planning methods can be reformulated in the ARF-framework, we describe two planning algorithms as instances of the RefineStep template algorithm.

4.2.1. *Fast Forward*

The planning algorithm Fast Forward (FF) [20] starts with the initial state and an empty plan. Repeatedly, the plan is extended with some actions, always adding to the end of the plan. For each of the possible extensions of the plan (first with one action, then with two actions, etc.), a heuristic value for the current state is calculated. The first possible extension leading to a state with a lower heuristic value is chosen.

The heuristic uses a relaxation of the planning problem to calculate a heuristic value efficiently: it is assumed that actions reproduce their consumed resources ($in(o) \rightarrow out(o) \cup in(o)$). Therefore, a so-called relaxed plan can be constructed where all actions with satisfied inputs are added (possibly even in parallel) until the goal state is reached. The heuristic value is the cost of all actions that are needed to reach the goal state.

Although FF does not use the presented refinement framework, and is not about producing resources, it is in fact a form of refinement planning with the following refinement step. Given a plan that represents a set of possible solutions, a new plan is constructed by extending this partial plan at the end. The extension is selected using the heuristic described above. In Algorithm 3 this algorithm is presented as an instance of the template given in Algorithm 2. As a proof-of-concept, this algorithm has also been implemented [25].

Algorithm 3 (RefineStep-FF $((I, P, \sigma, G), L)$)

Input: *An embedded plan* (I, P, σ, G) *and a plan library* L

Output: *A set of possible refinements* $\mathcal{P}$

begin

1. *take all subgoals* $R = (G\sigma - Out(P)\sigma) \cup (In(P)\sigma - I)$;
2. *use the following heuristic to compute a plan* Pi:
   - 2.1. $h :=$ *the heuristic costs of producing* R *from the current end state;*
   - 2.2. $h' := \infty$;
   - 2.3. **while** $h' \geq h$ **do**
     - 2.3.1. *breadth-first select a sequence of actions* P_1 *from* L *to extend* P;
     - 2.3.2. *determine a sub plan* P_1' *of* P *and functions* g_1 *and* h_1 *to combine* P *and* P_1;

2.3.3. $h' :=$ *the heuristic value of the end state of* $P \oplus_{P_1',g_1,h_1} P_1$;

2.3.4. $\mathcal{P} := \{P \otimes_{P_1',g_1,h_1} P_1\}$;

3. select the functions g_1 *and* h_1 *as computed in the previous step;*

4. $\mathcal{P} := \left\{ P \otimes_{P_i',g_i,h_i} P_i \middle| P_i \right\}$;

5. **return** $\mathcal{P}$;

end

4.2.2. *Plan adaptation*

As a second example, we show that the SPA [19] algorithm for plan adaptation can also be reformulated in the ARF. This system is based on the least commitment approach [29].

After retrieving an initial plan from their plan library, the SPA system starts an adaptation routine that can either *extend* the plan (adding further constraints or actions), or *retract* decisions that have been made. This adaptation algorithm performs a breadth-first search. The lists of nodes that has to be expanded is kept in a list of the form $\langle P, \uparrow\rangle$ or $\langle P, \downarrow\rangle$, meaning that a plan may be further refined (in case of a $\downarrow$) or that a decision may be retracted (signified by a $\uparrow$). Figure 7 shows what happens when a node is further refined or retracted from.

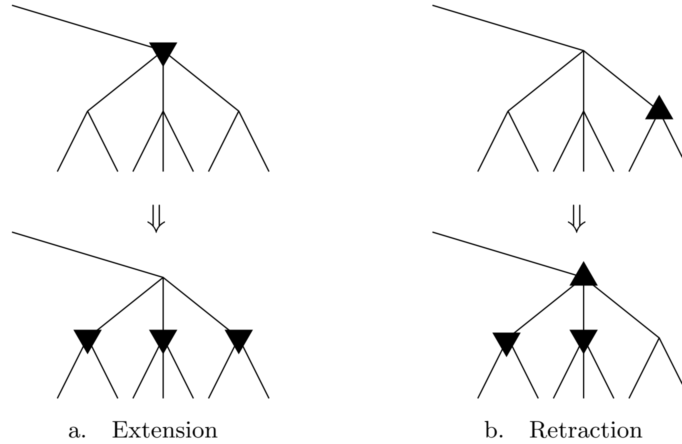

Figure 7. Extension and retraction as search in plan space. Extension replaces a plan tagged $\downarrow$ (represented by ▼) by its children, tagged $\downarrow$. Retraction replaces a plan tagged $\uparrow$ (▲ in the figure) by its parent (tagged $\uparrow$) and its siblings (tagged $\downarrow$).

Note that the refinement strategy of SPA does not necessarily reduce the set of candidate plans in every step. Actually, the set of candidate plans may grow during the refinement. The refinement strategy for SPA is shown in Algorithm 4. There is a twofold difference with the standard template: (i) depending on whether the plan is tagged $\uparrow$ or $\downarrow$ SPA can remove parts of the plan or add them; (ii) in the case of plan retraction,

step 4 includes an extra call to RefineStep-SPA. Since SPA requires the generation of sibling plans, we first calculate a set $\mathcal{P}'$ containing the parents (step 4.1), and calculate the children of these parents in step 4.3 (using a recursive call). Finally, step 4.4 combines the parents and their children, excluding the plan P as this is already processed.

Algorithm 4 (RefineStep-SPA $((I, P, \sigma, G), L)$)

Input: *An embedded plan* (I, P, σ, G) *and a plan library* L
Output: *A set of possible refinements* $\mathcal{P}$

begin

1. *select subgoals* $R \subseteq (G\sigma - Out(P)\sigma) \cup (In(P)\sigma - I)$;
2. *select one or more subplans* $P_i \in L$ *to attain* R *if* P *is tagged* $\uparrow$, *or the empty plan* $\diamondsuit$ *if* P *is tagged* $\downarrow$;
3. **for *each*** P_i **do**
   determine a subplan P_i' *of* P *and functions* g_i *and* h_i *to add* $P_i \neq \diamondsuit$ *to* P *or determine subplans* P_i' *of* P *that can be removed from* P *to free* R;
4. **if** P *is tagged* $\uparrow$ **then**
   4.1. $\mathcal{P}' := \left\{ \langle P \otimes_{P_i', g_i, h_i} P_i, \uparrow \rangle \,\middle|\, P_i \right\}$;
   4.2. *determine a suitable* σ_i *for each* $P_i \in \mathcal{P}'$;
   4.3. $\mathcal{P}'' := \bigcup_{P_i' \in \mathcal{P}'}$ RefineStep-SPA$((I, \langle P_i', \downarrow \rangle, \sigma_i, G), L)$;
   4.4. $\mathcal{P} := \mathcal{P}' \cup \mathcal{P}'' \setminus \{P, \downarrow\}$;
5. **else**
   5.1. $\mathcal{P} := \left\{ \langle P \otimes_{P_i', g_i, h_i} P_i, \downarrow \rangle \,\middle|\, P_i \right\}$;
6. **return** $\mathcal{P}$;

end

Remark. The refinement planning framework [22] for planning problems using propositional logic is strongly related to the algorithms presented in this section. The main difference lies in the distinct representation of the data structure that is used to represent partial solutions. In Kambhampati's [22] framework, so-called partial-plans are introduced to be able to represent all (dependency) constraints required for the many planning algorithms. These constraints can be divided into three categories.

1. Precedence constraints specify that one action should be included before another action in the eventual sequence of actions.

2. Interval preservation constraints specify that a formula should hold during a certain interval between two actions. Currently, the only used instantiations of such constraints are so-called causal links, used by least-commitment planners [29].

3. Point truth constraints specify that a formula should hold at a certain point before an action. In current planners these constraints are only used to specify open preconditions of actions.

All these different types of constraints are modeled by the dependency relation in the ARF. The dependency relation specifies why one action should be executed before another. This is equivalent to a causal link. The transitive closure of this relation can be used to model the precedence constraints. Finally, the fact that for an input resource of an action no dependency is specified yet, describes the open preconditions of actions.

5. Multi-agent aspects

So far, we have discussed planning and replanning from a single-agent perspective. In this section we show how the framework can be used to deal with multi-agent planning. To this end, we first proof that every (re)planning problem can be reduced to a *missing resources problem.* Next, we show how such missing resources problems can be solved (i) by using services other agents offer (service-based coordination), or (ii) by requesting missing resources from other agents (task-based coordination).

The following proposition shows that every (re)planning problem can be reduced to a (minimal) set of missing resources.

PROPOSITION 9. *Let (I, P, G) be an inadequate tuple and let $R^1_{missing}$ be a smallest set of resources such that P' can be applied to $I' = I \cup R^1_{missing}$ using a substitution σ, i.e., $In(P)\sigma \subseteq I'$. Furthermore, let $G' \subseteq G$ be the largest subset of G such that G' is satisfied by the result of applying P' to I', i.e., $G'\tau \subseteq I' - In(P)\sigma \cup Out(P)\sigma$ for a ground substitution τ. Finally, let $R^2_{missing}$ be a smallest set of resources satisfying the remaining set of goals $G - G'$, i.e., $R^2_{missing} = (G - G')\tau\theta$ for some ground substitution θ. Then the triple $((I \cup R^1_{missing} \cup R^2_{missing}), P, G)$ is adequate.*

Proof. Since P is applicable to $I' = I \cup R^1_{missing}$ using σ, it follows that $I' \models_{P'} I' - in(P')\sigma \cup Out(P)\sigma$. By monotony, $I'' \cup R^2_{missing} \models_P I' - in(P)\sigma \cup Out(P)\sigma \cup R^2_{missing}$. But then, since $G'\tau \subseteq I' - In(P)\sigma \cup Out(P)\sigma$, and $R^2_{missing} = (G - G')\tau\theta$, we immediately derive that

$$G\tau\theta = G'\tau\theta \cup (G - G')\tau\theta \subseteq I' \cup R^2_{missing} - in(P')\sigma \cup Out(P)\sigma.$$

Hence, $I \cup R^1_{missing} \cup R^2_{missing} = I' \cup R^2_{missing} \models_P G.$ □

For a single agent, missing resources can only be provided by its initial resources or be produced from the initial resources by some actions. In a multi-agent context, other agents can provide resources. For example, because other agents are capable of doing other actions. A *multi-agent planning problem* for a set of agents A is defined as a set of planning problems $\Pi_a = (\mathcal{O}_a, I_a, G_a)$, one for each agent $a \in A$.

We distinguish two ways to coordinate the exchange of resources in a multi-agent system. First, we discuss service-based coordination where an agent can use services *offered* by others. This approach has some strong similarities to the use of a plan library with plans provided by other agents. Secondly, we propose a method where agents *request* specific resources from others. This latter approach is a form of task allocation and we argue that it is a generalization of plan merging [7].

5.1. Service-based coordination

Service-based coordination is based on the idea that an agent can use the capabilities of other agents as if these are included as additional plans in its plan library. Depending on the environment, we distinguish two kinds of services. On the one hand, when agents trust each other, services are free plans that are offered to others. These plans can be included directly in the plan library of these other agents (*requesters*). Subsequently, these free plans can be treated in the same way as the other plans in the plan library. On the other hand, in a less secure environment, a service is a free plan of which only the input and output resources are published to other agents. In other words, this plan is not accessible to the requester.

In both cases, when an agent wishes to use a service from another agent (the *provider*), the agents have to agree on the actual resources that are produced. When the requester has completed its plan, it tells the provider that it wishes to include the provider's service S to produce a certain set of resources R_G using a set of resources R_I.[13] It is required that $in(S) \models R_I$ and $out(S) \models R_G$. The provider then verifies that it can indeed produce R_G from R_I using the underlying plan P_S and adds R_I to its available input resources, R_G to its goals and P_S to its plan.

By extending the plan library with services from other agents, we create a natural extension of single-agent planning to multi-agent planning. The only extension to the (re)planning algorithm is that it is now possible that an agent may have to contact other agents after creating a plan to inform them that services have been used (i.e., contracting the services). This form of multi-agent planning is an instance of the refine-

[13] In fact, for an plan that is not accessible, S is added as an (unknown) action in the requester's plan with a reference to the provider.

ment planning algorithm from the previous section (see Algorithm 1). The only difference is that in line 1.1 for each service that is used, the corresponding provider has to be informed that (part of) its service is used.

Whether serviced plans are accessible, does not change the basic outline of the algorithm but (at the cost of privacy) this method is more powerful: it allows agents to use only a subplan of the service. This way, they can combine parts of various services, when a single service (or plan of their own) cannot achieve the desired effect. This gives requesters greater control and flexibility.

An alternative to exchanging services is to let an agent inquire whether other agents can provide the resources it needs, without knowing which agent is able to provide these. This is a case where *task-based* coordination is needed.

5.2. Task-based coordination

An even more flexible solution to (re)planning problems can be offered in a multi-agent environment if agents are able to request missing resources from others without knowing beforehand which other agents are able to provide them these resources.

In this case, the contract net protocol [6] can be used to coordinate the communication between requesters (managers) and providers (contractors). As can be seen in Figure 5.2, this protocol uses three types of messages.

- Task(I_1, G_1) is sent by the requester agent to announce a task to get from a state I_1 to a state G_1. The agent (a_1 in Figure 5.2) that sends this message is called a provider (for this specific contract). Usually, such an announcement is sent to many other agents.

- Bid(I_2, P_2, G_2, c) contains information about in what way an agent (a_2) can deal with (part of) the task: an alternative initial state I_2, an alternative goal G_2, for which holds that $G_2 \cap G_1 \neq \emptyset$, and a plan P_2 to attain this goal from this initial state. This message can also include the costs the requester has to pay if it awards this agent a_2 the task (I_2, G_2). In multi-agent systems where agents do not trust each other, the plan P_2 may be omitted, just like in the service-based approach.

- Award(I_3, P_3, G_3) is sent only to an agent from which the requester previously has received a Bid(I_2, P_2, G_2, c). This means that the requester is prepared to pay the costs c for a sub plan P_3 of P_2.

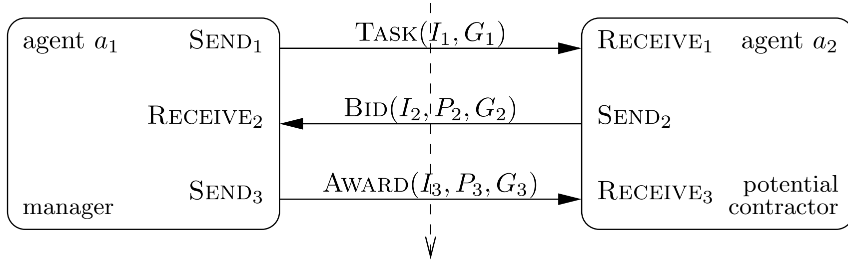

Figure 8. The messages that are exchanged between a requester and a provider to coordinate a task delegation.

Example. A task announcement can be done by an agent that has some missing resources in the sense of Proposition 9. Given an inadequate tuple (I, P, G), the initial state I_1 and the goals G_1 of the task can be found as follows. $I_1 = I - In(P)\theta$ and $G_1 = R^1_{missing} \cup R^2_{missing}$.

When a requester has received enough bids (I_2, P_2, G_2) to attain its goals, it needs to find a selection of these bids, and a series of operators such that the combination is an adequate plan. One possible way to do this, is by putting all received bids in a plan library, and using a refinement planning algorithm. For each bid where a (part of its) plan is used, the agent should receive a corresponding REWARD message.

Once a requester has awarded one or more agents with parts of the task, these results can be incorporated in the plan of the requester with the found plan operators. For the providers, the award (I_3, P_3, G_3) is added to their initial resource, their plan (using the addition operator), and their goals, respectively.

This method has a strong relation with the plan-merging algorithm [7]. In this previous work, we showed under which conditions resources of one agent can be used by another. The plan-merging algorithm itself describes that a request for resources G_1 can be dealt with by asking all agents for (a subset of) these resources. This is equivalent to a task announcement of $(\emptyset, G_1)$. Furthermore, in the plan-merging approach no plan is included in the bids and awards.

6. Conclusions and future work

For an agent to be autonomous, it must be able to decide which actions to take to accomplish its goals. Therefore, planning is a key aspect of agents. We started this paper by noting that there exists a number of objections to the way planning is conceptualized in a lot of planning systems. These planners work off-line, have no memory of what

problems they have solved in the past, or cannot be used to create multi-agent plans. While there are systems that tackle some of these problems, we are not aware of a consistent integrating approach. In this paper we have discussed a conceptually rather simple framework, the Action Resource Formalism (ARF), where on-line planning is combined with a history of past experience (in the spirit of case-based planning) and cooperative planning.

In the ARF actions are seen as processes that consume and produce resources. A plan in this framework specifies the dependencies between these consumed and produced resources. This explicit notion of dependencies between actions is required for almost any planning algorithm. Another advantage of this formalism is the fact that the result of an action is seen as a set of products (i.e., resources). This makes it easier to reason about exchanging results in a multi agent context.

Using the ARF, we described how plans can be combined using an addition operator $\oplus$, and how we can remove parts from a plan using a deletion operator $\ominus$. The combination of these operators $\otimes$ was shown to be complete, provided that the right plans are available to the agent. We also showed that this operator work with constant overhead, i.e., it is able to describe plan transformations rather efficiently.

An agent not only needs this kind of plan operators, it also needs to store plans that are successful or domain knowledge given by a human expert. Slightly extending the ARF, we have discussed a way of dealing with past plan experience using a plan library.

The last part of the framework is an extension to Kambhampati's [22] refinement planning. This extension makes it possible to describe replanning algorithms as well. The strength of the framework was illustrated by showing how a planning, a replanning, and a multi-agent planning technique fit into the framework. Such a uniform way to describe different types of planning algorithms is very useful for finding new variants and combining existing approaches. Furthermore, such a framework can help to compare different methods in a fair way.

To use the ARF for any task reduction approach, we need to extend the formalism with disjunctions of resources (super-resources). These disjunctions model the fact that a task sometimes can be fulfilled by producing very distinct resources. This allows a plan library like the one depicted in Figure 9. Here, the gap is not completely specified to allow both the P_1 and the P_2 plan to fit in the gap.

Task reduction (HTN) planning can also be integrated into the classical refinement planning approach [21, 28]. A refinement step can then also be a task reduction using one of the methods given in the plan library. In fact, this is quite similar to filling a gap in the plan. Using disjunctions of resources as described above, we hope to be able to show

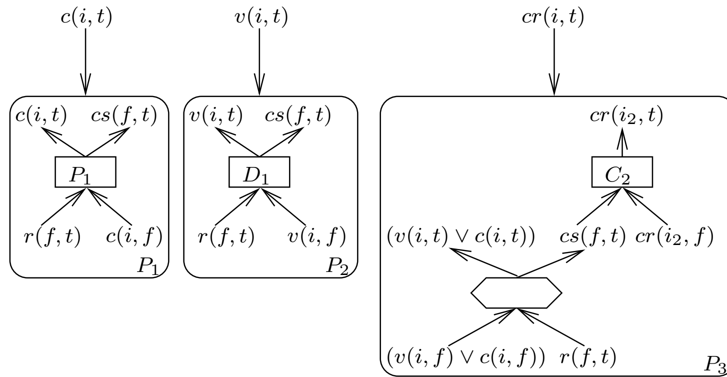

Figure 9. A more efficient way of storing the plan library of Figure 5.

that HTN planning can also be seen as a special case of our refinement template.

Future work focuses on three main issues: (i) extending the current framework to allow a more efficient plan library representation such as was just discussed, (ii) developing a continual algorithm, that uses the framework to fully integrate planning, replanning and execution, and (iii) developing a method to fill the plan library and maintain it. Further topics of interest are to show the relation with a disjunctive operator such as used in Graphplan [2].

It would also be interesting to see whether the planning template can be extended to allow the combination of several refinement strategies. This would, e.g., allow different planners to cooperate on solving the same problem.

References

1. Beek, P. V. and X. Chen: 1999, 'CPlan: A Constraint Programming Approach to Planning'. In: *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*. Menlo Park, CA.
2. Blum, A. L. and M. L. Furst: 1997, 'Fast Planning Through Planning Graph Analysis'. *Artificial Intelligence* **90**, 281–300.
3. Bond, A. H. and L. Gasser (eds.): 1988, *Readings in Distributed Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann Publishers.
4. Bonet, B. and H. Geffner: 2000, 'Planning as Heuristic Search'. *Artificial Intelligence*. Special issue on Heuristic Search.
5. Bonet, B. and H. Geffner: 2001, 'Heuristic Search Planner 2.0'. *AI Magazine* **22**(3), 77–80.
6. Davis, R. and R. Smith: 1983, 'Negotiation as a Metaphor for Distributed Problem Solving'. *Artificial Intelligence* **20**(1), 63–109.
7. de Weerdt, M. M., A. Bos, J. Tonino, and C. Witteveen: 2003, 'A Resource Logic for Multi-Agent Plan Merging'. *Annals of Mathematics and Artifi-*

cial Intelligence, special issue on Computational Logic in Multi-Agent Systems **37**(1–2), 93–130.
8. de Weerdt, M. M. and R. P. van der Krogt: 2002, ‘A Method to Integrate Planning and Coordination’. In: M. Brenner and M. desJardins (eds.): *Planning with and for Multiagent Systems*. Menlo Park, CA, pp. 83–88.
9. Decker, K. S. and V. R. Lesser: 1992, ‘Generalizing the Partial Global Planning Algorithm’. *International Journal of Intelligent and Cooperative Information Systems* **1**(2), 319–346.
10. DesJardins, M. E., E. H. Durfee, C. L. Ortiz, and M. J. Wolverton: 2000, ‘A Survey of Research in Distributed, Continual Planning’. *AI Magazine* **4**, 13–22.
11. Durfee, E. H. and V. R. Lesser: 1987a, ‘Planning Coordinated Actions in Dynamic Domains’. Technical Report COINS-TR-87-130, Department of Computer and Information Science, University of Massachusetts, Amherst, MA. Also published as [12].
12. Durfee, E. H. and V. R. Lesser: 1987b, ‘Planning Coordinated Actions in Dynamic Domains’. In: *Proceedings of the DARPA Knowledge-Based Planning Workshop*. pp. 18.1–18.10.
13. Durfee, E. H. and V. R. Lesser: 1987c, ‘Using Partial Global Plans to Coordinate Distributed Problem Solvers’. In: *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*. San Mateo, CA, pp. 875–883.
14. Ephrati, E. and J. S. Rosenschein: 1993, ‘Multi-Agent Planning as the Process of Merging Distributed Sub-Plans’. In: *Proceedings of the Twelfth International Workshop on Distributed Artificial Intelligence (DAI-93)*. pp. 115–129.
15. Georgeff, M. P.: 1983, ‘Communication and Interaction in Multi-Agent Planning’. In: *Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83)*. Menlo Park, CA, pp. 125–129. Also published in [3], pages 200–204.
16. Georgeff, M. P.: 1988, ‘Communication and Interaction in Multi-Agent Planning’. In: A. Bond and L. Gasser (eds.): *Readings in Distributed Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann Publishers, pp. 200–204. Also published as [15].
17. Gerevini, A. and I. Serina: 2000, ‘Fast Plan Adaptation through Planning Graphs: Local and Systematic Search Techniques’. In: *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS-00)*. pp. 112–121.
18. Hammond, K. J.: 1990, ‘Case-Based Planning: A Framework for Planning from Experience’. *Cognitive Science* **14**(3), 385–443.
19. Hanks, S. and D. Weld: 1995, ‘A Domain-independent Algorithm for Plan Adaptation’. *Journal of AI Research* **2**, 319–360.
20. Hoffmann, J. and B. Nebel: 2001, ‘The FF Planning System: Fast Plan Generation Through Heuristic Search’. *Journal of AI Research* **14**, 253–302.
21. Kambhampati, S.: 1995, ‘A Comparative Analysis of Partial Order Planning and Task Reduction Planning’. *SIGART Bulletin* **6**(1), 16–25.
22. Kambhampati, S.: 1997, ‘Refinement Planning as a Unifying Framework for Plan Synthesis’. *AI Magazine* **18**(2), 67–97.
23. Penberthy, J. and D. S. Weld: 1992, ‘UCPOP: A sound, complete, partial order planner for ADL’. In: *Proceedings of the Third International Conference on Knowledge Representation and Reasoning (KR&R-92)*. pp. 103–114.

24. Tonino, J., A. Bos, M. M. de Weerdt, and C. Witteveen: 2002, 'Plan Coordination by Revision in Collective Agent-Based Systems'. *Artificial Intelligence* **142**(2), 121–145.
25. van der Krogt, R. P., M. M. de Weerdt, L. R. Planken, and A. Biesheuvel: 2003, 'CABS planner'. http://www.pds.twi.tudelft.nl/~mathijs/.
26. von Martial, F.: 1991, 'Coordinating Plans of Autonomous Agents via Relationship Resolution and Communication'. Ph.D. thesis, Universität des Saarlandes.
27. Walsh, W. E. and M. P. Wellman: 1999, 'A Market Protocol for Decentralized Task Allocation and Scheduling with Hierarchical Dependencies'. In: *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*. pp. 325–332. An extended version of this paper is also available.
28. Wang, X. and S. Chien: 1997, 'Replanning Using Hierarchical Task Network and Operator-Based Planning'. Technical report, Jet Propulsion Laboratory Nasa.
29. Weld, D. S.: 1994, 'An Introduction to Least-Commitment Planning'. *AI Magazine* **15**(4), 27–61.
30. Wellman, M. P.: 1992, 'A General-Equilibrium Approach to Distributed Transportation Planning'. In: *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*. Menlo Park, CA, pp. 282–289.
31. Wellman, M. P., W. E. Walsh, P. R. Wurman, and J. K. MacKie-Mason: 2001, 'Auction Protocols for Decentralized Scheduling'. *Games and Economic Behavior* **35**(1–2), 271–303.
32. Williams, B. and P. Nayak: 1996, 'A Model-based Approach to Reactive Self-Configuring Systems'. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*. Menlo Park, CA, pp. 971–978.

Appendix

A. Proofs

PROPOSITION 10. *Let $P = (O, d, \theta)$ be a plan, I and R be disjoint sets of resources and G a set of goals. Then we have*

1. *If $I \models_P G$ then $I \models_P G'$ for every $G' \subseteq G$ and $I' \models_P G$ for every $I' \supseteq I$, that is $\models_P$ is anti-monotonic in its right argument and monotonic in its first argument.*

2. *For every ground substitution σ with $dom(\sigma) \supseteq res(P)$, $In(P)\sigma \models_P Out(P)\sigma$.*

3. *Let $S_1 + S_2$ denote the disjoint union of S_1 and S_2, that is $S_1 + S_2 = S_1 \cup S_2$ if $S_1 \cap S_2 = \emptyset$ and undefined otherwise. Then, if $In(P)\sigma + R$ is defined $In(P)\sigma + R \models_P Out(P)\sigma + R$.*

4. *If P is applicable to I then for every independent subset $S \subseteq prod(P)$ of general resources produced by P, P contains an initial subplan P' such that $I \models_{P'} S$.*

Proof. Each of the items of the proposition is proven in turn:

1. If $I \models_P G$, then there exists a substitution σ such that $In(P)\sigma \subseteq I$ and $G\sigma \subseteq (I - In(P)\sigma \cup Out(P)\sigma)$. But then for all $I' \supseteq I$, $In(P)\sigma \subseteq I \subseteq I'$ and thus $I' \models_P G$. Similarly, for $G' \subseteq G$, $G'\sigma \subseteq G\sigma \subseteq (I - In(P)\sigma \cup Out(P)\sigma)$, then $I \models_P G'$.

2. Note that in general, for a set of ground resources R and a substitution σ such that $In(P)\sigma \subseteq R$ we have $R \models_P R - In(P)\sigma \cup Out(P)\sigma$. Taking $R = In(P)\sigma$ gives the required result.

3. Note that P is applicable to $I - R$, since there clearly exists a substitution θ such that $In(P)\theta \subseteq (I - R) = In(P)\sigma$. Upon execution of the plan, it produces the set of resources $I - In(P)\sigma \cup Out(P)\sigma = (In(P)\sigma + R) - In(P)\sigma \cup Out(P)\sigma = R \cup Out(P)\sigma$ and thus, $In(P)\sigma + R \models_P Out(P)\sigma + R$.

4. Let S be an independent subset of $prod(P)$ for a plan $P = (O, d, \theta)$ then the set of producing actions is $producers = \{o \mid \exists r \in S \cdot r \in out(o)\}$. Let $O' = \bigcup_{o \in producers} \{o' | o' <_d o\}$, then $P' = (O', d', \theta')$ is an initial subplan to P, if d' is the restriction of d to $cons(O')$ and θ is an mgu for all dependent pairs $r, d'(r) \in d'$, since $O' \subseteq O$ and O' is downward closed with respect to $<_d$. For all resource $r \in S$, $d'^{-1}(r) = \bot$, since S is an independent subset of $prod(P)$. Hence, $S \subseteq Out(P')$ and thus $I \models_P S$. □