Metric Planning using Planning Graphs
- A First Investigation -

Jana Koehler

Institute for Computer Science
Albert Ludwigs University
Am Flughafen 17
79110 Freiburg, Germany
koehler@informatik.uni-freiburg.de

Abstract

After the extension of the GRAPHPLAN formalism to ADL, another important aspect of planning problems forms the focus for the development of IPP: resource-constrained planning problems. IPP's basic resource representation language BRL is motivated, and its syntax and semantics are defined. A sound and complete search algorithm is defined, which can be easily extended to handle conditional resource effects. Examples illustrate the use of the BRL formalism.

Technical Report No. 127

Contents

1 Technical Preliminaries

We begin straightforward with the definition of the technical framework, which will be used in this report. Important notions such as a planning problem, a plan, an operator, an action, and a state are defined. Two planning languages are considered: PDDL, which is a syntactic variant of the ADL formalism [Ped91] and BRL, IPP's experimental language to investigate resource-constrained planning problems. Both languages allow to formally investigate two important extensions of STRIPS:

- conditional effects, which extend the logical expressivity of a planning language,
- resource effects, which add non-logical constructs to a planning language.

This thesis treats both extensions individually. But the algorithms can be easily combined to handle conditional resource effects as it is shown in Section 11.2.

2 States, Planning Problems, Operators

Starting with version 4.0, explicit negation was introduced in IPP. This means, states are no longer described by sets of logical atoms, but now sets of ground literals (sometimes called *facts*) are admitted.

Definition 1 *The set of all ground literals is denoted with P. A* state *$S \in 2^P$ is a set of ground literals.*

A state is complete and consistent if for each ground literal $p \in P$ either p or $\neg p$ is true and contained in the state. If both p and $\neg p$ are true or if both are false and contained in the state, the state is inconsistent. Alternatively, if incomplete states are admitted one can interpret the falsehood of both p and $\neg p$ as an incomplete state description in which nothing is known about the truth of p.[1]

The inconsistent state, in which at least one ground atom and its negation are contained, is denoted with $\perp$ representing logical falsehood. Logical truth is represented with $\top$. It is used to represent the empty precondition or effect condition that is satisfied in every state.

Definition 2 *A* planning problem *$(\mathcal{O}, D, I, \mathcal{G})$ is a 4-tuple where $\mathcal{O}$ is the set of operators, D is the domain of discourse (a finite set of typed constants denoting objects), and I (the initial state) and $\mathcal{G}$ (the goals) are described by a logical formula.*

An operator is a more complex construct. It has a name, a parameter list, preconditions, and effects. The explicit use of a name is not really necessary from a theoretical point of view, but it is a widely agreed convention in the planning community.

Definition 3 *An* operator *is a 4-tuple consisting of*

1. a name*, which is a string,*

[1] IPP works on complete state descriptions.

2. a parameter list *of typed variables,*

3. the preconditions*,*

4. the effects.

Depending on the specific planning language a system accepts, the syntactic structure of preconditions and effects has to satisfy various syntactic restrictions.

Definition 4 *In a* STRIPS operator*, the preconditions and the effects are restricted to sets of literals. All variables have to occur as parameters of the operator.*

Definition 5 *A STRIPS action is an instantiation of the parameters of a STRIPS operator with constants of the corresponding type, where all literals have to be ground.*

A typical example is the **stack** action that puts a block named a on top of a block with name b if b is clear and a robot arm is holding a. As an effect, the robot arm becomes empty, a is clear, it is on b, but b is no longer clear and the robot arm is no longer holding a.[2]

stack(a,b):
:precondition *clear(b), holding(a)*
:effect *arm-empty, clear(a), on(a,b), ¬ clear(b), ¬ holding(a).*

In the following, simple STRIPS actions will be written without keywords, but as implications where the antecedent shows the preconditions and the consequent lists the unconditional effects:

stack(a,b): *clear(b), holding(a)*
⇒ *arm-empty, clear(a), on(a,b), ¬ clear(b), ¬ holding(a).*

Definition 6 *A* plan *is a partially ordered sequence of actions.*

3 Conditional Effects

The possibility to formulate conditional effects (sometimes called context-sensitive effects) significantly contributes to the expressivity of a planning formalism and is one of the important features of the ADL planning language [Ped91]. Depending on the context, in which an action is executed, it can now have different effects.

Definition 7 *An effect is* conditional *iff it has the form $pre_i \Rightarrow eff_i$ where pre_i and eff_i are sets of literals.*

Definition 8 *A conditional effect is* universally quantified *iff it has the form $\forall\, x\ pre_i \Rightarrow eff_i$. All variables occurring in the effect must either be universally quantified or must be parameters of the operator.*

[2]In this thesis, operator names are written with bold-face letters, and variable names are prefixed with a question mark. Keywords, which are prefixed with a colon, indicate the separate parts of an operator such as preconditions and effects.

Given $\mathcal{O}$, now possibly containing conditional effects, and D, the domain, one obtains the set of all *actions* O as the set of all possible ground instances of all operators in $\mathcal{O}$. Given an operator, it is generating the following set of actions: First, all possible type conforming instantiations of the declared parameters are systematically enumerated. For each instantiation, an action is generated in which all occurrences of variables are replaced by the corresponding constants in all parts of the operator. Universally quantified conditional effects are expanded into sets of ground conditional effects within each action.

Definition 9 *An* ADL action *o is a ground instance of an operator and has the form:*[3]

$$
\begin{aligned}
o :\ & \mathit{pre}_0 \\
& \phi_0 : \mathit{eff}_0 \\
& \phi_1 : \mathit{pre}_1 \Rightarrow \mathit{eff}_1 \\
& \vdots \\
& \phi_n : \mathit{pre}_n \Rightarrow \mathit{eff}_n
\end{aligned}
$$

The pre_i, eff_i *are sets of ground literals.* ϕ_0 *is denoting the preconditions of o,* eff_0 *are the unconditional effects, and* ϕ_1 *to* ϕ_n *are the conditional effects. The set of all effects* ϕ_0 *to* ϕ_n *of an action o is denoted with* $\Phi(o)$*.*

4 A Semantics for Parallel ADL Plans

One of the distinguishing features of GRAPHPLAN is its ability to produce *shortest* plans in the sense that it exploits maximal parallelism of actions in the plan. For standard STRIPS operators, it is relatively easy to define when two actions can be executed in parallel and what the result of the execution is, cf. [BF97]. In the case of actions with universally quantified and conditional effects, several semantics are possible that define the result of applying a parallel action set to a state in different ways. Two of them are discussed in the following.

Let us consider the following ground instance of the **move** operator in a problem with the two possible objects *letter, toy,* and the two possible locations *office, home* yielding the following action with one set of unconditional effects and two different conditional effects:

move(*office, home*)
:precondition *at-b(office)*
:effect *at-b(home)*, ¬ *at-b(office)*
in(letter) ⇒ *at(letter, home)*, ¬ *at(letter, office)*
in(toy) ⇒ *at(toy, home)*, ¬ *at(toy, office)*

The result of applying a single action to a given state can be easily defined. If the action is applicable, all action's effects whose effect condition hold in the state are added to the state description and all atoms that have been reverted by the action's effects are

[3]The ground instance of a universally quantified effect $\forall x\ [\mathrm{pre}_i(x) \Rightarrow \mathrm{eff}_i(x)]$ is the conjunction of ground instances $\bigwedge_{k=0}^{n}[\mathrm{pre}_i([x/a_k]) \Rightarrow \mathrm{eff}_i([x/a_k])]$ if the domain $DOM(x)$ of the typed quantified variable x is $\{a_0, a_1, \ldots, a_n\}$. $[x/a_k]$ denotes the instantiation of the variable x with the object constant a_k.

removed from it in order to maintain a consistent state. Trying to apply an action which is not applicable leads to an undefined state.[4]

Definition 10 *Let O be the set of ADL actions with conditional effects, i.e., all ground instances of the operators in $\mathcal{O}$ given in the normalform as defined in Definition 9*

$$\begin{aligned} o : \ & pre_0 \\ & \phi_0 : \mathit{eff}_0 \\ & \phi_1 : pre_1 \Rightarrow \mathit{eff}_1 \\ & \vdots \\ & \phi_n : pre_n \Rightarrow \mathit{eff}_n \end{aligned}$$

O^* be all sequences over O, and Res be a function from states and sequences of actions to states.

$$Res : 2^P \times O^* \longrightarrow 2^P$$

The result of applying a sequence containing the single action o to a state S_P is defined as

$$Res(S_P, \langle o \rangle) = \begin{cases} \left(S_P \cup \Phi(S_P, o)\right) \setminus \bar{\Phi}(S_P, o) & \text{if } S_P \models pre_0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

with

$$\Phi(S_P, o) = \bigcup_{S_P \models pre_i, i \geq 0} \mathit{eff}_i \qquad \text{and} \qquad \bar{\Phi}(S_P, o) = \{\neg p \mid p \in \Phi(S_P, o)\}$$

The result of applying a sequence of more than one action to a state is recursively defined as

$$Res(S_P, \langle o_1, \ldots, o_n \rangle) = Res(Res(S_P, \langle o_1, \ldots, o_{n-1} \rangle), o_n).$$

For a planning language of actions with only unconditional effects, the definition of Res can be extended to a set of parallel actions in a straightforward way such that the resulting state is uniquely defined [BF97]. However, when conditional effects are allowed, it is very difficult and even very restrictive to guarantee the uniqueness property of Res. In order to guarantee that a unique state results from the execution of a parallel set of ADL actions, one could come up with the following definition $Res^\star$:

Definition 11 *Let $Res^\star$ be a function from states and sequences of sets of actions to states*

$$Res^\star : 2^P \times (2^O)^* \longrightarrow 2^P$$

The result of applying the empty sequence of actions to a state is defined as $Res^\star(S_P, \langle\rangle) = S_P$. In the case of a sequence containing one non-empty set we obtain

$$Res^\star(S_P, \langle \{o_1, \ldots, o_n\} \rangle) = \begin{cases} \left(S_P \cup \bigcup_{i=1}^n \Phi(S_P, o_i)\right) \setminus \bigcup_{i=1}^n \bar{\Phi}(S_P, o_i) & \text{if (1) to (4) hold} \\ \text{undefined} & \text{otherwise} \end{cases}$$

[4]An alternative definition could be that the state remains the same.

(1) $\forall\, o_i : S_P \models pre_0(o_i)$

(2) $\forall\, o_i, o_j, i \neq j : \Phi(S_P, o_i) \cup \Phi(S_P, o_j) \not\models \perp$.

(3) $\forall\, o_k, o_j, k \neq j\ \forall\, pre_i(o_k)$ *with* $S_P \models pre_i(o_k)\ : Res(S_P, o_j) \models pre_i(o_k)$

(4) $\forall\, o_k, o_j, k \neq j\ \forall pre_i(o_k)$ *with* $S_P \not\models pre_i(o_k)\ : Res(S_P, o_j) \not\models pre_i(o_k)$

The recursive case of a sequence $Q = \langle Q_1, \ldots, Q_n \rangle$ *is defined as*

$$Res^\star(S_P, Q) = \begin{cases} Res^\star(Res^\star(S_P, \langle Q_1, \ldots, Q_{n-1}\rangle), \langle Q_n \rangle) & \textit{if } Res^\star(S_P, \langle Q_1, \ldots, Q_{n-1}\rangle) \\ & \textit{is defined} \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

Condition (1) requires that all preconditions of all actions in the set need to be satisfied in the state S_P. Condition (2) makes sure that all actions are independent of each other in the sense that for each pair of actions the unconditional and the conditional effects with satisfied effect conditions are consistent. Condition (3) requires that no precondition or effect condition that was valid in the state S_P is made invalid by an unconditional or conditional effect, while Condition (4) requires that no effect condition that was invalid in the state S_P is made valid by the effects of an action.

On one hand, this definition of $Res^\star$ leads to a unique resulting state similarly to what one obtains for simple STRIPS operators. On the other hand, this definition is far too restrictive and unnecessarily complicates the planning process when trying to satisfy conditions (1) to (4) during action selection. In particular, to satisfy (3) and (4) one has to proceed over all effect conditions and decide if it has to be made true or false. As a simple example for the restrictiveness of this semantics, consider the two actions with empty preconditions and one conditional effect:

$$\mathbf{o_1} : \top \qquad\qquad \mathbf{o_2} : \top$$
$$a \Rightarrow y\ ,\ \neg c \qquad\qquad b \Rightarrow x\ ,\ c$$

together with the goal $\{x, y\}$ and the state $S_P = \{a, b\}$. Trying to apply the action set to this state leads to an undefined state because c and $\neg c$ are both made true. But since we are not interested in the value of c (it is not a goal), we would rather be satisfied with obtaining a set of states as the result, one in which c holds and another one in which $\neg c$ holds. In fact, GRAPHPLAN also is too restrictive when defining interference of actions because the same phenomenon can already be observed for simple STRIPS actions [Kno94]. In the example by Knoblock, one action paints a table and the floor in red color. A second action paints a chair and the floor in blue color. The goal is to have a red table and a blue chair, but the color of the floor is irrelevant. GRAPHPLAN would mark **paint(blue)** and **paint(red)** as exclusive because the two actions interfere since one destroys the floor color the other has created. Thus, it would never consider them for parallel execution, i.e., the system only generates shortest parallel plans wrt. its own definition of interference. However, since the color of the floor is irrelevant, both actions are independent relative to the current goals. They can be executed in parallel, but a different state would result from each linearization. The only solution to this problem one can imagine, is to come up with a goal-dependent definition of interference, but it would change the basic planning algorithm significantly because mutual exclusion relationships between facts and actions had to be computed during search as they would now depend on the goal sets. Interference because of conflicting effects would become goal-dependent. It is not really clear if this effort is justified.

But for conditional effects, it is relatively easy to allow multiple states as the result of applying a parallel set of ADL actions. Being less restrictive wrt. the resulting states also avoids the problem that the planning algorithm has to decide for each effect condition if it has to be made true or false. This can be accomplished by the following definition:

Definition 12 *Let $\mathcal{S}_\mathcal{P}$ be a set of states and R be a function from sets of states and sequences of sets of actions to sets of states:*

$$R : 2^{2^P} \times (2^O)^* \longrightarrow 2^{2^P}$$

The result of applying the empty sequence is defined as

$$R(\mathcal{S}_\mathcal{P}, \langle\rangle) = \mathcal{S}_\mathcal{P}$$

For the sequence containing exactly one set set of parallel actions $Q = \{q_1, \ldots, q_n\}$ we define

$$R(\mathcal{S}_\mathcal{P}, \langle Q\rangle) = \begin{cases} \{T \in 2^P \mid S_P \in \mathcal{S}_\mathcal{P}, q \in Seq(Q), T = Res(S_P, q)\} & \text{if } Res(S_P, q) \text{ is defined} \\ & \forall\, S_P \in \mathcal{S}_\mathcal{P}, q \in Seq(Q) \\ \text{undefined} & \text{otherwise} \end{cases}$$

with $Seq(Q)$ denoting the set of all linearizations of the action set $Q = \{q_1, \ldots, q_n\}$. The result of applying a sequence of several sets of parallel actions is defined as

$$R(\mathcal{S}_\mathcal{P}, \langle Q_1, \ldots, Q_n\rangle) = \begin{cases} R(R(\mathcal{S}_\mathcal{P}, \langle Q_1, \ldots, Q_{n-1}\rangle), \langle Q_n\rangle) & \text{if } R(\mathcal{S}_\mathcal{P}, \langle Q_1, \ldots, Q_{n-1}\rangle) \\ & \text{is defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

5 Resource Effects and Resource Requirements

Besides a purely logical representation of actions, one is interested in specifying quantitative information. For example, an action requires a certain amount of a resource as its precondition and as its effect, it raises the available amount of another resource.

Definition 13 *A resource requirement is an expression of the form*

$$\langle rvar\rangle\langle comp\rangle\langle arex\rangle$$

*where $\langle rvar\rangle$ is a resource variable name, $\langle comp\rangle$ is an arithmetic comparison being either $=$ (*equal*), $\leq$ (*lower-or-equal*), or $\geq$ (*greater-or-equal*), and $\langle arex\rangle$ is an arithmetic term of one of the following forms*

- z
- $f(?x_1, \ldots, ?x_n)$
- $z\ [+ \mid - \mid * \mid \div]\ f(?x_1, \ldots, ?x_n)$
- $f(?x_1, \ldots, ?x_n)\ [+ \mid - \mid * \mid \div]\ z$

with z being a rational number and f being a function symbol whose arguments $?x_i$ must be logical parameters of the operator.

Typical examples of a resource requirement are $\$x \geq 5$ or $\$y \leq height(?x)$.[5]

Definition 14 *A resource effect is an expression of the form*

$$\langle rvar \rangle \langle ass \rangle \langle arex \rangle$$

where $\langle rvar \rangle$ is a resource variable name as above, $\langle ass \rangle$ is one of the following assignment operators $:=, +=, -=$ and $\langle arex \rangle$ being defined as above.

Typical examples of a resource effect are $\$x := 700$ or $\$y += distance(?x, ?y) * 0.3$.

Definition 15 *A resource-constrained action is an ADL action, which contains a set of resource requirements besides the logical preconditions and whose unconditional logical effects are augmented with a set of resource effects.*

The extension of operators with resource requirements and unconditional resource effects requires to adapt the notion of the initial state and goals accordingly.

Definition 16 *A resource-constrained initial state and a resource constrained goal are a set of ground literals and a set of resource requirements.*

6 IPP Input Formats

IPP accepts two input formats: a subset of the PDDL language[6] and its own language BRL.[7] Both languages come in a special input format and have incompatible syntax constructs.

BRL combines conditional logical effects with unconditional resource effects and resource requirements. Such a combination is straightforward because both language extensions, which are studied in this thesis, are independent of each other. An exact definition of the BRL syntax and semantics can be found in Section 8. A typical example of a BRL operator is the following:

[5] To distinguish logical variables names from resource variables names, the former start with a question mark, while the latter start with a dollar sign.

[6] PDDL stands for Planning Domain Description Language [M+98] and is the language that is used in the international planning systems competition. It has been implemented for IPP to participate in this competition.

[7] BRL stands for Basic Resource Representation Language.

move(*?from, ?to:room*)
:precondition *robby-at(?from) , $traveled_distance* $\leq$ *100*
:effect *robby-at(?to),* $\neg$ *robby-at(?from),*
$traveled_distance += distance(?from,?to)

PDDL allows for arbitrary function-symbol-free first-order formulas containing implication, disjunction, conjunction, negation, existential and universal quantification in the preconditions and effect conditions of an operator, as well as in the description of the goals. The exact definition of the PDDL language can be found in [M+98]. A typical example of a PDDL operator in the original syntax is the following:

assemble
:parameters *(?part ?whole - assembly)*
:precondition

```
(and (forall (?res - resource)
            (imply (requires ?whole ?res) (committed ?res ?whole)))
     (or (part-of ?part ?whole) (transient-part ?part ?whole))
     (available ?part)
     (forall (?prev - assembly)
            (imply (assemble-order ?prev ?part ?whole) (incorporated ?prev ?whole))))
```

:effect

```
(and (incorporated ?part ?whole)
     (not (available ?part))
     (when (and (not (exists (?p - assembly)
                         (and (part-of ?p ?whole)
                              (not (= ?p ?part))
                              (not (incorporated ?p ?whole)))))
                (not (exists (?tp - assembly)
                         (and (transient-part ?tp ?whole)
                              (incorporated ?tp ?whole)))))
     (and (complete ?whole) (available ?whole))))
```

With a few transformations [GK97], a PDDL operator can be translated into an equivalent set of STRIPS actions. For IPP, the transformations are used to simplify precondition, effect conditions, and effects into sets of ground literals, but the conditional effects are preserved, because they are handeled directly by the planner:

- First-order formulas in preconditions, effect conditions, and effects are equivalently transformed into disjunctive prenex normal form.
- Due to the finite domain assumption (remember that D is a finite set of typed objects) quantified formulas can be easily expanded into ground formulas. A universal quantifier is translated into a conjunction, e.g., a formula $\forall\ ?x : p(?x)$ is translated into $p(a) \wedge p(b) \wedge p(c)$ if the variable $?x$ has a domain containing the three constants a, b, c. An existential quantifier is translated into a disjunction, e.g., $\exists\ ?x : p(?x)$ would be translated into $p(a) \vee p(b) \vee p(c)$.

- Disjunctive preconditions $\text{pre}_1 \vee \text{pre}_2$ lead to new operators each containing one of the preconditions, but inheriting the same original effect. Correspondingly, disjunctive effect conditions lead to new conditional effects within the same operator.[8]

- A first-order goal description is replaced by a new atomic goal and an operator is added that has the original goal description as its preconditions and the new atomic goal as its effect.

Considering their expressivity, both languages can express planning problems that are represented in STRIPS and ADL. Beyond this, PDDL includes extensions to deal with numeric constraints in planning problems and to describe hierarchical task network planning tasks. Both extensions cannot be handled by IPP. BRL offers an extension to deal with resource-constrained planning problems, which is not contained in PDDL.

7 The Limitation of ADL Representations

Consider the small planning problem depicted in Figure 1. A robot has to carry a number of balls from one location A to another location B. The robot's ability to grasp and transport objects is very limited, i.e., it is more than unlikely that it can carry more than 2 objects at a time. This means, its transportation capacity can be considered as a limited resource a plan has to respect.

Figure 1: A robot scenario involving resources.

In the small robot example, ADL can still be used to represent the problem. The two gripper hands are modeled as objects, each ball is given a name, and the two locations are objects, too. A predicate **free** is used to keep track whether the robot is holding an object in a gripper hand or not. This yields the following problem representation for an example of 6 balls:

room: *A, B*
object: *Ball1, Ball2, Ball3, Ball4, Ball5, Ball6*
gripper: *left, right*

initial: *robby-at(A), free(left), free(right),* $\forall$ *?x:object at(?x, A)*
goal: $\forall$ *?x:object at(?x, B)*

[8] Note that disjunctive effects are not allowed in PDDL because the language excludes nondeterminism.

move(*?from, ?to:room*)
:precondition *robby-at(?from)*
:effect *robby-at(?to),* $\neg$ *robby-at(?from).*

pick-up(*?obj:object, ?room:room, ?gripper:gripper*)
:precondition *at(?obj, ?room), robby-at(?room), free(?gripper)*
:effect *carry(?obj, ?gripper),* $\neg$ *at(?obj, ?room),* $\neg$ *free(?gripper).*

drop(*?obj:object, ?room:room, ?gripper:gripper*)
:precondition *carry(?obj, ?gripper), robby-at(?room)*
:effect *at(?obj, ?room), free(?gripper),* $\neg$ *carry(?obj, ?gripper).*

IPP generates the following plan to solve this problem:

```
time step  0: pick(Ball1,A,left) pick(Ball2,A,right)
time step  1: move(A,B)
time step  2: drop(Ball1,B,left) drop(Ball2,B,right)
time step  3: move(B,A)
time step  4: pick(Ball3,A,left) pick(Ball4,A,right)
time step  5: move(A,B)
time step  6: drop(Ball3,B,left) drop(Ball4,B,right)
time step  7: move(B,A)
time step  8: pick(Ball5,A,left) pick(Ball6,A,right)
time step  9: move(A,B)
time step 10: drop(Ball5,B,left) drop(Ball6,B,right)
```

Not very surprisingly, the runtime of IPP explodes when the number of balls increases. Two problems are caused by such a pure logical encoding:

1. The planning problem is *over-specified*, because each ball and each gripper is a named individual. Thus, the planner has to assign a specific ball to a specific gripper at a specific time step. For this particular planning task, the goals are reachable for the first time in time step 3, but a solution is not found until time step 11. This means that the planner needs to perform a complete search between time steps 4 to 10 before it can find out that the problem is unsolvable at these time steps. During this search, all possible permutations of ball configurations are tried.

2. The plan is *too specific* because it orders balls and assigns them to a specific gripper. When executing such a plan on a real robot, plan execution becomes much more complicated because of the need to perform a very precise object identification. Instead of grasping an arbitrary ball with any free gripper, a specific choice and assignment is forced by the plan.

Besides this, many practical planning problems involve to consider quantities of resources that can hardly be expressed in logic. Typical resource-constrained planning problems occur in logistics domains. For example, flying from one airport to another requires a certain amount of fuel and the transportation capacity of vehicles is of course limited.

It is widely agreed that planning systems need to be able to solve resource-constrained problems in order to become practically relevant. Therefore, the BRL planning language was designed to support the representation of resource information and IPP was extended to handle problems stated in BRL.

8 The Language BRL

The basic resource representation language BRL is an extension of ADL and offers the following general features:

- resource requirements of actions can be specified in preconditions,
- resource effects of actions can occur in its unconditional effects,
- database query schema support a compact representation of resource requirements and effects.

The planning algorithm allows IPP to generate plans that satisfy *resource constraints*. It does not yet enable the system to flexibly optimize the use of resources in a plan. This comes mainly from two reasons: First, very different optimization functions can be imagined when resources play a role in a planning task. Second, a different type of search algorithm is needed that takes into a account the desired optimization function. Currently, there are no techniques available to support the representation and the use of optimization functions in a classical planning system. The current IPP planning algorithm guarantees that all resource constraints, which are specified in the actions or as a goal, are satisfied by the plan. A plan is generated such that any constraint is satisfied, but it is impossible to guarantee that a plan minimizing or maximizing resource consumption is found. Such an extension is subject to future work.

8.1 Actions that Provide, Produce, and Consume Resources

In order to reason about resources, a planning operator description is extended in the following way:

1. Following the ordinary logical preconditions *resource requirements* can be specified.
2. Effect descriptions are extended to specify which of the resource variables is *provided*, *produced*, or *consumed* by the action.
3. Database query schemata allow for a compact representation of resource requirements and resource effects.

As an example, consider the following actions that represent an extended version of the *gripper* domain from the beginning of this chapter.

The BRL encoding also uses the **move** operator from the ADL encoding, but additionally keeps track of the distance the robot has traveled, which it computes from a database of distances between the various rooms. The idea behind this simple operator is to avoid that the robot runs out of power. It can only move to another room if it has traveled less than 100 units, otherwise it needs to recharge its batteries, which simply resets the traveled distance back to 0.

The picking and dropping of a particular ball using a particular gripper is replaced with the more abstract **pick-a-ball** and **drop-a-ball** operators. The resource variable *$gripper* is used to record the number of balls that the robot holds in the grippers and the resource variables *$balls(A)* and *$balls(B)* record the number of balls in each room.

move*(?from, ?to:room)*
:precondition *robby-at(?from) , $traveled_distance $\leq$ 100 - distance(?from,?to)*
:effect *robby-at(?to), $\neg$ robby-at(?from),*
$traveled_distance += distance(?from,?to)

pick-a-ball*(?room:room)*
:precondition *robby-at(?room), $gripper $\leq$ 1, $balls(?room) $\geq$ 1*
:effect *$gripper += 1,*
$balls(?room) -= 1

drop-a-ball*(?room:room)*
:precondition *robby-at(?room), $gripper $\geq$ 1*
:effect *$gripper -= 1,*
$balls(?room) += 1

recharge*(?room:room)*
:precondition *robby-at(?room)*
:effect *$traveled_distance := 0*

The operator **move** is a *producer* of *$traveled_distance*. It specifies that the robot can only move from a room *?from* to a room *?to* if it is in *?from* and has traveled less than 100 units.[9] As an effect of this operator, the robot is in *?to*, but no longer in *?from*, and its traveled distance is *increased relatively* to the value it has in the state, in which the operator is applied. BRL allows the distinguished binary predicates $=, \leq, \geq$, which mean that the current amount of a resource variable $\$x$ specified as the first argument must be either *equal*, *lower-or-equal*, or *greater-or-equal* the arithmetic expression following as the second argument. An arithmetic expression can contain a database query schema such as *distance(?x,?y)* together with an arithmetic term built out of the four basic mathematical operations and rational numbers, e.g., $3 * distance(?x, ?y) + 6.33$. When the operator gets instantiated, the query schema is instantiated into a database query such as *distance(RoomA, RoomB)* for which the specific value is found in the associated database.

Pick-a-ball is a *producer* of *$gripper*, i.e., it *increases* its value relative to the value this resource has in the state, in which this operator is executed. The operator is also a *consumer* of *$balls(?room)*, because it *decreases* the number of balls in a room relative to its current value. **Drop-a-ball** shows the inverse effects. It is a *consumer* of *$gripper* and a *producer* of *$balls(?room)*.

Recharge is a *provider* of *$traveled_distance*, which recharges the robot's batteries in order to avoid that it runs out of power, i.e., it sets the traveled distance back to 0. A provider of a resource assigns a specific value to a resource *independently* of the value it has in the state where the operator is applied.

All effects are restricted to positive numbers. For example, one cannot declare *$gripper += -500*, i.e., that an action is a producer, but the instruction would in fact decrement the value of the resource.

[9]For each resource variable a unit measure is assumed that remains implicit in the planner. In the current implementation, IPP does not support conversion from one unit into another such as inches into meters, for example.

8.2 Planning Problems Involving Resources

The specification of a planning problem includes the usual declaration of constants and their types, i.e., only the two rooms remain as the only logical constants

```
room:  A B;
```

and the database information (since no connection to a real database is implemented yet) such as

```
database:  distance(B,A)    5
           distance(A,B)    5 .
```

The specification of the initial state specifies the usual logical facts and either assigns exact values or an initial interval to *all* resource variables that are involved in the planning problem. In the example, a logical fact specifies that the robot is in room A and the initial values of the resource variables specify that it is holding no balls, 6 balls are in room A, and 0 balls are in room B.

initial: *$gripper= 0, $balls(A)= 6, $balls(B)=0, robby-at(A), $traveled_distance = 0*

The goal specification comprises a single resource goal only, namely that the number of balls in room B should be equal to 6:

goal: *$balls(B)=6*

8.3 A Semantics for BRL

As a next step, a meaning and semantics is assigned to resource expressions. The formal semantics also allows to formally prove properties of the IPP planning algorithm for resource-constrained planning problems. The basis for the semantics is the following normal form of resource-constrained actions:

Definition 17 *A resource-constrained action in BRL normal form has the following structure*

$$\begin{aligned} o :\ & \mathit{pre}_0, \mathit{req}_0 \\ & \phi_0 : \mathit{eff}_0, \mathit{res}_0 \\ & \phi_1 : \mathit{pre}_1 \Rightarrow \mathit{eff}_1 \\ & \vdots \\ & \phi_n : \mathit{pre}_n \Rightarrow \mathit{eff}_n \end{aligned}$$

where pre_0 *are the logical preconditions of the action o,* eff_0 *are its unconditional logical effects, and* ϕ_1 *to* ϕ_n *are its conditional logical effects.* req_0 *are the resource requirements and* res_0 *are the unconditional resource effects, which must be unique, i.e., for each resource variable* $\$x$ *only one effect can be specified.*

The logical components of this action are interpreted as usual. It remains to interpret its resource requirements req_0 and its resource effects res_0. Furthermore, a meaning has to be assigned to resource requirements that are allowed to occur in state descriptions.

- A resource requirement is an expression of the form

$$< rvar >< comp >< number >$$

where $< rvar >$ is a resource variable name, $< comp >$ is an arithmetic comparison being either $=$ (*equal*), $\leq$ (*lower-or-equal*), or $\geq$ (*greater-or-equal*), and $< number >$ is a rational number $z \in \mathbb{Q}$ resulting from the simplification and instantiation of the database query schema and the arithmetic expressions.

- A resource effect is an expression of the form

$$< rvar >< ass >< number >$$

where $< rvar >$ is a resource variable name as above, $< ass >$ is one of the following assignment operators $:=, +=, -=$, and $< number >$ is a positive rational number $z \in \mathbb{Q}$ resulting from the simplification and instantiation of the database query schema and the arithmetic expressions.

The interpretation of a resource variable in a state is a rational number. Similarly, to the logical variables, which are assigned the values $\top$ or $\bot$, a resource variable is assigned a rational number. For example, a state S can thus be characterized by the set $\{a = \top, \neg b = \top, c = \bot, \$x = 7.56, \$y = 15\}$.

Definition 18 *Let P denote the set of all ground literals. Let $\$X$ be the set of all resource variables. A state S is a set of ground literals and a set of pairs $\langle \$x_i, z_i \rangle$ assigning a rational number $z_i \in \mathbb{Q}$ to each resource variable $\$x_i \in \$X$.*

$$S \subseteq P \cup (\$X \times \mathbb{Q})$$

A state description thus consists of two parts—the logical part S_P consisting of a set of ground literals and the resource part $S_\$$ consisting of the resource variable/value-pairs. $S_\$$ is required to be consistent, i.e., each $\$x_i \in \$X$ occurs at most once in the state description and is assigned a unique value.

BRL allows *resource requirements* in the form of equations or inequations to occur in the *initial state*, the *goals*, and the preconditions of an action.

Definition 19 *The interpretation I of*

- *an equation $x = z$ is $I(\$x = z) = \langle \$x, z \rangle$,*
- *an inequation $\$x \geq z$ is $I(\$x \geq z) = \{\langle \$x, z_i \rangle\}$ with $z \leq z_i \leq +\infty$,*
- *an inequation $\$x \leq z$ is $I(\$x \leq z) = \{\langle \$x, z_i \rangle\}$ with $-\infty \leq z_i \leq z$.*

Note that an inequation describes a set of states. If a state description or resource requirement contains several inequations for the same resource, the intersection of the interpreting sets yields the possible values for the resource.

Resource effects in actions induce state-dependent or state-independent changes in the values of resource variables. Note that the logical preconditions and effects are completely independent of the resource requirements and resource effects of an action. When applying a BRL action to a state, the logical effects only change the set of ground literals that characterize the state, while the resource effects only change the values of the resource variables. Therefore, the result of applying an action can be defined using two independent functions. In Section 4, Definition 12 dealt with the logical part. The subsequent Definitions 21 and 23 deal with the resource part of the state and the action only.

First, one needs a definition that specifies under which conditions a state satisfies a resource requirement.

Definition 20 *A state $S_\$$ with $\langle \$x, z\rangle \in S_\$$ satisfies a resource requirement $req_0(o, \$x)$ of an action o on resource variable $\$x$ if and only if z is one of the values that interpret the resource requirement, i.e.,*

$$S_\$ \models req_0(o, \$x) \ \textit{iff}\ z \in I(req_0(o, \$x))$$

$S_\$$ satisfies all resource requirements of an action o (written $S_\$ \models req_0(o)$) iff it satisfies the requirements on each resource variable.

Definition 21 *Let O be a set of resource-constrained actions given in the normal form as defined in Definition 17, let O^* be all sequences over O, and Res be a function from states and sequences of actions to states.*

$$\mathsf{Res} : (\$X \times \mathbb{Q}) \times O^* \longrightarrow (\$X \times \mathbb{Q})$$

The result of applying a sequence containing the single action o to a state $S_\$$ is defined as

$$\mathsf{Res}(S_\$, \langle o\rangle) = \begin{cases} \langle \$x_i, z'\rangle & \textit{if } S_\$ \models req_0 \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

For each resource variable $\$x_i$ its new value z' is obtained as follows. Let z be the value of $\$x_i$ in $S_\$$, i.e., let $\langle \$x_i, z\rangle \in S_\$$ hold, then[10]

$$z' = \begin{cases} z + z_o & \textit{if } res_0(o) = \$x_i += z_o & o \textit{ is producer of } \$x_i \\ z - z_o & \textit{if } res_0(o) = \$x_i -= z_o & o \textit{ is consumer of } \$x_i \\ z_o & \textit{if } res_0(o) = \$x_i := z_o & o \textit{ is provider of } \$x_i \\ z & \textit{otherwise} & o \textit{ does not affect } \$x_i \end{cases}$$

The result of applying a sequence of more than one action to a state is recursively defined as

$$\mathsf{Res}(S_\$, \langle o_1, \ldots, o_n\rangle) = \mathsf{Res}(\mathsf{Res}(S_\$, \langle o_1, \ldots, o_{n-1}\rangle), o_n).$$

[10]Remember that there is only one effect per resource variable allowed to occur in the action effects, i.e., for each resource variable exactly one of the cases must apply.

When a parallel set of actions O is applied, a resource can be affected by several resource effects, each originating from another action $o \in O$. For example, considering an arbitrary choice of actions, one action could consume the resource, while another one would produce it. However it is very difficult to imagine what the result of applying such general choices of actions would be. Therefore, a parallel set of actions is restricted such that it yields a unique resulting state when applied, i.e., it needs to be *resource-compatible* with respect to its resource effects.

Definition 22 *Two actions o_i and o_j are* resource-compatible *wrt. a resource $\$x$ iff one of the following conditions is satisfied:*

1. *If o_i is a provider of $\$x$, then o_j is not allowed to affect $\$x$, and vice versa.*
2. *If o_i is a consumer of $\$x$, then o_j must either also be a consumer of $\$x$ or not affect $\$x$, and vice versa.*
3. *If o_i is a producer of $\$x$, then o_j must either also be a producer of $\$x$ or not affect $\$x$, and vice versa.*

A set of actions is resource-compatible iff all actions in the set are pairwise resource-compatible wrt. all resource variables.

Note that a resource-compatible set can only contain a set of producers or a set of consumers or a *single* provider for a given resource variable.

Definition 23 *Let $\mathsf{R}^\star$ be a function from states and sequences of sets of actions to states*

$$\mathsf{R}^\star : (\$X \times \mathbb{Q}) \times (2^O)^* \longrightarrow (\$X \times \mathbb{Q})$$

The result of applying the empty set of actions to a state is defined as $\mathsf{R}^\star(S_\$, \langle\rangle) = S_\$$. In the case of a sequence containing exactly one non-empty set of parallel actions $O = \{o_1, \ldots, o_n\}$ we obtain

$$\mathsf{R}^\star(S_\$, \langle O \rangle) = \begin{cases} \langle \$x_i, z' \rangle & \textit{if } \forall\, o_i \in O : S_\$ \models \textit{req}_0(o_i) \textit{ and } O \textit{ is resource-compatibel} \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

For each resource variable $\$x_i$ its new value z' is obtained as follows. Let z be the value of $\$x_i$ in $S_\$$, i.e., let $\langle \$x_i, z \rangle \in S_\$$ hold, then

$$z' = \begin{cases} z + \Sigma_{i=1}^{i=n}\, z_{o_i} & \textit{if } \$x_i \textit{ is produced by } O \\ z - \Sigma_{i=1}^{i=n}\, z_{o_i} & \textit{if } \$x_i \textit{ is consumed by } O \\ z_{o_i} & \textit{if } \$x_i \textit{ is provided by } O \\ z & \textit{otherwise} \end{cases}$$

The recursive case of a sequence $O = \langle O_1, \ldots, O_n \rangle$ is defined as

$$\mathsf{R}^\star(S_\$, \langle O_1, \ldots, O_n \rangle) = \mathsf{R}^\star(\mathsf{R}^\star(S_\$, \langle O_1, \ldots, O_{n-1} \rangle), O_n).$$

If a state description contains an inequation, a set of resource-constrained states $\mathcal{S}_\$ = \{S_{\$_1}, S_{\$_2}, \ldots, S_{\$_n}\}$ is specified. To define the result of applying a sequence containing a parallel set of actions to a set of resource-constrained states, the function R is introduced.

Definition 24 *Let* R *be a function from sets of states and sequences of sets of actions to sets of states*

$$\mathsf{R} : 2^{(\$X \times \mathbb{Q})} \times (2^O)^* \longrightarrow 2^{(\$X \times \mathbb{Q})}$$

The result of applying the empty set of actions is defined as

$$\mathsf{R}(\mathcal{S}_\$, \langle\rangle) = \mathcal{S}_\$$$

For the sequence containing exactly one set of parallel actions we define

$$\mathsf{R}(\mathcal{S}_\$, \langle Q\rangle) = \begin{cases} \bigcup_{S_{\$_i} \in \mathcal{S}_\$} \mathsf{R}^\star(S_{\$_i}, Q) & \textit{if } \forall\, i : \mathsf{R}^\star(S_{\$_i}, Q) \textit{ is defined} \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

The recursive case is defined analogously to the $\mathsf{R}^\star$ *function.*

Finally, the logical part and the resource-constrained part of a state and an action can be put together. The Res and Res functions define what happens when a single action is applied to a single state. The R and R functions define what happens when a set of parallel actions is applied to a set of states.

Definition 25 *Let* $S = S_P \cup S_\$$ *be a resource-constrained state. Let* o *be a BRL action. The result of applying the sequence containing the single action* o *to* S *is defined as*

$$\mathsf{Result}(S, \langle o\rangle) = \begin{cases} Res(S_P, \langle o\rangle) \cup \mathsf{Res}(S_\$, \langle o\rangle) & \textit{if both are defined} \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

For the sequence containing a parallel set of actions O *applied to a set of states* $\mathcal{S} = \{(S_{P_1} \cup S_{\$_1}), (S_{P_2} \cup S_{\$_2}), \ldots\}$ *one obtains:*

$$\mathsf{Result}^\star(\mathcal{S}, \langle O\rangle) = \begin{cases} \bigcup_{S_i \in \mathcal{S}} \left(R(\{S_{P_i}\}, \langle O\rangle) \times \mathsf{R}(\{S_{\$_i}\}, \langle O\rangle) \right) & \textit{if both are defined} \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

The recursive case is defined as before.

9 The Search Algorithm

Only very few planning systems exist that have been extended to deal with resource-constrained planning problems. In all attempts, complex models of resources have been merged with complex time models making it almost impossible to obtain complete algorithms showing an acceptable runtime behavior.

An early attempt of integrating time into a planning system is Allen's interval logic [All83, AKPT91]. Experimental implementations using this formalism exist [AK83], but the algorithms remain incomplete, which can also be explained by the intractability results presented in [Hir95, NB95].

A more recent attempt of dealing with resources and time in a planning system are the planners ZENO [PW94] and IxTeT [GL94, LG95]. ZENO possesses a sound and complete planning algorithm, but cannot generate plans containing more than a dozen actions, although a goal ordering heuristic is used to guide the planner in subgoal selection. IxTeT is slightly faster, but uses an incomplete temporal constraint satisfaction algorithm. In [CO95, CS97], a constraint solver is also proposed as the underlying machinery to find a plan.

BRL is the first attempt to separate the problem of dealing with resource constraints from the problem of having a richer temporal model of actions. The motivation comes mainly from two sources. First, there are many interesting problems for a planner which is able to handle resource constraints. Second, a moderate extension of planning formalisms has turned out to be successful in the past, for example when one thinks of the transition from STRIPS to ADL.

9.1 General Overview

The IPP search algorithm combines the ADL search algorithm to handle logical goals together with *interval arithmetics* to handle resource goals. Interval arithmetics is very simple and does not generate a significant overhead for the planner. The problem lies in the treatment of *resource conflicts*, which opens a huge search space for any planning system, cf. the discussion in [Hir95]. By systematically trying all possible conflict resolution possibilities, a complete search algorithm is obtained.

In a nutshell, the algorithm works as follows: To extract a valid plan from the planning graph, a backward search is performed that generates a plan, which is logically valid. Then this plan is propagated forward in order to determine its effect on the resource variables. If a conflict occurs, conflict-reducing actions are added to the plan. Then the backward search starts again from the time step where the plan has been modified.

The search algorithm is presented using the following terminology:

- $\mathcal{G}_n$ are the logical goals at time step n,
- $\mathcal{R}_n$ are the resource goals at time step n,
- *max* is the last fact level in the graph, i.e., the last action level in the graph has index $max - 1$,
- S_n^G are the logical facts that are satisfied by the set of states that is reached after executing a plan of length $n - 1$,
- S_n^R are the resource values that are satisfied by the set of states that is reached after executing a plan of length $n - 1$,
- S_0^G, S_0^R are thus the facts and resource values, which are satisfied by the initial state(s).

The resource goals are represented using intervals. Remember that they result from resource requirements in the goals and in the preconditions of actions, where they can be specified by inequations. The interpretation of an inequation is a set of possible values for a given resource variable, which can be characterized by the lower and upper bounds on the values. Besides this, the initial and goal states are not required to be unique. In fact when using inequations, one represents a set of valid initial or goal states that differ wrt. the possible values for the resources, but coincide wrt. the valid logical facts. The aim of the planning process is to find a plan that is valid for all initial states and reaches at least one of the goal states. For example, if one specifies $\$x \geq 10$ in the initial state then a plan is seeked that works for all possible values $10, \ldots, \infty$. If one specifies $\$x \geq 10$ in the goal state, it is sufficient to construct a plan that achieves one of the possible values for $\$x$.

The algorithm uses the following inference modules:

- **Choose-actions** receives as input a pair of goals $\langle \mathcal{G}_n, \mathcal{R}_n \rangle$. It returns a set of actions Δ_{n-1} together with the pair of new goals $\langle \mathcal{G}_{n-1}, \mathcal{R}_{n-1} \rangle$ or fails.
- **Add-actions** receives as input a resource goal R_n, a state description S^G_{n-1}, S^R_{n-1}, and a set of actions Δ_{n-1}. It returns a modified set of actions Δ'_{n-1}, and a pair of new goals $\langle \mathcal{G}_{n-1}, \mathcal{R}_{n-1} \rangle$ or fails.
- **Propagate-and-check** receives as input a state description S^R_{n-1} and a set of actions Δ_{n-1}. It computes the new state description S^R_n and performs the test $S^R_n \models \mathcal{R}_n$. It returns true if the test is satisfied, false otherwise.

Figure 2 summarizes how the modules interact. Given the original goals at time step max, **choose-actions** selects a valid set of actions at each time step until it reaches the initial state. If no such set exists (anymore), the planner backtracks to the next higher level in the graph. If **choose-actions** fails at the last level, no plan with max time steps exists and the graph is extended to $max+1$. After having arrived in the initial state (time step 0), the termination test is performed. If the logical goals and the resource goals are satisfied, a valid plan has been found.

If the termination test fails, **propagate-and-check** is activated. It takes the known values of the resource variables at a time step $n-1$, the set of actions that has been selected for this time step and determines the values of the resource variables at time step n. Then it checks if these values satisfy the resource goals at time step n. If this is the case, a valid plan has been found and the search terminates. If the resource goals are not satisfied, conflict-reducing actions are considered by **add-actions**. If such a choice of actions has been found at a time step n, the goals at this time step are recomputed and a new call of **choose-actions** is initiated. It starts from the goals at n and searches for a new plan towards the initial state because any previous choices of actions between n and 0 have to be reconsidered because of the recomputed goals.

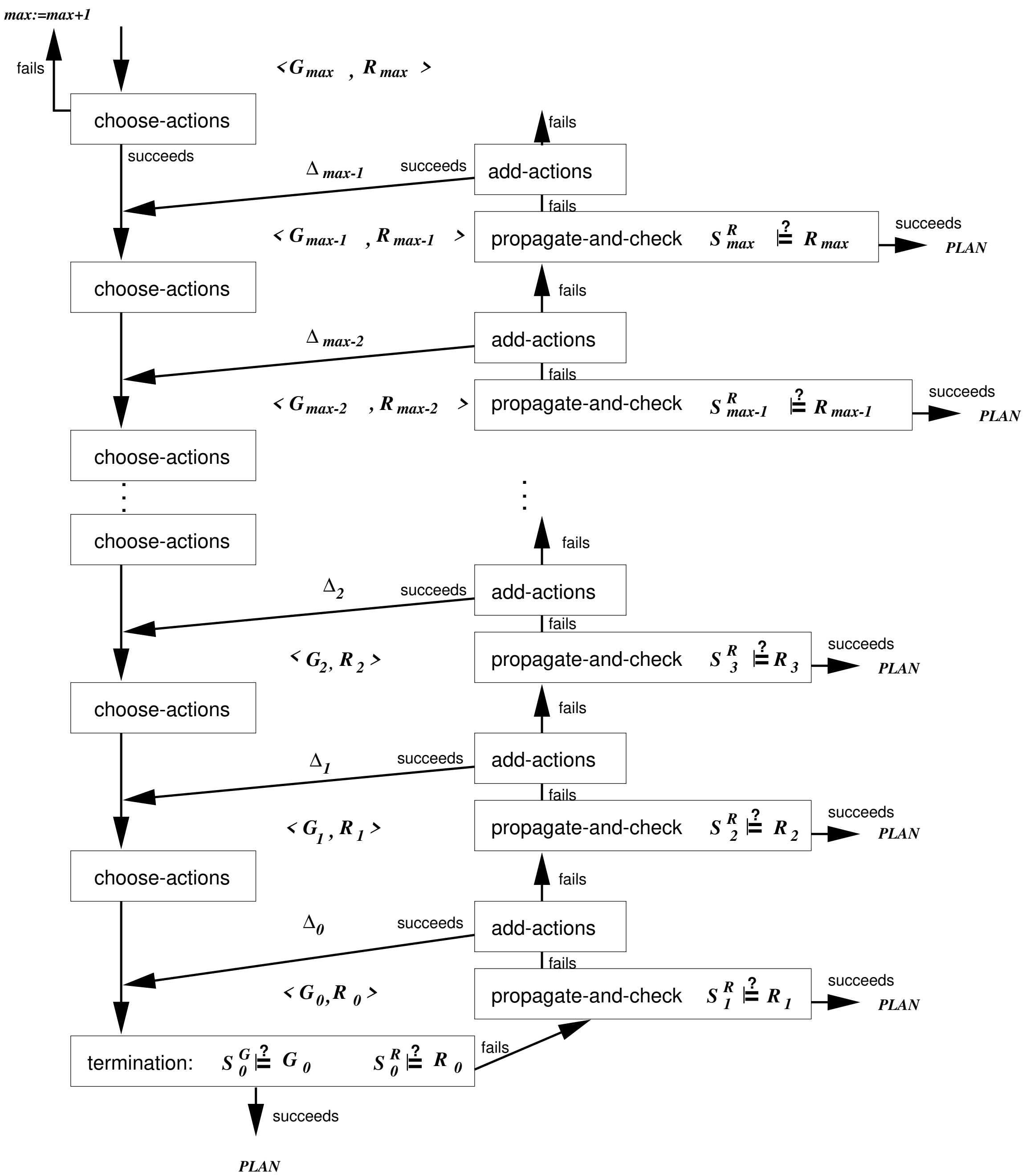

Figure 2: Overview over the search algorithm.

If no choice of conflict-reducing actions can be found by **add-actions** at a time step n, either because no such choice exists at all or all choices have already been tried, then **propagate-and-check** resumes at the next time step $n+1$ and checks if the plan can be repaired at a time step higher in the graph. In the following, each of the modules is specified in detail.

9.2 Choosing a Valid Set of Actions

Given a set of logical goals $\mathcal{G}_n$ and a set of resource goals $\mathcal{R}_n$ at time step n, the search algorithm selects a set of actions Δ_{n-1} such that Δ_{n-1} achieves the logical goals and is resource-compatible.

As described, in Chapter 4, IPP selects a minimal set of conflict-free actions Δ_{n-1} to achieve the logical goals. Additionally, it performs the following tests on Δ_{n-1}:

1. All actions $o \in \Delta_{n-1}$ must be pairwise resource-compatible (Definition 22).
2. If there is a resource goal $\mathcal{R}_n(\$x) = [min_n(\$x), max_n(\$x)]$ for a resource variable $\$x$ and Δ_{n-1} contains a provider of $\$x$, i.e., an action has an effect of the form $\$x := z$ then $z \in [min_n(\$x), max_n(\$x)]$ must hold.

If one of the tests is violated, Δ_{n-1} is discarded and another choice of actions is tried. If no choice is possible, which satisfies all conditions, IPP backtracks.[11]

Given a valid choice Δ_{n-1}, the goals $\mathcal{G}_n$ and $\mathcal{R}_n$, the new goals $\mathcal{G}_{n-1}$ and $\mathcal{R}_{n-1}$ at the next time step are computed as follows. The logical goals $\mathcal{G}_{n-1}$ are obtained as usual from the preconditions, effect conditions of selected effects of the actions in Δ_{n-1}. As before, IPP performs the conflict resolution to resolve potential conflicts between conditional effects.

In order to determine the set of new resource goals $\mathcal{R}_{n-1}$, IPP works as follows:

Let $\mathcal{R}_n(\$x) = [min_n(\$x), max_n(\$x)]$ be the resource goal for resource variable $\$x$ at time step n. Since Δ_{n-1} must be resource-compatible, one of the following situations must occur.

Case 1: No action $o \in \Delta_{n-1}$ affects $\$x$. The resource goal remains unchanged:

$$\mathcal{R}_{n-1}(\$x) := \mathcal{R}_n(\$x)$$

Case 2: Δ_{n-1} contains a single provider of $\$x$. The new resource goal for $\$x$ becomes arbitrary:

$$\mathcal{R}_{n-1}(\$x) := [-\infty, +\infty]$$

Case 3: Δ_{n-1} contains a set of producers $o_1, \ldots o_k$ of $\$x$. Their combined effects push the goal interval to the left:

$$\begin{aligned} min_{n-1}(\$x) &:= min_n(\$x) - \Sigma_{i=1}^{i=k}\, z_{o_i} \\ max_{n-1}(\$x) &:= max_n(\$x) - \Sigma_{i=1}^{i=k}\, z_{o_i} \end{aligned}$$

Case 4: Δ_{n-1} contains a set of consumers $o_1, \ldots o_k$ of $\$x$. Their combined effects push the goal interval to the right:

$$\begin{aligned} min_{n-1}(\$x) &:= min_n(\$x) + \Sigma_{i=1}^{i=k}\, z_{o_i} \\ max_{n-1}(\$x) &:= max_n(\$x) + \Sigma_{i=1}^{i=k}\, z_{o_i} \end{aligned}$$

Finally, the resource requirements of each action $o \in \Delta_{n-1}$ are added to $\mathcal{R}_{n-1}$ and IPP tests if the resulting new resource goals are consistent. This is the case, if a non-empty resource goal is obtained for each resource variable:

[11] **Choose-actions** at a time step n can either be activated from **choose-actions** or from **add-actions** at time step $n + 1$.

$$\forall\ \$x\ \ \mathcal{R}_{n-1}(\$x) \cap_{o \in \Delta_{n-1}} \text{req}_0(o, \$x) \not\models \emptyset$$

Technically, the goal interval that has been computed by IPP so far is intersected with all intervals representing the resource requirements of the selected actions wrt. the same resource variable. If the intersection becomes empty, the new resource goals and the resource requirement of at least one action from the set are contradictory and they cannot be satisfied at the same time. For example, if IPP has determined $\mathcal{R}_{n-1}(\$x) = [3, 8]$, but one of the selected actions requires $\$x \geq 10$ in order to be applicable, then $[3, 8] \cap [10, +\infty] = \emptyset$.

If the test fails, Δ_{n-1} is discarded and another choice is tried. If the test succeeds, the new resource goals are obtained from the intersection:

$$\forall\ \$x\ \ \mathcal{R}_{n-1}(\$x) := \mathcal{R}_{n-1}(\$x) \cap_{o \in \Delta_{n-1}} \text{req}_0(o, \$x)$$

The algorithm terminates successfully in the initial state iff

1. $S_o^G \models \mathcal{G}_o$, i.e., the logical goals at time step 0 hold in the initial state.
2. $S_o^R \models \mathcal{R}_o$, i.e., all possible initial values for the resource variables fall within the required range at time step 0 (Definition 20).

Technically, IPP tests if each interval, which is obtained from the resource requirements in the initial state, is a subinterval of the interval which represents the corresponding resource goal at time step 0.

After successful termination, the plan $\mathcal{P} = \langle \Delta_0, \ldots, \Delta_{max-1} \rangle$ is returned as a solution for the planning problem.

If the termination test fails, because the logical goals are unsatisfied, IPP backtracks and tries to find a another plan. If the termination test fails, because only the resource goals are unsatisfied, but the logical goals are satisfied, then IPP enters the forward propagation phase and tries to fix possible resource conflicts.

9.3 Forward Propagation and Resource Conflicts

The forward propagation is entered if IPP has found a logically valid plan, but the termination test on resources has failed. This means, the planner has constructed a complete plan, which is sound with respect to the logical goals at each time step. To be sound with respect to the resource goals, at each time step n the condition $S_n^R \models \mathcal{R}_n$ must be satisfied.

The purpose of **propagate-and-check** is to verify exactly this condition. Given a state description S_{n-1}^R and a set of actions Δ_{n-1}, the new state description S_n^R is determined using the $\mathsf{R}^\star$ function from Definition 23. The propagation can return an undefined result if the resource requirements of an action in Δ_{n-1} are violated. In this situation, the set of actions cannot be executed and is invalid. Control is referred back to **choose-actions**, which needs to select another Δ_{n-1}.

If the propagation yields a defined state, $S_n^R \models \mathcal{R}_n$ is tested based on Definition 20.

If the test succeeds, IPP returns a plan $\mathcal{P} = \langle \Delta_0, \ldots, \Delta_{max-1} \rangle$ as a solution for the planning problem. As will be shown in the soundness proof below, in this case, $S_k^R \models \mathcal{R}_k$

follows for all time steps $n \leq k \leq max$ in the graph. Technically, S_n^R and $\mathcal{R}_n$ both contain one interval for each resource variable $\$x$ and **propagate-and-check** tests if the interval from S_n^R is contained in the interval from $\mathcal{R}_n$. If no resource goal is specified for a resource variable, $\mathcal{R}_n$ contains the interval $[-\infty, +\infty]$ for this variable, i.e., the test is trivially satisfied.

If the test fails, IPP analyses the resource conflict. Let us consider the possible conflict situations in more detail. Let $[smin_n(\$x), smax_n(\$x)]$ be the interval of possible values for $\$x$ at time step n after having executed the plan prefix $\langle \Delta_0, \ldots, \Delta_{n-1} \rangle$. Let $[min_n(\$x), max_n(\$x)]$ be the goal interval for this variable at time step n.

A necessary condition for the test to succeed is that the length of the state variable must be lower or equal the length of the goal variable.

$$|smax_n(\$x) - smin_n(\$x)| \leq |max_n(\$x) - min_n(\$x)|$$

If this condition is violated, the planner needs to shorten the length of the state interval. The only possible way to succeed, is to add a provider of $\$x$ because producers or consumers can only "push" intervals into a certain direction.

Thus if $|smax_n(\$x) - smin_n(\$x)| \not\leq |max_n(\$x) - min_n(\$x)|$, IPP activates **add-actions**, which tries to add a provider o with effect $\$x := z$ to Δ_{n-1} that needs to satisfy the following conditions:

1. $z \in [min_n(\$x), max_n(\$x)]$,
2. $\Delta_{n-1} \cup \{o\}$ must be resource-compatible and
3. $\text{eff}_0(o) \cup \mathcal{G}_n \not\models \perp$, the logical unconditional effects of o must be consistent with the logical goals $\mathcal{G}_n$.

If one of the conditions is violated o is not a valid choice and the conflict cannot be resolved. In this situation, control is referred back to **choose-actions**, which needs to select another Δ_{n-1}.

If $|smax_n(\$x) - smin_n(\$x)| \leq |max_n(\$x) - min_n(\$x)|$ is satisfied, but the state interval is not contained in the goal interval, two possible conflict situations can occur, see Figure 3.

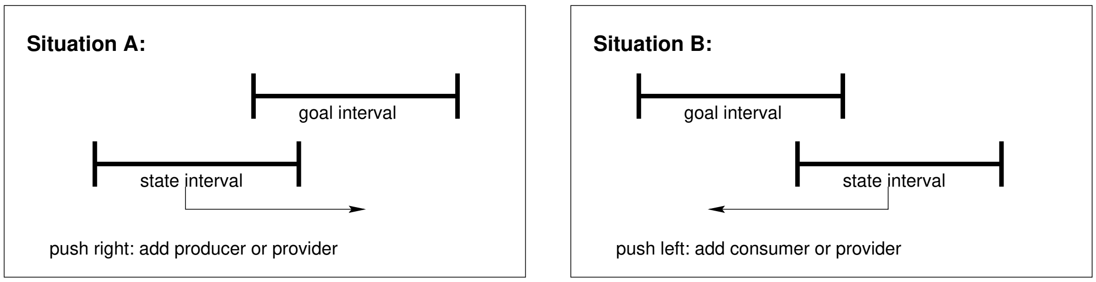

Figure 3: Resolution of resource conflicts.

In situation A, the state interval lies "left" of the goal interval and needs to be pushed to the right. To achieve such a push, a producer or provider of $\$x$ must be added.

In situation B, the state interval lies "right" of the goal interval and needs to be pushed to the left. To achieve such a push, a consumer or provider of $\$x$ must be added.

A provider must satisfy the conditions as stated above in order to be a valid choice. For a producer or consumer, only the last two conditions must be satisfied, i.e., it must be resource-compatible and its unconditional logical effects must be consistent with the goals.

This proceeds yields an extended set of actions Δ_{n-1}, which reduces at least one resource conflict. For this set, the new resource goals and the new logical goals are computed as before and IPP replans given these new goals at time step $n-1$.

If the new goals are inconsistent, **add-actions** needs to try another choice of repairing actions at n. If no choice exists that reduces the conflicts, it fails and activates **propagate-and-check** at time step $n+1$, i.e., it tries to resolve the conflict at a time step higher in the graph towards the goals.

9.4 An Example

Let us consider a very simple example to illustrate the complex behavior of the search algorithm. Given the following set of operators

move*(?from, ?to:room)*
:precondition *robby-at(?from)*
:effect *robby-at(?to), ¬ robby-at(?from)*

pick-a-ball*(?room:room)*
:precondition *robby-at(?room), $gripper $\leq$ 1, $balls(?room) $\geq$ 1*
:effect *$gripper += 1*
$balls(?room) -= 1

drop-a-ball*(?room:room)*
:precondition *robby-at(?room), $gripper $\geq$ 1*
:effect *$gripper -= 1*
$balls(?room) += 1

how does IPP construct a plan for the problem of transporting one ball to room B:

initial: *$gripper= 0, $balls(A)= 6, $balls(B)=0, robby-at(A)*
goal: *$balls(B)=1*

Since the logical goals are empty, the planning graph is only expanded until time step 0, i.e., only the initial fact level is built. But the termination test on the resource goals fails because $\$balls(B) = 0 \not\models \$balls(B) = 1$, i.e., $[0,0] \not\subseteq [1,1]$. No **propagate-and-check** is possible, because there is only one level in the graph. Thus, the graph is expanded further. Let us assume that IPP has failed on a graph of depth 2, which is now expanded to depth 3. Starting from this level, the empty set of actions is selected at each time step, which leave the resource goals unchanged. This yields the following situation:

level	logical goals/actions	$\$gripper$	$\$balls(A)$	$\$balls(B)$
$F3$:	$\emptyset$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[1,1]$
$A2$:	$\emptyset$			
$F2$:	$\emptyset$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[1,1]$
$A1$:	$\emptyset$			
$F1$:	$\emptyset$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[1,1]$
$A0$:	$\emptyset$			
$F0$:	robby-at(A)	$[0,0] \subseteq [-\infty,+\infty]$	$[0,6] \subseteq [-\infty,+\infty]$	$[0,0] \not\subseteq [1,1]$

Again, the termination test in the initial state fails. **Propagate-and-check** now determines the possible values for the resources at time step 1 given the empty plan, which yields $\$gripper \in [0,0]$, $\$balls(A) \in [0,6]$, and $\$balls(B) \in [0,0]$.

A problem occurs on $\$balls(B)$, which is required to be in [1,1]. Since the state interval lies left of the goal interval, only a producer or provider is considered by **add-actions**. The only possible choice is **drop-a-ball(B)**, which is added to the empty set of actions at level 0. The new logical goals result with *robby-at(B)* and the new resource goals result with $\$balls(B) \in [0,0]$ and $\$gripper \in [1,+\infty]$. Since the logical goals are not satisfied in the initial state, backtracking over **add-actions** is triggered. No other choice is possible and therefore **propagate-and-check** is activated at the next time step, where the same conflict situation occurs. Again, **drop-a-ball(B)** is the only possible choice. Starting from fact level 1 to replan, **move(B,A)** is the only choice of actions that yields a logically correct plan in the initial state:

level	logical goals/actions	$\$gripper$	$\$balls(A)$	$\$balls(B)$
$F3:$	$\emptyset$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[1,1]$
$A2:$	$\emptyset$			
$F2:$	$\emptyset$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[1,1]$
$A1:$	**drop-a-ball(B)**			
$F1:$	robby-at(B)	$[1,+\infty]$	$[-\infty,+\infty]$	$[0,0]$
$A0:$	**move(B,A)**			
$F0:$	robby-at(A)	$[0,0] \not\subseteq [1,+\infty]$	$[0,6] \subseteq [-\infty,+\infty]$	$[0,0] \subseteq [0,0]$

Unfortunately, the resource goals are not satisfied because this time, a conflict on $\$gripper$ occurs. This conflict cannot be repaired by **add-actions** between time steps 0 and 1 and thus, backtracking over **drop-a-ball(B)** as a conflict-reducing action is triggered finally. Since no other choice is possible, **add-actions** at time step 1 fails and activates **propagate-and-check** at fact level 2. Again, the same conflict situation occurs, as IPP still has the empty choice of actions at each time step. The conflict triggers again the **drop-a-ball(B)** action, but this time at action level 2. At action level 1, either a no-op or **move(B,A)** can be considered to achieve the logical goals. After selection of **move(B,A)**, a no-op could be considered for execution in the initial state, which would yield a logically valid plan:

level	logical goals/actions	$\$gripper$	$\$balls(A)$	$\$balls(B)$
$F3:$	$\emptyset$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[1,1]$
$A2:$	**drop-a-ball(B)**			
$F2:$	robby-at(B)	$[1,+\infty]$	$[-\infty,+\infty]$	$[0,0]$
$A1:$	**move(B,A)**			
$F1:$	robby-at(A)	$[1,+\infty]$	$[-\infty,+\infty]$	$[0,0]$
$A0:$	**no-op**			
$F0:$	robby-at(A)	$[0,0] \not\subseteq [1,+\infty]$	$[0,6] \subseteq [-\infty,+\infty]$	$[0,0] \subseteq [0,0]$

Because of the conflict on $\$gripper$, a producer of this resource needs to be considered for conflict resolution at action level 0. Two possibilities exist: **pick-a-ball(A)** and **pick-a-ball(B)** of which only the former will succeed and yield a solution to the planning problem. The termination test is finally satisfied:

level	logical goals/actions	$\$gripper$	$\$balls(A)$	$\$balls(B)$
$F3:$	$\emptyset$	$[-\infty, +\infty]$	$[-\infty, +\infty]$	$[1, 1]$
$A2:$	**drop-a-ball(B)**			
$F2:$	robby-at(B)	$[1, +\infty]$	$[-\infty, +\infty]$	$[0, 0]$
$A1:$	**move(B,A)**			
$F1:$	robby-at(A)	$[1, +\infty]$	$[-\infty, +\infty]$	$[0, 0]$
$A0:$	**pick-a-ball(A)**			
$F0:$	robby-at(A)	$[0, 0] \subseteq [0, 0]$	$[0, 6] \subseteq [-\infty, +\infty]$	$[0, 0] \subseteq [0, 0]$

10 Soundness and Completeness

This section proves the search algorithm for resource-constrained planning problems sound and complete.

Since the resource-constrained part of the language and the logical part of of the language are independent of each other, we only prove soundness for the resource goals wrt. the resource-constrained part of the initial state.

Theorem 1 (Soundness) IPP*'s search algorithm is sound for BRL. Let* $(\mathcal{O}, D, S_0^R, \mathcal{R})$ *be a resource-constrained planning problem and* $\mathcal{P} = \langle Q_1, Q_2, \ldots, Q_{max-1}\rangle$ *be the plan, which is returned by* IPP*. Then*

$$\mathsf{R}^\star(S_0^R, \langle Q_1, Q_2, \ldots, Q_{max-1}\rangle) \models \mathcal{R}$$

Since all resource variables are independent of each other, it is sufficient to show the soundness of the plan for one of the resource goals that is arbitrarily chosen. The proof proceeds by induction over the length of the plan.

Proof:

Let $[min(\$x), max(\$x)]$ be the resource goal for the resource variable $\$x$. Let $[min_0(\$x), max_0(\$x)]$ be the available range for $\$x$ in the initial state. The proof shows that for each $z \in [min_0(\$x), max_0(\$x)]$ holds

$$\mathsf{R}^\star(z, \langle Q_1, Q_2, \ldots, Q_{max-1}\rangle) \in [min(\$x), max(\$x)]$$

Due to the soundness of IPP for ADL, we can presuppose that $\mathcal{P}$ is logically sound.

base case: $\mathcal{P} = \langle\emptyset\rangle$

For the empty plan, the $\mathsf{R}^\star$ function returns z. Given the empty set of actions and the resource goal $[min(\$x), max(\$x)]$, IPP uses $[min(\$x), max(\$x)]$ as the current resource goal in the initial state on which it performs the termination test. Because a solution was returned, this test must have succeeded, i.e., $z \in [min(\$x), max(\$x)]$ holds.

Induction step: $\mathcal{P} = \langle Q_1, Q_2, \ldots, Q_{max-1}\rangle$

Let $1 \leq k \leq max - 1$. The induction hypothesis assumes that for each z holds $\mathsf{R}^\star(z, \langle Q_1, Q_2, \ldots, Q_{k-1}) \in [min_k(\$x), max_k(\$x)]$, i.e., the resource goals at time step k are satisfied. It remains to show that $\mathsf{R}^\star(z', Q_k) \in [min_{k+1}(\$x), max_{k+1}(\$x)]$ follows for some arbitrary $z' \in [min_k(\$x), max_k(\$x)]$.

Note that Q_k is a set of resource-compatible actions and thus, only one of the subsequent cases can apply. Furthermore, $\mathsf{R}^\star$ is defined when applied to some arbitrary $z' \in [min_k(\$x), max_k(\$x)]$ and Q_k because the search algorithm guarantees that each z' satisfies the resource requirements of each action in Q_k by construction.

Case 1: No action in Q_k affects $\$x$.

In this case, the search algorithm propagates $[min_{k+1}(\$x), max_{k+1}(\$x)]$ to level k and merges them with the resource requirements of each action in Q_k, which can only yield a narrower subinterval, i.e.,

$$[min_k(\$x), max_k(\$x)] \subseteq [min_{k+1}(\$x), max_{k+1}(\$x)]$$

Because of $\mathsf{R}^\star(z', Q_k) = z'$, the proposition follows in this case.

Case 2: Q_k contains a single provider wrt. $\$x$.

The search algorithms determines $[min_k(\$x), max_k(\$x)]$ solely from all resource requirements of the actions in Q_k. This ensures that $\mathsf{R}^\star(z', Q_k)$ is defined and yields $min_{k+1}(\$x) = max_{k+1}(\$x) = z''$ for a provider effect of the form $\$x := z''$. Because of the fact that the search algorithm selects only those providers for which $z'' \in [min_{k+1}(\$x), max_{k+1}(\$x)]$ holds, the proposition follows.

Case 3: Q_k contains a set of consumers wrt. $\$x$.

The search algorithm takes $[min_{k+1}(\$x), max_{k+1}(\$x)]$ and shifts it by the constant amount of $+\Sigma_{i=1}^{i=l} z_{o_l}$. Intersecting this interval with the resource requirements yields the resource goals at k as a subinterval of this shifted interval. The $\mathsf{R}^\star$ function is defined such that this subinterval will be shifted back by exactly the same constant amount, i.e., $z' \in [min_{k+1}(\$x), max_{k+1}(\$x)]$ follows immediately.

Case 4: Q_k contains a set of producers wrt. $\$x$.

The proof proceeds analogously, but with $-\Sigma$ instead of $+\Sigma$.

∎

Theorem 2 (Completeness) IPP*'s search algorithm is complete for BRL. Let* $(\mathcal{O}, D, S_0^R, \mathcal{R})$ *be a planning problem, for which a solution* $\mathcal{P} = \langle Q_1, Q_2, \ldots, Q_{max-1}\rangle$ *of length* $k = |Q_1| + |Q_2| + \cdots + |Q_{max-1}|$ *exists. Then* IPP *returns a plan solving this problem after atmost* k *expansion steps of the graph.*

Proof: First, if IPP finds a solution after k' expansion steps with $k' < k$, then nothing needs to be proven. Second, we note that IPP cannot terminate before or after k steps as no termination test has been included yet. Thus, it needs to arrive at the kth level of the graph if it has not found a solution earlier.

Completeness follows from two observations:

1. IPP is complete for ADL, i.e., given a set of logical goals $\mathcal{G}_n$ at some time step n, it systematically tries all possible choices of actions Δ_{n-1} to achieve these goals. This was proven in [Koe99].

2. IPP's resolution of resource conflicts is complete, i.e., it tries all possible ways of reducing a conflict at all time steps that necessarily need to be considered in the graph. This will be proven in the following.

Given some goals $\mathcal{G}_n$ and $\mathcal{R}_n$ and a possible choice of actions Δ_{n-1}, which achieves $\mathcal{G}_n$, the following situations can occur:

IPP is not visiting the action level $n-1$ again. Then it must have found a plan and Δ_{n-1} must be a set of actions, which are part of the solution.

IPP backtracks to action level $n-1$. This can happen in two situations. First, at some time step between $n-1$ and 0, the logical goals have been unsatisfiable. Thus, a new set Δ_{n-1} must be selected and completeness follows from IPP's completeness for ADL. Second, at some time step between $n-1$ and 0, the resource goals have been unsatisfiable. The backtracking mechanism returns to level $n-1$ with S^R_{n-1}, the range of values for each resource that is achieved at time step $n-1$ by the current plan. When applying Δ_{n-1} given this possible range of resource values the following can happen:

1. S^R_n becomes undefined. In this case, Δ_{n-1} is not applicable and cannot be made applicable by further adding actions to it, because some of the resource requirements of actions in Δ_{n-1} are violated. Thus Δ_{n-1} must be an invalid choice given the plan prefix $\langle \Delta_0, \ldots, \Delta_{n-2} \rangle$. Another Δ_{n-1} must definitely be selected as no solution can be obtained from the current choice.

2. S^R_n is defined, but does not satisfy $\mathcal{R}_n$. This happens when the goal interval does not cover the state interval, because either the state interval is shifted or it is too long. In order to remedy a shift, only shifting effects can be used, i.e., only a producer or consumer, which shifts the state interval into the right direction, needs to be considered. If the state interval is too long, its size can only be reduced by resetting the resource value using a provider. Any other actions cannot reduce a conflict and therefore do not need to be considered. Given that IPP systematically tries all possibilities of adding a single conflict-reducing action, it can completely enumerate all possible conflict resolutions for a given Δ_{n-1} and will find a solution of atmost length k.

If no conflict-reducing choice exists anymore, i.e., the state interval cannot be influenced such that at least one conflict is reduced, the only way to eliminate the conflict is by changing the goals $\mathcal{R}_n$, because these have been shown to be unsatisfiable since all possible plan prefixes have been systematically explored. Thus, conflict resolution continues at all time steps n to max and this way will explore all possible ways of changing the goals $\mathcal{R}_n$.

■

11 Outlook

This section discusses the issue of termination and how conditional resource effects can be handled. Finally, it is argued that further extending the planning language such that resource dependencies can be expressed, will pose a general problem for any backward-searching planner.

11.1 Termination

IPP uses UBTree in order to effectively handle the subset memoization problem for sets of logical goals [HK99]. For resource goals, a completely different technique is necessary as goals are no longer represented by sets of ground atoms, but instead by sets of intervals over rational numbers. Figure 4 illustrates the sets of resource goals

$$\{\$money \in [0, 270], \$debts \in [133.33, 750]\}$$

and

$$\{\$gripper \in [2, 2], \$balls(B) \in [4, 4], \$balls(A) \in [0, 6]\}$$

In the example, the resource goals represent a rectangle in the 2-dimensional space bounded by the individual resource intervals. In the second example, the goals represent a line in the 3-dimensional space because two of the resources do not range within intervals, but require an exact value.

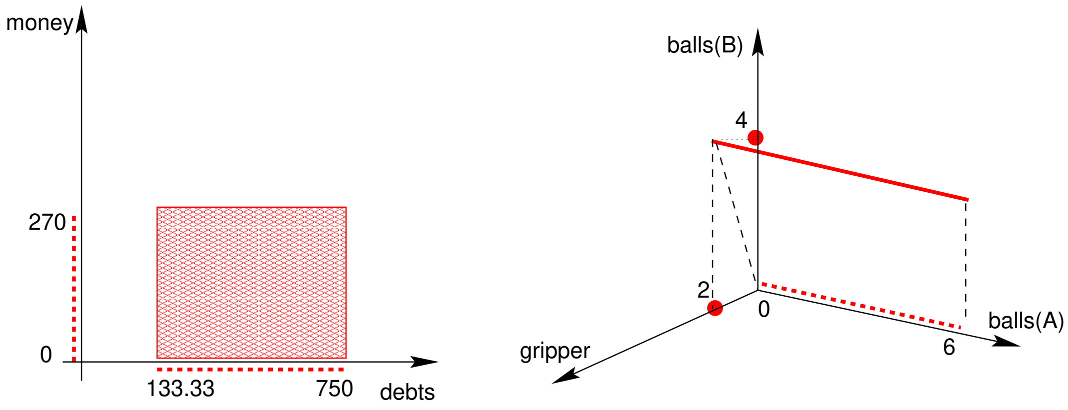

Figure 4: Graphical illustration of example resource goals.

In the general case of n resources, the resource goals at a particular time step spawn an n-dimensional iso-oriented hypercube. Obviously, the memoization and matching of resource goals requires to solve a geometrical covering problem in the n-dimensional space.

For a given planning problem, the planner needs to construct and maintain a set of hypercubes at each time step representing the unsolvable resource goals. If a new resource goal is constructed, it's hypercube has to be matched against the ones already stored. If the query hypercube is completely contained within the space spawned by the stored hypercubes, then the new resource goals are unsatisfiable. Figure 5 illustrates the problem in the 2-dimensional space.

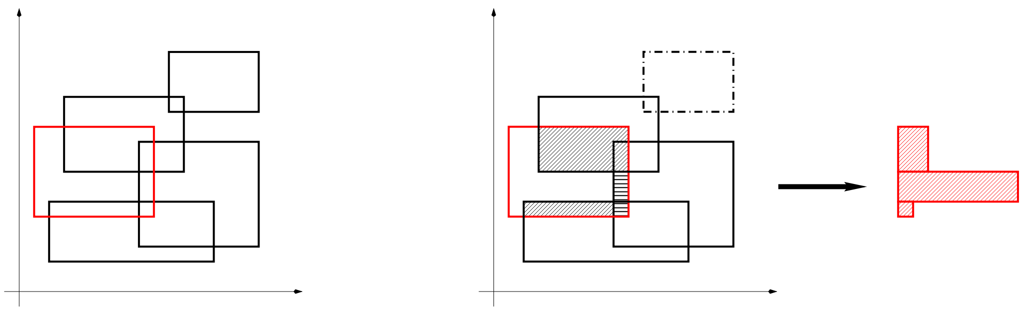

Figure 5: The resource goal memoization problem in the 2-dimensional space.

For any BRL planning problem, the planner will usually construct a set of logical goals together with a set of resource goals at each time step. Such a combined goal is unsolvable if UBTree returns an unsolvable subset of logical goals and the geometrical matcher returns a complete covering of the hypercube representing the resource goals. The covering cannot be formed by arbitrary hypercubes that have been memoized. Instead, only those hypercubes can be considered that have associated logical goal sets, which are subsets of the current goals. In the most simple case, a subset of the logical case has an associated hypercube that completely covers the current resource goals. Then one can immediately decide that the current goal set is unsolvable.

One can easily imagine that thousands or even millions of goal intervals need to be stored for larger planning problems. When matching a query hypercube, only a fraction of them will intersect with it, but in any case an efficient matching technique is required. For example, in Figure 5 the upper right-most rectangle is irrelevant. The development of such a technique lies beyond the scope of this thesis.[12]

The memoization of resource goals is a necessary prerequisite to reduce the search space for the planner and to develop a termination test, which provides something analogue to the termination test for the logical goals. Currently, no termination of IPP on general BRL planning problems can be guaranteed. A termination test based on memoized resource goals is subject to future work.

11.2 Extension to Conditional Resource Effects

One can easily combine both language extensions that have been considered so far and obtain conditional resource effects of the form $\mathrm{pre}_i, \mathrm{req}_i \Rightarrow \mathrm{eff}_i, \mathrm{res}_i$.

Until now, the result of applying a single action o in a state S becomes undefined if its logical preconditions or its resource requirements are unsatisfied in S. For ADL, it was defined that unsatisfied effect conditions just prevent the effect to occur, but they cannot yield an inconsistent state.

For conditional resource effects, one could adopt the same solution. For example, if a conditional effect requires $\$x \geq 5$, then it would not happen if $\$x < 5$ holds. However such a semantics requires to perform a conflict resolution over conditional resource requirements and seems to yield some unintuitive results.

Another possibility is that the resulting state becomes undefined if a resource condition is not satisfied, but the logical effect condition is satisfied. This requires that either all resource conditions of an action must be established by a plan or a conditional effect must be prevented over its logical effect condition only. For this approach, the standard ADL conflict resolution would be sufficient and it also seems to yield more plausible resulting states.

The following operator **enter-bus** illustrates the consequences of both approaches.

enter-bus(*?s:station*)
:precondition *bus-at(?s), door-closed*
:effect *door-open*
$\forall$ *p passenger :* *waiting(?p,?s),* $\Rightarrow$ *boarded(?p)* $\wedge$ $\$seats - = 1$

The operator describes what happens when a bus stops at a station. It will open the doors and all waiting passengers will enter the bus and diminish the number of available seats. Instantiation of this operator with two passengers yields the following action

[12]Currently, the problem is the topic of a diploma thesis that is supervised by the author. First complexity-theoretic results show that that the problem is co-NP-complete.

enter-bus*(sea-shore)*
:precondition *bus-at(sea-shore), door-closed*
:effect *door-open*
$\quad$ *waiting(max,sea-shore)* $\Rightarrow$ *boarded(max)* $\wedge$ $\$seats- = 1$
$\quad$ *waiting(mary,sea-shore)* $\Rightarrow$ *boarded(mary)* $\wedge$ $\$seats- = 1$

Assuming that $\$seats$ must be greater or equal 0 in all states, one obtains $\$seats \geq 2$ as a resource precondition for this action if both passengers are allowed to board. If only one seat is available in the state where the plan schedules the action for execution, then under the first semantics, some passengers cannot board and either Max or Mary have to remain outside the bus. Which passenger is affected, remains unclear because it depends on the linearization of the conditional effects. If *boarded(mary)* is a goal, then any linearization should yield a plan, i.e., in the case of one seat the problem is unsolvable, since Max could enter first. There is also no reasonable way of preventing one of the conditional effects because both affect the same resource.

Under the second semantics, the state becomes undefined if there is a passenger waiting for which no free seat is available because the individual conditional resource effects have to be combined. Each resource effect must therefore be respected when planning and the result of executing the action is independent of an execution order for the conditional effects.

The search algorithm can be easily adapted to deal with conditional resource effects. Instead of selecting a producer, consumer, or provider, one selects a particular producing, consuming, or providing effect. Its resource requirements are merged into the new resource goals and its logical effect conditions are merged into the new logical goals. Conflicting conditional effects can only be prevented by unsatisfied logical effect conditions, while resource requirements always need to be satisfied if the logical effect conditions are satisfied.

11.3 Resource Dependencies and Backward Search

BRL limits all resource effects to be positive rational numbers, i.e., resources change independently of each other. However, one can imagine that many planning problems require to represent resource dependencies. For example, the following **get-gas** operator defines what happens at a gas station when we want a fully-tanked car. The gas level in the tank is set to 40, but the price we have to pay depends on what has been in the tank before getting gas.

get-gas*(?x:location)*
precondition: *at(?x)*
effect: *\$gas := 40*
$\quad$ *\$price += - 1.5 * \$gas + 60*

When this action is selected at a particular level, one can define a set of resource-compatible actions such that no other effect in the choice of actions is allowed to change the dependency variable \$gas, i.e., its value depends solely on the actions that change it *before* the **get-gas** action takes place. But during a backward search, these actions are not known at the time point when the new goal for $\$price$ must be determined, i.e., the goal for $\$price$ is unknown. In fact, one needs to generate a *complete* plan (candidate) without knowing what the goal for $\$price$ is. After arriving in the initial state, the termination test cannot be performed as no goal for $\$price$ is available. This must first be determined by forward-propagation. The unknown goal will most likely cause the planner to generate many complete non-solutions.

Here, a forward-search algorithm seems to be more appropriate. Any resource dependency can be completely resolved at each time step because the exact value of the dependency variable is known. Furthermore, conflict resolution becomes easier because no propagation of conflicts over time steps is necessary, but a standard backtracking strategy will be sufficient. A first sketch of such a forward-search algorithm has been described in [BL98], but practical tests showed that this approach leads to extremely inefficient domain-independent planners. No effective method to guide a forward-searching planning algorithm is available. Besides this, no forward-planning method for ADL has ever been worked out - although this seems to be rather straightforward.

References

[AK83] J. Allen and J. Koomen. Planning using a temporal world model. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 741–747, Karlsruhe, Germany, August 1983.

[AKPT91] J. Allen, H. Kautz, R. Pelavin, and J. Tenenberg. *Reasoning about Plans.* Morgan Kaufmann, USA, 1991.

[All83] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[BF97] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2):279–298, 1997.

[BL98] A. Blum and J. Longford. Probabilistic planning in the Graphplan framework. unpublished manuscript, 1998.

[CO95] A. Cesta and A. Oddi. DDL.1: A formal description of a constraint representation language for physical domains. In M. Ghallab and A. Milani, editors, *Proceedings of the 3rd European Conference on Planning*, pages 341–352. IOS Press, Amsterdam, Washington, Tokyo, 1995.

[CS97] A. Cesta and C. Stella. A time and resource problem for planning architectures. In Steel [Ste97], pages 117–129.

[GK97] B. Gazen and C. Knoblock. Combining the expressivity of UCPOP with the efficiency of Graphplan. In Steel [Ste97], pages 221–233.

[GL94] M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In Hammond [Ham94], pages 61–67.

[Ham94] K. Hammond, editor. *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems.* AAAI Press, Menlo Park, 1994.

[Hir95] R. Hirsch. Intractability in the Allen and Koomen planner. *Computational Intelligence*, 11(4):553–564, 1995.

[HK99] J. Hoffmann and J. Koehler. A new method to index and query sets. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 462–467. Morgan Kaufmann, San Francisco, CA, 1999.

[Kno94] C. Knoblock. Generating parallel execution plans with a partial-order planner. In Hammond [Ham94], pages 98–103.

[Koe99] J. Koehler. Handling of conditional effects and negative goals in ipp. Technical Report 128, University of Freiburg, Institute of Computer Science, 1999. available at http://www.informatik.uni-freiburg.de/˜ koehler/ipp.html.

[LG95] P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1643–1649. Morgan Kaufmann, San Francisco, CA, 1995.

[M+98] D. McDermott et al. *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Comitee, 1998.

[NB95] Bernhard Nebel and Hans-Jürgen Bürckert. Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra. *Journal of the ACM*, 42(1):43–66, 1995.

[Ped91] E. Pednault. Generalizing nonlinear planning to handle complex goals and actions with context-dependent effects. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 240–245. Morgan Kaufmann, San Francisco, CA, 1991.

[PW94] J. Penberthy and D. Weld. Temporal planning with continuous change. In *Proceedings of the 12th National Conference of the American Association for Artificial Intelligence*, pages 1010–1015. AAAI Press, MIT Press, 1994.

[Ste97] S. Steel, editor. *Proceedings of the 4th European Conference on Planning*, volume 1348 of *LNAI*. Springer, 1997.