
Annals of Mathematics and Artificial Intelligence **37:** 93–130, 2003.

A resource logic for multi-agent plan merging

Mathijs de Weerdt *, André Bos **, Hans Tonino and Cees Witteveen

Delft University of Technology, Faculty of Information Technology and Systems, P.O. Box 356, 2600 AJ Delft, The Netherlands
E-mail: {m.m.deweerdt, a.bos, j.f.m.tonino, c.witteveen}@its.tudelft.nl

In a multi-agent system, agents are carrying out certain tasks by executing plans. Consequently, the problem of finding a plan, given a certain goal, has been given a lot of attention in the literature. Instead of concentrating on this problem, the focus of this paper is on cooperation between agents which already have constructed plans for their goals. By cooperating, agents might reduce the number of actions they have to perform in order to fulfill their goals. The key idea is that in carrying out a plan an agent possibly produces *side products* that can be used as resources by other agents. As a result, an other agent can discard some of its planned actions. This process of exchanging products, called *plan merging*, results in distributed plans in which agents become dependent on each other, but are able to attain their goals more efficiently. In order to model this kind of cooperation, a new formalism is developed in which side products are modeled explicitly. The formalism is a resource logic based on the notions of *resource*, *skill*, *goal*, and *service*. Starting with some resources, an agent can perform a number of skills in order to produce other resources which suffice to achieve some given goals. Here, a skill is an elementary production process taking as inputs resources satisfying certain constraints. A service is a serial or parallel composition of skills acting as a program. An operational semantics is developed for these services as programs. Using this formalism, an algorithm for plan merging is developed, which is anytime and runs in polynomial time. Furthermore, a variant of this algorithm is proposed that handles the exchange of resources in a more flexible way. The ideas in the paper will be illustrated by an example from public transportation.

Keywords: multi-agent planning, cooperation, resource logic

AMS subject classification: 68T27, 68W15

1. Introduction

1.1. Plan-based controlled agents

One of the design goals of an agent system is to make the programming of the system easier than in conventional systems. Instead of supplying the system with a list of low-level (machine) instructions, an agent is normally told what to do in terms of a

* Supported by the Seamless Multimodal Mobility (SMM) research program.
** Supported by the Freight Transport Automation and Multimodality (FTAM) research program. Both programs are carried out within the TRAIL research school for Transport, Infrastructure and Logistics.

high-level goal it has to fulfill. In this way, the user is relieved from the task to *program* the agent; the agent itself has to find out a way to satisfy the mission goal.

Generally speaking, there are two design principles to achieve this kind of behavior: One way is by designing a so-called *reactive* agent. A reactive agent has a set of rules that given a certain state of the world and the goal to achieve immediately tell the agent what kind of action to perform. In general, the action will change the current state of the world into a new one in which either the goal is satisfied or in which goal satisfaction is easier than in the original situation. Another way to realize goal-finding behavior is to design so-called *deliberative* agents. Instead of immediately reacting to a new situation, a deliberative agent first tries to find a *sequence* of actions, or a *plan*, of which the agent can show that it will transform the current situation into one in which the mission goal is satisfied. To facilitate this planning process, each of the agents is equipped with a set of *skills* or *actions* that allow them to change the state of the world. Each of these skills can only be applied if the world satisfies a certain condition, that is encoded by a set of required *resources* of a skill that must be present in the world before a skill may fire. Application of a skill results in producing new resources that may cause other skills to become firable. In this way, a sequence of skills can be constructed such that from an initial situation a desired state of the world can be obtained.

In general, the computational burden of a reactive agent is less than that of a deliberative agent, as the solution strategy to satisfy a goal has been designed into the reactive rule set, where a deliberative agent has to compute this strategy for each new situation. The advantage of a deliberative architecture, on the other hand, is that such an agent is able to satisfy a goal also in situations not anticipated during the design of the agent, and that an explicit plan representation can be used for further processing. An explicit plan representation may guide (i) replanning in case contingencies arise during the execution of the plan, and (ii) negotiation if two or more agents want to cooperate, as we describe in this paper.

1.2. Multi-agent systems

Another design goal of an agent system is that it is able to *cooperate* with other agents. Usually, this requirement stems from the observation that a mission goal cannot be satisfied by a single agent, but requires a joint effort of several agents, coordinating their plans and sharing their resources and goals.

Although this observation points out a necessary reason to cooperate, another, almost equally important reason should not be overlooked: even if agents are able to solve problems on their own, they might prefer to cooperate as sharing their resources and coordinating their activities may save costs or allows them to accept new orders to be executed in parallel.

This observation refers to applications such as *supply chain management* and *travel assistance* in which individual planning and cooperation phases can be clearly distinguished. For example, a (chain) manager first assigns each agent to a part of a complex task via a *task allocation process* (see for example the work by Shehory and Kraus [25]),

or each agent personally receives some requests by its customers. So, each of the agents has been assigned (sub)goals for which they construct a plan. Finally, the chain manager, or the agents themselves analyze their plans and may try to cooperate in order to satisfy (additional) constraints on costs or performance. In such a case, cooperation can be accomplished by plan revision: to reduce the total cost of his plan an agent tries to revise part of his plan by exchanging resources and goals with other agents.

1.3. Plan merging

Several potential problems may arise during merging of two plans. Plans may *conflict* because they require the same resource for correct execution, or the prescribed order of actions may lead to dead-lock, e.g., both have to wait for the other to finish a certain action. In order to realistically model real-life situations, we must track different attributes of resources – such as fuel, time, space, and capacity – in a numerical fashion. Furthermore, as one agent may take over responsibility of another one to satisfy a (sub)goal, there must be some form of *cost and profit model*.

In the literature, some attention already has been given to the development and analysis of efficient (approximately or optimal) plan merging methods as discussed in section 4. We did not encounter, however, computationally feasible approaches for solving the plan merging problem, dealing with the abovementioned, often numerical, constraints.

1.4. Our approach

In this paper, we take a special approach to multi-agent cooperative planning. As in the previously described situation of supply chain management, we assume that each of the agents has a plan available to satisfy their (sub)goals. In order to lower their costs, two or more agents investigate whether they can cooperate. Two agents can cooperate if one agent produces a resource – one that is not directly needed for goal satisfaction, but as a byproduct – that is needed by the other agent. Both agents may agree to cooperate by exchanging resources. That is to say, the agent producing the resource allows the other agent to use it. As a result, the resource that was originally produced as a byproduct becomes a subgoal for the agent. This idea has already been introduced [22,27], but now we present a thorough logic framework and we show how this process of exchanging resources can be controlled by an auction. We assume one of the agents acts as an auctioneer and collects all requests of all agents. For each request an auction is held, in which all agents, other than the requesting agent, may offer to deliver the requested resource(s). The auctioneer ensures that all constraints are satisfied and that the exchange actually takes place, resulting in a cost reduction. Note that we assume that agents are always willing to cooperate.

The structure of this paper is as follows. First, we will introduce a new framework for representing plans. The main features of this framework are (i) the ability to treat plans as first-class citizens, (ii) the distinction between resources, goals and side-products of resource production processes, and (iii) the possibility to model attributes of

resources and constraints over attributes in a numerical fashion. We give an operational semantics in terms of a transition system for plans expressed in our resource-based plan formalism. In this framework, we also show that a set of cooperating agents can optimize their plans if they exchange resources. This optimization process is called *plan merging*.

Next, we further investigate this plan merging process by presenting an algorithm for it. We give two variants for the plan merging algorithm: one running in polynomial time, but in which only resources can be exchanged that have similar attributes; and one that is more liberate in that the attribute values need not necessarily be similar, but for which a polynomial running time can not be guaranteed.

Finally, we end with a comparison with related work, a description of future work, and conclusions.

2. The resource-skill framework

The formal framework we introduce is built upon three primitive notions: (i) *resources*, which are the objects manipulated by an agent, (ii) elementary production processes, called *skills*, used by an agent to consume resources and to produce other resources, and (iii) the *goals* an agent has to realize.

On top of these notions, we introduce *resource schemes* to represent collections of resources, and *services* as sequential and independent compositions of skills to model more complex (generic) production processes. We consider services to be the resource production *programs* an agent might use in order to achieve some goals, given some set of initial resources. Moreover, an agent has the possibility to *modify* these programs if he is cooperating with other agents to reduce the costs of his service or the costs of the services of other agents.

The key ideas we present in this section are the following:

1. We introduce services, which are compositions of skills, as programs an agent can use to realize its goals, given some set of initial resources. A transition system is specified to formalize the operational semantics of these services as programs.
2. It is assumed that skills cannot be used for free. Therefore, an agent will try to reduce the costs of his service by removing one or more skills, while maintaining goal realizability by executing a *service reduction* process. We present some reduction techniques that maintain goal realizability and hardly affect the structure of the remaining part of the service. As an outcome, we show how goal realizability can be maintained at the cost of increasing serial dependencies between skills.
3. From an abstract point of view, the services of several agents together can be viewed as one aggregated service consisting of the independent composition of the individual agent services. Initially, each of the agents uses its own service to achieve its own goals, independently from the other agents. An agent now has the opportunity to reduce its production costs by removing skills, while maintaining goal realizability, and using some *free resources* of other agents. As a result, such an agent will

become dependent on other agents. In our framework, this means that as a result of such reductions, the original independent composition of services will go through a number of *sequential refinements* of the aggregated service. We then show that the intra-agent service reduction methods already presented also can be used to model inter-agent cooperation by resource-sharing.

4. Two distinct forms of inter-agent resource sharing are proposed. In the first one, instead of using a service as a generic production process, the specialization (instantiation) of the service with respect to the resources and goals given is used. Such "frozen" forms of services are called *ground plans*. They allow for a restricted, but computationally attractive, form of resource sharing between agents. In the second form, resource sharing is more flexible: instead of using only resources that fit into the current plan, other resources might be used as well, provided that they can be used to realize the current goals of the agent.

The results on service and plan reduction will be used in the next section to develop plan and service merging algorithms for agent cooperation. While in the formal framework, we do not distinguish between *intra-agent dependencies*, i.e., dependencies between skills in the service of a single agent, and *inter-agent dependencies* that relate services of different agents, in these algorithms we will make this distinction.

To discuss our framework into detail, first we introduce a many-sorted (resource) language to describe primitive notions like resources, resource schemes, goals and goal schemes. Then we describe skills, services and operations on services. Much of our concepts and notational conventions have been taken from logic programming and are easy to grasp.

2.1. Language

We use a many-sorted first-order language $\mathcal{L}$ containing:

- Sorts: $Sorts = \{s_1, s_2, \ldots, s_n, \ldots\}$.
- Variables: $Var_s = \{x_s^1, \ldots, x_s^n, \ldots\}$ for each $s \in Sorts$; we use $Var = \bigcup_{s \in Sorts} Var_s$ to denote the set of all variables.
- Predicate symbols: $Pred = \{p, q, \ldots\}$ with rank $r : Pred \to Sorts^*$.
- Function symbols: $Fun_s = \{f, g, \ldots\}$ and rank $r : Fun_s \to Sorts^+ \times \{s\}$ for each $s \in Sorts$.
- Constants: $Con_s = \{c_s^1, \ldots, c_s^n, \ldots\}$ for each $s \in Sorts$.

Equality is dealt with as usual; the sets $Term = \bigcup_{s \in Sorts} Term_s$ of many-sorted *terms* and *Form* of formulae are defined in the usual inductive way, where we distinguish *ground terms* and *ground formulae* from general terms and formulae that may contain variables. If x is a variable of sort s, we use $x : s$ to indicate its sort.

2.2. Resources and resource schemes

A *resource fact*, or simply *resource*, in our language $\mathcal{L}$, is an $\mathcal{L}$-*atomic* ground formula $p(t_1 : s_1, t_2 : s_2, \ldots, t_n : s_n)$. The terms s_j are also called the *attributes* of resource p with attribute *values* t_j. As an example, take a resource fact

$$taxi\big(1234 : id, adam : loc, [90, 100] : time\big)$$

referring to the taxi with identification 1234 at location Amsterdam at time interval [90, 100].[1]

Often, we do not need to point out a specific individual resource: it is sufficient if the resource we want has certain properties that might be satisfied by more than one specific resource. That is, we are looking for an arbitrary resource belonging to some *class* of resource facts instead of a specific unique resource. Using the language $\mathcal{L}$, such a class of resources or *resource scheme* is denoted by a (not necessarily ground) $\mathcal{L}$-atomic formula.

Resource schemes are useful to denote arbitrary resources that are ground instances of a resource scheme. For example,

$$taxi(x : id, adam : loc, I : time)$$

is a resource scheme denoting some taxi at Amsterdam, where we do not care about the time interval I it is available.

Given some resource r and a resource scheme rs we would like to know whether r indeed is a ground instance of rs. Hence, we need *substitutions*. As usual, a substitution θ is a finite set of substitution pairs of the form $x_i = t_i$, with $x_i \in Var$ and $t_i \in Term$, satisfying the standard conditions. That is: (i) variables and terms match, i.e., if $x_i \in Var_s$, then $t_i \in Term_s$; (ii) $i \neq j$ implies $x_i \not\equiv x_j$; (iii) x_i does not occur as a variable in t_i. The domain $dom(\theta)$ of a substitution θ is the set of variables x for which there exists a substitution pair $x = t$ in θ. If θ and σ are substitutions, $\theta - \sigma$ denotes the set of substitution pairs $x = t$ from θ such that $x \notin dom(\sigma)$. If $dom(\theta) \cap dom(\sigma) = \emptyset$, $\theta \cup \sigma$ is the substitution consisting of all substitution pairs from θ and σ. A substitution θ is said to be *ground*, if for every pair of the form $x_i = t_i$ contained in θ, t_i does not contain any occurrence of a variable in *Var*. The empty substitution is denoted by ε.

We say that a resource r is an *instance* of the resource scheme rs if there exists a ground substitution θ such that $rs\theta = r$. If we have a set RS of resource schemes, we would also like to know whether a given set R of resources *satisfies* RS, i.e., whether R is a typical set of resources denoted by RS. Here we have to be careful: it is not enough to have a substitution θ such that $RS\theta = R$ since for each $rs \in RS$ we have to find a *unique* resource $r \in R$ as an instance of rs. So an individual resource cannot be used to satisfy more than one resource scheme. Therefore, we do not allow a substitution θ such that for two different schemes $rs, rs' \in RS$ we have $rs\theta = rs'\theta$. Hence, we need to introduce *resource-identity preserving* substitutions:

[1] In this paper we use *adam* to denote the city Amsterdam and *rdam* to denote the city Rotterdam.

Definition 1. Let RS be a set of resource schemes. A substitution θ is said to be *resource-identity preserving* with respect to RS, if for every $rs_1, rs_2 \in RS$, $rs_1 \neq rs_2$ implies $rs_1\theta \neq rs_2\theta$.

Hence, an identity-preserving ground substitution with respect to RS gives rise to an injection $RS \rightarrow R$. It is very easy to check, given a set of resource schemes RS and a substitution θ, whether θ is identity-preserving: note that for every substitution θ we have $|RS\theta| \leqslant |RS|$. By definition 1, we have $|RS\theta| \geqslant |RS|$. Hence, if θ is resource-identity preserving, then $|RS| = |RS\theta|$. Analogously, if θ is a substitution such that $|RS| = |RS\theta|$, it immediately follows that $rs_1 \neq rs_2$ implies $rs_1\theta \neq rs_2\theta$. Summarizing this line of reasoning, we have:

Proposition 2. A substitution θ is resource-identity preserving with respect to a given set of resource schemes RS iff $|RS| = |RS\theta|$.

Definition 3. A set of resources R *satisfies* a set RS of resource schemes, denoted by $R \models RS$, if there exists a resource-identity preserving ground substitution θ with respect to RS, such that $RS\theta \subseteq R$. To indicate the substitution θ explicitly, we write $R \models_\theta RS$.

Sometimes, we need to find a collection of resources that not individually, but together satisfy some conditions. Corresponding classes of resources can be obtained by introducing *extended resource schemes*.

Definition 4. An *extended resource scheme* is a tuple (RS, C), where (i) RS is a *set* of resource schemes, and (ii) C is a set of constraints for terms with variables occurring in resources in RS.[2]

Remark. Note that every set R of resources is a special set of resource schemes. An arbitrary set RS of resource schemes is a special case of an extended resource scheme, where the set C of constraints equals {**true**}.

Of course, we would also like to know whether a given set R of resources *satisfies* a set of extended resource schemes RS.

Definition 5. A set of resources R *satisfies* an extended resource scheme (RS, C), abbreviated as $R \models (RS, C)$, if there exists some resource-identity preserving ground substitution θ with respect to RS such that

1. $RS\theta \subseteq R$,
2. $\models C\theta$, i.e., every constraint reduces to **true** under θ.

[2] We do not feel the need to specify exactly the type of constraints we will use; it suffices to use standard relations.

If we want to specify the substitution θ used to satisfy the extended resource scheme, we write $R \models_\theta (RS, C)$.

To give an example, consider a passenger $p(x_1 : loc, I_1 : time)$ at location x_1 at time interval I_1, waiting for a (taxi)-ride $ride(x_2 : loc, y : loc, c : cap, I_2 : time)$ from location x_2 to y, having c free seats and occurring in time interval I_2. To help the passenger it is required that $x_1 = x_2$, $c > 0$, and $I_1 \cap I_2 \neq \emptyset$. This is expressed by the following extended resource scheme:

$$\big(\{p(x_1 : loc, I_1 : time), ride(x_2 : loc, y : loc, c : cap, I_2 : time)\}, \\ \{x_1 = x_2, I_1 \cap I_2 \neq \emptyset, c > 0\}\big).$$

Every set R of resources containing at least one ride and one passenger resource meeting these constraints will satisfy this extended resource scheme. In this example we used x_1 and x_2 to denote the same location, because of the constraint $x_1 = x_2$. Later in this paper we will use only one variable x and omit this constraint.

2.3. Goals

Some resources an agent already has, others he needs to obtain. In general, such an agent does not care about obtaining a specific (unique) resource as a goal g, but only a resource having some specific properties. So we will conceive an individual goal g as a resource *scheme*. Even more generally, we may want to obtain a set of goals meeting some constraints. Therefore, we define goals and *goal schemes* analogously to resources and extended resource schemes as follows:

Definition 6. A *goal* g is a resource scheme. A *goal scheme* Gs is an extended resource scheme $Gs = (G, C)$, where G is a set of goals.

As goal schemes are extended resource schemes, we also use $R \models Gs$ to express that the resource set R satisfies the goal scheme Gs.

2.4. Skills

Suppose we are given some set of resources R and we want to obtain some set of resources R' satisfying a given goal scheme Gs. We clearly need a way to transform the set R into this set R'. To this end, we introduce *skills* as (elementary) resource consuming and resource production processes.

Definition 7. A *skill* is a rule of the form $s : RS' \leftarrow (RS, C)$. Here, s is the name of the skill, RS' is a set of resource schemes containing only variables that occur in the extended resource scheme (RS, C), i.e., $var(RS') \subseteq var(RS)$.

Intuitively, the meaning of a skill s is as follows: whenever there is a set of resources R and a ground substitution θ such that $R \models_\theta (RS, C)$, the set R is consumed

by s and a new set of resources $R' = RS'\theta$ will be produced.[3] This set R' is uniquely determined by θ and RS.

We identify the set of constraints C occurring in a skill s by C_s. Furthermore, $var(s) = var(RS)$ denotes the set of variables occurring in s, $in(s) = RS$ denotes the set of input resource schemes, and $out(s) = RS'$ is the set of output resource schemes.

Definition 8. Skill application to resources. Let $s : RS' \leftarrow (RS, C)$ be a skill and let R, R' be sets of resources. We say that R' *can be produced from* R *using* s, abbreviated by $R \vdash_s R'$, if there is a resource-identity preserving ground substitution θ with respect to $RS \cup RS'$ such that[4]

1. $R \models_\theta RS$;
2. $\models_\theta C$, i.e., every constraint in C reduces to **true** under θ; and
3. $R' = (R - RS\theta) \cup RS'\theta$, i.e., the resources $RS\theta$ which are consumed by s are removed from R, while the resources $RS'\theta$ produced by s are added to the remaining set of resources.

Example 9. To give two concrete examples, take the following skills:

$$\begin{aligned}
\textbf{drive_rdam}\colon\ & taxi\big(rdam : loc, \big[t_1 + t(x, rdam), t_2\big] : time\big),\\
& ride\big(x : loc, rdam : loc, 2 : cap, [t_1, t_2] : time\big)\\
& \leftarrow \big(taxi\big(x : loc, [t_1, t_2] : time\big),\ \emptyset\big);\\
\textbf{travel}\colon\ & p\big(y : loc, \big[t_3 + t(x, y), t_2\big] : time\big)\\
& \leftarrow \big(p\big(x : loc, [t_1, t_2] : time\big),\\
& ride\big(x : loc, y : loc, c : cap, [t_3, t_3] : time\big),\\
& \big\{t_1 \leqslant t_3, t_3 + t(x, y) \leqslant t_2, c \geqslant 1\big\}\big).
\end{aligned}$$

To explain the last skill, **travel**: for a passenger to travel from location x to location y, two resources are needed. First of all, the passenger should be at location x at a certain time t_1 and secondly, a ride (produced by a taxi) should be available at a time $t_3 \geqslant t_1$ and with capacity at least 1. The time the passenger arrives at location y is calculated using a pre-defined time–distance matrix $t(x, y)$. The constraint $t_3 + t(x, y) \leqslant t_2$ ensures that the passenger arrives before his deadline t_2.

A ride resource can be produced by the first skill: take the resource set $R = \{taxi(adam, [10, \infty])\}$, suppose that $t(adam, rdam) = 40$ and let $\theta = \{x = adam, t_1 = 10, t_2 = \infty\}$. Then $\{taxi(adam, [10, \infty])\} \vdash_{\textbf{drive\_rdam}} \{taxi(rdam, [50, \infty]), ride(adam, rdam, 2, [10, 10])\}$.

[3] That is, every skill is range restricted, meaning that from ground resources only ground resources can be produced.

[4] Note that by proposition 2 this immediately implies that θ is identity-preserving with respect to RS as well as with respect to RS'.

Skills represent generic production processes that may be applied to different sets of resources. Since the same skill may be used more than once, we should be able to distinguish several uses of the same skill; therefore we introduce *skill variants*.

Definition 10. Two skills $s_1 : RS'_1 \leftarrow (RS_1, C_1)$ and $s_2 : RS'_2 \leftarrow (RS_2, C_2)$ are said to be *variants* if there exist substitutions θ and ψ such that

1. $RS'_1\theta = RS'_2, RS_1\theta = RS_2, C_1\theta = C_2$;
2. $RS'_2\psi = RS'_1, RS_2\psi = RS_1, C_2\psi = C_1$; and
3. $var(s_1) \cap var(s_2) = \emptyset$.

It is not difficult to see that whenever two skills s_1 and s_2 are variants, the substitutions θ and ψ involved have to be resource-identity preserving with respect to $RS_1 \cup RS'_1$ and $RS_2 \cup RS'_2$, respectively.[5]

2.5. Resources, skills and uniqueness postulates

As remarked before, resources are considered to be *unique* objects in our framework. Since we are dealing with skills that consume and produce resources, it is not sufficient to deal with uniqueness in a (current) set of resources. Instead, we want uniqueness to be preserved also in these generic resource consumption and production processes. We capture these uniqueness requirements in the form of the following uniqueness postulates:

Postulate UN1. Every resource r is a unique object, uniquely identified by a resource fact.

Postulate UN2. Whenever a resource r has been consumed by a skill application it can never be reproduced by any sequence of skill applications.

Postulate UN3. A resource r produced by a a skill variant s is uniquely determined by s and the unique input resources of s.

Postulate UN4. A skill is a generic production process. A goal is a generic resource object. Therefore, a skill can never require a unique resource to be given as one of its input resources and a goal can always be satisfied by more than one possible resource.

These uniqueness postulates can be satisfied if every resource has a special identification attribute *id*. For each resource r, the *id* attribute has a unique value. We also consider a countable set of skill variants; each such a skill variant has a unique identification number. Whenever a resource r appears as output of a skill, the value of the *id*-attribute of r is determined by the concatenation of the values of the input identification labels and the skill-identification number. Hence, the value of the identification

[5] It suffices to observe that $RS'_1\theta = RS'_2$ implies $|RS'_1| \geqslant |RS'_2|$ and $RS'_2\psi = RS'_1$ implies $|RS'_2| \geqslant |RS'_1|$. Hence, $|RS'_1| = |RS'_2| = |RS'_1\theta|$, implying that θ is resource-identity preserving with respect to RS'_1. Analogously for RS_1, RS'_2 and RS_2.

attribute of a resource r will reflect its production history and – given unique initial values of input resources – we are guaranteed to preserve uniqueness of every resource ever produced or consumed. Note that the last postulate requires that we can never use specific values of the *id*-attribute as one of the constraints on the variables of input resources. In fact, it is sufficient to assume that the *id*-attribute is never mentioned in the set of constraints of a skill s or goal scheme.

In the sequel, we assume that every resource has such an identification attribute *id* and we define two resources to be *similar* if they are identical with the possible exception for the value of their *id*-attributes.

Definition 11. Resources r, r' having the same predicate name and attributes and attribute values, with the possible exception of values of the *id* attribute, are said to be *similar* resources, abbreviated as $r \approx r'$. Analogously, if there exist resource sets R, R' and a bijection $f : R \to R'$ such that for every $r \in R, r \approx f(r)$, we say that R is similar to R', abbreviated as $R \approx R'$.

Analogously, for substitutions σ and σ', we write $\sigma \approx \sigma'$ if they are identical with the possible exception of the values assigned to the *id*-attribute.

Note that according to postulate UN3, similar resources are indistinguishable for skills or goals.

2.6. Services

In general, most of the goals are not realizable by applying just a single skill on the set of available resources. Instead they will require the execution of several skills sequentially and/or concurrently to reach some goals.

Of course, we might use definition 8 and the transitive closure of the production relation $\vdash_S = \bigcup_{s \in S} \vdash_s$ to describe the set of resources sets R' derivable from a given resource set R. However, we will not do so. The reason is that we are not primarily interested in *finding* the skills to produce a set of resources from an initial set of resources – this surely is a difficult planning problem, – but in *using* a given (structured) set of skills to produce a set of resources.

Instead of determining a complete plan from scratch, we assume the existence of such a *skill program*, also called a *service*, and we will use these services to test whether they can be used to obtain some given goal, and, if so, whether they can be simplified to reach these goals.

We will assume that such a service consists of a composition of (unique) skills or variants of skills. The set of services Ss over the set S of skill variants is inductively defined as follows.

Definition 12.

1. E is a service over $\emptyset$, called the *empty service*.
2. Every skill s is a service over the set $\{s\}$.

3. If π_1 and π_2 are services over S_1 and S_2 and $S_1 \cap S_2 = \emptyset$, then $(\pi_1; \pi_2)$ and $(\pi_1 \parallel \pi_2)$ are services over $S_1 \cup S_2$.

Here ; denotes sequential composition and $\parallel$ denotes the independent composition operator. The service $\pi_1; \pi_2$ expresses that starting the execution of skills in π_2 is dependent upon completion of all skills in π_1, while $\pi_1 \parallel \pi_2$ denotes two services that have to be executed completely independent from each other, that is, no resource produced or consumed by π_1 can be consumed by a skill in π_2 and vice-versa and there are no execution constraints between skills in the two services.

If Ss is a service over a set S of skills, $Skills(Ss) = S$ denotes the set of skills used in Ss. Without loss of generality, we can always assume that whenever $s, s' \in Skill(Ss)$ are two different skills then $var(s) \cap var(s') = \emptyset$.

2.7. A transition system specification for services

A service Ss is a kind of program an agent can execute on a given set R of resources to obtain some set of goals. During execution of skills in a service, the agent creates new sets of resources that in turn might be changed by executing the remaining part of the service. So we identify a *state* of an agent A by a tuple (Ss, R) where Ss is the (remaining part of a) service to be applied on the current set of resources R. After successful execution of the complete service Ss, its final state will be (E, R') for some set of final resources R'.

To specify a transition relation between the states of an agent and especially between initial states (Ss, R) and final states (E, R'), we use *transition systems* (cf. [23]) to define the operational semantics of skill applications, sequential composition and independent composition. Each of the following definitions specifies a transition rule defining the results a single skill application, and the composition operators have on the current state of the agent. Together they inductively define the transition relation $\rightarrow$ between the states of an agent.

During the execution of skills, the agent creates substitutions θ used to bind skill variables to attribute values in resources. For bookkeeping purposes, we will often use such substitutions as subscripts in the transition relation $\rightarrow$.

First of all, we need reflexivity: by using an empty substitution, a transition can be made from a state to itself.

Definition 13. Reflexivity:

$$(\pi, R) \rightarrow_{\varepsilon} (\pi, R).$$

Given a skill s and a set of resources R, the skill can be executed on R whenever we can find an identity preserving ground substitution θ that (i) satisfies the constraints C_s and (ii) applied to $in(s)$ selects a subset of the resources (cf. definition 8).

Definition 14. Transition rule for skill application:

$$\frac{in(s)\theta \subseteq R,\ \models_\theta C_s,\ R' = (R - in(s)\theta) \cup out(s)\theta}{(s, R) \rightarrow_\theta (E, R')},$$

The sequential composition $\pi_1; \pi_2$ of two services π_1 and π_2 using resources in R requires the execution of all skills in π_1 on R (resulting in an intermediate set of resources produced), before π_2 can be applied. We distinguish two cases: one to execute (part of) π_1 (definition 15) and one to execute π_1 and (a part of) π_2 (definition 16).

Definition 15. Transition rule for sequential composition – reduction:

$$\frac{(\pi_1, R) \rightarrow_\theta (\pi_1', R') \rightarrow_{\theta'} (E, R'')}{(\pi_1; \pi_2,\ R) \rightarrow_\theta (\pi_1'; \pi_2,\ R')}.$$

Definition 16. Transition rule for sequential composition – elimination:

$$\frac{(\pi_1, R) \rightarrow_{\theta_1} (E, R'')\ ,\ (\pi_2, R'') \rightarrow_{\theta_2} (\pi_2', R') \rightarrow_{\theta_2'} (E, R''')}{(\pi_1; \pi_2,\ R) \rightarrow_{\theta_1 \cup \theta_2} (\pi_2',\ R')}.$$

Note that these two rules define any state transition for a sequential composition of services, so we do not need the transitive closure of $\rightarrow$.

The specification of the independent composition $\pi_1 \parallel \pi_2$ using resources in R is somewhat more involved. The reason is that this operator is used to express that two sub-services (i) have to be completely independent with respect to their resource needs and (ii) run concurrently. These concurrency aspects, however, have to be specified by the independent and successful execution of the subservices separately.

Definition 17. Transition rule for independent composition – reduction:

$$\frac{\begin{array}{l}(\pi_1, R_1) \rightarrow_{\theta_1} (\pi_1', R_1') \rightarrow_{\theta_1'} (E, R_1''),\\ (\pi_2, R_2) \rightarrow_{\theta_2} (\pi_2', R_2') \rightarrow_{\theta_2'} (E, R_2''),\ R_1 \cap R_2 = \emptyset\end{array}}{(\pi_1 \parallel \pi_2,\ R_1 \cup R_2) \rightarrow_{\theta_1 \cup \theta_2} (\pi_1' \parallel \pi_2',\ R_1' \cup R_2')}.$$

We note that if $R_1 \cap R_2 = \emptyset$, as a consequence of the uniqueness postulates, we also have $R_1' \cap R_2' = \emptyset$.

Finally, we need two transition rules to deal with the borderline cases $E \parallel \pi$ and $\pi \parallel E$ to identify them with π.

Definition 18. Transition rules for $E \parallel \pi$ and $\pi \parallel E$ – elimination:

$$\frac{(\pi,\ R) \rightarrow_\theta (\pi',\ R')}{(E \parallel \pi,\ R) \rightarrow_\theta (\pi',\ R')}, \qquad \frac{(\pi,\ R) \rightarrow_\theta (\pi',\ R')}{(\pi \parallel E,\ R) \rightarrow_\theta (\pi',\ R')}.$$

Example 19. Consider the drive and travel skills discussed before. Let *Ss* be the set {**drive_rdam**, **travel**} and let R be the following set of resources {*taxi*(*adam*, [100, ∞]),

$p(adam, [80, \infty])\}$. Using the transition specifications defined in definition 14 we derive:

$$\begin{aligned}&\big(\textbf{drive\_rdam},\ \{taxi(adam, [100, \infty]),\ p(adam, [80, \infty])\}\big)\\&\quad\rightarrow_{\theta_1} \big(E,\ \{taxi(rdam, [140, \infty]),\ ride(adam, rdam, 2, [100, 100]),\\&\qquad p(adam, [80, \infty])\}\big) \quad \text{and}\\&\big(\textbf{travel},\ \{taxi(rdam, [140, \infty]),\ ride(adam, rdam, 2, [100, 100]),\\&\qquad p(adam, [80, \infty])\}\big) \rightarrow_{\theta_2} \big(E,\ \{taxi(rdam, [140, \infty]),\\&\qquad ride(adam, rdam, 1, [100, 100]),\ p(rdam, [140, \infty])\}\big)\end{aligned}$$

and thus by definition 16 and reflexivity:

$$\begin{aligned}(Ss, R) \rightarrow_{\theta_1\cup\theta_2} \big(E,\ \{&taxi(rdam, [140, \infty]),\ ride(adam, rdam, 1, [100, 100]),\\&p(rdam, [140, \infty])\}\big),\end{aligned}$$

where $\theta_1 = \{x = adam, t_1 = 100, t_2 = \infty\}$ and $\theta_2 = \{x = adam, y = rdam, t_1 = 80, t_2 = \infty, t_3 = 100, c = 2\}$.

Using the transition rules defined, we now use $\rightarrow$ as the reflexive closure of the relation specified by these rules.

As an immediate consequence of the preceding definitions and the uniqueness postulates, we mention the following properties of the relation $\rightarrow$.

Proposition 20. If $(\pi, R) \rightarrow_\theta (\pi', R')$ then

(i) (resource monotony) for every set of resources R_1, $(\pi, R \cup R_1) \rightarrow_\theta (\pi', R' \cup R_1)$;

(ii) (irrelevance of unused resources) $(\pi, R - R'') \rightarrow_\theta (\pi', R' - R'')$ for every $R'' \subset R \cap R'$;

(iii) (resource production genericity) if $R_1 \approx R$ then there exists some $R_1' \approx R'$ and a substitution $\theta' \approx \theta$ such that $(\pi, R_1) \rightarrow_{\theta'} (\pi', R_1')$.

Proof. By easy induction on the structure of the service π. □

2.8. Goal satisfaction, service reduction and sequential refinement

Given these transition rules, we now are able to define whether a goal scheme Gs can be satisfied by a service Ss.

Definition 21. Let Ss be a service over the set S, R a set of resources and $Gs = (G, C)$ a goal scheme. We say that (Ss, R) *satisfies* Gs, abbreviated as $(Ss, R) \models Gs$, if there exists an R' such that $(Ss, R) \rightarrow (E, R')$ and $R' \models Gs$. If $(Ss, R) \rightarrow_\theta (E, R')$, $R' \models_\sigma Gs$ and we want to specify these substitutions explicitly, we write $(Ss, R) \models_{\theta,\sigma} Gs$.

2.8.1. Service reduction

Using a service Ss to obtain some goals, it may be the case that not every skill occurring in Ss is needed to satisfy the goals. If the use of a skill is not for free, it might be profitable for an agent to look for skills that might be omitted from a service if the goals and initial resources are known. In this subsection we will develop some results about such *service reduction methods*. In particular, we will study the consequences of removing one skill from a service, while leaving nearly untouched the remaining structure of the service.

Definition 22. Let Ss be a service and s a skill occurring in Ss. The *elementary reduction* of Ss with respect to s, denoted by $Ss - s$, is the service obtained from Ss by substituting E for s in Ss. Given a service Ss and a set $S = \{s_1, \ldots, s_n\}$ of skills, the reduced service $Ss - S$ is obtained by repeatedly removing a skill $s \in S$ from Ss.

First we show that the loss of a skill in a service can be easily compensated by adding the right resources in order to maintain goal realizability. Intuitively, these additional resources should be similar to the output resources of the skill. We will use this result further on and therefore we present a rather elaborate proof.

Proposition 23. If $(Ss, R) \models_{\theta,\sigma} Gs$ and s is a skill occurring in Ss, then $(Ss - s,\ R \cup out(s)\tau) \models_{(\theta-\tau),\sigma} Gs$, where $\tau \subset \theta$ is a substitution with $dom(\tau) = var(s)$.

Proof. By induction on the structure of Ss we show that $(Ss, R) \to_\theta (E, R')$ for some substitution θ, implies that there exists some substitution $\tau \subset \theta$ with $dom(\tau) = var(s)$, such that

$$\big(Ss - s,\ R \cup out(s)\tau\big) \to_{\theta-\tau} \big(E,\ R' \cup in(s)\tau\big).$$

Then $(Ss, R) \models_{\theta,\sigma} Gs$ implies that $R' \cup in(s)\tau \models_\sigma Gs$ and therefore, $(Ss - s, R \cup out(s)\tau) \models_{(\theta-\tau),\sigma} Gs$.

So it remains to present the proof by induction.

1. Suppose $Ss = s$ and $(s, R) \to_\theta (E, R')$ for some substitution θ. By definition 14, it follows that $R' = (R - in(s)\theta) \cup out(s)\theta$. Since $\to$ is reflexive, and $\tau = \theta$, $(Ss - s, R \cup out(s)\theta) = (E, R \cup out(s)\theta) \to_\varepsilon (E, R \cup out(s)\theta) = (E, R' \cup in(s)\theta)$.

2. Suppose $Ss = (\pi_1; \pi_2)$ and $(Ss, R) \to_\theta (E, R')$.
   Hence, we have $(\pi_1; \pi_2,\ R) \to_{\theta_1} (\pi_2, R'') \to_{\theta_2} (E, R')$, where $\theta = \theta_1 \cup \theta_2$. But this implies that

   $$(\pi_1, R) \to_{\theta_1} \big(E, R''\big) \tag{1}$$

   and

   $$\big(\pi_2,\ R''\big) \to_{\theta_2} \big(E,\ R'\big). \tag{2}$$

We distinguish two cases:

(a) If $s \in \mathit{Skills}(\pi_1)$, then, by induction hypothesis, we derive from equation (1):

$$\big(\pi_1 - s, R \cup \mathit{out}(s)\tau\big) \rightarrow_{\theta_1 - \tau} \big(E, R'' \cup \mathit{in}(s)\tau\big),$$

where $\tau \subset \theta_1$. By monotonicity (see proposition 20(i)), it follows from equation (2) that

$$\big(\pi_2, R'' \cup \mathit{in}(s)\tau\big) \rightarrow_{\theta_2} \big(E, R' \cup \mathit{in}(s)\tau\big)$$

and combining these derivations, noting that $Ss - s = (\pi_1 - s; \pi_2)$, we derive

$$\big(Ss - s, R \cup \mathit{out}(s)\tau\big) \rightarrow_{\theta - \tau} \big(E, R' \cup \mathit{in}(s)\tau\big).$$

(b) If $s \in \mathit{Skills}(\pi_2)$, then, by induction hypothesis, we have

$$\big(\pi_2 - s,\ R'' \cup \mathit{out}(s)\tau\big) \rightarrow_{\theta_2 - \tau} \big(E, R'' \cup \mathit{in}(s)\tau\big),$$

where $\tau \subset \theta$. But then, again by proposition 20(i) and by equations (1) and (2), it follows that

$$\big(Ss - s, R \cup \mathit{out}(s)\tau\big) \rightarrow_{\theta - \tau} \big(E, R' \cup \mathit{in}(s)\tau\big).$$

3. Suppose $Ss = \pi_1 \parallel \pi_2$ and $(Ss, R) \rightarrow_\theta (E, R')$.
By the definition of $\parallel$ we must have $R = R_1 \cup R_2$ for some R_1, R_2 and

$$(\pi_1, R_1) \rightarrow_{\theta_1} \big(E, R_1'\big) \quad \text{and} \quad (\pi_2, R_2) \rightarrow_{\theta_2} \big(E, R_2'\big),$$

where $\theta = \theta_1 \cup \theta_2$ and $R' = R_1' \cup R_2'$.
Without loss of generality we may assume that $s \in \mathit{Skills}(\pi_1)$. Hence, by induction hypothesis,

$$\big(\pi_1 - s, R_1 \cup \mathit{out}(s)\tau\big) \rightarrow_{\theta_1 - \tau} \big(E, R_1' \cup \mathit{in}(s)\tau\big),$$

where $\tau \subset \theta_1$. By the uniqueness postulates UN1–UN3, it follows that $(R_1 \cup \mathit{out}(s)\tau) \cap R_2 = \emptyset$ and $(R_1' \cup \mathit{in}(s)\tau) \cap R_2' = \emptyset$. Hence, using definition 17, we derive

$$\big(Ss - s, R \cup \mathit{out}(s)\tau\big) \rightarrow_{\theta - \tau} \big(E, R' \cup \mathit{in}(s)\tau\big). \qquad \square$$

This proposition captures the intuitively obvious idea that a loss of some skill in a service always can be compensated for by (i) adding some (initial) resources, (ii) without affecting the structure of the remaining part of the service.

In general, however, this is not a solution we are interested in. Instead, we would like to find a solution (i) *without* affecting the initial set of resources available and (ii) *hardly affecting* the structure of the remaining part of the service.

The idea is that sometimes we do not need to add resources if we can use resources produced by the service Ss itself that are not involved in the goal satisfaction process. That is, whenever we remove a skill s we might try to compensate for its removal by

using *free resources* generated by other skills in the service to substitute for the output resources of s.

To guarantee that such a substitution of resources does not affect the goal realizability when executing the remaining part of the service, we have to analyze what requirements such resources have to satisfy in order to be used. This will force us to analyze the *dependency structure* between a given skill and other skills in a service.

Secondly, we have to make clear what is meant by a hardly changing the structure of the remaining service in order to adapt to the loss of some skill. To this end we will introduce the notion of a *sequential refinement* of a service.

2.8.2. Free resources and dependencies

Suppose some skill s is removed from Ss and let r be a resource produced by s that is used as input resource of some skill s' sequentially dependent on s. So we need to replace r. Suppose that a resource r' produced by some skill s'', is used to replace r. As a consequence, a sequential dependency will be created between s'' and s'.

This immediately implies that the production of such a *free resource* r' should not depend on the skill s nor on any skill sequentially dependent on s in the service Ss.

So, let Ss be a service containing a skill s. We define the (sub)service $Ss_{\text{post}(s)}$ as the service containing all skills in Ss dependent on s, as follows:

Definition 24.

$$Ss_{\text{post}(s)} = \begin{cases} E, & \text{if } Ss = s, \\ (\pi_{\text{post}(s)}; \pi'), & \text{if } Ss = (\pi; \pi') \text{ and } s \text{ occurs in } \pi, \\ \pi'_{\text{post}(s)}, & \text{if } Ss = (\pi; \pi') \text{ and } s \text{ occurs in } \pi', \\ \pi_{\text{post}(s)}, & \text{if } Ss = (\pi \parallel \pi') \text{ and } s \text{ occurs in } \pi, \\ \pi'_{\text{post}(s)}, & \text{if } Ss = (\pi \parallel \pi') \text{ and } s \text{ occurs in } \pi'. \end{cases}$$

As an easy consequence of the last definition we mention the following proposition:

Proposition 25. If $s \in \mathit{Skills}(Ss)$ and $s' \in \mathit{Skills}(Ss_{\text{post}(s)})$ then

$$\mathit{Skills}(Ss_{\text{post}(s')}) = \mathit{Skills}\big((Ss_{\text{post}(s)})_{\text{post}(s')}\big) \subseteq \mathit{Skills}(Ss_{\text{post}(s)}).$$

It follows that no resource produced by a skill in $Ss_{\text{post}(s)}$ can be used as a free resource to compensate for the removal of s from Ss.

Now we are ready to define the set of free resources $\mathit{Free}(Ss, s, R, \theta)$ that might be used to replace resources r produced by some skill s removed from Ss.

Definition 26. Let $(Ss, R) \rightarrow_\theta (E, R')$ and $s \in \mathit{Skills}(Ss)$. The *set of free resources* $\mathit{Free}(Ss, s, R, \theta)$ that can be used if s is removed from Ss equals the set R_{free} satisfying $(Ss, R) \rightarrow_\theta (s; Ss_{\text{post}(s)}, R_{\text{free}})$.

That is, the set R_{free} equals the set of resources produced by the part of the service Ss not containing s or any service sequentially dependent on s in Ss using the original bindings θ that have been produced when Ss was applied to R.

2.8.3. Sequential refinement

We use a sequential refinement of a service to capture the idea of *minimal changes* in the structure of a service. Intuitively speaking, Ss' is a sequential refinement of Ss iff sequential precedence constraints in Ss are not violated. This notion can be easily defined using the set of dependencies in a service.

Definition 27. Let Ss and Ss' be services. Then Ss' is a *sequential refinement* of Ss, abbreviated $Ss' \preceq Ss$, iff

(i) $\mathit{Skills}(Ss) = \mathit{Skills}(Ss')$, and

(ii) for every $s \in Ss$, $\mathit{Skills}(Ss_{\text{post}(s)}) \subseteq \mathit{Skills}(Ss'_{\text{post}(s)})$, i.e., Ss' respects all sequential dependencies occurring in Ss.

An important property of sequential refinements is that the transition relations are preserved.

Lemma 28 (Sequential refinement and transition relation). If it holds that $(Ss, R) \rightarrow_\theta (E, R')$ and $Ss' \preceq Ss$ then $(Ss', R) \rightarrow_\theta (E, R')$.

Proof. Easy but tedious by induction on the composition of Ss. □

Using these notions of refinements and free resources, we will state now one of the main results of this section. The following theorem characterizes (i) the requirements a set of free resources has to satisfy and (ii) the changes that have to be made in the remaining part of the skill in order to guarantee goal realizability. Intuitively, the theorem states that whenever a skill s is removed from a service Ss used and the set of free resources is sufficient to realize the goal starting with the sub-service $Ss_{\text{post}(s)}$, we always can find a *sequential refinement* of $Ss - s$ that realizes Gs using the *original* set of resources.

Theorem 29. Let $(Ss, R) \models_{\theta,\sigma} Gs$, $s \in \mathit{Skills}(Ss)$ and F_s be the set of free resources $\mathit{Free}(Ss, s, R, \theta)$. If $(Ss_{\text{post}(s)}, F_s) \models Gs$, there exists a sequential refinement $Ss' \preceq (Ss - s)$ such that $(Ss', R) \models Gs$.

Proof. Suppose the conditions are verified. So, let $(Ss, R) \models_{\theta,\sigma} Gs$, $s \in \mathit{Skills}(Ss)$, $F_s = \mathit{Free}(Ss, s, R, \theta)$ and suppose $(Ss_{\text{post}(s)}, F_s) \models Gs$. We define the service Ss_{pre} to be the sub-service obtained from Ss by substituting E for every skill occurring in $\{s\} \cup \mathit{Skills}(Ss_{\text{post}(s)})$ and we define the service Ss' as the sequential composition of Ss_{pre} and $Ss_{\text{post}} = Ss_{\text{post}(s)}$: $Ss' = (Ss_{\text{pre}}; Ss_{\text{post}})$. It now suffices to verify the following claims. □

Claim 1. $Ss' \preceq (Ss - s)$; i.e., Ss' is a sequential refinement of Ss.

Claim 2. $(Ss', R) \models Gs$.

Proof of claim 1. By definition of Ss', it immediately follows that $Skills(Ss') = Skills(Ss - s)$. Let s' be an arbitrary skill in $Ss - s$. We verify that $Skills((Ss - s)_{post(s')}) \subseteq Skills(Ss'_{post(s')})$, distinguishing the following cases:

- s' occurs in Ss_{pre}. Then

$$Skills\big((Ss - s)_{post(s')}\big) \subseteq Skills\big((Ss_{pre})_{post(s')}\big) \cup Skills(Ss_{post(s)}) = Skills\big(Ss'_{post(s')}\big).$$

- s' occurs in Ss_{post}. Then we have

$$Skills\big((Ss - s)_{post(s')}\big) = Skills\big((Ss_{post})_{post(s')}\big) = Skills\big(Ss'_{post(s')}\big).$$

Therefore, $Ss' \preceq (Ss - s)$. □

Proof of claim 2. Since $F_s = Free(Ss, s, R, \theta)$, by definition, it follows that $(Ss, R) \rightarrow_\theta (s; Ss_{post(s)}, F_s)$. Hence, according to the definition of Ss_{pre} we have

$$\big(Ss - Skills(s; Ss_{post(s)}), R\big) = (Ss_{pre}, R) \rightarrow_{\theta_1} (E, F_s),$$

where $\theta_1 \subset \theta$ is the restriction of θ to $var(Ss_{pre})$. Now assuming that $(Ss_{post(s)}, F_s) \models Gs$, there exists a substitution σ and a set of resources R' such that $(Ss_{post(s)}, F_s) \rightarrow_\sigma (E, R')$ and $R' \models Gs$.

Since $Ss' = (Ss_{pre}; Ss_{post(s)})$ we now immediately derive that

$$\big(Ss', R\big) \rightarrow_{\theta_1 \cup \sigma} \big(E, R'\big)$$

and therefore, $(Ss', R) \models Gs$. □

Observe that $\theta_1 \cup \sigma$ does not need to be contained in θ. The reason is that a completely different set of bindings might be produced by $Ss_{post(s)}$ to derive a set of resources satisfying Gs.

There is, however, a special case in which the same bindings used in the execution of Ss can be used to satisfy the set of goals in the reduced service. Suppose that a service Ss is used to produce a set of resources R' from R using a substitution θ. Moreover, it is known that $R' \models_\sigma Gs$. Let s be a skill to be removed and let F_s the set of of free resources not depending on s. Suppose now that we are able to find a set of resources R_s similar to the set $out(s)\theta$ of resources to be replaced such that (i) R_s contains output resources of Ss or input resources of s, i.e., $R_s \subseteq (F_s \cap R') \cup in(s)\theta$ and (ii) R_s is not involved in the binding of goals, that is: $R' - R_s \models_\sigma Gs$.

We show that we can use R_s to replace the resources originally produced by s if we use a sequential refinement of $Ss - s$.

Theorem 30. Let $(Ss, R) \rightarrow_\theta (E, R')$, $R' \models_\sigma Gs$, $s \in Skills(Ss)$ and let $F_s = Free(Ss, s, R, \theta)$ be the set of free resources.

If there is a set $R_s \subseteq (F_s \cap R') \cup in(s)\theta$ such that

(1) $R_s \approx out(s)\theta$ and

(2) $(R' - R_s) \models_\sigma Gs$,

then there exists a sequential refinement $Ss' \preceq Ss - s$ such that for substitutions $\theta' \approx \theta$ and $\sigma' \approx \sigma$ it holds that $(Ss', R) \models_{\theta',\sigma'} Gs$.

Proof. Suppose that all the conditions hold. Let $Ss_{\text{pre}} = (Ss - s) - Skills(Ss_{\text{post}(s)})$. From the definition of F_s it follows that $(Ss, R) \to_{\theta_1} (Ss - Ss_{\text{pre}}, F_s)$ where θ_1 is θ restricted to Ss_{pre}. But this implies that

$$(Ss_{\text{pre}}, R) \to_{\theta_1} (E, F_s).$$

Define the service $Ss_s = Ss_{\text{pre}}; s; Ss_{\text{post}(s)}$. Clearly, Ss is a sequential refinement of Ss and by lemma 28, we have

$$(Ss_s, R) \to_{\theta_1} (s; Ss_{\text{post}(s)}, F_s) \to_{\theta-\theta_1} \big(E, R'\big).$$

Then it suffices to prove that $(Ss_{\text{post}(s)}, F_s) \models_{\rho,\sigma'} Gs$ for some $\rho \approx (\theta - \theta_1)$ and $\sigma' \approx \sigma$.

Since $(s; Ss_{\text{post}(s)}, F_s) \to_{\theta-\theta_1} (E, R')$, by proposition 23 it follows that

$$\big(Ss_{\text{post}(s)}, F_s \cup out(s)\tau\big) \to_{\theta-\theta_1-\tau} \big(E, R' \cup in(s)\tau\big),$$

where $dom(\tau) = var(\sigma)$.

Since $R_s \subseteq (R' \cap F_s) \cup in(s)\theta$, by proposition 20(ii) we also have

$$\big(Ss_{\text{post}(s)}, F_s \cup out(s)\tau - R_s\big) \to_{\theta-\theta_1-\tau} \big(E, R' \cup in(s)\tau - R_s\big),$$

Now $out(s)\theta \approx R_s$ and since τ equals θ restricted to $var(s)$, we also have $out(s)\tau \approx R_s$. Therefore, by proposition 20(iii):

$$(Ss_{\text{post}(s)}, F_s \cup R_s - R_s) \to_{\theta-\theta_1-\tau} \big(E, R''\big),$$

where $R'' \approx (R' \cup in(s)\theta - R_s)$.

Hence, $(Ss_{\text{post}(s)}, F_s) \to_{\theta-\theta_1-\tau} (E, R'')$. Finally, since $(R' - R_s) \models_\sigma Gs$, we also have $R' \cup in(s)\theta - R_s \models_\sigma Gs$. But then, by postulate UN3, it follows that $R'' \models_{\sigma'} Gs$ for some $\sigma \approx \sigma'$. Using $\rho = \theta - \theta_1$, we have $(Ss_{\text{post}(s)}, F_s) \models_{\rho,\sigma'} Gs$. □

2.9. Sequential refinement and multi-agent cooperation

The elementary reductions of services can be applied to the services of single agents but also – in a more interesting way – to the services of agents together. Suppose that $A_1, \ldots, A_n$ are agents using (disjoint) sets of resources R_i and services S_i to satisfy goal schemes $Gs_i = (G_i, C_i)$. Clearly, an independent combination $Ss = Ss_1 \parallel Ss_2 \parallel \ldots \parallel Ss_n$ of their services can be considered as a model of their joint services and we can easily derive

$$(Ss, R_1 \cup R_2 \cup \cdots \cup R_n) \models \Big(\bigcup G_i, \bigcup C_i\Big).$$

That is, the agents jointly are able to satisfy the joint goals $Gs = (\bigcup G_i, \bigcup C_i)$ using their joint resources $R = R_1 \cup R_2 \cup \cdots \cup R_n$.

Hence, the service reduction techniques discussed sofar for the single agent case can be easily extended to deal with the multi-agent case and can be used to form the formal basis for modeling agent cooperation, if we are prepared to model both intra- and inter-agent independent skill execution using the independent combination operator $\|$.

Imagine that, initially, the agents run their services concurrently to satisfy their individual goals. We assume that the cost of executing a service is the sum of the costs associated with the applications of the individual skills and the costs of the resources occurring in R.

Although the concurrent composition of services is sufficient to satisfy the goals, it will often contain more skills and uses more resources than is necessary to satisfy all the goals. Hence, the agents might try to reduce the costs of their service (i) by reducing the number of skill applications, (ii) without increasing the set of resources available, while (iii) maintaining goal-realizability. The techniques they can apply essentially are based on the service reduction theorems we have presented above. Basically, we distinguish two main techniques for inter-agent cooperation.

1. *Ground-plan based merging.* In the previous section we used a service Ss to apply them to a resource set R thereby generating a ground substitution θ and producing a set of resources R'. This set in turn was shown to satisfy a given goal scheme $Gs = (G, C)$ by using a substitution σ such that $G\sigma \subseteq R'$ while $\models C\sigma$. It is not difficult to see that $Ss\theta$ is an independent/sequential composition of ground instances of skills taking as input a set of resources R and producing R'. The substitution σ informs us in a unique way about the subset $G\sigma$ of R' that is used to satisfy the goal scheme. We call the triple $(R, Ss\theta, G\sigma)$ a *ground plan* for achieving Gs from R using θ.
   Assuming that all the agents have such ground plans for realizing their goals, they might try to reduce their ground plans, that is try to remove some of the ground instances of the skills used, without affecting the substitutions used in other skills and the goal bindings. Such reductions can only be achieved if upon the removal of such a ground instance $s\theta$, the agent can find a set of resources similar to $out(s)\theta$ if $s\theta$ is used in the goal production. Theorem 30 provides the essentials of such ground-plan based merging processes.
2. *Flexible-plan based merging.* While ground-plan based merging can be achieved rather efficiently, in many respects it is rather inflexible. Sometimes, an agent might use resources of another agent after a removal of its own skill, but then he has to adapt some of the remaining skill applications because the resources provided are not similar to the ones to be replaced. They even might have some consequences in the goal to end-product bindings. Merging techniques based on an exchange of non-similar resource sets can be based on theorem 29. These merging techniques require the use of the original skill and testing of constraints. It is therefore more flexible, but in general computationally more involved.

We will discuss both variants in the next section.

Example 31. Consider again the drive and travel skills discussed before, let $Ss = \{\textbf{drive\_rdam}; \textbf{travel}\}$ and let $t(adam, rdam) = 40$. Furthermore, let $R = (\{taxi(adam, [100, \infty]), p(adam, [80, \infty])\}$ be a set of resources, and $(G, C) = (\{p(rdam, [t'', \infty])\}, \{t'' \leqslant 150)\})$ be a goal scheme. Then $P = (R, Ss\theta, G\sigma)$ with $\theta = \{x = adam, y = rdam, t_1 = 100, t_2 = \infty, t_3 = 100, c = 2\}$ and $\sigma = \{t'' = 140\}$ is a ground plan for (G, C) using Ss.

3. Plan merging

For agents acting in a common environment, on one hand, it is often necessary to cooperate, for example, to reduce their costs. On the other hand, agents usually like to keep some information to themselves. The following example illustrates this difficulty.

Example 32. Consider two taxi companies each having their own plans for their taxis. Such companies do not like to reveal all their plans to a third party, because they do not trust each other, but they do like to cooperate, because taxis often drive to a location without transporting any passengers. It may happen that such an empty ride coincides with the route one of the clients of the other company would like to take. In this case, one company could sell its – otherwise unused – ride to the other, thereby reducing the costs of its plan.

In the previous section, we have shown that a skill can be deleted from a service without affecting goal-realizability, at the cost of introducing additional dependencies between skills (theorem 29), or by using resources from the set of free resources (theorem 30). In the multi-agent setting, however, we do not have one service representing the plans of all agents. Because the agents like to keep information to themselves, each agent has its own service. The dependencies between skills of services of separate agents are different from the usual dependencies within a service, because these dependencies are not visible to the agents. We represent the dependency of an agent a_1 on an agent a_2 by introducing an additional goal for a_2 similar to an additional initial resource of agent a_1.

In this section we first describe a plan merging algorithm based on replacing the output resources of a skill by free resources that are similar in the sense of definition 11. We call this algorithm *ground plan merging*. Although validity of resource replacement can easily be checked, it may be inefficient as we use a too strict notion of similarity. Therefore, we refine the ground plan merging algorithm to a more flexible plan merging algorithm, where we only require replacements to match the same constraints as the output resources of the deleted skill. That is, all constraints on subsequent skills and goals that depend on these resources must be satisfied by the new resources. This algorithm, called *flexible plan merging*, is described in section 3.2, followed by an analysis of the time-complexity of both algorithms.

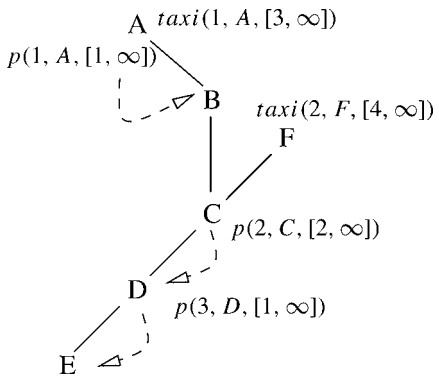

Figure 1. A layout of a city annotated with initial resources.

3.1. Ground plan merging

The goal of ground plan merging is to remove as many skills as possible by exchanging ground resources between agents, while retaining goal-realizability. In a plan merging algorithm for a multi-agent system, each agent should retain local control, and be able to decide for itself what information or resources to share, and what not. We illustrate this problem by elaborating on example 32. To discuss the exchange of resources in a simple way, we use a graph representation of a ground plan that allows us to address resources in a plan directly.

Definition 33. Let $(R, Ss\theta, G\sigma)$ be a ground plan. Then, the *plan graph* F associated with this ground plan is the bipartite directed acyclic graph $F = (S \cup R', A)$, where $S = Skills(Ss)$, and $R' = \bigcup_{s \in S}(in(s) \cup out(s))\theta$, i.e., the set of all resources, including intermediate resources, produced by Ss; and $A = \{(r, s) \mid r \in in(s)\theta\} \cup \{(s, r) \mid r \in out(s)\theta\}$, i.e., the set of all arcs connecting resources to skills using or producing these resources.

Example 34. In the examples in this section we use resources fitting the resource scheme $p(number : N, location : \{A, \ldots, F\}, time : \mathcal{I})$ to describe passengers at a location taken from the set $\{A, \ldots, F\}$ at a time specified by an interval $I \in \mathcal{I}$, and we use the resource scheme $taxi(number : N, location : \{A, \ldots, F\}, time : \mathcal{I})$ to describe a taxi that is available at a location $\in \{A, \ldots, F\}$, during a specific time interval.[6] Consider a simple street plan of a city as shown in figure 1. The dashed arrows point to the goal location of the passengers. We assume an agent a_1 that has the disposal of skills **drive**_y, with y taken from the set $\{A, \ldots, F\}$, and **travel**, as described in example 9, and of the initial resources $\{taxi(2, F, [4, \infty]), p(3, D, [1, \infty])\}$. Furthermore, we assume that a_1 has to attain the goal: $(\{p(3, E, [t_1, t_2])\}, \{t_1 \leqslant 8\})$, that is, passenger 3 should be at location E before time 8. We assume that agent a_1 has a

[6] Note that in these examples, the identification attribute *id* of a resource (as defined in section 2.5) is omitted to improve readability.

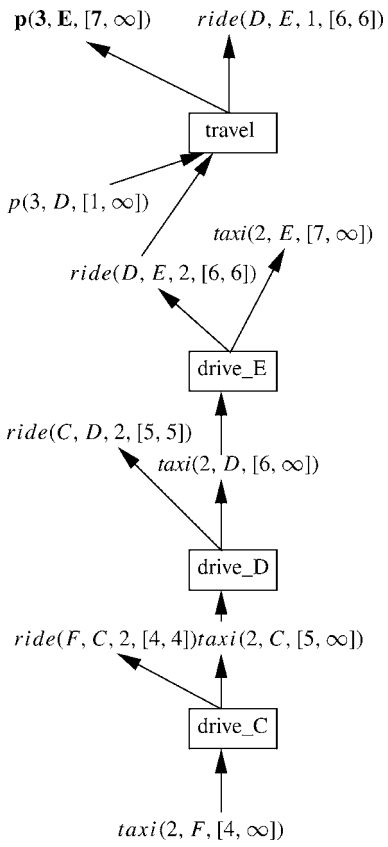

Figure 2. A plan graph for agent a_1 to reach the goal $(\{p(3, E, [t_1, t_2])\}, \{t_1 \leqslant 8\})$, that is displayed in boldface.

plan available to attain these goals, such as shown in figure 2. We also assume an agent a_2 having the disposal of the same skills, but a different (disjunct) set of initial resources: $\{taxi(1, A, [3, \infty]), p(1, A, [1, \infty]), p(2, C, [2, \infty])\}$, and a goal set: $(\{p(2, D, [t_1, t_2]), p(1, B, [t_3, t_4])\}, \{t_1 \leqslant 8, t_3 \leqslant 5\})$, that is, passenger 2 should be at D before 8 and passenger 1 should be at B before 5, leading to a plan indicated by figure 3. The problem is how to discover which of the nine skills can be removed by exchanging resources, without constructing a global plan.

In this section, we use an auction mechanism to solve this problem. We assume one of the agents, or a trusted third party, acts as the auctioneer. All agents deposit requests with this auctioneer. Each request corresponds with the removal of a skill from an agent's plan, and thus has an expected cost reduction. A skill can be removed if for each output resource that is needed by another skill or as a goal, a replacement can be found. The auctioneer deals with the request with the highest potential cost reduction first. Right before the auction is started, the requesting agent is asked for the specific set of resources R that has to be replaced by resources of other agents. This

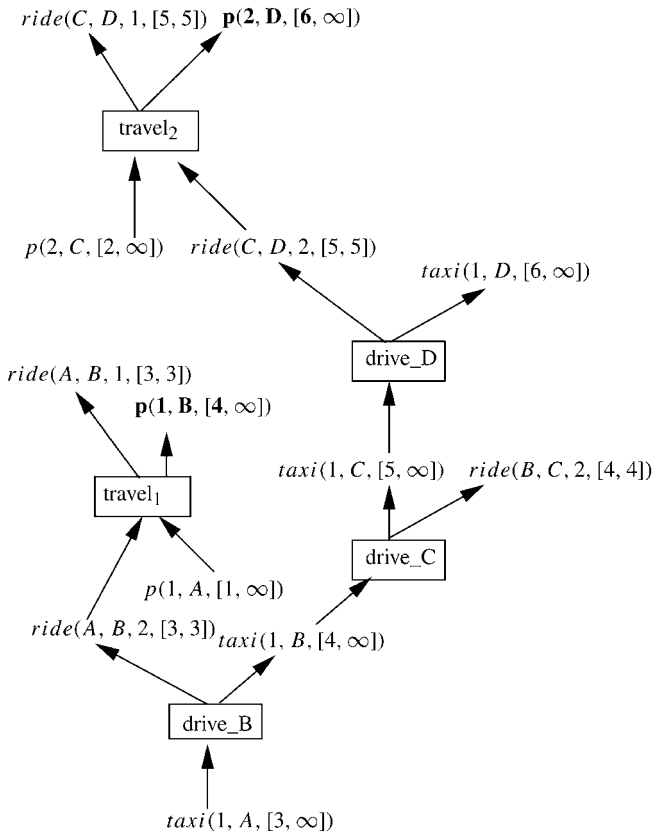

Figure 3. This figure shows a plan graph for agent a_2 to reach the following two goals: $(\{p(2, D, [t_1, t_2]), p(1, B, [t_3, t_4])\}, \{t_1 \leqslant 8, t_3 \leqslant 5\})$.

set cannot be included in the initial request, since it can change because agents may give resources they could use themselves to other agents (that had requests of a higher priority).

To put up an auction for a request of an agent a, the set of resources R is sent to each agent, except to a. The agents return all their free resources for which there is a similar (see definition 11) one in the request set R, and include the price of each of their offered resources.[7] When all bids are collected by the auctioneer, it selects for each requested resource the cheapest bid. If for each resource in R a replacement can be found, the auctioneer tells the requesting agent a that it may discard the corresponding skill(s). The replacing resources are marked as goals for the providing agents, and become additional 'initial' resources for agent a. If not all resources can be replaced, the request is retried after completion of all other requests. This process is repeated until none of the auctions has been successful. Algorithm 1 describes this auction algorithm in more detail.

[7] In this paper we omit a discussion on bid strategies and costs of resources for agents, but focus on the overall structure of the distributed algorithm.

Algorithm 1 (GROUND_PLAN_MERGING(A)).

1. retrieve requests from all agents A
2. **repeat**
   - 2.1. **while** some requests left **do**
     - 2.1.1. get the request with the highest priority
     - 2.1.2. ask the requesting agent a for the required resources R
     - 2.1.3. **for each** $a' \in A, a' \neq a$ **do**
       ask a' for free resources similar to resources in R add these to R'
     - 2.1.4. let $R'' \subseteq R'$ be the cheapest set, such that $R'' \approx R$
     - 2.1.5. **if** successful, i.e., $R'' \approx R$ **then**
       actually replace R by R'' for agent a and mark all resources from R'' as goals for their respective delivering agent
     - 2.1.6. **if** failed **then**
       store this failed request in a secondary storage
   - 2.2. restore all failed requests
3. **until** there has not been a successful replacement

In the following we continue example 34 by showing the effect of plan merging on the plans of the agents.

Example 35. Running GROUND_PLAN_MERGING for agents a_1 and a_2, as described in example 34, results in the following plan modifications. Each agent returns a set of requests to the auctioneer. For a_1 this is at most {(drive_C, 1), (drive_D, 2), (drive_E, 3), (travel, 4)}, and for a_2 {(drive_B, 1), (drive_C, 1), (drive_D, 2), (travel$_1$, 1), (travel$_2$, 3)}. Each request consists of a resource and a number indicating the potential cost reduction. In this example each skill has costs one. The requests are put in a priority queue based on this cost reduction. Subsequently, the one with the highest potential cost reduction is selected, that is (travel, 4) by a_1, and the required resource(s) are determined: $\{p(3, E, [7, \infty])\}$. Unfortunately, agent a_2 does not have this resource in its free resource set. So this request is stored in the secondary storage. Most of the other requests do not lead to a success either. The only request that allows for plan optimization is (drive_D, 2) by agent a_2. The required resource for this request is $\{ride(C, D, 2, [5, 5])\}$ and a_1 happens to have this resource available. This resource is then used in the plan of agent a_2 to replace the similar resource produced by drive_D, and so two skills can be removed from this plan: drive_D and therefore also drive_C, as it then does not produce any product anymore that is used by another skill or marked as a goal. The resource $ride(C, D, 2, [5, 5])$ is marked as a goal for agent a_1, resulting in the merged plan depicted in figure 4.

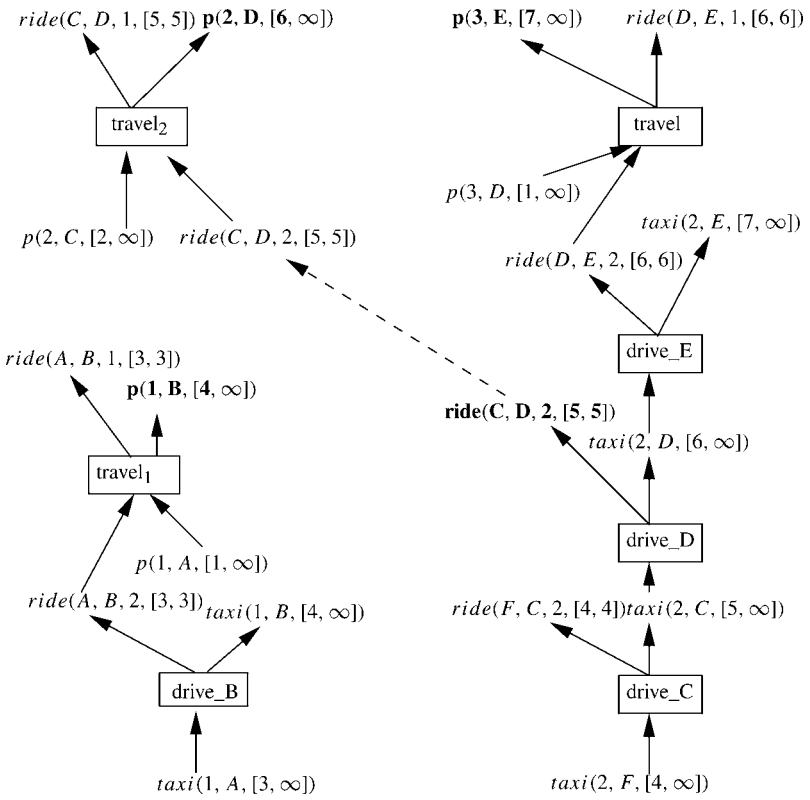

Figure 4. (a) the merged plan for agent a_2 and (b) agent a_1.

The GROUND_PLAN_MERGING algorithm is an any-time algorithm, because it can be stopped at any moment. If the algorithm is stopped, it will still return an improved set of agent plans, because this algorithm used a greedy policy, i.e., dealing with the requests with the largest potential cost reduction first. In the next section we describe a more flexible version of such an any-time merging algorithm.

3.2. Flexible plan merging

The ground plan merging algorithm can be too restrictive, because only resources that are similar, are accepted as a replacement. In general, a replacement should be an instantiation of the same resource scheme and satisfy the same constraints. These constraints are (i) the constraints of the skill or goal that uses this resource, and (ii) indirectly constraints on subsequent skills and the goal constraints. We first give an example of a situation in which the ground plan merging algorithm does not give the desired result.

Example 36. In example 35 we have seen how a resource can be exchanged in a ground plan to improve the joint efficiency. Suppose now that the initial resources of a_1 are $\{taxi(2, F, [3, \infty]), p(3, D, [1, \infty])\}$ instead of $\{taxi(2, F, [4, \infty]), p(3, D, [1, \infty])\}$.

A planner now might come up with a plan where every resource is produced one time unit earlier. Intuitively, it is clear that a similar exchange would still be possible. However, the original plan merging algorithm is too restrictive to discover this possibility, because the requested resource $\{ride(C, D, 2, [5, 5])\}$ by agent a_2 and the free resource $\{ride(C, D, 2, [4, 4])\}$ by agent a_1 are not similar.

In the flexible plan merging algorithm, a request is not specified by a set of resources, but by a set of resource schemes together with a set of constraints on the values of the attributes of these schemes. All constraints on the requested resource schemes are extracted from the services involved in the plan by the requesting agent. Each agent returns only those free resources that satisfy at least one of the resource schemes, such that the set of constraints is satisfiable. It is the task of the auctioneer to find a combination of those returned resources that satisfies both the set of resource schemes and all constraints (preferably with minimal costs). In general, the complexity of the flexible plan merging algorithm (algorithm 2) that solves this problem is exponential for the auctioneer, but in practice the time complexity is usually limited (see section 3.3).[8]

Algorithm 2 (FLEXIBLE_PLAN_MERGING(A)).

1. retrieve requests from all agents A
2. **repeat**
   - 2.1. **while** some requests left **do**
     - 2.1.1. extract the request with the highest priority
     - 2.1.2. let a be the agent placing this request for skill s
     - 2.1.3. using function GET_CONSTRAINTS(a, s) (see algorithm 3) get the required resource schemes and the constraints C from agent a
     - 2.1.4. **for each** $a' \in A, a' \neq a$ **do**
       return all free resources r of a', such that $\exists rs \in RS : r \models rs$ and $C(r)$ is satisfiable, to the auctioneer, storing it in the set R'
     - 2.1.5. $R'' =$ the cheapest combination $R'' \subseteq R'$ such that $R'' \models RS$ and $C(R'')$ is satisfiable
     - 2.1.6. **if** successful **then**
       replace RS by R'' and update agent a's plan; mark all resources from R'' as goals for their respective delivering agent
     - 2.1.7. **if** failed **then** store this failed request in a secondary storage
   - 2.2. restore all failed requests
3. **until** there has not been a successful replacement

[8] Unfortunately, in general, when the constraints range over all resource schemes and resources satisfy any of the resource schemes, this problem is NP-hard.

In step 2 of algorithm 2, the constraints on the requested resources are determined. This procedure is somewhat complicated, because a requested resource r may be used by a skill s having constraints on it. The skill s may produce resources R that due to range-restrictedness (see definition 7), are functionally dependent on this resource r. These resources R, in turn, may be used by other skills or as a goal, again introducing constraints on r (indirectly).

The set of all constraints C on the requested resources can be derived as follows. For an agent a, a set of requested resources R_x, the corresponding set of resource schemes RS_x, a set of resources R from the plan of a, and a set of constraints C on R, we define a recursive function, GET_CONSTRAINTS_ON(a, R_x, RS_x, R, C). This function creates a resource scheme RS corresponding to R, such that the attributes of RS are expressed as a function of the attributes of RS_x, and returns all constraints on RS_x, including the constraints C. Let R_I denote the set of initial (ground) resources. In case $R \subseteq R_x \cup R_I$, the set of constraints C already is of the correct type (on RS_x and some constants) and may be returned, and the attributes of R are also correct (either constant or a function of attributes of R_x).

If there are resources in R that are not ground and not in R_x, then the attributes of the resources R need to be expressed as a function of the attributes of $R_x \cup R_I$. For each resource $r \in R$ that is not in $R_x \cup R_I$, there is a skill s that produces r. The attributes of r are functionally dependent on the input resources of s. A recursive call of GET_CONSTRAINTS_ON the input resources of s, ensures that the attributes of the input resources, and thus the attributes of r can be expressed as a function of the attributes of $R_x \cup R_I$, and so can the constraints C. The GET_CONSTRAINTS_ON algorithm (see algorithm 3) is initiated with the set of free resources and the goal constraints.

Algorithm 3 (GET_CONSTRAINTS(a, s)).

1. Let R_x be the set of output resources of s to be replaced.
2. Let $RS_x \subseteq out(s)$ be the set of resource schemes representing R_x.
3. Let R be the set of free resources of the plan of agent a.
4. Let C be the set of goal constraints to be satisfied by agent a.
5. Let S_{comp} be the (global) set of completed skills, initially $\emptyset$.
6. Return RS_x and the constraints on RS_x, using the function:

   GET_CONSTRAINTS_ON (a, R_x, RS_x, R, C).

   1. Let CB be $\emptyset$.
   2. Determine the set of skills S producing R by tracing one step backwards in the plan graph of agent a.
   3. Let $S' = S - S_{\text{comp}}$ be the skills that have not been dealt with.
   4. Add all skills from S' to S_{comp}.

5. **for each** $s \in S'$ **do**
   5.1. determine the set R' of input resources of s;
   5.2. get constraints on RS_x, given output resources R' and constraints on s (GET_CONSTRAINTS_ON (a, R_x, RS_x, R', C_s));
   5.3. add these constraints to the constraint base CB.
6. **for each** resource $r \in (R - R_x)$ **do**
   express the attributes of r in terms of the attributes of the input resources of the skill s that produces r in the plan of agent a.
7. Evaluate the constraints C on the attributes of R.
8. Add them to the constraint base CB.
9. **return** CB.

We now show how the constraints introduced in example 36 are derived.

Example 37. The set of resource schemes needed for the request (drive_D, 2) of agent a_2 is $\{ride(C, D, c, [t, t])\}$ and the constraint base is $\{c \geqslant 1,\ 2 \leqslant t,\ t \leqslant 7\}$. The first two constraints come directly from the constraints of the drive skill. The third constraint is a result of an evaluation of the goal constraint combined with the functional dependency of the goal $p(2, D, [t_1, t_2])$ on the *ride* resource: $t_1 = t + 1 \leqslant 8$. It is easy to see that the resource $ride(C, D, 2, 4)$ matches these constraints.

3.3. Complexity analysis

In this section we analyze the time complexity of both plan merging algorithms. We start by giving the complexity analysis of FLEXIBLE_PLAN_MERGING, because, as we will see, GROUND_PLAN_MERGING is a special case of FLEXIBLE_PLAN_MERGING. First, however, we make some assumptions. Let n be the number of skills in the joint plan of all agents. We suppose there is a constant $c \geqslant 1$ such that for all skills s holds that $|in(s)|$, $|out(s)| \leqslant c$, and $|C_s| \leqslant c$, and thus (because a request is related to the removal of one skill) for all requests G also $|G| \leqslant c$. Then, because the number of resources related to a skill is bounded by this constant c, we may conclude that the number of resources is $O(n)$, as is the time complexity of one traversal through the plan graph.

Proposition 38. Given a set of skills S, the worst-case time complexity of the FLEXIBLE_PLAN_MERGING algorithm is $O(n^{2+c_S})$, where $c_S = \max_{s \in S}\{|in(s)|, |out(s)|, |C_s|\}$.

Proof. In step 1 of algorithm 2 the requests of the agents are collected and the cost-reductions determined. This can be done by one traversal of the plan of the agent, and thus takes $O(n)$ time. The repeat loop (step 2) is stopped once there is no change.

A change is the removal of a skill. So this loop, in the worst case, is traversed $O(n)$ times. In the worst case, the while-loop (step 2.1) inside this repeat-loop may be executed $O(n)$ times as well, since there are at most $O(n)$ requests.

We discuss the steps of this while-loop that, at first sight, take more than constant time.

Finding the request with the highest priority can be done, e.g., by a priority queue, taking aggregated $O(1)$ time. Step 2.1.3 determines the constraints on the request.[9] This is algorithm 3. Because of the set of completed skills, each skill is considered only once, so, given that the functions can be evaluated in constant time, this step takes $O(n)$ time. Step 2.1.4 takes $O(n)$ time as well, given that constraints can be evaluated in constant time, because at most $O(n)$ free resources are considered. In step 2.1.5, in the worst case, all combinations of size c_S from the set of $O(n)$ resources need to be considered, taking $O(\binom{n}{c_S}) = O(n^{c_S})$ time steps. Finally, step 2.1.6 may take $O(n)$ time, because in some cases a whole sub-tree may be deleted.

All other steps take constant time, as can be easily seen. The time complexity of the FLEXIBLE_PLAN_MERGING algorithm is $O(n)$ for the repeat-loop, times $O(n)$ for the while-loop, times $O(n^{c_S})$ for the contents of the while loop, resulting in a total time complexity of $O(n^{2+c_S})$. □

Proposition 39. The GROUND_PLAN_MERGING algorithm has worst-case time complexity of $O(n^3)$.

Proof. We use the proof of proposition 38. It can be easily seen that the complexity of the steps of GROUND_PLAN_MERGING is equal to the equivalent steps of FLEXIBLE_PLAN_MERGING. There is, however, one important difference: step 2.1.5 in FLEXIBLE_PLAN_MERGING of $O(n^c)$ is related to step 2.1.4 of GROUND_PLAN_MERGING of $O(n)$. Therefore the total time complexity of GROUND_PLAN_MERGING is $O(n^3)$. □

The time-complexity of FLEXIBLE_PLAN_MERGING is caused by the dependencies between the resource schemes. For example, it can be seen that, if (i) for each resource scheme *rs*, the set of constraints on *rs* only involves *rs*, and (ii) the set of resources that satisfies *rs* and C_{rs} is disjoint from sets of resources satisfying other resource schemes, then the selection of the cheapest combination can be done in linear time, like in GROUND_PLAN_MERGING. Proposition 41 bounds this dependency by a constant, depending on the set of goal schemes and the set of skills used. First, we need to define a so-called *dependency constant* for a set of constraints on a set of resource schemes. We use $res(C) = RS$ to denote the set of resource schemes whose attributes occur in the constraints in C (so all attributes occurring in C belong to RS).

[9] For a detailed description and analysis of priority queues see, e.g., Cormen et al. [4].

Definition 40. For a set of constraints C on a set of resource schemes RS, the *dependency constant*, d_C is defined as follows: if there exists a partition $C_1 \cup \cdots \cup C_m$ of C such that:

(1) $res(C_1) \cup \cdots \cup res(C_m)$ is a partition of RS, and

(2) there is no resource r that for $i \neq j$ satisfies both C_i and an $rs_i \in res(C_i)$, as well as C_j and an $rs_j \in res(C_j)$,

then we define the *dependency constant* d_C to be the size of the largest part of the partition of resource schemes, i.e., $d_C = \max_{1 \leqslant i \leqslant m} |res(C_i)|$.

Proposition 41. Given the set of all goal schemes $\mathcal{GC}$ and the set of all skills S. If

$$d = \max\{\max_{s \in S} d_{C_s}, \max_{(G,C) \in \mathcal{GC}} d_C\}$$

then the time complexity of FLEXIBLE_PLAN_MERGING is $O(n^{2+d})$.

Proof. By definition of d, we know that the resource schemes can be partitioned into sets of at most size d. Choosing a combination of resources with minimal costs that satisfies such a set of resource schemes costs, in the worst case, $O((c \cdot n)^d)$ time steps, where $0 \leqslant c \leqslant 1$. If for all k sets of resource schemes, each set i is possibly satisfied by at most $O(c_i \cdot n)$ resources, with $\sum c_i = 1$, then we know that the total time complexity of the step 2.1.5 in FLEXIBLE_PLAN_MERGING is $\sum_{1 \leqslant i \leqslant k} O((c_i \cdot n)^d) = O(n^d)$. By the proof of proposition 38, the result follows. □

The following example illustrates these properties.

Example 42. In our model of the taxi company domain, all skills have at most two input and two output resource schemes. So we have $c = 2$, and therefore the plan merging algorithm has time complexity $O(n^4)$.

Using proposition 41, we derive the following: in one case (the travel skill) we have a skill constraint with $d = 2$ (see definition 40). If we further require that all goals (G, C) are modeled such that $d_C \leqslant 2$, in other words, no goals may be specified so vague that one resource may satisfy more than two goals, this also leads to a time complexity of $O(n^4)$.

4. Discussion

We proposed a formalism in which we are able to express all kinds of constraints on all kinds of attributes. Our plan merging algorithm deals with these constraints and reallocates resources to improve efficiency in a distributed way. Our algorithm is an approximation, using a greedy heuristic, and can be stopped at any time. We also give some restrictions on the problem that allows a very efficient plan merging. Furthermore, we deal with private knowledge of the agents. Dealing with conflicts between agents,

insincere agents, interleaving planning and execution, and replanning is future work (see section 4.3).

This work is strongly related to other work in both the multi-agent community, especially multi-agent planning and multi-agent problem solving, and the computational logic community. In this section we first relate our work on plan merging to other plan merging methods. In section 4.2 we mention some other logics that are used for planning and we show how they are different from our approach. We conclude with a discussion on future work.

4.1. Relation with other plan merging methods

In the beginning of the eighties Rosenschein [24] proposed ways to deal with conflicts between agents that can be resolved by synchronizing the execution of their plans. He showed what to communicate to resolve undesired interactions, so-called *conflicts*. He did not show, however, how to improve the efficiency of plans by cooperation, nor how to cope with quantitative resources like capacity or costs. In our framework we have not paid much attention to resolution of conflicts. However, we assume an initial distribution of all available resources. This resolves most of the conflicts, since an agent may not sell a resource it needs itself.

The work of Georgeff [13,14] on multi-agent planning was also not to improve a set of plans, but to ensure that two or more plans do not conflict. In [14] Georgeff defines a so-called *process model* to formalize the actions open to an agent. Part of a process model are the *correctness conditions*, defined over the state of the world, that must be valid before execution of the plan may succeed. Two agents can help each other by changing the state of the world in such a way that the correctness conditions of the other agent become satisfied. Of course, changing the state of the world may help one agent, but it may also interfere with another agent's current plan. That is, one agent may change the state of the world in such a way that the correctness conditions of another agent become invalid. In our current work we assume that agents already have correct plans available that realize the set of goals before they agree to cooperate to optimize their plans. The notion of correctness conditions could be helpful when we incorporate situations where cooperation between agents is needed to realize the set of goals.

Logic and planning were combined by Stuart [26] to find out whether individual plans can be synchronized. Using Propositional Temporal Logic, constraints are generated from plans to specify the feasible states of the environment. These constraints are given to a theorem prover to generate sequences of communication acts that guarantee that no event will fail. Just as Georgeff, Stuart focuses on the resolution of conflicts, and not on potential positive relations between plans.

Our framework can also be compared with the PGP (Partial Global Plan) framework described by Durfee et al. ([6,7]) and Decker and Lesser ([5]). In this PGP-framework and its extension GPGP, agents are allowed to cooperate by exchanging parts of their local plans to develop common plans. The main difference with our approach to cooperation is that in our framework exchanging parts of local plans is more involved,

since exchanging parts of a plan will usually change not only the agent commitment structure but will also affect the structure of the local plans involved.

Plan merging was also studied by von Martial [18,19]. He considers several types of relations between plans, divided into conflicts and positive relations. Plans are represented hierarchically and need to be exchanged among the agents to determine such relations. If possible, relations are solved or exploited at top-level. If this is not possible, a refinement of the plans is made and the process is repeated. For each specific type of plan relationship a different solution is presented. Our plan merging algorithm is applied at a low-level representation of the plan, not needing any domain specific information, nor communicating the complete plan of one agent to another.

Clement and Durfee [2,3] describe an *interleaved* planning process, where different agents jointly construct a plan to establish a set of goals. They use the class of planners called Hierarchical Task Network (HTN) planners like von Martial's hierarchical plans. To avoid unnecessary backtracking, the agents must recognize whether their plans interfere before the plans are fully refined. Clement and Durfee derive the so-called *plan summary information* from the hierarchical plan operators. Using this summary information, they can reason about possible interactions between different agents before the individual plans are completely specified. It would be interesting to analyze how in a similar way summary information can be derived and used to recognize opportunities to optimize plans of individual agents.

Both to improve efficiency and to resolve conflicts, restrictions may be introduced on individual plans to ensure efficient merging. This work by Yang [28] and Foulser [12] is also useable to merge alternative plans to reach the same goal. Furthermore, some approximation algorithms for plan merging are described by the same authors. They do not, however, deal with costs, quantitative constraints on resource attributes, and their algorithms are not distributed.

Finally, we would like to mention the work on plan merging by Ephrati and Rosenschein [8–10]. Their approach resembles our work the most, because they also developed a distributed polynomial algorithm to improve social welfare (i.e., the sum of the benefits of all agents). Through a process of group constraint aggregation, agents incrementally construct an improved plan by voting about joint actions. They even propose algorithms to deal with insincere agents and to interleave planning, coordination, and execution. But, neither they do pay attention to dealing with numerical constraints, time, and costs. The main difference between their approach and ours, is that our algorithms can be formulated by a resource logic, while theirs use a description of the state of the world by propositions, like in STRIPS, as is discussed in the next section.

4.2. Relation with other planning logics

Several planning logics have been proposed in the literature that are more or less close to the one developed in this paper. The most popular of these are STRIPS that is based on first-order logic [11], and extensions of it like PDDL [21] and Functional STRIPS [1].

STRIPS resembles our formalism with respect to the fact that for each operator explicitly is stated which atomic facts are to be deleted from the current state and which have to be added to create the new state. The atomic facts in the STRIPS formalism are, so to speak, the resources of the resource-skill formalism. This enables embedding STRIPS in the latter. However, unlike our formalism, STRIPS is not suited to directly handle parallel actions, to deal with quantities like capacities, distances, costs, and time, and to model cooperation between agents.

The resource-skill formalism has also a strong resemblance to the revision specification programs developed by Marek and Truszcynski [17]. In these programs rules not only are used to create other atoms or resources but also to remove them given the presence of some input atoms and the absence of others. Although their formalism uses Horn-like rules, skills can be simulated by revision specification programs. The reverse way is also possible, i.e., modeling revision processes using the resource-skill formalism. This result is interesting since it opens the door to applying logic programming techniques to our formalism.

An important difference of our work and the work by Marek and Truszcynski, is the fact that the resource-skill formalism deals with constraints and services as the composition of skills. Furthermore, we developed plan merging algorithms for which our plan representation is very suited.

Another planning logic that is very close to ours is the one developed by Masseron et al. [20]. One important difference is the fact that in their formalism it is possible to include *nondeterministic actions*, i.e., actions with an outcome that is uncertain albeit limited to a set of given outcomes. On the other hand, our formalism can handle constraints, whereas the planning logic of Masseron et al. does not. For their variant, they constructed a correct and complete embedding in a fragment of *linear logic*.

Linear logic, invented and developed by Girard (see, e.g., [16]) is a so-called *substructural logic* that emerges if structural rules such as *weakening* and *contraction* are discarded. Since resources, be it in the sense of our formalism or that of Masseron et al., can be only used once (no absorption), linear logic can be considered a good candidate as a planning logic.

So it seems quite natural to study the relationship between planning and linear logic. However, the fragment of linear logic used for the embedding, is very restricted. It turns out that skills and resources have to be translated as a *linear theory*, using only one linear logical connective (multiplicative conjunction) in the deterministic case, and two (multiplicative conjunction and additive disjunction) in the nondeterministic one.

In their paper, Masseron et al. do not touch the problem of plan merging. In our opinion our presentation of the planning formalism is better suited to study the operational behavior of planning. Our choice of describing the operational semantics of plans by a transition system (cf. Structural Operational Semantics as introduced by Plotkin [23]) underlines this idea. We also have given evidence of the usefulness of our planning formalism to model plan merging.

4.3. Future work

Many applications induce additional requirements on our formalism or algorithms. For example, we would like to develop a mechanism to solve conflicts between agents, such as both agents requiring the same resource. Until now, we assumed all resources have initially been distributed over all agents. This is not realistic in all circumstances, e.g., in cases where agents have to share (scarce) resources. Therefore, we would like to adapt our algorithms to this kind of situations. This could be done, for instance, by incorporating auctioning mechanisms in which agents have to make bids for allocating resources.

Another important issue for future research is the interleaving of planning and execution. Our algorithms are developed from the point of view that agents already have made up their plans before cooperating with other agents. This is a very static view of reality. It would be much more realistic to interleave the process of planning and cooperation, for it is often during planning that the necessity of cooperation seems to become inevitable.

The last point is also of importance in the light of replanning. Replanning is necessary when circumstances are not as expected beforehand due to unforeseen events. For instance, a transportation agent might have to change his route on the way because of a severe traffic jam. So, planning has to be a dynamic process. From this observations it is clear that we have to direct our research to planning as well. Since, from a computational point of view, planning is a difficult problem, we have to develop approximation techniques for this problem.

There are many other directions in which our formalism or our algorithms could be extended. To mention two of these: how to cope with systems in which not all agents are to be trusted (insincere agents), how do we model the various kinds of relationships that are possible between agents (sociological agents). Finally, we are about to test our algorithms with data from public transportation. In this way we hope to substantiate our claim that our cooperation algorithms may improve plan efficiency in a substantial way.

Acknowledgements

We like to thank Roman van der Krogt for his comments on this paper and for his help with the formalism and the algorithms. Furthermore, we greatly appreciate the detailed comments of the reviewers that really helped us to improve this paper.

References

[1] B. Bonet and H. Geffner, Functional STRIPS: a more general language for planning and problem solving, in: *Proceedings of the Logic-Based AI Workshop* (1999).

[2] B.J. Clement and E.H. Durfee, Theory for coordinating concurrent hierarchical planning agents using summary information, in: *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)* (1999) pp. 495–502.

[3] B.J. Clement and E.H. Durfee, Top-down search for coordinating the hierarchical plans of multiple agents, in: *Proceedings of the Third International Conference on Autonomous Agents (Agents-99)* (1999) pp. 252–259.

[4] T. Cormen, C. Leiserson and R. Rivest, *Introduction to Algorithms* (MIT Press/McGraw-Hill, 1990).

[5] K.S. Decker and V.R. Lesser, Designing a family of coordination algorithms, in: *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence*, Lake Quinalt, WA (1994) pp. 65–84.

[6] E.H. Durfee, *Coordination of Distributed Problem Solvers* (Kluwer Academic, Dordrecht, 1988).

[7] E.H. Durfee and V. Lesser, Using partial global plans to coordinate distributed problem solvers, in: *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, Milan, Italy (1987) pp. 875–883.

[8] E. Ephrati and J.S. Rosenschein, Multi-agent planning as a dynamic search for social consensus, in: *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, Chambéry, France (1993) pp. 423–429.

[9] E. Ephrati and J.S. Rosenschein, Multi-agent planning as the process of merging distributed sub-plans, in: *Proceedings of the Twelfth International Workshop on Distributed Artificial Intelligence* (1993) pp. 115–129.

[10] E. Ephrati and J.S. Rosenschein, Divide and conquer in multi-agent planning, in: *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, WA (1994) pp. 375–380.

[11] R.E. Fikes and N. Nilsson, STRIPS: a new approach to the application of theorem proving to problem solving, Artificial Intelligence 5(2) (1971) 189–208.

[12] D. Foulser, M. Li and Q. Yang, Theory and algorithms for plan merging, Artificial Intelligence Journal 57(2–3) (1992) 143–182.

[13] M. Georgeff, Communication and interaction in multi-agent planning, in: *Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83)*, Washington, DC (1983) pp. 125–129. See also [15].

[14] M. Georgeff, A theory of action for multi-agent planning, in: *Proceedings of the Fourth National Conference on Artificial Intelligence (AAAI-84)*, Austin, TX (1984) pp. 121–125. Also published in: *Readings in Distributed Artificial Intelligence*, eds. A.H. Bond and L. Gasser (Morgan Kaufmann, San Mateo, CA, 1988) pp. 205–209.

[15] M. Georgeff, Communication and interaction in multi-agent planning, in: *Readings in Artificial Intelligence*, eds. A. Bond and L. Gasser (Morgan Kaufmann, San Mateo, CA, 1988) pp. 200–204.

[16] J.-Y. Girard, Linear logic, Theoretical Computer Science 50 (1987) 1–102.

[17] V. Marek and M. Truszczynski, Revision specifications by means of programs, in: *Proceedings of European Workshop, JELIA'94,* York, UK, September 1994 (Berlin, 1994) pp. 122–136.

[18] F. von Martial, Interactions among autonomous planning agents, in: *Decentralized AI – Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-89)*, eds. Y. Demazeau and J. Müller (North-Holland, 1990) pp. 105–119.

[19] F. von Martial, *Coordinating Plans of Autonomous Agents*, Lecture Notes in Artificial Intelligence, Vol. 610 (Springer, Berlin, 1992).

[20] M. Masseron, Generating plans in linear logic, Theoretical Computer Science 113(2) (1993) 349–370.

[21] D. McDermott, PDDL – the planning domain definition language, Technical Report TR-98-003, Yale Center for Computational Vision and Control (1998).

[22] B.-J. Moree, A. Bos, H. Tonino and C. Witteveen, Cooperation by iterated plan revision, in: *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-00)* (2000).

[23] G. Plotkin, An operational semantics for CSP, in: *Proceedings IFIP TC2 Working Conference: Formal Description of Programming Concepts – II*, Amsterdam, ed. D. Bjørner (1983) pp. 199–223.

[24] J.S. Rosenschein, Synchronization of multi-agent plans, in: *Proceedings of the Second National Conference on Artificial Intelligence (AAAI-82)*, Pittsburgh, PA (1982) pp. 115–119. Also published in:

Readings in Distributed Artificial Intelligence, eds. A.H. Bond and L. Gasser (Morgan Kaufmann, San Mateo, CA, 1988) pp. 187–191.

[25] O. Shehory and S. Kraus, Methods for task allocation via agent coalition formation, Artificial Intelligence 101(1–2) (1998) 165–200.

[26] C.J. Stuart, An implementation of a multi-agent plan synchronizer, in: *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, Los Angeles, CA (1985) pp. 1031–1033. Also published in: *Readings in Distributed Artificial Intelligence*, eds. A.H. Bond and L. Gasser (Morgan Kaufmann, San Mateo, CA, 1988) pp. 216–219.

[27] M.M. de Weerdt, A. Bos, H. Tonino and C. Witteveen, Fusion of plans in a framework with constraints, in: *Proceedings of the ISCS Conference on Intelligent Systems and Applications (ISA-00)* (2000) pp. 393–399.

[28] Q. Yang, D.S. Nau and J. Hendler, Merging separately generated plans with restricted interactions, Computational Intelligence 8(4) (1992).